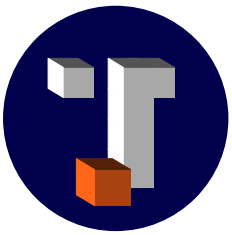
UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN



JERARQUÍA DINÁMICA DE ESQUEMAS PARA LA GENERACIÓN DE COMPORTAMIENTO AUTÓNOMO

TESIS DOCTORAL

José María Cañas Plaza

Ingeniero de Telecomunicación

Madrid, 2003

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS TELEMÁTICOS
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE MADRID

TESIS DOCTORAL

JERARQUÍA DINÁMICA DE ESQUEMAS PARA LA GENERACIÓN DE COMPORTAMIENTO AUTÓNOMO

Autor:
José María Cañas Plaza
Ingeniero de Telecomunicación

Director:
Vicente Matellán Olivera
Doctor en Informática

Tutor:
Gregorio Fernández Fernández
Doctor Ingeniero de Telecomunicación

Madrid, 2003

TESIS DOCTORAL:	Jerarquía dinámica de esquemas para la generación de comportamiento autónomo
AUTOR:	José María Cañas Plaza
DIRECTOR:	Vicente Matellán Olivera
TUTOR:	Gregorio Fernández Fernández

El tribunal nombrado para juzgar la Tesis arriba indicada, compuesto por los siguientes doctores:

PRESIDENTE:	Dr. D. Juan Antonio de la Puente Alfaro
VOCALES:	Dr. D. Pablo Bustos García de Castro
	Dr. Alberto Ruiz García
	Dr. Joaquín López Fernández
SECRETARIO:	Dr. D. Ángel Álvarez Rodríguez

acuerda otorgarle la calificación de

Madrid, de de 2003

El Secretario del Tribunal

Agradecimientos

Me resulta dificil acabar esta tesis sin la sensación de que más que un punto concreto es un proceso. Un camino largo, incierto y duro, pero que debe ser recorrido sin atajos para que sea valioso. Aunque sigue habiendo viaje por delante, momento es de parar a merendar, disfrutar de esta recompensa puntual, y de reconocer la ayuda que muchas personas me han prestado hasta aquí.

En primer lugar agradecer a Vicente su acogida en los momentos delicados, sus sabios consejos (que siempre me superan), y el entendimiento al que hemos llegado; ¡estoy muy contento de que trabajemos juntos!. A Reid, por su hospitalidad año tras año y por enseñarme que trabajar en robótica además de reflexionar sobre estupendas ideas, consiste en bajar a la arena de la programación para materializarlas.

Segundo, dar las gracias a los amigos con quienes comparto curiosidad por estos temas y que en mayor o menor medida han influido en mis ideas a través de conversaciones "profundas": Pablo, Lía, Alberto, Pedro Enrique, Jorge, Joaquín ... Da gusto poder hablar de cosas interesantes con vosotros :-) En este mismo sentido, aunque de manera absolutamente desconocida para ellos, quiero expresar mi gratitud hacia quienes me han servido de referencia e iluminan el camino con su perspectiva: Arkin, Brooks, y en especial a Lorenz y Tinbergen, que tantos años de ventaja nos sacan aún a los robóticos.

Tercero, quiero recordar a los amigos, por su cariño y por estar siempre ahí: Javi, Marilú, Dani, Maribel, JuanCarlos, Josean, Ricardo, Manuel, Rodrigo, Victor, los amigos de Teleco.... Muy en especial a Lía, por tantos esfuerzos pasados juntos. También a los demás colegas del Instituto de Automática por el tiempo disfrutado allí: Montse, Pablo Yañez, Mauricio Tupia, Angela, Fernando, Oscar. A los amigos de Carnegie Mellon, por su afecto y por las largas noches de laboratorio: Joseph, Shyjan, Greg, Enrique. A los compañeros del GSYC, por enriquecerme con sus vastos conocimientos y por compartir una apuesta vital similar.

Cuarto, corresponder a mis alumnos del grupo, porque a pesar de requerir muchas atenciones y esfuerzo, son un soplo de aire fresco. También quiero reconocer a los caminos equivocados, por enseñarme como no se deben hacer las cosas, aunque fuera por la vía más dura.

Finalmente dedicar este trabajo a mi familia, y a Susana, por su amor infinito, su impaciencia y el pragmatismo al que me exponen siempre.

José María
Madrid, Noviembre 2003

Resumen

La robótica móvil es una rama importante de la robótica, y persigue la construcción de robots con objetivos, capaces de desenvolverse por sí mismos en entornos desconocidos y parcialmente cambiantes sin necesidad de supervisión. Típicamente los robots móviles constan de sensores, actuadores y procesadores. Aunque los avances tecnológicos de los últimos años nos han dado los sensores más precisos, los mejores actuadores y los procesadores más potentes, la manera de combinarlos para generar comportamiento inteligente sigue siendo un problema abierto.

La arquitectura conceptual de un robot es la organización de sus capacidades de actuación, percepción y procesamiento para generar un repertorio de comportamientos autónomos. Ella determina las conductas observables que exhibe un robot móvil. La arquitectura se materializa en unos programas, los cuales dan vida al hardware del robot y realizan el enlace entre los datos sensoriales y los comandos a los actuadores. Cuando el robot sólo tiene que hacer una cosa, o la información sensorial está directamente relacionada con la tarea, entonces su programación es relativamente sencilla. En este caso, casi cualquier organización resuelve el problema. Sin embargo, la cuestión se complica cuando se quiere integrar en el mismo robot un abanico amplio de comportamientos, y cuando hay sensores que entregan un volumen desbordante de datos, y no todos son interesantes para la tarea. Entonces la organización se convierte en un factor crítico para que el robot pueda resolver de modo autónomo sus tareas.

La arquitectura que proponemos en esta tesis es la *Jerarquía Dinámica de Esquemas*, JDE. En ella se concibe el comportamiento autónomo como la imbricación de dos cuestiones diferentes: la percepción y el control. La percepción se encarga de elaborar la información interesante para la tarea y/o la situación actual. El control toma decisiones de actuación sobre esa información, de manera que la conducta observable sea inteligente, es decir, finalista y sensible al entorno. Ambas se cuantizan en unas unidades pequeñas llamadas *esquemas*. Así, tendremos esquemas perceptivos y esquemas de actuación.

Cada esquema es un flujo iterativo con algún objetivo, que se puede activar o desactivar a voluntad y cuyo funcionamiento se puede modular a través de parámetros. Siguiendo la inspiración de los modelos etológicos JDE propone una colección de esquemas para generar comportamiento, todos ellos funcionando en paralelo y organizados en jerarquía. Un esquema perceptivo puede despertar a otros para componer su propio estímulo desde la información que perciben sus hijos. Un esquema de actuación activa a los esquemas perceptivos que elaboran la información que él necesita para determinar correctamente la actuación adecuada (percepción orientada a la tarea). Además puede emitir comandos a los actuadores, o si es un esquema complejo, despertar a varios esquemas hijo para que le ayuden a conseguir su propio objetivo. Todos sus hijos competirán continuamente por el control porque sólo el ganador tomará el control de los actuadores. Este diseño jerárquico concurrente permite generar fenómenos de atención, percibir estímulos complejos y acotar la complejidad de decidir qué hacer en el siguiente instante (selección de acción). Además facilita la reutilización de partes y la escalabilidad del sistema a comportamientos más complejos.

La arquitectura ha sido implementada en una infraestructura software que materializa los mecanismos propuestos. Además con ella se ha diseñado una colección de esquemas perceptivos y de actuación que permiten generar un conjunto de conductas observables en diferentes plataformas robóticas y percibir varios estímulos relevantes para robots de interiores.

Abstract

Mobile robotics is an important field inside robotics, it aims to make purposive robots, able to move themselves autonomously through unknown and partially dynamic environments. Tipically mobile robots are endowed with sensors, actuators and processors. Technology improvements have delivered most precise sensors, best actuators and fastest processors ever. Despite that, the way they should be combined to provide intelligent artificial behavior remains an open issue.

The conceptual architecture of a robot is the organization of its actuation, perception and processing capabilities with the aim of generating a whole set of autonomous behaviors. It determines the observable behaviors exhibited by the mobile robot. The architecture is implemented in some programs, which establish the connection between the sensor data and the actuator commands. When the robot has to do only one task, or when the sensor information is closely related to the task at hand, then the robot programming is a relatively easy thing to do. In such a case, almost any organization does the job. Nevertheless, when a whole set of behaviors are to be included in the robot habilities, or when there are sensors delivering a huge amount of data, most of them irrelevant to the task, then the architecture becomes much more difficult to deal with. In such a case, the organization becomes a critical issue for the robot to be able to carry out all its tasks.

The proposed architecture in this thesis is named *Dynamic Schema Hierarchy* (in Spanish, JDE). It sees the autonomous behavior as a combination of two different topics: perception and control. Perception builds the relevant information from the sensor data, according to the goals and the environment situation. Control makes actuation decisions based on such information in order to generate intelligent observable behavior, that is, goal oriented and sensitive to the environment situation. Both, perception and control, are decomposed into small units called *schemas*. So that, we have perceptive schemas and actuation schemas.

Each schema is an iterative thread with a specific goal, which can be actioned and deactivated at will, and whose behavior can be tuned through some parameters. Following ideas borrowed from the ethology, JDE proposes a collection of schemas to generate behavior, where all of them are running concurrently and organized into a single hierarchy. A perceptive schema can activate others in order to create and update its own stimulus based on the information set up by its children. An actuation schema actions several perceptive schemas to build the information it requires to make the correct decision about the suitable actuation (task oriented perception). In addition, it can send commands to the actuators or, if it was complex enough, awake other actuation schemas for them to help it to achieve its own goal. All its actuation child schemas will continuously contend, since only the winner will send commands to the actuators. This hierarchical design allows for an attention mechanism, to perceive complex stimuli and to bound the complexity of making the decision about what to do next (action selection). It also easies the code reutilization and the scalability of the whole system to more complex behaviors.

The architecture has been implemented in a software suite which embodies the proposed mechanisms. Using such suite we have developed a whole set of perceptive and actuation schemas, which allow the generation of several observable behaviors on different robotic platforms, and to perceive some relevant stimuli for indoor robots.

Índice general

Capítulo 1

Introducción

> *The early results were like a cold shower. While the pure reasoning programs did their jobs about as well and fast as college freshmen, the best robot-control programs, besides being more difficult to write, took hours to find and pick up a few blocks on a tabletop, and often failed completely, performing much worse than a six-month-old child.*
> Hans Moravec – Robot: Mere machine to transcendent mind, 1999

A grandes rasgos el problema que abordamos en esta tesis es conseguir que un robot se mueva de manera independiente e inteligente en cierto entorno. El marco en el que encuadramos este problema es la robótica, y en concreto la robótica móvil. En las secciones 1.1 y 1.2 presentamos el contexto genérico de la robótica, sus orígenes industriales, algunas aplicaciones relevantes y el contexto más reducido de la robótica móvil. De modo más específico circunscribimos el problema que abordamos en esta tesis a la generación de comportamiento autónomo en robots móviles. En la sección 1.3 definimos algunos términos frecuentes en ese ámbito: comportamiento, arquitectura, autonomía, robot, etc. En particular, presentamos la idea de arquitectura cognitiva del robot, que determina su comportamiento, e introducimos las principales cuestiones que ésta debe resolver. Los objetivos concretos y estructurados de la tesis se presentan en la sección 1.4. También esbozamos la respuesta que proponemos, es decir, la arquitectura que planteamos como solución, y sus características principales en la sección 1.5. Finalmente en la 1.6 se describe la estructura de esta memoria.

1.1. Robótica

En esta primera sección se describe el contexto global en el que se desarrolla esta tesis. Haremos una introducción genérica a la robótica, repasando sus orígenes industriales y recorriendo algunos ejemplos de las aplicaciones en las que se muestra útil.

Quizá lo primero sea introducir qué entendemos por robot. Aunque posteriormente lo definiremos con precisión, de momento podemos asumir que un robot es una máquina que puede sensar la realidad, medir alguna característica suya y además es capaz de interactuar con su entorno, de actuar en él provocando cambios. Estas dos capacidades lo caracterizan. Un robot no es una computadora, es algo más. Utiliza los ordenadores o los microprocesadores como elementos de cómputo, pero un robot tiene más elementos, como los sensores y los actuadores, que lo sumergen y conectan inevitablemente con la realidad física.

En cuanto a su fisonomía, los robots pueden presentarse en múltiples configuraciones. Lo realmente distintivo es su funcionalidad, no su apariencia. Hay robots estáticos, cuya base no se mueve de un punto fijo, y robots móviles que pueden desplazarse por el entorno. Dentro de los móviles hay robots con ruedas y robots con patas. Hay robots humanoides que tienen forma similar al cuerpo humano, algunos que se asemejan a insectos, otros a coches, etc..

En cuanto a su función, el hombre siempre se ha servido de herramientas para su propio beneficio. Herramientas que extendían las capacidades perceptivas y de actuación del ser humano: un microscopio para ver cosas más pequeñas, un telescopio para ver cosas más lejanas, un martillo para golpear con más fuerza, una grúa, etc. Estos utensilios son instrumentos que el hombre maneja y que en cierto

modo extienden el alcance o la fuerza de sus brazos, sus ojos o sus manos. Un paso más en esta línea es la construcción de máquinas que realizan alguna tarea de modo automático. Esta idea ha estado presente en la cultura occidental desde hace al menos dos siglos. De hecho forma parte del corazón mismo de la revolución industrial, que aplica esas máquinas al proceso productivo, a la fabricación de bienes. Por ejemplo, los telares automáticos del siglo XIX realizaban distintos dibujos en la tela entrelazando hilos de diferentes colores, según le indicaban tarjetas perforadas insertadas previamente en el sistema.

Desde comienzos del siglo XX hasta la actualidad la automatización ha ido conquistando terreno, ampliando las tareas que se pueden automatizar, desde las máquinas herramienta que fabrican piezas de modo preciso hasta el piloto automático de los aviones. En estos espacios el papel de las personas se ha desplazado a tareas que la máquina por sí misma no es capaz de realizar o a labores de supervisión.

Dentro de este enfoque utilitarista nació precisamente la palabra *robot*, que proviene de la novela RUR (Rossum's Universal Robots) que el autor checoslovaco Karel Capek publicó en 1923 [Capek, 1923]. En ella dos pequeños seres artificiales de forma humana responden perfectamente a las órdenes de su creador, aunque al final acaban rebelándose contra él. Al referirse a estos seres el autor les llama robots, derivación del vocablo checo *robota* que significa trabajo obligatorio.

Este perfil utilitarista es el que ha calado en la definición de la Real Academia Española [Española, 1992], que define `robótica` como *técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales*. Esos aparatos son los robots, que define como: *(1) ingenio electrónico que puede ejecutar automáticamente operaciones o movimientos muy varios. (2) autómata.*

Desde sus orígenes la robótica ha tenido una motivación industrial que ha impulsado su desarrollo. De hecho, la mayoría de los robots que existen en la actualidad trabajan en fábricas, muchos de ellos en cadenas de fabricación de automóviles. Un hito paradigmático en este escenario son los brazos articulados tipo PUMA (Programmable Universal Manipulator for Assembly). Se utilizan con frecuencia para soldar, pintar, transporte de materiales, etc. tal como ilustra la figura 1.1. Por ejemplo los puntos de soldadura se le dan predefinidos en una lista con la que se programa el recorrido del brazo. Las principales ventajas que ofrece un brazo robotizado frente a un operario humano para estas labores repetitivas son la precisión y la rapidez. Desde el punto de vista del empresario también hay que considerar que los robots no hacen huelga, no se aburren y pueden trabajar 24 horas al día, todos los días del año. Además salvando cierta inversión inicial, su coste de mantenimiento puede ser menor que el equivalente para que la misma labor la realicen operadores humanos.

Figura 1.1: *Robots soldadores en una factoría de coches (izda.) y brazo transportando materiales (dcha.)*

Los robots también aparecen en otros escenarios fuera de la fábrica. En entornos altamente peligrosos para las personas se utilizan robots para reemplazarlas. Por ejemplo, se utilizan robots teleoperados para explorar zonas de alta radiación en el interior de reactores nucleares, para rastrear en las profundidades oceánicas, para sondear volcanes o pirámides, para inspeccionar y desactivar

bombas o minas. En estos casos el robot suele estar equipado con cámaras de video y es guiado remotamente por un operador humano.

En los últimos años está creciendo el uso de robots para tareas agrícolas [Stentz *et al.*, 2002]. Las tareas de fumigación o cosecha son potencialmente automatizables, porque son repetitivas y tediosas, e incluso conllevan cierto riesgo químico para el operario humano. Por ejemplo, hay robots recolectores de tomates, melones, pepinos o champiñones, que utilizan visión artificial para identificar el fruto adecuado [García Pérez, 2004]. Otra aplicación relevante en esta línea son los cosechadores automáticos de cereales, que recorren de modo semiautónomo los inmensos campos de cereales segando y recogiendo el grano.

Las exploraciones espaciales suponen otro campo de aplicación de la robótica. Se han utilizado robots en varias misiones. Por ejemplo, la misión Pathfinder puso al robot Sojourner en la superficie de Marte en 1997. El robot tenía 6 ruedas y básicamente se movía teleoperado desde la Tierra. Tomó varias imágenes, analizó el suelo y algunas rocas del planeta rojo, enviando los resultados a la Tierra. En esta línea la NASA ha invertido y sigue invirtiendo mucho en proyectos que desarrollan robots capaces de desenvolverse en entorno hostil tan lejano (como el programa de telerobotica espacial[1]).

Uno de los campos insospechados por donde está creciendo la oferta robótica es el ocio y el entretenimiento. Por ejemplo, el perrito Aibo de Sony[2] se vende como mascota y los robots humanoides desarrollados por Honda y Sony en los últimos años avanzan en esta línea. También el ladrillo de Lego[3] se vende como juguete imaginativo. En este sentido hay que destacar también la RoboCup[4], que es una competición internacional anual en la que equipos de robots juegan al fútbol en distintas categorías.

También hay tímidos intentos de introducir los robots en el hogar para automatizar tareas domésticas. Una muestra es el aspirador robótico[5] deambula por una habitación de forma autónoma sin chocar con las paredes, tragando pelusas y deslizándose debajo de las camas y los sofás. Sus sensores evitan que choque contra las paredes o los muebles, o se caiga por el hueco de las escaleras.

En general, el uso de los robots va creciendo paulatinamente, aumentando el rango de sus aplicaciones a medida que éstos aumentan en autonomía y funcionalidad. Precisamente la falta de autonomía es el principal cuello de botella que ha impedido el uso generalizado de robots. En la mayoría de las aplicaciones robóticas de hoy día los robots utilizados no tienen autonomía, son teleoperados o autómatas que ejecutan una sola tarea. Aunque algunos escenarios admiten teleoperación, en ciertas aplicaciones la autonomía es un requisito ineludible y en la mayoría, una característica muy ventajosa. Por ejemplo, en robots para exploraciones espaciales la teleoperación se hace demasiado lenta cuando se alejan de la Tierra y las ondas de radio tardan demasiado en llegar. En los robots de entretenimiento la gracia radica en que sean mascotas independientes, con su propias capacidades, y no meras marionetas pasivas. En las tareas agrícolas la ventaja principal se consigue al automatizar de modo completo el laboreo. Con ello se reduce al mínimo la intervención humana, quizás a una sencilla supervisión. Del mismo modo sucede con las tareas domésticas.

En resumen, para realizar tareas y aplicaciones más complejas necesitamos robots móviles que sean completamente autónomos. Precisamente a este ámbito se dirige esta investigación. El avance hacia la autonomía amplía los horizontes y la utilidad de la robótica. Como iremos viendo, este salto en funcionalidad supone resolver o dar respuesta a muchos problemas a los que los sistemas teleoperados no tienen que enfrentarse porque finalmente todas las decisiones las toma el operador humano.

1.2. Robótica móvil

En esta sección delimitamos un poco más el contexto de la tesis, que situamos dentro de la robótica móvil. El objetivo fundamental de este campo es la construcción de robots capaces de desenvolverse por sí mismos en entornos desconocidos y parcialmente cambiantes sin necesidad de supervisión. Conseguir esta autonomía de modo satisfactorio es un problema muy complejo. Un robot móvil autónomo muestra

[1] http://ranier.oact.hq.nasa.gov/telerobotics_page/telerobotics.shtm
[2] http://www.aibo.com
[3] http://www.legomindstorms.com
[4] http://www.robocup.org
[5] http://www.roombavac.com

cierta inteligencia en su comportamiento, porque conlleva que sea capaz de afrontar muy distintas situaciones de modo satisfactorio y pueda reaccionar ante cambios en el entorno sin ayuda del operador humano.

El camino hacia los robots móviles actuales ha sido largo. Los primeros proyectos en este área datan de principios de los años 70, cuando por primera vez se reúnen en una misma plataforma sensores, motores y procesadores [Serradilla, 1997]. Desde entonces se ha ido consolidando como un campo específico dentro de la robótica, en el cual el comportamiento principal es el movimiento y donde hay una especial preocupación por la autonomía. Además la evolución de este campo ha ido definiendo sus propios problemas como la navegación, la construcción de mapas, la localización, etc.

El robot Shakey [Nilsson, 1969], que aparece en la figura 1.2, es un pionero dentro la robótica móvil, y fue construido en 1970 en el Stanford Research Institute (SRI). Tenía una cámara, motores y sensores odométricos. Era capaz de localizar un bloque en su entorno y empujarlo lentamente. El procesamiento de las imágenes se realizaba en un ordenador fuera del robot. Dentro del mismo instituto, un digno sucesor fue el robot Flakey, construido en 1984 con numerosos avances tecnológicos sobre Shakey, y que ya incluía en su interior ordenadores para realizar a bordo cualquier cómputo necesario para su comportamiento.

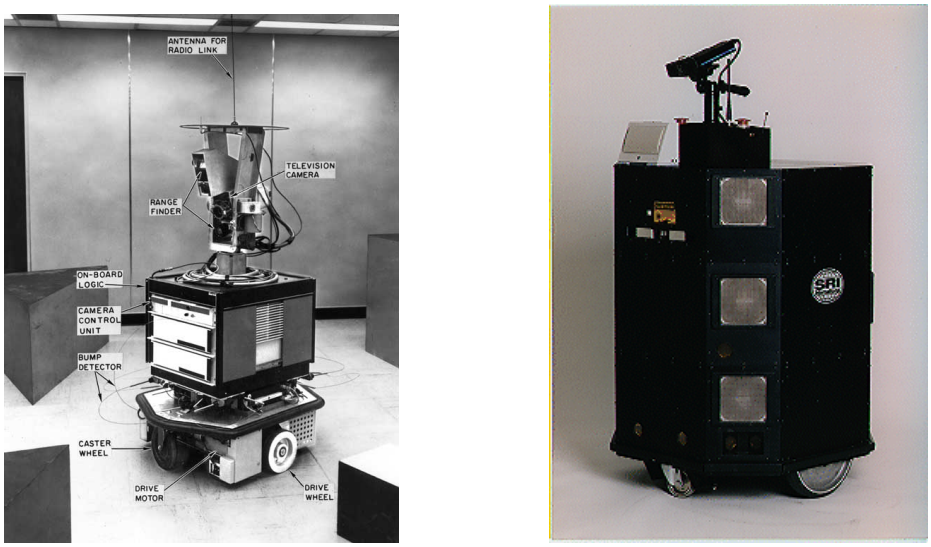


Figura 1.2: *Robots Shakey (izda.) y Flakey de SRI (dcha.)*

Actualmente no es ciencia ficción ver a un robot móvil guiando a los visitantes por un museo[6], navegando por entorno de oficinas[7], o conduciendo un coche de un extremo a otro de Estados Unidos (NavLab-CMU[8]). Estos prototipos son el fruto de la investigación de varias universidades y centros situados a la vanguardia de la robótica, como Carnegie Mellon University, el Massachusetts Institute of Technology o el Stanford Research Institute.

La robótica móvil es un campo interdisciplinar donde confluyen muchas áreas de conocimiento como la electrónica, la mecánica, la informática, la automática, etc. Los sensores y actuadores de los robots son dispositivos electrónicos que además de los transductores necesarios, contienen una circuitería propia para interactuar con los procesadores del robot, bien recogiendo sus órdenes de actuación, bien enviándoles los datos sensoriales. Los procesadores del robot son también circuitos electrónicos, que se benefician de los continuos avances en microelectrónica y concentran en muy pequeño espacio una enorme capacidad de cómputo. Todos estos componentes suelen ir en placas, insertadas de algún modo en la estructura física del robot, en su cuerpo mecánico. El diseño mecánico del robot condiciona sus capacidades motrices, de actuación e incluso de percepción. Por ejemplo, una cuestión relevante es determinar dónde situar los sensores, porque ello afecta a la información que el

[6] Minerva: http://www.cs.cmu.edu/∼minerva
[7] Xavier: http://www.cs.cmu.edu/∼xavier
[8] http://www.ri.cmu.edu/labs/lab_28.html

robot puede recibir del entorno. Otro ejemplo son los robots con patas, en los que el diseño mecánico condiciona enormemente los movimientos posibles. En cuanto a la informática, ha entrado de lleno en este campo porque los robots móviles son máquinas digitales programables. Suelen llevar a bordo microprocesadores o incluso ordenadores personales de propósito general para ejecutar los programas que dan vida al robot. Cuestiones como lenguajes de programación, algoritmos, estructuras de datos, comunicación entre programas, ingeniería del software, reutilización de código, etc. se muestran útiles a la hora de programar robots móviles.

En un plano conceptual la generación de comportamiento autónomo artificial también es un problema multidisciplinar. Al ser una temática amplia tiene intersección con muchos campos de investigación en los cuales se apoya. Por ejemplo, conceptos utilizados en robótica como la planificación, el uso de la lógica, el razonamiento probabilístico, la teoría de la evidencia, la lógica borrosa, etc. le han venido de la inteligencia artificial. El uso de controladores PID, los conceptos de realimentación e iteración de control se han recogido de la teoría de control. Del mismo modo también recibe influencias de la etología, la teoría de la decisión, la psicología, etc. Nos vamos a concentrar ahora en tres de las disciplinas que consideramos más han influido en el desarrollo de la robótica móvil en cuanto a sus principios: la teoría de control, la Inteligencia Artificial y, en sentido laxo, la biología.

Teoría de control. Dentro de la línea de automatización de sistemas han surgido disciplinas como la cibernética y la *teoría de control*. De estos campos la robótica ha heredado ideas básicas como la realimentación, el ciclo de control, la regulación, etc.. Por ejemplo, es dificil pensar hoy día en un robot que no cierre algún bucle de control, a cierto ritmo sobre algún actuador.

El problema de control y el de robótica pueden considerarse inicialmente homólogos, pues ambos persiguen que un sistema se comporte de cierta manera. Si en el escenario de control se entiende el comportamiento como la interacción entre el sistema a controlar y el controlador, en robótica se entiende como la interacción entre el entorno y el propio robot. Como argumentaremos posteriormente las aplicaciones típicas de robótica desbordan los planteamientos clásicos de un problema de control, debido principalmente a la imposibilidad de realizar modelos matemáticos completos de la situación.

Por ello es difícil plantear el diseño global de un robot como un controlador clásico. Sin embargo, sí es habitual que los robots actuales incorporen pequeños componentes de control, principalmente en el bajo nivel. Por ejemplo, los controladores PID para el manejo de articulaciones motorizadas o motores, que admiten referencias como objetivo y materializan bucles de control para conseguirlas.

Inteligencia artificial. La idea de construir máquinas que piensen por sí mismas ha seducido a muchos investigadores a lo largo de los tiempos. Siempre ha sido un desafío y un reto formidable. Una de las plasmaciones de este ansia es el propio campo de la Inteligencia Artificial (IA), que adoptó ese nombre en la conferencia de Darmouth, en 1956. Su principal objetivo es emular el procesamiento de información humano con computadoras electrónicas. Dentro de ella se abordan problemas como el procesamiento de lenguaje natural, el reconocimiento de imágenes, el aprendizaje, la demostración de teoremas, etc. Dos ejemplos paradigmáticos son la construcción de un programa que juegue bien al ajedrez, y un traductor automático. Ambos se consideraban un reflejo de inteligencia aplicada de los programas.

La psicología cognitiva, que trata de entender el funcionamiento de la inteligencia humana, ha ejercido gran influencia sobre la IA, que a su vez persigue poder reproducirla en una máquina. En este sentido, quizá apoyado en un razonamiento introspectivo, el paradigma dominante en la IA ha sido el de la descomposición funcional de la inteligencia en módulos especialistas (lenguaje, visión, razonamiento, etc.) y la lógica como motor de la deliberación, que se plasma como cierto proceso de búsqueda dentro de las alternativas.

Su ascendiente sobre la robótica móvil ha sido muy intenso. Por ejemplo, aportando la idea de que el comportamiento se puede generar como la ejecución de cierto plan calculado de antemano. Dicho plan es fruto de una deliberación sobre cierta representación del mundo, que tiene en cuenta los objetivos del robot. Por ejemplo, una ruta como secuencia de tramos intermedios que acaban llevando al robot al punto destino. La IA enfatiza la planificación y el modelado de la realidad como ingredientes fundamentales de la inteligencia en los robots.

Biología. A finales de los 80 los robots desarrollados, fundamentalmente basados en la IA, eran lentos y poco robustos, incapaces de realizar con soltura tareas aparentemente sencillas como navegar por un pasillo y reaccionar a obstáculos imprevistos. Motivado por estas limitaciones nació el paradigma reactivo, inspirado en las asociaciones estímulo respuesta típicas del conductismo. En él se genera comportamiento asociando una respuesta de actuación a ciertos estímulos sensoriales, bien directamente [Brooks, 1986], bien a través de transformaciones sensorimotoras [Arbib y Liaw, 1995].

Más allá de la revolución reactiva, la biología y dentro de ella la etología, suponen un modelo a seguir por los robots autónomos, en cuanto que estudian el comportamiento en los seres vivos y sus mecanismos causales. Los seres vivos son una solución real a la generación de comportamiento inteligente, al menos lo suficiente para sobrevivir en la naturaleza, y por lo tanto fuente de inspiración. En sentido contrario, la robótica le sirve a estas disciplinas como un banco de pruebas, como escenario de laboratorio en el que ensayar modelos y verificar hipótesis. Es pues, una relación bidireccional.

En los últimos años se ha vuelto la mirada a la biología y en particular a la etología buscando claves que permitan simplificar y hacer más robustos los comportamientos de los robots [Mallot y Franz, 1999]. Varios conceptos nacidos en biología como la homeóstasis se han propuesto como mecanismos para la selección de acción [Tyrrell, 1994]. También los movimientos de las abejas, capaces de volver siempre a su colmena y orientarse con el sol, de las moscas equilibrando el flujo óptico en ambos ojos, sirven de ejemplo para las técnicas de navegación en robots. La percepción gestáltica y el uso de invariantes visuales perceptivas como en los picados de los cormoranes para pescar [McFarland y Bösser, 1993] puede facilitar el desarrollo de comportamientos en robots, en apariencia muy sofisticados. También son destacables los trabajos que reproducen artificialmente el comportamiento de una rana [Corbacho y Arbib, 1995] o una mantis religiosa [Arkin *et al.*, 2000].

Aunque se ha progresado mucho, aún estamos lejos de las expectativas levantadas por el cine o la literatura. Por ejemplo, robots móviles tan versátiles como C3PO, R2D2 de la Guerra de las galaxias, o los que aparecen en Blade runner o Terminator siguen siendo a día de hoy ciencia ficción. De hecho estamos muy lejos que conseguir una máquina que pase con buena nota el test de Turing [Turing, 1950]. En este terreno la imaginación ha ido muy por delante de la realidad y ha levantado unas expectativas demasiado prometedoras. Expectativas que no se han satisfecho, lo que provoca cierta frustración y descrédito. En este sentido las comunidades robótica y de inteligencia artificial se han encontrado con unas barreras muy complejas y difíciles de superar: el lenguaje natural, el aprendizaje, autonomía, adaptación, etc. La construcción de máquinas que se comporten de modo autónomo sigue siendo un reto y un problema abierto.

En el terreno material, el crecimiento de la robótica móvil se ha beneficiado enormemente de la revolución electrónica que hemos experimentado en las últimas décadas. Las mejoras tecnológicas y en microelectrónica han disparado la capacidad de cómputo disponible y el desarrollo de los más variados sensores, motores y dispositivos. Por ejemplo, sensores precisos como el láser, el GPS, o las cámaras en color, y motores de continua, servos, etc. aparecen frecuentemente como parte del equipamiento de los robots móviles. La velocidad de los computadores se ha multiplicado en los últimos años considerablemente y esto ha hecho factible la materialización de algunas aproximaciones a la robótica móvil que demandan gran capacidad de cálculo. Sin embargo, a pesar de estas mejoras tecnológicas, de disponer de los procesadores más rápidos, los sensores y actuadores más avanzados, el cómo combinarlos para generar comportamiento autónomo sigue siendo un problema abierto. Los progresos en el hardware han puesto en evidencia la importancia de una buena organización interna, que se ha convertido en un factor crítico para conseguir el comportamiento autónomo robusto. Precisamente a esa organización interna dedicamos la siguiente sección, presentando las cuestiones que debe resolver y las principales dificultades que ha de superar para ello.

1.3. Generación de comportamiento autónomo en un robot móvil

Una vez presentado el marco genérico de la robótica y de la robótica móvil, en este apartado vamos a circunscribir de manera más específica el contenido de esta tesis a un problema concreto. En particular estudiamos la generación de comportamiento autónomo en un robot móvil, entendiendo como tal la capacidad de conseguir que un robot actúe de modo inteligente en cierto entorno. En esta

sección vamos a definir los términos más significativos que manejaremos en la memoria, delimitando los conceptos principales y la interpretación que les damos. En especial precisamos qué entendemos por robot móvil, comportamiento y arquitectura. De este modo podremos acotar mejor en el siguiente apartado los objetivos concretos de la tesis, qué cosas caen dentro del estudio y qué cosas quedan fuera.

1.3.1. Definiciones

Robot móvil. Un *robot* puede ser considerado como un sistema complejo dotado de varias capacidades básicas, percepción, acción, procesamiento y memoria, que interactúa con su entorno para conseguir ciertos *objetivos*. De modo más específico llamamos *robot móvil* al robot que puede desplazarse por el entorno, y esta capacidad forma parte fundamental de su naturaleza. En este tipo de robots los comportamientos motrices son los más frecuentes. Los comportamientos del robot deben servir para algo, resolver tales o cuales tareas concretas, tener ciertos objetivos. Estos pueden ser órdenes provinientes de un usuario humano, lo que abre una perspectiva utilitarista, u objetivos implícitos como *sobrevivir*.

Los ingredientes fundamentales de un robot son los sensores, actuadores y procesadores. Los *sensores* le permiten medir alguna característica del mundo o de sí mismo, captar información de su entorno, enterarse de lo que ocurre a su alrededor o dentro de él mismo. De este modo su comportamiento podrá ser sensible a las condiciones del entorno en el cual robot se haya inmerso. La RAE [Española, 1992] define sensor como: *Cualquier dispositivo que detecta una determinada acción externa, temperatura, presión, etc., y la transmite adecuadamente.* Quizá habría que amplizar esa definición para incluir que los sensores también pueden medir alguna característica interna o introspectiva del propio robot, como su nivel de batería, el giro de sus ruedas, etc.. En el mercado hay gran cantidad de sensores: sensores de luz, de temperatura, ultrasónicos, infrarrojos, láser para detectar obstáculos, cámaras, sensores cuenta vueltas, sensores de contacto, de presión, etc..

Los *actuadores* son dispositivos que le permiten ejercer alguna acción sobre el medio, o simplemente materializar un movimiento. Los actuadores hacen posible al robot influir sobre su entorno de modo activo, modificarlo o sencillamente moverse a través de él. Por ejemplo los motores de continua, los servos, las válvulas neumáticas, etc. En el caso de los robots móviles éstos cuentan por definición con la capacidad de movimiento por su entorno, no están anclados a una localización fija. En general esto se consigue con ruedas que los motores hacen girar o con patas que los motores desplazan, aunque también hay robots autónomos que vuelan o nadan.

El tercer ingrediente fundamental de todo robot móvil son los *procesadores*, que ejecutan los programas del robot. Aunque el enlace entre los datos sensoriales y las consignas a los actuadores se puede construir como cierta conexión fija inalterable, en general se puede configurar a voluntad a través de programas. Estos programas son los que leen los datos sensoriales y gobiernan los movimientos del robot, determinando de este modo el comportamiento final observable del sistema. Por ello, conseguir que el robot se comporte de cierto modo se traduce en escribir los programas de control que gobiernan al robot para que lo consiga. Los procesadores utilizados en robótica móvil van desde microcontroladores específicos hasta ordenadores personales, disponibles en cualquier tienda de informática.

Los entornos en los que se desenvuelven los robots móviles se clasifican típicamente en interiores, si están dentro de edificios, como oficinas o laboratorios, y de exteriores, si se desarrollan sin techo, en campo abierto. Pueden estar preparados para la navegación de los robots, como las fábricas con hilos enterrados que les sirven de guía, o no tener modificación especial ninguna. Pueden tener cierta estructura, como las regularidades de habitaciones, puertas, esquinas, etc. o ser más heterogéneos, como los entornos naturales. Pueden ser dinámicos, si por ejemplo aparecen personas moviéndose, o estáticos.

Comportamiento autónomo. Una vez presentados los componentes de un robot, definimos qué entendemos por comportamiento autónomo. La RAE [Española, 1992] define comportamiento como *conducta, manera de portarse*. Si ahondamos un poco más define conducta como *porte o manera con que los hombres gobiernan su vida y dirigen sus acciones* y portarse como *actuar o proceder de*

cierta manera. Sin alejarnos del significado normal, nosotros nos referiremos al comportamiento del robot móvil como su modo de actuar en cierto entorno, principalmente a su manera de moverse.

Por *autónomo* entendemos un sistema que toma decisiones por sí mismo, sin necesidad de supervisión humana, de modo independiente. El robot tiene objetivos, asignados por un operador humano o implícitos. Sin embargo cómo conseguirlos, cómo procesar los datos sensoriales en cada momento, cómo y cuánto mover los motores, etc. deben ser decisiones propias del robot. Esta autonomía en la consecución de objetivos implica cierta inteligencia en el comportamiento observable. De hecho con frecuencia utilizaremos autónomo e inteligente como sinónimos. Sin profundizar en definiciones entendemos por *inteligente*, en sentido laxo, algo que un observador humano, externo, pueda juzgar como inteligente. Básicamente si consigue los objetivos de modo eficiente y es sensible a las condiciones de su entorno. Es decir si el comportamiento está orientado hacia sus objetivos, si reacciona ante las condiciones del entorno, y si es eficiente.

En ningún momento vamos a hablar de un robot inteligente, sino de comportamiento inteligente, ya que lo que vamos a valorar como inteligente o no, atendiendo a sus consecuencias, es comportamiento del robot en su entorno [McFarland y Bösser, 1993]. Este modo de entender la inteligencia asume dos principios fundamentales. El primero es que es necesario disponer de un *cuerpo físico* capaz de interactuar con el entorno para materializar ese comportamiento. El segundo es que si no hay entorno no hay comportamiento inteligente. Los patrones de comportamiento apropiado surgen de la interacción del agente con el entorno, y no sólo del agente de manera intrínseca.

Siguiendo la clasificación que da Brooks en [Brooks, 1991a] el problema del comportamiento se puede dividir en varios campos: el nivel *micro*, el nivel *macro* y el nivel *multitud*. El nivel *micro* se encarga de comportamientos unitarios como seguir una pelota, atravesar una puerta, mover un brazo, etc.. El nivel *macro* se encarga de la integración de todo un repertorio de comportamientos *micro* en un solo individuo. El nivel *multitud* pone énfasis en comportamientos sociales, de conjuntos de individuos. Esta tesis estudia el nivel *macro*, centrándose en la arquitectura que hilvana un conjunto de comportamientos de nivel *micro*.

Arquitectura. Desde un enfoque sistémico un robot móvil se puede contemplar como un sistema que tiene por entradas los datos sensoriales y por salidas los posibles comandos sobre los actuadores. En este ámbito, de modo informal, entendemos que arquitectura es todo lo que hay entre los sensores y los actuadores. De manera más precisa llamamos arquitectura cognitiva, conceptual, o simplemente *arquitectura* de un robot autónomo a *la organización de sus capacidades sensoriales, de procesamiento y de acción para conseguir un repertorio de comportamientos inteligentes interactuando con un cierto entorno*. La arquitectura determina los comportamientos que exhibe un robot autónomo.

Desde nuestra perspectiva la arquitectura conceptual debe responder a dos cuestiones principales: ¿Qué hacer ahora? (selección de acción [Tyrrell, 1992]) y ¿qué es interesante ahora en el entorno? (atención). Su misión fundamental es organizar los procesos internos del robot de modo que sea capaz de seleccionar la percepción y actuación apropiadas en cada momento de acuerdo con los objetivos del robot y con las condiciones del entorno en las que se encuentra. Así pues, la arquitectura marca qué estimulos aceptar, buscar o elaborar, y cómo calcular en cada momento la acción más apropiada que debe ejecutarse ante esa percepción. Resumiéndolo en dos puntos, una arquitectura ha de incluir: capacidad de percepción selectiva y de actuación adecuada.

La importancia de la arquitectura radica en que los robots móviles son sistemas con un alto grado de complejidad, y cuanto más complejo es un sistema más relevancia cobra el papel de la organización de sus componentes. Normalmente para tareas sencillas hay muchas maneras de abordarlas y el modo en que se resuelven, *el cómo*, casi no tiene importancia. Sin embargo al aumentar la complejidad la organización influye más en el resultado, puede incluso llegar a ser determinante: con una buena organización se consigue el objetivo y con una mala no. Por ejemplo, el papel de la arquitectura es más visible cuando la complejidad aumenta, bien porque el robot deba exhibir un repertorio completo de diferentes comportamientos, bien porque los sensores proporcionan una gran cantidad de datos del entorno, no todos relevantes. Además *el cómo* puede influir notablemente en la calidad del sistema, ya sea en su funcionamiento, en el tiempo de desarrollo necesario para construirlo o en otros aspectos. Para lograr sistemas repetibles y adaptables su organización interna ha de ser clara, accesible y fácilmente

modificable. El estudio de cómo organizar estos procesos internos en robots es lo que denominamos arquitectura. Avances en la arquitectura conducen a comportamientos aun más complejos y a aumentar la autonomía de modo fiable.

Existen otras visiones de arquitectura, muy relacionadas entre sí pero diferentes, aunque unas se apoyen en otras. Por ejemplo, el robot tiene su *arquitectura mecánica* y sus programas establecen cierta *arquitectura software*. En esta tesis nos centraremos en los problemas asociados a la arquitectura conceptual y sus implicaciones, pero también implementaremos nuestra propuesta en unos programas determinados y en el hardware material de un robot físico concreto. En cuanto a la configuración mecánica, los dispositivos sensoriales como un láser o una cámara, los dispositivos de actuación como motores o brazos, o las placas con los procesadores son componentes que constituyen el *hardware* del sistema. Son las partes tangibles, que se pueden tocar. Todos ellos se organizan en una cierta estructura física, configurando un cuerpo robótico. En esta arquitectura mecánica importan cuestiones como tamaños, posiciones, conexiones, etc. . Esta arquitectura mecánica es la que finalmente interacciona con el mundo y materializa los comportamientos.

La arquitectura hardware en sí misma no hace nada motu proprio, es inerme. Lo que da vida a estos componentes materiales es el *software*, los programas. Ellos recogen la información proporcionada por los sensores y calculan los comandos que se envían a los actuadores, determinando de este modo el comportamiento del robot. Si queremos que el robot se comporte de determinada manera tenemos que programarlo para que lo haga. Esos programas siguen cierta arquitectura software. Puede ser algo tan sencillo como un programa único en bucle infinito que ejecuta cíclicamente la secuencia: leer datos de los sensores, pensar, actuar. También puede ser varios procesos concurrentes que se comunican entre sí enviándose datos, estableciendo flujos de información entre ellos. O quizá habrá procesos que se dediquen a elaborar cierta información de salida desde sus datos de entrada, procesos que tomen decisiones, etc. Las distintas arquitecturas conceptuales se concretan en cierta arquitectura software, es decir en los programas que se ejecutan en los procesadores del robot.

En una primera aproximación generar un comportamiento autónomo se puede ver como un simple problema de control, en el que se trata que un sistema físico actúe de determinada forma en cierto entorno. Sin embargo, el problema de la arquitectura es más complejo que un problema de control. Hablamos de un controlador cuando tiene por entrada señales con la información relevante que están definidas con nitidez, cuando los objetivos no cambian de naturaleza (si acaso cambian las referencias) y cuando lo "único" que hay que hacer es elegir entre determinadas posibilidades de actuación para conseguir el objetivo. Por el contrario, hablamos de arquitectura cuando incluimos la tarea de construir o seleccionar esas señales de entrada y los sensores ofrecen una cantidad desbordante de información no específica. También cuando la naturaleza de los objetivos puede varíar enormemente o cuando el robot tiene varios objetivos, con prioridades dinámicas que dependen de las necesidades actuales y de la situación del entorno. Tenemos un problema de arquitectura cuando el sistema ofrece un *repertorio* muy variado de comportamientos y hablamos de la interacción entre unos comportamientos y otros, cambio de un modo a otro, etc. Problemas clásicos de control como el mantenimiento de cierta referencia se pueden incluir como un componente más de la arquitectura, pero ésta última aborda un problema más amplio. El control es el análogo al nivel *micro* de Brooks [Brooks, 1991a], y la arquitectura aborda el nivel superior, el nivel *macro*, que combina varios comportamientos unitarios de nivel *micro* en el mismo robot.

No existe *la* arquitectura de control válida para todos los entornos y para todos los comportamientos. De hecho, incluso un animal con escasa capacidad de aprendizaje tiene un repertorio de comportamientos limitado, aunque amplio. Por ejemplo, un pingüino no sabe escribir. Mejor aún, el ganso que empolla sus huevos no sabe recoger el huevo fuera del nido con las alas, porque el mecanismo que tiene implementado en su sistema nervioso es recogerlo con el cuello [Tinbergen, 1987]. Su arquitectura le provee de un comportamiento y no de otro, aunque tiene el hardware necesario para ambos.

1.3.2. Cuestiones de arquitectura

Una vez definida la arquitectura conceptual de un robot como el corazón que organiza sus capacidades para generar comportamiento, en esta sección vamos a introducir algunas de las cuestiones concretas que consideramos debe abordar. Primero, la relación entre las actuaciones posibles y los objetivos del robot, que no siempre es sencilla, y se enmaraña cuando los objetivos son varios o cambiantes y cuando las situaciones a afrontar son muy diversas. Segundo, la intrincada correspondencia entre los datos sensoriales disponibles y la información útil para la tarea, que no suele ser una simple biyección.

Relación entre acciones y objetivos. Un robot debe tomar decisiones de actuación continuamente. Para ello tiene muchas posibilidades de actuación en cada instante, más cuantos más grados de libertad tenga. En principio todos los valores admitidos por sus actuadores son posibles, pero sólo un subconjunto limitado de ellos conduce a la satisfacción de los objetivos. Cómo elegir la actuación más beneficiosa es una tarea difícil puesto que debe atender tanto a la situación del entorno como a los objetivos que tenga en ese momento. A este problema se le llama *selección de acción* [Tyrrell, 1992] e incluye preguntas similares a ¿cómo debe actuar el sistema en la siguiente iteración?, ¿cómo se calcula el valor que se comanda a los distintos actuadores del sistema? Las implicaciones de una decisión actual en la consecución futura de cierto objetivo suelen ser complicadas, y en algunos casos, impredecibles. Además generalmente hay que asumir que a la hora de calcular la acción de control no toda la información necesaria va a estar disponible.

El robot suele tener varios objetivos simultáneamente, algunos relacionados con su propia supervivencia y otros con las tareas que tiene asignadas, algunos explícitos y otros implícitos. Por ejemplo, además de querer avanzar hacia cierto punto, un robot con patas nunca debe perder la estabilidad. La relación entre los distintos objetivos también es una cuestión delicada, que incluso puede ser cambiante dependiendo del estado del robot y el grado de satisfacción de otros objetivos. Como veremos en el capítulo 2 hay dos grandes familias en la literatura para regularla: el arbitraje, que elije un único objetivo ganador, y la fusión de comandos, que establece una solución de compromiso.

Además, cuando los actuadores están relacionados entre sí aparece el problema de la *coordinación* para conseguir un comportamiento coherente. Por ejemplo, todos los grados de libertad de una pata articulada deben trabajar en la misma línea para conseguir movimiento coherente de esa pata. Del mismo modo todas las patas del robot deben estar sincronizadas de algún modo para conseguir el movimiento global.

Información compleja, incierta y dispersa. El robot percibe su entorno a través de los sensores. La relación entre los objetivos actuales del robot y la información del entorno que interesa en cada momento es complicada. En muchas ocasiones el flujo de información sensorial es desbordante y contiene muchos datos irrelevantes para la tarea en curso. Así sucede por ejemplo cuando se trabaja con visión artificial. La cantidad de datos por segundo que suministra una cámara es enorme, pero sólo cierta información contenida en el flujo de imágenes es interesante. Por ello es conveniente que la arquitectura ofrezca algún mecanismo de *atención* que le permita ignorar los datos sensoriales no significativos y concentrarse en los relevantes. Filtrar los estímulos tiene la ventaja añadida de que permite no desperdiciar tiempo de cómputo en datos que no interesan.

La correspondencia entre la información útil para la tarea, que llamaremos genéricamente *estímulos*, y los datos sensoriales del robot también es compleja. Aunque se pueden tener sensores específicos que directamente dan la información relacionada con la tarea, es más frecuente disponer de sensores generalistas como una cámara, sensores de distancia, etc.. En este caso los estímulos necesitan ser elaborados con algún procesamiento específico sobre los datos sensoriales. Por ejemplo, si la naturaleza del estímulo no es percibible con la señal de un único sensor, se puede utilizar información procedente de varios sensores, iguales o diferentes. También puede ser conveniente elaborar los estímulos con información redundante, de más de un sensor, para reducir incertidumbres y errores sensoriales. Por ejemplo, para estimar la posición de un robot en exteriores es conveniente combinar la información de diferentes tipos de sensores como el GPS, la odometría, la brújula y el giróscopo. En otras situaciones es necesario mantener algún tipo de registro temporal de la señal. Por ejemplo, para caracterizar la

velocidad de un obstáculo si se pretende trabajar con estímulos móviles y distinguirlos de los estáticos. La arquitectura debe incorporar en ella técnicas y algoritmos que extraigan y construyan los estímulos desde los datos sensoriales, lo que puede implicar abstracción, memoria, fusión, etc..

Un problema consustancial a los sensores es su *incertidumbre*, que obliga a que el robot maneje información no fiable, ruidosa o errónea. Adicionalmente en entornos no estructurados la percepción es más difícil que en entornos preparados, donde la realidad se ha simplificado para que sea más fácil al robot identificar los estímulos relevantes.

En resumen, en entornos complejos mucha de la información de interés no es directamente percibible sin procesamiento, y mucha de la información disponible no es relevante o fiable. La arquitectura incorpora la complejidad del entorno en su organización y tiene que ser capaz de extraer los estímulos necesarios para poder calcular la acción más adecuada. Esto conlleva tanto filtrar los estímulos relevantes del abundante flujo de datos sensoriales como posibilitar el procesamiento que elabora los estímulos complejos desde las señales sensoriales. La arquitectura conceptual debe responder a preguntas como: ¿Qué estímulos interesan y se perciben en este momento?, ¿cómo se construyen desde los datos sensores crudos? Como veremos en el capítulo 2 han surgido en la literatura muchos enfoques de esta cuestión. Desde las aproximaciones reconstructoras que buscan reconstruir el entorno del robot partiendo de la información sensorial, hasta la percepción orientada a tarea que engancha directamente los estímulos a la actuación.

Además de estos dos puntos principales la arquitectura debe abordar otras cuestiones como el tratamiento del tiempo, el equilibrio entre deliberación y reactividad, y la dialéctica entre centralizado versus distribuido. Cualquier arquitectura conceptual se sitúa en algún punto dentro de estos tres ejes.

Deliberación versus reactividad. Entendemos por *reactividad* la capacidad para reaccionar a tiempo [Simmons, 1994]. En este sentido los robots han de ser reactivos a estímulos del entorno, incluso los imprevistos, y tener mecanismos para detectar con rapidez estas situaciones de sorpresa. También deben incorporar los resortes para actuar apropiadamente ante ellas. Apropiadamente quiere decir a tiempo y sin perder de vista el o los objetivos del robot. Por ejemplo, ante un obstáculo imprevisto o un nuevo hueco en su trayectoria el robot ha de recalcular el camino que le lleva a su meta. Las novedades detectables no sólo han de ser contingencias, en el sentido de Payton [Payton, 1990], también nuevas oportunidades que surjan y que se pueden aprovechar.

Pero el comportamiento del robot no puede estar únicamente influído por el entorno, los objetivos propios también son condicionantes muy importantes para sus acciones. Una de las aproximaciones que orientan las acciones del robot hacia sus objetivos es la *deliberación*, que introduce en sus decisiones cierto razonamiento explícito sobre las metas. Un ejemplo de deliberación es la planificación, que elucubra de antemano sobre cierto modelo del mundo y concluye un plan de actuación. Este plan es una secuencia de actuaciones cuya ejecución conduce a la consecución de los objetivos. La deliberación puede incluir varios niveles de abstracción y la división de una tarea en sub-tareas más sencillas [Simmons, 1994]. Esta descomposición de la complejidad resulta útil cuando algunos de los objetivos del robot son lo suficientemente complicados como para que resulte necesario desmembrarlos en tareas más simples. La planificación ofrece otra ventaja que es la anticipación en el tiempo.

Como veremos en el capítulo 2 históricamente la deliberación ha ejercido gran influencia como tendencia para generar comportamiento en robots. A finales de la década de los ochenta se detectó también la necesidad de reactividad para que los comportamientos fueran flexibles y ágiles. Hoy día ambas características se reconocen como necesarias en un robot, y surge la pregunta de cómo combinarlas. ¿Cómo se funden las capacidades reactiva y la deliberativa? En general hay que establecer un balance entre ambas tendencias. De este equilibrio va a depender en gran medida que se aprovechen las ventajas de ambos enfoques o se sufran sus inconvenientes. En la práctica este compromiso depende en gran medida del entorno. Si el entorno es muy dinámico e impredecible entonces los esfuerzos por planificar no compensan.

Centralizado versus distribuido. Una dialéctica que se plantea a la hora de diseñar la arquitectura es su distribución en pequeñas unidades o su concentración en un único nodo central que se encarga de todo. No es necesario que la distribución sea en el hardware, con microcontroladores

distintos, tan real como ésta es una distribución software donde cada unidad se implementa con un flujo de ejecución diferente, aunque sea sobre el mismo procesador. En general, la distribución puede ser ventajosa porque elimina el cuello de botella de un único programa pendiente de todo, que en casos con visión se atasca claramente. Al poder paralelizar el procesamiento en varias unidades concurrentes se puede ganar en capacidad de cómputo. Otra de las ventajas de la distribución es que favorece la reutilización de partes, unas pueden utilizar a otras.

Al distribuir se introduce el problema de la coordinación entre los distintos componentes. Por ejemplo, cómo se combinan las recomendaciones de actuación de varios módulos [Rosenblatt y Pyton, 1989][Saffiotti, 1997]. Se pueden mezclar o seleccionar la salida de uno sólo cada vez, pero la salida conjunta debe ser única porque los actuadores son únicos.

Cada arquitectura se sitúa en algún punto de esta dialéctica de distribución. Se puede distribuir tanto el control como la percepción, y siempre en mayor o menor medida. Además no hay un criterio universal de partición, existen muchos: descomposición funcional [Crowley, 1985], niveles de competencia [Brooks, 1986], jerarquía de agentes[Fischer *et al.*, 1994] o controladores[Albus, 1993], un módulo por objetivo, un módulo por comportamiento [Arkin, 1989b], etc.

Tratamiento del tiempo. Una cuestión que debe tener en cuenta la arquitectura es la escala temporal de las decisiones y de la información que el robot maneja. Hay decisiones de actuación de corto plazo, decisiones de medio plazo y decisiones de largo plazo. Cada una se toma a su propio ritmo sobre cierta información. Por ejemplo, la decisión de ir a cierto punto de destino lejano puede tener un horizonte temporal de varios minutos. Esa decisión a largo plazo, condiciona que se persiga un subobjetivo local próximo, alcanzable en algunos segundos. Dentro de esa decisión de medio plazo se emiten consignas de velocidad a los motores, de corto plazo, digamos cada 50 ms, que se cierran en rápidos bucles PID del orden de microsegundos. Las decisiones a largo plazo condicionan a las de menor horizonte temporal.

La información sobre la que se toman decisiones de actuación también tiene su propia dinámica temporal. Los datos sensoriales crudos suelen ser rápidos. Por ejemplo, en un robot con sónares pueden haber nuevos datos ultrasónicos cada 200 ms. Los estímulos que se elaboran también tienen sus ritmos, su latencia (tiempo en detectarse) y su persistencia (tiempo en recordarlo en ausencia de evidencia sensorial). Del mismo modo que se toman decisiones de actuación a distintos ritmos también se toman decisiones de percepción.

Desde esta perspectiva cualquier arquitectura ha de cuidar a qué ritmo se toman las decisiones y sobre qué información, si ésta es fresca u obsoleta. Y debe adaptar los ritmos y la dinámica de las decisiones con los de la información sobre la que se apoyan.

Una vez descritos los problemas principales que la arquitectura debe abordar, vamos a presentar algunas dificultades que los complican y a las que la arquitectura también debe dar respuesta. Por ejemplo, el dinamismo del entorno supone un escollo adicional cuando el escenario en el que queremos que trabaje el robot no es estático y en él se están produciendo constantemente cambios, algunos debidos al propio robot, otros no, y muchos de ellos imprevisibles. También las limitaciones que el cuerpo físico impone condicionan la arquitectura y obligan a que su diseño sea cuidadoso.

Entorno dinámico y abierto. Los entornos en los que operaban los primeros robots móviles eran estáticos y cerrados. Eran estáticos porque los objetos del entorno en que se desenvolvía el robot no se movían por sí mismos. Además se verificaba la *hipótesis de mundo cerrado*, que asume que los únicos cambios en el mundo son los debidos a la propia actuación del robot. En estos entornos la relación entre las acciones del robot y los objetivos es mucho más simple que en el caso dinámico, porque el entorno es más predecible. Por ejemplo, los objetos no cambian de lugar si el robot no los mueve, lo cual simplifica las inferencias y el encadenamiento de razonamientos.

Sin embargo estos escenarios estáticos y cerrados son poco frecuentes en las aplicaciones robóticas reales. Normalmente el entorno del robot es dinámico y abierto. Abierto en el sentido de que existen otros agentes que pueden cambiar el estado del mundo, que evolucionan a su albedrío y que participan en la realidad igual que el propio robot. Dinámico, porque la situación puede cambiar con el tiempo. Cuanto menos estructurado sea más cambios imprevisibles son susceptibles de aparecer. Que el mundo

sea cambiante obliga al robot a una percepción continua, que capture esos cambios, y que relacione la nueva situación con los objetivos. Las arquitecturas han de considerar estas posibilidades y reaccionar ante ellas si procede.

Recursos limitados. El hecho de contar con un cuerpo físico dota al robot de ciertas capacidades pero también implica unas limitaciones. Limitaciones en la capacidad de percepción del entorno, en la capacidad de proceso, y en la capacidad de actuar sobre el entorno.

En cuanto a la percepción ya hemos mencionado que los sensores reales ofrecen información del entorno, pero sujeta a errores, incertidumbres y además tienen un alcance finito. En cuanto a la capacidad de cómputo, los procesadores del robot tienen cierta velocidad y gracias a ella pueden realizar cierto número de cálculos por segundo, pero no más. Por ello no todos los algoritmos son implementables en el robot real sin perder reactividad. El tiempo de procesador se convierte en un recurso más, y es finito. La arquitectura ha de contemplar estas limitaciones y repartir el tiempo de cómputo de los procesadores entre las distintas tareas u objetivos del robot, priorizando las que se consideran más urgentes en cada momento. Esta asignación de prioridades puede no estar explícita pero siempre ha de existir para que el robot sea capaz de reaccionar a tiempo a las circunstancias.

Con respecto a las limitaciones de actuación la arquitectura también debe tener en cuenta las restricciones físicas del hardware a la hora de elegir la mejor acción a realizar, ya que ésta ha de ser realizable por el robot. Por ejemplo, los motores reales de un robot tienen una aceleración máxima insuperable. Si en un instante dado el robot está en una posición, con una orientación y velocidad determinadas, entonces en el instante posterior, no todas las posiciones, orientaciones y velocidades son alcanzables. Además las actuaciones suelen llevar también una incertidumbre asociada.

Diversidad. La *diversidad* es otra fuente de complicación cuando hay que generar comportamiento. Por ejemplo, la variedad de situaciones concretas que el robot debe afrontar con éxito es enorme. Esta variedad hace difícil generar comportamiento únicamente mediante una tabla que relacione las posibles condiciones sensoriales con las actuaciones correctas para cada caso. La cantidad de situaciones posibles es desorbitada y por ello no se pueden anticipar todas ellas (si acaso, detectar invariantes en esas condiciones sensoriales y responder a ellos).

La diversidad de comportamientos que el robot debe exhibir para completar su funcionalidad es otra fuente de dificultad. El robot no tiene que hacer una sola cosa, debe ser capaz de ofrecer un repertorio variado para que sea útil. Esto contrasta con algunos programas de IA, como el jugador de ajedrez Deep Blue, que *sólo* tiene que calcular la mejor jugada en cada turno.

1.3.3. Criterios de evaluación

Una vez introducidas algunas cuestiones que una arquitectura conceptual de un robot debe abordar, en esta sección vamos a describir algunos criterios que permiten valorar esa arquitectura. Estas características miden las prestaciones, la calidad de la arquitectura. Para esta evaluación se puede tomar un punto de vista externo, observando o juzgando el comportamiento que la arquitectura provoca. También se pueden emplear criterios internos, que necesitan examinar su funcionamiento interno y los mecanismos que proporcionan esas propiedades. En cualquier caso la evaluación de una arquitectura es una cuestión delicada. No hay un repertorio universalmente admitido de comportamientos, ni una batería de pruebas que sirva como patrón medidor de cada una de estas características. La evaluación será forzosamente subjetiva y discutible.

Con mayor o menor fragilidad en las conclusiones, el proceso de evaluación permite comparar distintas arquitecturas. La heterogeneidad es el principal escollo para establecer comparaciones justas. Es tanta la variedad de entornos y aplicaciones de los diferentes robots que es muy difícil establecer unos patrones cuantitativos que sirvan para cotejar objetivamente distintas arquitecturas. Esto explica el escaso número de trabajos comparativos en la literatura. Pese a esta dificultad sí puede resultar útil establecer el contraste, aunque sea en términos cualitativos.

Una recapitulación de las cualidades deseables en las arquitecturas se puede encontrar en [Ollero Baturone, 2001] y en [Medeiros, 1998]. Los criterios principales que consideramos fundamentales y que utilizaremos para juzgar una arquitectura son la reactividad, la finalidad, la robustez y la

escalabilidad. Dependiendo del grado en que satisfaga ciertas características deseables diremos que una arquitectura es mejor o peor.

1. **Reactividad: orientación al entorno**. La reactividad es la capacidad de responder a tiempo. Como ya hemos dicho, esto implica dos partes: la percepción a tiempo del estímulo ante el cuál es necesario reaccionar, y la capacidad de ejecutar a tiempo la acción más adecuada. La arquitectura ha de ser capaz de aprovecharse de las circunstancias de la situación concreta [Tyrrell, 1993a], detectar los cambios en el entorno y ser oportunista. Es lo que Ollero [Ollero Baturone, 2001] llama adaptabilidad.

2. **Finalidad: orientación a objetivos**. La arquitectura ha de ser capaz de no perder de vista los objetivos del sistema para que en el comportamiento observable no influya sólamente el entorno sino también los objetivos propios del robot. Es parte de la propiedad de eficiencia que define Ollero en [Ollero Baturone, 2001].

3. **Robustez**. Es deseable que cualquier arquitectura sea robusta frente a fallos de partes del sistema, frente a incertidumbre y frente a la falta de información sobre el entorno. En este sentido puede incluir mecanismos de redundancia. Conviene que la arquitectura cumpla los objetivos con sensores un ligeramente peores, con actuadores levemente peores, con la falta de cierta información y en entornos ligeramente distintos. También es deseable que la arquitectura sea portable a otros entornos similares y que funcione en ellos, es decir, que no haya sobreadaptación a un entorno concreto o a ciertas condiciones de trabajo. Este rasgo incluye la fiabilidad de Ollero [Ollero Baturone, 2001].

4. **Escalabilidad y versatilidad**. Una cualidad importante es la escalabilidad, definida como facilidad para añadir nuevas funcionalidades a las arquitecturas. Relacionada con esta propiedad está la *composición de comportamientos*, que permite a los nuevos aprovechar la funcionalidad existente de los antiguos. Es el *exploitation principle* de Minsky [Minksy, 1986]. También es deseable que la arquitectura sea versátil. Por ejemplo, que sea fácil modificarla para objetivos similares, o que incorpore con facilidad nuevos dispositivos hardware. Estas dos características tienen que ver con la flexibilidad de la arquitectura a la hora de extenderla, bien añadiendo funcionalidad (escalabilidad), bien modificando la existente para comportamientos parecidos (versatilidad).

Los criterios señalados hasta ahora son generalistas. Otro aspecto a tener en cuenta en la evaluación de una arquitectura es su adecuación a la tarea concreta que debe resolver. En este sentido aparecen criterios más específicos, relacionados con la aplicación inmediata del robot. Por ejemplo, criterios de eficiencia y rendimiento: mantener bajo el consumo de recursos, realizar la tarea en poco tiempo, etc. Si la tarea fundamental es de navegación puede ser interesante medir rasgos como: velocidad media, distancia media a obstáculos, suavidad de trayectoria, energía consumida, etc..

1.4. Objetivos de esta tesis

Como hemos avanzado en las secciones anteriores el problema que abordamos es la generación de comportamiento artificial en un robot móvil. Tal y como indica el título, la hipótesis que se examina en esta tesis es que el comportamiento autónomo se puede generar organizándolo en función de esquemas articulados en jerarquías dinámicas. Para explorar esta propuesta hay que diseñar una arquitectura cognitiva basada en esquemas que dé respuesta a las principales cuestiones de arquitectura, no sólo a los aspectos de control, sino también los aspectos perceptivos. Ese es el objetivo principal de la tesis.

En el aspecto de actuación hay que estudiar la distribución del control en esquemas, su jerarquización y reconfiguración dinámica. También hay que diseñar mecanismos distribuidos de selección de acción, y en general de combinación de comportamientos: bien arbitraje o bien fusión entre varios. En el lado perceptivo hay que profundizar en percepción distribuida, explorando las facilidades que ofrece para realizar fusión sensorial, mecanismos de atención, de interpretación y de percepción de estímulos dependientes (jerarquías de esquemas perceptivos).

Esta tesis no pretende ser una solución definitiva a todo el abanico de cuestiones que surgen en ese empeño. Sí pretende avanzar un paso en busca de explicaciones *causales* o etiológicas del comportamiento inteligente. No pretende sólo proponer una respuesta a las principales cuestiones de arquitectura, sino haber hecho las preguntas correctas para capturar mejor el resbaladizo problema del comportamiento en robots. A grandes rasgos el planteamiento del problema ya lo hemos esbozado en la sección 1.3, en el resto de la tesis exploraremos una hipótesis concreta, la jerarquía de esquemas, que permita generar comportamiento autónomo en los robots. En este sentido la tesis se sitúa lejos de teorías teleonómicas o meras descripciones generales del comportamiento, porque lo que se quiere es reproducir ese comportamiento observable en el hardware actual. También se distancia de explicaciones antropomórficas que atribuyen intenciones o pensamientos a los robots (precisamente lo que buscamos es cómo generar esas intenciones o esos pensamientos). Esas hipótesis no son válidas si no les acompaña una teoría detrás que aclare, explique su funcionamiento y que resista de modo convincente una buena batería de preguntas razonables.

Ligados al objetivo principal de diseño de la arquitectura cognitiva hay varios secundarios, algunos con más peso específico que otros, que enumeramos a continuación.

1. Analizar el estado del arte en arquitecturas de robots. El estudio bibliográfico permitirá analizar las diferentes propuestas surgidas en los últimos 40 años para generar comportamiento en robots móviles, identificando líneas comunes, aportes originales y agrupándolas en paradigmas.

2. Evaluar las prestaciones de la arquitectura propuesta frente a criterios deseables en cualquier arquitectura de control, señalando sus virtudes y sus debilidades aunque sea de modo cualitativo. Además nuestra propuesta debe enmarcarse y contrastarse con las principales paradigmas propuestos hasta el momento, comentando las similitudes, diferencias y las aportaciones más relevantes.

3. Desarrollar una arquitectura software que materialice en programas la distribución del comportamiento en esquemas y los mecanismos de activación, modulación, selección de acción y percepción distribuida descritos conceptualmente.

4. Generar un repertorio limitado de comportamientos autónomos de navegación en un robot real siguiendo la arquitectura cognitiva propuesta. Dado el estado actual, fundamentalmente cualitativo, de las investigaciones en el campo de la robótica móvil estimamos que la mejor vacuna contra la especulación vacua es el apoyo en implementaciones experimentales. Los prototipos reales proporcionan realimentación sobre las ventajas e inconvenientes de la arquitectura. Validaremos la viabilidad de nuestra propuesta implementándola en un robot real.

   Se desarrollarán dentro de la arquitectura esquemas de percepción que permitan identificar estímulos complejos significativos para robots en escenarios de oficina, como personas moviéndose o puertas. Estos estímulos son muy relevantes para entornos de interiores y son relativamente complejos. Permiten explorar el uso de técnicas de interpretación, estímulos compuestos y de fusión que ofrece la jerarquía de esquemas. Un aspecto perceptivo en particular que la tesis estudiará es la representación de obstáculos dinámicos, y cómo capturar la dinamicidad de ciertos estímulos.

   Además se materializarán varios conductas básicas en robots reales, como el seguimiento de personas, la navegación local, etc.. También se construirá un sistema con la arquitectura propuesta que ofrezca un repertorio variado de comportamientos para un jugador de la RoboCup. Un objetivo explícito es probar la arquitectura en varios plataformas diferentes, no restringir los experimentos a un robot concreto.

Requisitos. El robot particular que usaremos para nuestros experimentos y que utilizaremos como referencia es un Pioneer 2DXE. La arquitectura software y los comportamientos de demostración han de funcionar en esta plataforma. El entorno en el que se desenvolverá este robot es un entorno de oficina, dinámico y sin marcadores artificiales especiales. Explícitamente evitamos el uso de sensores de posición absoluta como GPS. Estas restricciones condicionan aún más los problemas a los que la

arquitectura debe dar respuesta. Por ejemplo, la percepción se hace más exigente que en entornos manipulados para facilitar el funcionamiento del robot. Sin embargo se gana en versatilidad: no es necesario alterar el entorno con marcas especiales para que el robot se comporte correctamente.

Para mayor autonomía exigiremos que toda la información que tiene el robot sobre su entorno la construya él mismo desde sus sensores. En este sentido no permitiremos la inserción de ningún mapa externo. Además trabajaremos con los estímulos directamente disponibles o recientemente sensados, permitiendo una memoria de corto plazo, siempre egocéntrica y de alcance local.

Lo que no son objetivos. Expuestos los objetivos, quedarían fuera de esta tesis los restantes campos. Indicaremos en esta sección los aspectos de la robótica que explícitamente quedan fuera del ámbito de este trabajo. En particular, dejamos fuera de esta tesis el aprendizaje, la interacción social y el lenguaje natural. La modificación de conductas y el aprendizaje de nuevos comportamientos quedan fuera porque nos parecen demasiado complicados en el estado actual de las investigaciones en este campo. Hemos preferido centrarnos en la arquitectura que ofrece un repertorio comportamientos fijos. En términos etológicos, nos centramos en comportamientos instintivos, gracias a los cuales un mismo animal puede exhibir todo un repertorio de comportamientos y donde se pueden estudiar mecanismos de activación, percepción, reutilización de actuadores, etc.. Quizá cuando haya más comprensión de esos mecanismos que generan los comportamientos innatos será el momento de abordar el aprendizaje y la incorporación de conductas aprendidas al sistema. Por ahora, no.

También queda fuera de este trabajo el comportamiento de grupos, el nivel multitud de Brooks [Brooks, 1991a]. Así cuestiones como la coordinación se abordan desde el punto de vista interno, entre los posibles centros nerviosos o comportamientos del robot, no como coordinación entre individuos. En esta misma línea no se aborda la comunicación con otros robots, ni siquiera con el operador humano.

Otro punto interesante que queda fuera de los objetivos de la tesis es la localización en un marco absoluto y en general la construcción de representación o mapas del entorno a largo plazo. De modo más genérico no abordamos la memoria a largo plazo, su formato, los mecanismos de inserción en ella, de olvido, etc. Nos limitamos a representar las inmediaciones del robot, su entorno más reciente y próximo, y con ello generar un repertorio de conductas.

1.5. Propuesta de arquitectura: Jerarquía Dinámica de Esquemas

Una vez presentadas las cuestiones que la arquitectura de un robot autónomo debe resolver y planteado el objetivo principal de diseñar una arquitectura, vamos a esbozar en esta sección nuestra propuesta. Recibe el nombre de Jerarquía Dinámica de Esquemas (JDE) y es el fruto de nuestros posicionamientos ante esas cuestiones y de cómo planteamos el problema del comportamiento autónomo. En esta sección haremos una somera presentación de sus fundamentos, dejando los detalles para el capítulo 3.

Básicamente se articula en tres puntos que describiremos a continuación. Primero, el comportamiento se fragmenta en unidades pequeñas llamadas *esquemas*. Esta cuantización facilita la reutilización de partes de la arquitectura. Segundo, la arquitectura separa dos problemas distintos: la percepción y el control. Son problemas muy imbricados, pero distintos. Veremos que esta distinción aporta grandes ventajas desde el punto de vista de implementación. Combinando estos dos principios tendremos esquemas perceptivos y esquemas de control. Tercero, los esquemas, tanto perceptuales como de control, se pueden combinar dinámicamente en jerarquías.

1.5.1. Esquemas

Inspirándonos en los trabajos de Arbib [Arbib y Liaw, 1995] y Arkin [Arkin, 1989b] cuantizaremos la arquitectura en pequeñas unidades de comportamiento llamadas esquemas. Definimos un esquema como un flujo de ejecución independiente con un objetivo. Puesto que la arquitectura aborda tareas de percepción y de actuación, tendremos esquemas perceptivos y esquemas de control respectivamente. Los perceptivos elaboran representación y los de actuación, también llamados esquemas motores, toman decisiones de control. De modo más específico entendemos por *esquema* un *flujo de ejecución*

que se encarga de la construcción de la representación (esquema perceptivo) o de la actuación (esquema motor) para conseguir cierto objetivo. Estos esquemas son similares a las *action units* de [Nilsson, 1969], en el sentido que cada uno resuelve su tarea concreta.

Los esquemas son flujos de ejecución iterativos, es decir que funcionan cíclicamente. Además son interrumpibles en cualquier momento, su ejecución se puede detener al final de cada ciclo y relanzar cuando se desee. Es decir se pueden activar y detener a voluntad. Como no todas las acciones y percepciones posibles son útiles en el momento presente, unos esquemas estarán despiertos y otros dormidos. Los esquemas son además modulables, es decir, admiten parámetros de modulación que sesgan ligeramente su funcionamiento. Para conseguir cierto comportamiento se despierta y modula a los esquemas perceptivos que elaboran los estímulos necesarios y a los esquemas motores que materializan las respuestas adecuadas conforme al comportamiento deseado. Esa es la interfaz genérica de utilización de los esquemas: su activación y su modulación a través de parámetros.

La fragmentación de la percepción y el control en pequeños unidades modulables favorece la reutilización de esquemas para diferentes comportamientos, previsiblemente con parametrizaciones diferentes, como veremos en el capítulo 3. Además esta cuantización recoge las evidencias encontradas por Arbib en neurofisiología, y es perfectamente compatible con las hipotetizadas por Lorenz en la etología, en concreto con los centros instintivos de comportamiento [Lorenz, 1978].

1.5.2. Arquitectura: percepción y control

Entendemos por tanto que el problema del comportamiento autónomo se divide en dos partes imbricadas: la percepción y el control. Para conseguir comportamiento completamente autónomo del robot, la arquitectura debe contemplar ambas y sus interrelaciones. Por un lado la percepción se encarga de conseguir la información relevante para los objetivos del robot en cada momento. Por otro, asumiendo que tenemos esa información, que puede ser incierta e incompleta, el control se encarga de calcular la respuesta adecuada, entre todas las opciones posibles, para conseguir los objetivos.

Quizá una novedad importante de esta tesis sea la inclusión explícita de los aspectos perceptivos dentro de la arquitectura. En otros muchos enfoques [Maes, 1990] la percepción se asume resuelta, o introducida a priori por un humano [Simmons y Koening, 1995] y a nuestro juicio es una de las piezas clave para conseguir comportamiento inteligente. Al incluirla de forma explícita en la arquitectura el robot aumenta su nivel de autonomía, porque es capaz no sólo de tomar independientemente decisiones de actuación, sino de elaborar de modo autónomo la información sobre la que aquellas se toman.

No se puede pensar en control sin percepción porque ésta es la base sobre la cual se deciden las actuaciones. Tampoco es concebible pensar en percepción sin control en el caso de un robot autónomo: se percibe *para* actuar. JDE contempla explícitamente el caracter subsidiario de la percepción, porque no existe per se, sino que nace orientada a la tarea. Los esquemas perceptivos van atados a los esquemas de actuación, y son éstos los que activan a aquellos. No sólo para comportamientos de navegación, sino para cualquiera del repertorio.

Percepción. El problema de la percepción es suficientemente complejo y decisivo en el comportamiento como para merecer un tratamiento en profundidad. Aunque está muy imbricado con el control, tradicionalmente no ha demandado tanto interés de la comunidad robótica como aquel, y normalmente se ha visto simplificada a los datos sensoriales crudos, situando el problema en el sensor, o bien se ha considerado como una disciplina separada. Otros investigadores simplemente asumían que tenían una percepción correcta y completa [Maes, 1990], sin preocuparse de cómo se conseguía. Por el contrario nosotros creemos que gran parte de la complejidad de generar comportamiento inteligente reside en la capacidad de elaborar esa información, de extraer los estímulos relevantes desde los sensores y de simultanear esa tarea con la toma de decisiones, con el control propiamente dicho. Una percepción pobre limita la complejidad del comportamiento conseguido. Creemos que a la hora de generar comportamiento la percepción es más de la mitad del problema. Si se tienen los estímulos adecuados el control es relativamente fácil.

Definimos *estímulo* como una abstracción que captura información del mundo (incluido el propio robot) relevante para los objetivos. El concepto estímulo se interpone entre los sensores y la actuación directa, y permite escalar en complejidad. Esta distinción aparece también en la naturaleza. Hay

animales que tienen sensores específicos para una tarea concreta como los detectores de murciélagos que tienen algunas polillas (esos sensores no sirven para ninguna otra cosa). Sin embargo, es mucho más frecuente la presencia de sensores generalistas que sirven como base para una multitud de comportamientos. Por ejemplo, los ojos, oídos u olfato de los animales sirven para muchas cosas, sobre todo en animales superiores. Para estos sensores conviene la definición del concepto estímulo, mucho más cercano a la tarea concreta que el simple sensor. Elaborar esos estímulos desde la información sensorial cruda requiere cierto procesamiento perceptivo, cómputos o memoria. Los estudios de psicología sobre la percepción visual corroboran esta postura. Así, un estímulo puede no estar contenido directamente en los datos de un único sensor.

Muchos estímulos serán relevantes para los objetivos del robot, pero no todos los estímulos presentes en los datos sensoriales interesan al mismo tiempo o con la misma fuerza. El estímulo estará presente o no en la realidad, eso no depende del robot. Sí depende de él estar buscándolo activamente o tener despiertos los algoritmos que lo perciben caso de que exista. Al hablar aquí de atención estamos hablando de predisposición interna, de activar los procesos que en caso de que el estímulo exista en la realidad lo interioricen al robot, hacen que lo perciba y caracterice.

JDE proporciona un mecanismo de atención a través de la activación selectiva de esquemas perceptivos. En determinado instante sólo están activos los esquemas perceptivos que elaboran o buscan los estímulos que pueden interesar en ese momento. El procesamiento perceptivo se concentra en ellos, no desperdiciando esfuerzos en percibir estímulos ajenos a la tarea en curso o a la situación actual. Este mecanismo también se puede ver como una percepción situada, que selecciona qué estímulos buscar dependiendo de la situación concreta. En este sentido diferimos de otras muchas arquitecturas que consideran la percepción fija y buscan siempre todos los estímulos posibles. Para nosotros los sensores pueden estar midiendo continuamente las condiciones del entorno, pero la percepción tiene en todo momento una predisposición hacia un subconjunto de estímulos.

Además el estado de atención en que se encuentra el robot es dinámico pues no siempre interesan los mismos estímulos. Esa prediposición será hacia unos estímulos en cierta situación, y hacia otros en contextos diferentes. Un riesgo de la atención es que si se hace demasiado restrictiva entonces se corre el peligro de no ver más que lo que se quiere ver, perdiendo la detección de estímulos imprevistos interesantes. En general conviene tener la percepción enfocada, pero con los ojos abiertos a posibles nuevas cosas relevantes.

Una tendencia acentuada de la percepción en robótica es la que persigue reconstruir el entorno del robot. Esta *percepción reconstructivista* es frecuente desde los primeros robots móviles, que tratan de generar un mapa del entorno como paso previo antes de que el robot pueda navegar. Este mapa es una representación objetiva del entorno, con el que establece una correspondencia biyectiva. Ejemplos actuales en esta línea son los trabajos de Thrun con mapas 2D [Thrun *et al.*, 1998] y 3D desde láser o los que reconstruyen el entorno desde visión estéreo [Bustos *et al.*, 2001]. En contraste con esta aproximación nuestra arquitectura no impone esa representación objetiva. Si bien puede incluir esquemas que elaboren esos mapas, también puede manejar *percepción innata* [Lorenz, 1978], mucho más subjetiva. La percepción innata no compone un cuadro sobre la situación y actúa en consecuencia. Más bien parece que responde a estímulos separados, que se consideran aditiva pero independientemente [Lorenz, 1978]. Suelen ser estímulos sencillos, discriminativos pero burdos. Esto abre posibilidades en la robótica a enfoques como la psicología ecológica de Gibson o la escuela Gestáltica.

Control. Asumiendo que se tiene toda la información pertinente, ¿cómo obtener la respuesta adecuada? Existen muchas herramientas utilizadas en este propósito como la teoría de control clásico, el control borroso, la lógica, los *simuladores del mundo* (que simulan los efectos todas las actuaciones posibles y eligen la actuación actual en función de esas predicciones), tablas de situación-acción, etc.. Cualquiera de estas técnicas es utilizable dentro de JDE, el único requisito que se exige es el respeto a la interfaz obligada por nuestro diseño: esquemas activables a voluntad y modulables a través de parámetros.

En general cada esquema de control materializa un comportamiento objetivo. Si el comportamiento es sencillo el propio esquema emite directamente los comandos o las consignas a los actuadores del

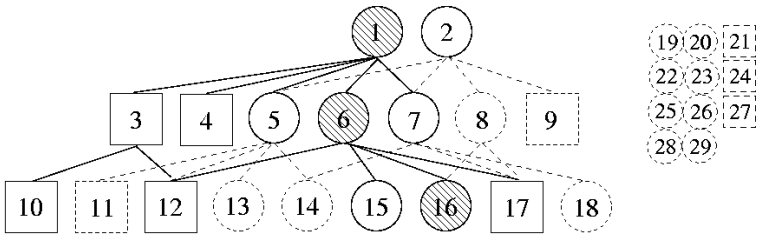


Figura 1.3: *Jerarquía de esquemas y pila de esquemas inactivos*

robot. Si es más complejo activa a otros esquemas motores y los modula continuamente a través de los parámetros para que lo hagan. En este sentido JDE permite la descomposición de una tarea en varias subtareas más simples.

En JDE el control se imbrica con la percepción porque cada esquema motor activa a los esquemas perceptivos que elaboran la información que él necesita para determinar correctamente la actuación adecuada. En cualquier caso la activación de nuevos esquemas hijo, tanto perceptivos como de control, no es bloqueante. Puesto que los esquemas son flujos independientes el padre sigue funcionando iterativamente a su ritmo, en paralelo con los hijos.

La figura 1.3 muestra el estado de la jerarquía de esquemas de la arquitectura en cierto instante. Los esquemas motores despiertos se representan por círculos sólidos, y los esquemas perceptivos activos por cuadrados sólidos. Los esquemas con contornos discontinuos estan dormidos o inactivos, esperando que alguien los despierte para ser de utilidad.

Así el esquema motor 1 activa los esquemas perceptivos 3 y 4 para que busquen y elaboren los estímulos que él necesita. También despierta a los esquemas de control 5, 6 y 7, y los modula para que reaccionen a los estímulos elaborados por 3 y 4 según sus propios intereses. Ellos materializan las respuestas motoras que le interesan al esquema padre, y consiguen los objetivos del esquema 1 a la vez que persiguen los genuinamente suyos.

El esquema 1 no despierta sólo a los esquemas hijo convenientes para la situación actual, sino que activa a todos los esquemas de nivel inferior que tratan situaciones que puedan presentarse en ese contexto. De esta manera los esquemas inferiores son activados y modulados por los superiores, recursivamente, conformado así una única jerarquía para cada comportamiento observable. Esta jerarquía acaba arraigando en su parte inferior en los sensores para los esquemas perceptivos y en las consignas a los actuadores para los esquemas motores.

1.5.3. Combinación en jerarquía

Como hemos anticipado, los esquemas se organizan en capas, en jerarquías que dependen del comportamiento deseado. Para exhibir comportamientos distintos los esquemas se reorganizan en jerarquías diferentes. La jerarquía se construye cuando la salida de un esquema motor son activaciones de otros esquemas de nivel inferior y sus parámetros de modulación. También cuando un esquema perceptivo activa otros esquemas perceptivos más sencillos para elaborar su propio estímulo. Por ejemplo, en la figura 1.3, el estímulo que elabora el esquema 3 está compuesto por los que elaboran los esquemas perceptivos 10 y 12. El estudio de la combinación de esquemas en jerarquía es un aporte de la tesis respecto de otros enfoques basados en esquemas. En percepción esta estratificación permite tener unos estímulos dependientes de otros. Como veremos en el capítulo 3 esto resulta útil para interpretar estímulos de bajo nivel y tener estímulos cuya naturaleza escapa a la modalidad de un único sensor, desde el cual sólo se puede tener una percepción parcial. En control permite manejar actuaciones más complicadas. En ambos casos la jerarquía permite elevar el grado de abstracción y aumentar la complejidad del comportamiento generado.

Para que se puedan combinar en jerarquías es necesario que la salida de un esquema pueda conectarse a la entrada de otros. La interfaz definida en nuestra propuesta así lo establece. Todos los esquemas deben respetar esa interfaz, aunque su contenido dependa de los comportamientos concretos que se quieran conseguir.

La activación de varios esquemas motores hijo obliga también a pensar en un mecanismo de

arbitraje. En JDE pueden haber varios esquemas motores despiertos por nivel en cada instante, pero sólo uno de ellos puede estar realmente activo, generando órdenes de control. JDE propone un mecanismo de arbitraje distribuido como mecanismo de coordinación entre comportamientos. Por ejemplo, los esquemas 5, 6 y 7 de la figura 1.3 están todos preparados, pero sólo el 6 gana la competición por el control en ese periodo. Es el único que emite control hacia abajo, activando a su vez los esquemas perceptivos 12 y 17, y los de actuación 15 y 16. Si hubiera ganado el esquema 5 habría despertado a los esquemas 11,12,13 y 14.

Según explicaremos en el capítulo 3, este arbitraje se basa en *regiones de activación* que el esquema padre configura para cada uno de sus esquemas motores hijo utilizando los estímulos disponibles por sus esquemas perceptivos hijo. Cada hijo chequea el estado de sus hermanos y si a él le toca ganar el control, según su región de activación. De esta manera también detecta vacíos y solapes de control, en el caso de que no haya ningún hermano activo o quieran activarse varios, respectivamente. En estos casos el padre decidirá qué hacer. Así el algoritmo de arbitraje se distribuye entre padres e hijos, de manera que la arquitectura no necesita ningún ente especial. Otras aproximaciones sí necesitan un árbitro central como [Rosenblatt y Pyton, 1989] para materializar la coordinación entre distintos módulos. En JDE todo son esquemas, resultando una arquitectura uniforme.

1.6. Estructura de la tesis

En este primer capítulo se ha trazado el contexto en el que se desarrolla la tesis, partiende desde la definición de la robótica, que es el más genérico, hasta el comportamiento autónomo de robots móviles, que es el más específico. Se han presentado las definiciones de los términos más usuales de la robótica como comportamiento, arquitectura y autonomía. Esto ha permitido enunciar de modo nítido el problema que abordamos en esta tesis y esbozar la solución que proponemos. En el resto de los capítulos explicaremos en detalle los distintos aspectos de la arquitectura propuesta.

En la segunda parte (estado del arte) describiremos distintas aproximaciones que han surgido en la literatura y que conforman el estado del arte en arquitecturas. Comentaremos los paradigmas que se identifican en la bibliografía y las arquitecturas concretas que materializan y son representativas de cada uno de los enfoques. Haremos una revisión crítica de todas ellas, sopesando sus pros y sus contras.

En el tercer capítulo (JDE) explicaremos en detalle la arquitectura de percepción y control que se propone para generar comportamiento autónomo en robots móviles. Se detallan los mecanismos incluidos en el diseño de la arquitectura, como la activación, la selección de acción, la atención y la interpretación y se hace especial énfasis en cómo la organización en jerarquía las facilita. Además se compara cualitativamente con las aproximaciones descritas en el capítulo segundo.

El cuarto capítulo (implementación) detalla la materialización de la arquitectura conceptual propuesta en una arquitectura software que se ejecuta en un robot concreto. Se presenta el soporte de programación con el cual se han implementado los mecanismos de activación, desactivación, modulación, selección de acción, etc.. presentados en el tercer capítulo. También se describe aqui toda la infraestructura software que ha sido necesario desarrollar para llevar la arquitectura a varios entornos concretos.

El quinto capítulo (experimentación) describimos un conjunto de esquemas perceptivos particulares, que elaboran estímulos relevantes para un robot de interiores. Por ejemplo, los que construyen una rejilla dinámica de ocupación, capaz de detectar los obstáculos en los alrededores del robot. También los que se encargan de percibir puertas. Todos ellos se han desarrollado siguiendo la ordenación de esquemas propuesta por JDE. Además se describen varios comportamientos concretos que se han realizado, tanto en el robot de referencia como en otros escenarios robóticos como la RoboCup. Estos comportamientos validan la arquitectura realizada y se han construido utilizando el soporte descrito en el cuarto capítulo.

Cerramos esta memoria con un último capítulo (conclusiones) que resume las aportaciones de este trabajo, las soluciones que plantea, las limitaciones que exhibe y las líneas de continuidad que se atisban para explorar en el futuro inmediato.

Capítulo 2

Estado del arte

Creo que releer es más importante que leer, salvo que para releer se necesita haber leído
Jorge Luis Borges

En este capítulo revisamos los distintos enfoques que históricamente han ido apareciendo para generar comportamiento artificial inteligente en robots móviles. Con este repaso se describe el estado del arte actual de las arquitecturas de control que se han venido utilizando para generar comportamiento autónomo. Entre ellas hemos identificado varias corrientes diferentes, cada una de las cuales se articula alrededor de ciertas ideas troncales. En este capítulo exponemos los paradigmas principales y las ideas que hemos encontrado características de cada uno de ellos, haciendo hincapié en las soluciones que aportan y en las limitaciones que exhiben. El análisis no pretende ser exhaustivo pero sí enfatizar los conceptos y las dificultades más importantes que han ido surgiendo a medida que estos enfoques evolucionaban.

Las fronteras entre los distintos enfoques no son nítidas, y las arquitecturas concretas suelen tener ingredientes de varios de ellos, aunque pongan más énfasis en unos que en otros. No obstante, dentro de cada escuela comentaremos algunas arquitecturas particulares, las más representativas. Ellas ilustran diferentes materializaciones de los principios de cada paradigma, a modo de ejemplo.

Encuadramos cada escuela en la definición establecida en el capítulo anterior para la arquitectura de un robot, que engloba las cuestiones de percepción y de control, como la construcción autónoma de estímulos, la atención, la selección de acción y la descomposición de la complejidad. Comentamos qué soluciones plantea cada una de las arquitecturas propuestas a esas cuestiones. Las diferentes respuestas dadas en las arquitecturas descritas nos permitirán compararlas. Adicionalmente evaluaremos someramente cada uno de los enfoques atendiendo a los criterios presentados anteriormente, como la robustez, la reactividad, la finalidad y la escalabilidad.

Tradicionalmente el estado del arte en este campo se ha dividido en tres grandes corrientes: los sistemas deliberativos, los reactivos y los híbridos. Clasificaciones similares existen en [Murphy, 2000; Arkin, 1998; Arkin, 1995; Coste-Maniére y Simmons, 2000] y en muchas tesis de los años 90 [Schneider Fontán, 1996; Serradilla, 1997]. Sobre esa clasificación típica hemos preferido distinguir entre los sistemas reactivos puros y los sistemas basados en comportamientos (SBC). Como luego veremos, aunque tienen muchos puntos en común, los SBC suponen un paso adicional sobre los reactivos puros porque aumentan el repertorio del robot utilizando la composición de varios comportamientos individuales.

Hemos añadido una corriente adicional que se ha ido confirmando en los últimos años: la que busca inspiración en la etología para diseñar las arquitecturas de los robots [Tyrrell, 1993a; Blumberg, 1997; Arkin *et al.*, 2003]. Esta tendencia refleja el movimiento del campo hacia nuevas áreas, quizá fruto de cierto estancamiento dentro de las aproximaciones más tradicionalmente robóticas en cuanto a las cuestiones de arquitectura.

También hemos decidido incorporar al estado del arte otra aproximación: la teoría clásica de control y sistemas. A pesar de que normalmente se queda fuera de las referencias de robótica móvil, nos parece otro modo más de generar comportamiento, principalmente útil en entornos muy controlados. Dedicaremos la primera sección a describir sus características y limitaciones

fundamentales. Argumentaremos que en general no es válido para la arquitectura de un robot porque es difícil elaborar los modelos exactos del entorno y su funcionamiento que el enfoque demanda.

2.1. Teoría clásica de control y sistemas

En general, controlar un sistema es modificar su comportamiento para que evolucione de una forma determinada. En este sentido la teoría de control también es una alternativa posible a la hora de conseguir que un robot se comporte de una determinada manera. Históricamente la ingeniería de control ha tenido aplicaciones industriales directas logrando que cierto sistema se comportara del modo adecuado. Sistemas como plantas químicas, centrales térmicas, circuitos amplificadores, aeronaves, coches, etc. incluyen cierto control que garantiza su estabilidad y su buen funcionamiento.

Dentro del escenario del control se identifican varios componentes interrelacionados. La *planta* es el sistema a controlar y el objetivo de control es la forma en que se quiere que el sistema evolucione. El *controlador* propiamente dicho interactúa con la planta para conseguir el objetivo. Un objetivo típico de control es la regulación, que persigue que la señal de salida alcance y mantenga un nivel de referencia o que no se salga de ciertos márgenes. Otros objetivos son la optimización (por ejemplo minimizar el consumo o el coste), el seguimiento o simplemente alcanzar cierto estado final dentro del espacio de posibles estados.

A lo largo de los últimos 130 años se han ido acumulando técnicas y métodos matemáticos que permiten un tratamiento sistemático del problema de control, tanto del análisis de sistemas como de la síntesis de controladores. Estas técnicas conforman la teoría clásica de control, que incluye identificación y modelado de sistemas, transformada de Laplace, transformada Z, análisis de estabilidad, métodos de control estocástico, robusto y adaptativo, pasando por el análisis en el espacio de estados y el control óptimo. Estas herramientas han permitido controlar de modo preciso a los sistemas dinámicos cada vez más complicados, no sólo las plantas determinísticas lineales e invariantes con el tiempo, que son las más sencillas, sino también los sistemas no lineales e incluso sistemas estocásticos. También permiten analizar la estabilidad de un sistema, sus puntos de equilibrio y la síntesis del controlador adecuado para que cierta planta satisfaga unos objetivos de control determinados.

Todas estas técnicas se basan en una fuerte caracterización del sistema a controlar, en un modelo de su funcionamiento. El modelo describe su comportamiento alrededor de un determinado punto de funcionamiento o en un cierto rango de condiciones de trabajo, y puede tener varias formas. Por ejemplo, para ciertos sistemas se tiene su *función de transferencia*, bien en el dominio del tiempo, bien en el dominio de la frecuencia, que expresa cómo evolucionan sus salidas en función de sus entradas. Otros formatos útiles del modelo son las ecuaciones diferenciales, la respuesta al impulso, las ecuaciones en diferencias y los autómatas para sistemas con dinámicas discretas.

De hecho, el primer paso en el diseño de un controlador para un sistema concreto consiste en obtener ese modelo matemático de la planta, es decir caracterizar el funcionamiento del sistema. Aquí la teoría clásica ofrece técnicas de identificación de sistemas y estimación de parámetros. El modelo se puede destilar analizando la física, mecánica o química del sistema, es decir, analizando el interior del sistema. Otro enfoque, tipo caja negra, consiste en observar desde fuera el comportamiento del sistema y utilizar un modelo patrón parametrizado. Con las entradas y salidas observadas se estiman los parámetros que adaptan el patrón a la planta concreta, obteniendo un modelo matemático cuyo comportamiento coincide con el del sistema real.

Una vez que se dispone del modelo de planta hay un conjunto de técnicas sistemáticas de síntesis de controlador, que determinan el controlador adecuado para esa planta y los objetivos de control que se requieren. La mayor dificultad de todo el problema reside normalmente en conseguir un buen modelo del sistema a controlar, que puede incluso cambiar su comportamiento con el tiempo. Para contemplar esta posibilidad hay sistemas que dinámicamente reestiman el modelo de la planta y corrigen al controlador para adaptarlo al nuevo comportamiento del sistema.

En cuanto al número de variables, el caso más sencillo de sistema es aquel que tiene una única variable de entrada y una única variable de salida. En estos sistemas, denominados SISO (*Single Input*

Single Output), la percepción está muy clara: es la variable de entrada, en general continua, la cual está siempre disponible y no hay que elaborarla, ya viene dada. No hay abstracción ni mecanismos de atención posibles. La percepción se sitúa en cierto sensor, que es quien proporciona esa señal de entrada. La actuación también es relativamente sencilla, es la variable de salida, cuyo valor hay que determinar para cada instante. Es el controlador el que realiza la selección de acción adecuada. No hay descomposición de complejidad, por cuanto que los valores del controlador suelen atacar directamente a los actuadores de la planta. En los casos más complejos el sistema puede tener muchas señales de entrada y tiene que determinar el valor de múltiples variables de salida. En estos casos, llamados MIMO (*Multiple Input Multiple Output*) el control se complica pero responde en esencia al mismo planteamiento.

2.1.1. Control en lazo abierto y realimentación

Se dice que un sistema está controlado *en lazo cerrado* cuando el valor de salida del controlador se calcula en base a la diferencia entre la salida de la planta y el valor deseado, es decir, la salida de la planta realimenta la acción del controlador. Por este motivo al control en lazo cerrado también se le denomina *control realimentado*. La figura 2.1 en concreto 2.1(a) muestra un ejemplo de sistema controlado siguiendo ese esquema. Según especifica [Ogata, 1990] "el controlador aplica a la planta controlada una variable manipulada para corregir o limitar la desviación de la salida respecto del valor deseado". El *error(t)* entre la salida real y el objetivo constituye una entrada más para decidir con qué valor alimentar a la planta.

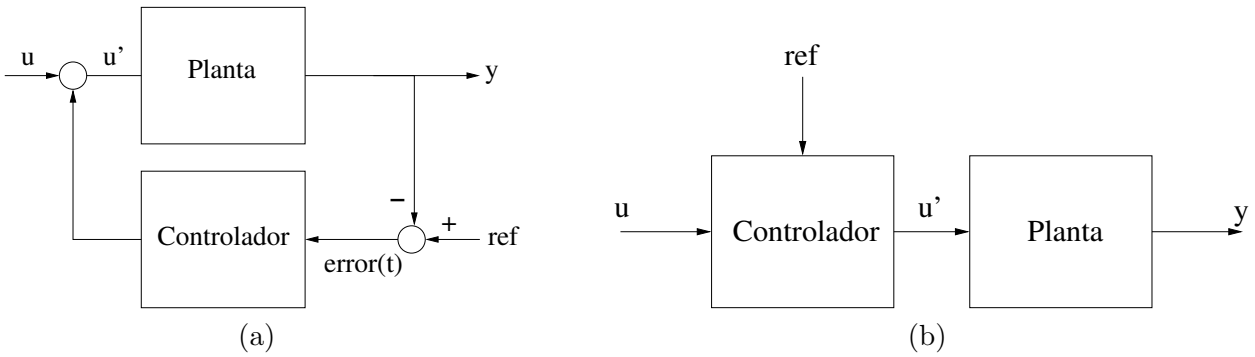


Figura 2.1: *Ejemplo de bucle de control realimentado (a), y de control en lazo abierto (b)*

Los sistemas realimentados son indispensables en la práctica porque son robustos frente a pequeñas variaciones en las condiciones de trabajo. Es la realimentación negativa la que confiere esa robustez frente a perturbaciones. No obstante, si se diseña mal el controlador realimentado se puede comprometer la estabilidad del sistema conjunto, que puede entrar en oscilaciones indeseadas (realimentación positiva).

Alternativamente un sistema de control se dice *en lazo abierto* si la salida del controlador no se ve influída por el valor de salida de la planta. Tal y como muestra la figura 2.1, en concreto la 2.1(b). Por esto a veces también se llama técnica de control hacia delante (*feedforward*). Estos controladores siguen una técnica indirecta, completamente basada en modelo para generar la salida adecuada. Consiste en invertir las ecuaciones de funcionamiento de la planta (cuando sea posible) y obtener la entrada que debemos dar a la planta si queremos que su salida sea tal o cual. Por ejemplo, tal y como muestra la figura 2.1(b), si la dinámica del sistema viene descrita por $y(t) = R(u(t))$ este enfoque calcula la dinámica inversa del sistema, $u(t) = R^{-1}(y(t))$ y con ello estima qué salida de control introducir al sistema $u'(t) = R^{-1}(ref(t))$. Esa es la salida que genera el controlador y que alimenta como entrada a la planta, para que la salida global sea la deseada $y(t) = R(R^{-1}(ref(t))) = ref(t)$.

En general las técnicas hacia delante dependen críticamente de la integridad del modelo y son más frágiles que los sistemas realimentados frente a ruidos y perturbaciones. Sin embargo cuando ese modelo exacto es obtenible consiguen un control muy preciso, que se anticipa a cualquier desviación de la salida, mientras que con la realimentación se necesita cierta desviación para corregir el comportamiento.

2.1.2. Limitaciones del control clásico

Debido a la necesidad de un modelo matemático preciso, como ecuaciones diferenciales o la función de transferencia, las técnicas clásicas de control no digieren bien la incertidumbre o los modelos incompletos. Por ejemplo, no son capaces de sintetizar controladores adecuados si sólo se tiene conocimiento incierto o parcial acerca del comportamiento del sistema. Esta situación contrasta por ejemplo con un operador humano que sí es capaz de utilizar información imprecisa o lingüística sobre el sistema para controlarlo de modo efectivo. En este sentido, en los últimos años se han desarrollado nuevas técnicas de control *inteligente* [Passino, 1995], también llamado *soft-computing*, que se apoyan en la lógica borrosa, los algoritmos genéticos o las redes neuronales.

El control borroso admite modelos del sistema expresados en términos lingüísticos o difusos, que capturan muy bien el conocimiento impreciso e incluso incompleto de un sistema. Describiremos con más detalle estos controladores más adelante, con algún ejemplo de arquitecturas completas de robots basadas en ellos. Los algoritmos genéticos se pueden utilizar en este contexto para sintonizar controladores. Con ellos se puede agilizar la búsqueda en el espacio de posibles parámetros del controlador simulando de antemano la aplicación un cierto conjunto de parámetros y teniendo una *función de salud* (*fitness*) que evalúa sus resultados. Los controladores basados en redes neuronales pueden utilizarse para aprender el control correcto desde una colección de pares entrada-salida, que describe con ejemplos cuál es la salida de control adecuada ante cierta situación de entrada.

En general las técnicas clásicas de control resultan útiles cuando se pueden conseguir modelos precisos de comportamiento de la planta a controlar. Si ese modelo analítico o funcional se puede obtener, entonces las herramientas clásicas son muy potentes y permiten un control muy preciso. Sin embargo para sistemas complejos resulta imposible obtener ese modelo, con lo que estas técnicas devienen en inútiles para el problema general. Es decir, consiguen resultados muy buenos y precisos para escenarios sencillos modelizables pero no escalan bien cuando el problema a controlar crece en complejidad. Este es el gran talón de Aquiles de la teoría clásica de control y sistemas a la hora de abordar el comportamiento complejo en robótica móvil. El escenario típico de un robot moviéndose por un entorno dinámico de oficina con muchos sensores ya constituye un sistema demasiado complicado como para conseguir un modelo cuantitativo significativo que caracterice su dinámica. Si además introducimos que el escenario es dinámico, multiobjetivo y que los objetivos concretos pueden cambiar radicalmente a lo largo del tiempo entonces el sistema se complica aún más, haciendo inviable abordarlo con estas técnicas. Por lo tanto no son una buena solución para la arquitectura de percepción y control de un robot móvil. Sin embargo, sí se pueden utilizar para solucionar problemas parciales, como generar un comportamiento específico en cierto contexto.

2.1.3. Controladores Proporcional-Integral-Derivativos

Debido a estas limitaciones la teoría de sistemas y control no se utiliza como solución para la arquitectura de control, pero sí se emplea con profusión en los distintos componentes de un robot móvil. Un ejemplo típico son los controladores Proporcional-Integral-Derivativo (PID), que aparecen en muchas plataformas móviles como el B21 (Irobot), Robuter (Robosoft), Pioneer (ActivMedia) como módulos de bajo nivel para mantener una velocidad comandada, muy próximos a los motores físicos. Por ejemplo, en la figura 2.2 se observa la evolución de la velocidad real cuando está regulada con un controlador PID. Estos controladores también son frecuentes en las articulaciones de los brazos robóticos y de los robots con patas, en este caso estableciendo la realimentación básica desde las posiciones medidas.

Los controladores PID siguen una ley de control realimentado guiada totalmente por el error, definido como la diferencia entre la salida real del sistema y la referencia. En concreto, la ecuación genérica es (2.1), que establece la salida de control como suma de tres componentes: la componente Proporcional al error, la proporcional a la Derivada del error y la proporcional a la Integral del error.

$$u'(t) = K_p(ref - y(t)) + K_d\dot{y}(t) + K_i \int_{t_o}^{t} (ref - y(t))\, dt \tag{2.1}$$

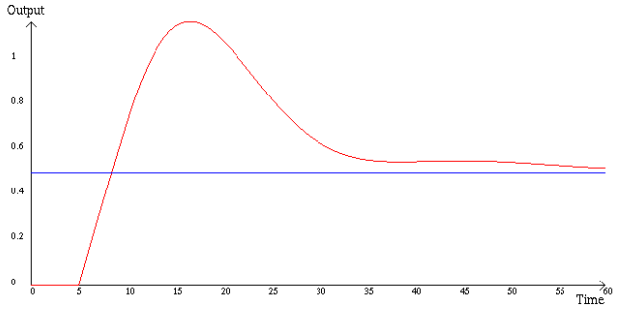


Figura 2.2: *Ejemplo salida controlada con controlador PID*

La componente P hace que el sistema conjunto quiera compensar cualquier desviación respecto de la referencia $ref(t)$. Sin embargo por sí sola provoca siempre una desviación en la salida del sistema conjunto. Para corregir esta desviación sistemática se introduce la componente I, que además acelera la reacción del sistema conjunto frente a diferencias con el valor de referencia.

Los controladores PI suelen tener una respuesta rápida, en ocasiones esa rapidez les hace oscilar porque no frenan la salida del control $u'(t)$ cuando la salida conjunta $y(t)$ se está acercando a la referencia. Para corregir esas oscilaciones se utiliza la componente D o de amortiguación, que aminora la corrección cuando el error tiende a disminuir.

En las plataformas móviles normalmente los programas de control autónomo desembocan periódicamente en ciertas consignas de velocidad de tracción y de giro, pero no se encargan de generar la señal que alimenta directamente a los motores. Los fabricantes ofrecen un microcontrolador con un tiempo de ciclo mucho menor que materializa los controladores PID para el motor de tracción y para el de giro. La realimentación se establece entre las consignas de velocidad entregadas por los programas, que se toman como referencia, y la estimación de velocidad desde los encoders de cada motor. Ciertas características físicas como la aceleración máxima del motor determinan los parametros PID con los que opera en cada momento, que son ajustados por el fabricante.

Estos controladores realimentados sencillos resultan suficientes para muchas aplicaciones, por ejemplo cuando se requiere seguir una consigna de referencia, por lo que su uso está muy extendido. Una de sus ventajas es que al ser realimentados son robustos frente a perturbaciones, cualquiera que sea su origen. Sin embargo su principal ventaja es que los parámetros K_p,K_d y K_i se pueden ajustar heurísticamente para adaptarlo a la planta concreta, sin necesidad de modelar el comportamiento de la planta. Aunque en este caso, no tendremos modelo del sistema conjunto, ni el análisis de estabilidad, ni el diseño óptimo sistemático del controlador. Este ajuste a mano sustituye a un ajuste basado en el modelo de la planta que sería más óptimo, pero que requeriría el esfuerzo de conseguir el modelo.

Por supuesto, si el sistema es suficientemente complejo no se puede abordar con PID, pero muchos sistemas sencillos se pueden controlar con ellos, sin necesidad de modelar la planta. No son, pues, una solución posible al problema de arquitectura.

2.2. Sistemas deliberativos simbólicos

Como ya avanzamos en la introducción, la disciplina de la *Inteligencia Artificial* ha influído notablemente en el desarrollo de la robótica móvil. Las arquitecturas tradicionalmente consideradas como *deliberativas*, o de la escuela simbólica son las herederas de las investigaciones en IA clásica y fueron las primeras en surgir en el mundo de la robótica móvil para generar comportamiento, siendo las dominantes hasta finales de los años 80.

En general, estas arquitecturas asumen que generar comportamiento en un robot consiste en ejecutar una secuencia de acciones calculada de antemano (*plan*), que se elabora razonando sobre cierto modelo simbólico del mundo. El robot necesita una descripción simbólica del estado de su entorno (por ejemplo, un mapa de ocupación, con sus paredes, pasillos y puertas, o una relación de hechos lógicos sobre ese estado), y de su funcionamiento (por ejemplo, el efecto de sus propias acciones). La inteligencia del robot se halla principalmente en el planificador, el cual delibera sobre los

símbolos del modelo del mundo y genera un plan de actuación explícito. El plan elaborado contiene el curso deseado de la acción en el futuro para conseguir el objetivo. Por ejemplo, la secuencia de operadores para conseguir el estado objetivo a partir del estado actual o la ruta óptima para llegar al punto destino.

Los orígenes epistemológicos de este enfoque están en la filosofía cartesiana que considera el alma como el ente que decide el comportamiento [McFarland y Bösser, 1993]. Hunde sus raíces en el cognitivismo y representa toda una teoría del funcionamiento de la inteligencia. Era natural que la Inteligencia Artificial clásica buscara refrendo probando su capacidad de generar comportamiento inteligente en cuerpos robóticos. Su aporte a la robótica ha sido fundamental en el desarrollo de esta última, introduciendo ideas como el manejo de símbolos, la planificación y la jerarquía para abordar la complejidad.

En esta sección introduciremos cuatro muestras representativas de las ideas fundamentales del enfoque, como son el robot Shakey, el robot Hilare, la arquitectura SOAR y la arquitectura RCS. Estas arquitecturas han sido elegidas por su carácter fundacional e ilustrativo, más que por ser recientes. A continuación describiremos los principales puntos comunes a las arquitecturas de este paradigma, como son la necesidad de planificación y de modelado del mundo, el uso de jerarquía y el bucle típico de ejecución sensar-modelar-planificar-actuar.

2.2.1. Shakey

El robot Shakey[1] se considera el primer robot móvil controlado por computador y supone un buen ejemplo de este paradigma (figura 1.2). Esta plataforma era capaz de navegar en una habitación con obstáculos, y de empujar un bloque de un punto a otro. La arquitectura de control de Shakey está dividida en dos niveles, el inferior y el superior, según muestra la figura 2.3(a). El nivel inferior lo constituyen varias funciones especializadas, llamadas *unidades de acción*. Cada una de ellas tiene su modelo y su deliberador propios. Son unidades en el sentido de que una vez comenzada su ejecución no se detiene hasta que bien termina con éxito o bien se detecta alguna situación anómala. Además se apoyan unas en otras. Por ejemplo, la unidad encargada de llevar al robot a un punto lejano se apoya en un planificador de caminos y en el ejecutor de esa ruta que le hace pasar por todos los puntos intermedios.

El planificador de caminos utiliza un modelo del mundo en forma de rejilla multiresolución de ocupación [Nilsson, 1969], como la mostrada en la figura 2.3(b). Sobre esta representación planifica cuál es el mejor camino para alcanzar el punto destino desde la localización actual, utilizando un algoritmo geométrico. Ese modelo del mundo se actualiza periódicamente desde las imágenes tomadas a bordo, las cuales son analizadas buscando bordes conexos, para caracterizar los contornos de los objetos del entorno. Una vez encontrados se hipotetiza que la parte inferior de la imagen corresponde al suelo, por lo que en los primeros bordes encontrados subiendo por la imagen empiezan los obstáculos. De este modo se consigue un perfil de proximidad desde la imagen, que es utilizado para actualizar el mapa de ocupación en forma de rejilla.

El nivel superior de la arquitectura conceptual de Shakey se encarga de deliberar sobre las acciones del robot en un nivel de abstracción más elevado. Para ello hace deducciones lógicas sobre una representación del mundo expresada en forma de lógica proposicional. El planificador elegido inicialmente en este nivel era el *Question-Answering System* (QA-3), basado en los métodos de demostración de teoremas. Se utilizaba con vocación generalista, de manera que el mismo deliberador pudiera utilizarse para muchas tareas, sin más que cambiar la descripción del mundo y las consecuencias de las acciones. Posteriormente se incorporó el planificador STRIPS [Fikes y Nilsson, 1971], otra variante del *General Problem Solver*, basada en análisis de medios-fines. *STanford Research Institute Problem Solver* (STRIPS) basa sus decisiones acerca de cuáles operadores elegir en una tabla de diferencias: si el estado final no se puede conseguir con la aplicación de un único operador, entonces elige el operador que más reduce la diferencia entre el estado predicho siguiente y el objetivo. De esta manera es capaz de encadenar operadores que llevan hasta el estado deseado. Los efectos lógicos de un operador se almacenan en forma de lista, la lista de postcondiciones, que inlcuye tanto hechos

[1] http://www.sri.com/technology/shakey.html

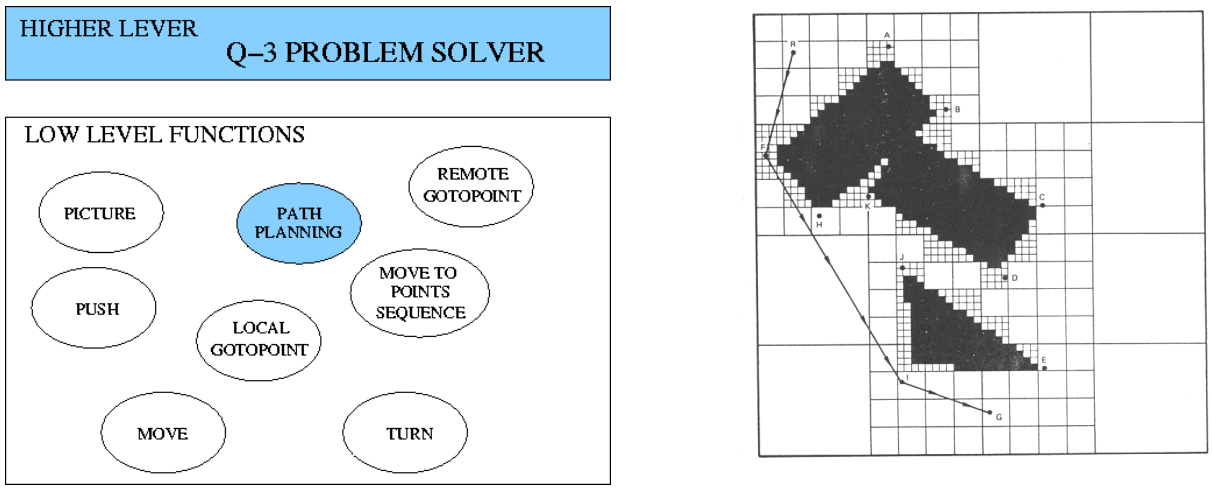


Figura 2.3: *Arquitectura del robot Shakey (a) y representación geométrica del mundo (b).*

que dejan de ser ciertos (`delete-list`) como nuevos hechos verdaderos (`add-list`). Además a los operadores se les asocia una lista de precondiciones, que deben ser ciertas para que el operador sea aplicable (`preconditions-list`).

2.2.2. Hilare

Otra muestra significativa de este paradigma deliberativo es el robot Hilare[2]. Para ir de un punto a otro utiliza el módulo de navegación, que se describe en la figura 2.4(a), y que incluye un planificador, un controlador de ejecución y varios submódulos (*Specialized Processing Module*, SPM) necesarios para construir y actualizar el modelo del mundo [Giralt *et al.*, 1983].

La arquitectura de Hilare describe topológicamente el entorno utilizando un grafo, y geométricamente con mapas de segmentos conectados, como ilustra la figura 2.4(b). Submódulos como `mapping` y `define place` de la figura 2.4(a) se utilizan para construir y actualizar ese modelo del mundo. Ellos incorporan los algoritmos para construir segmentos y nodos desde los datos sensoriales, y actualizar la representación interna.

Análogamente la planificación de navegación incorpora dos niveles. El primero es un nivel global que opera sobre el grafo topológico y descompone la misión de ir a un punto en una secuencia de puntos intermedios, utilizando el submódulo `routing`. En un segundo nivel, para cada punto intermedio el planificador busca la ruta entre los segmentos locales, empleando el submódulo `path search`. Adicionalmente el controlador de ejecución es un sistema de reglas que monitoriza el perfecto seguimiento del plan.

Otra característica de la arquitectura de decisión de Hilare [Giralt *et al.*, 1983] es la descomposición en módulos especializados. Su arquitectura global se compone de varios módulos de decisión (SDM), cada uno especializado en cierta tarea. El SDM que aparece en la figura 2.4(a) sólo es uno de ellos, precisamente el dedicado a la navegación y control de movimiento. El sistema completo tiene también un SDM para el manejo del manipulador, un SDM especialista en comunicación y lenguaje natural y un SDM encargado del modelado de objetos y comprensión de la escena.

En la misma línea de Hilare, otro ejemplo similar es la arquitectura de Crowley para un robot móvil [Crowley, 1985]. Como representación del entorno utiliza un modelo simbólico del mundo, descompuesto en un nivel global y otro local. En el nivel local se utilizan segmentos como primitiva perceptiva, los cuales se construyen y actualizan desde los datos sensoriales. En el nivel global se divide el espacio en regiones convexas, en una suerte de mapa topológico (*network of places*). Sobre esta última representación un planificador de caminos destila un plan, el cual se especifica como una secuencia de

[2] http://www.laas.fr/~matthieu/robots/

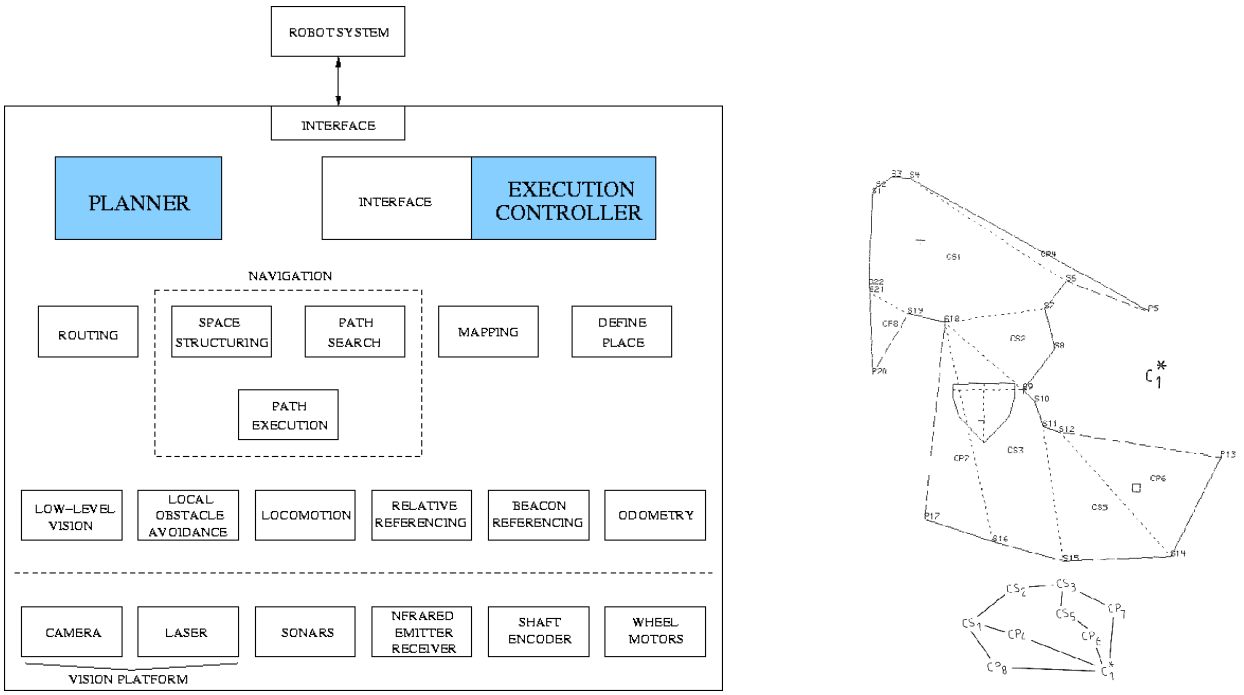


Figura 2.4: *Módulo de navegación y control de movimiento (a) y descripción con segmentos del entorno del robot Hilare (b).*

lugares intermedios. La ejecución local de ese plan genera el comportamiento de navegación deseado. De hecho, esta arquitectura está específicamente diseñada para el comportamiento de navegación. A continuación veremos dos arquitecturas deliberativas con una vocación más generalista, aplicables a problemas de comportamiento más amplios.

2.2.3. SOAR

La arquitectura SOAR [Laird *et al.*, 1987] refleja toda una teoría unificada de la inteligencia, que hunde sus raíces en la hipótesis de sistema de símbolos físicos de Newell y Simon [Newell y Simon, 1976]. Esta hipótesis enuncia que un sistema de símbolos físicos tiene los medios necesarios y suficientes para mostrar inteligencia. Esta arquitectura ha sido acompañada siempre por implementaciones reales que materializan sus ideas. Aunque no ha sido aplicada a robots móviles reales con profusión, contiene las ideas inspiradoras del paradigma deliberativo como la manipulación de símbolos y la descomposición de tareas en subtareas. Reune todos los ingredientes que se adivinaban subyacentes al comportamiento inteligente: aprendizaje, resolución de problemas, aplicación a un rango amplio de tareas, actuación dirigida a conseguir objetivos, etc.

En esencia SOAR es un sistema que representa de modo explícito las tareas, y contiene unos procesos simbólicos que manipulan esa representación. De modo genérico, el conocimiento de fondo de SOAR se almacena y expresa en forma de sistemas de reglas (sistemas de producción). Una de sus características fundamentales es que plantea cualquier actividad simbólica dirigida a conseguir un objetivo como un problema de búsqueda del estado deseado en un espacio de representación del problema determinado (*problem space hypothesis*). Esta formulación se apoya en la definición de estados para cada tarea y operadores cuya aplicación permite pasar de un estado a otro. Esa búsqueda es el corazón de la manipulación simbólica, y está guiada por heurísticos que se denominan preferencias. Esos heurísticos son específicos de cada tarea y recortan el espacio de búsqueda, acelerando el avance hasta el objetivo.

SOAR construye dinámicamente un árbol de objetivos, descomponiendo automáticamente las tareas en subtareas. Los subobjetivos se crean automáticamente cuando el sistema no puede resolver el avance o la resolución de algún procesamiento (*universal subgoaling*). Por ejemplo, se crea un

subobjetivo para la elección del operador adecuado en cierto estado y otro para la selección de la implementación del operador elegido. Cada uno de ellos se convierte en un problema de búsqueda en cierto subespacio de ese nuevo problema concreto. La progresión de las subtareas se monitoriza continuamente, así como su terminación.

El sistema se completa con una técnica general de aprendizaje basada en la experiencia, llamada empaquetado (*chunking*). A grandes rasgos, cuando un proceso de búsqueda se invoca para resolver una subtarea, el mecanismo almacena y empaqueta el resultado y el contexto. La próxima vez que se requiere resolver una subtarea similar ya no se dispara el proceso de búsqueda, sino que se entrega directamente el resultado que ofreció la primera invocación.

La arquitectura SOAR y sucesivas adaptaciones suyas se han utilizado con éxito para desarrollar varias aplicaciones comerciales y de investigación[3]. Se ha empleado en programas que aprenden e interpretan texto en inglés, en agentes que examinan grandes bases de datos, etc.. De modo más específico se ha utilizado para controlar un brazo robótico a partir de visión [Laird y Rosenbloom, 1990], para navegación de entornos simulados (por ejemplo en el desarrollo de un piloto de combate en simuladores militares), y para controlar el movimiento de jugadores automáticos en un videojuego. Una perspectiva más amplia de las aplicaciones y de la evolución temporal de esta arquitectura desde sus orígenes se tiene en [Laird y Rosenbloom, 1996].

2.2.4. RCS

El modelo de referencia *Real-time Control System* (RCS) ha sido desarrollado por James Albus [Albus, 1993; Albus, 1996]. Basa su teoría de la inteligencia en tres componentes principales: el procesamiento sensorial (*Sensory Processing*, SP), el modelado del mundo (*World Modelling*, WM) y la descomposición de tareas (*Task Decomposition*, TD). El modelo ha sufrido diferentes ampliaciones desde sus orígenes, contándose hasta cuatro generaciones. La misión del procesamiento sensorial (SP) es crear y actualizar desde los sensores la conexión entre un modelo interno y el mundo real. Este módulo recoge los datos sensoriales, y los procesa, filtrándolos si es necesario y realizando su integración e interpretación. Además se encarga de la detección y el reconocimiento de entidades y eventos en el mundo real, casándolos con sus representaciones internas, haciendo un uso intensivo para ello de predicciones y medidas de semejanza.

El módulo de modelado del mundo (WM) mantiene una representación interna del mundo exterior. Actúa a modo de buffer intermedio entre el SP y el TD, tal y como muestra la figura 2.5. De una parte genera predicciones para SP y actualiza el modelo desde los datos que le suministra. De otra, responde a las necesidades de información del módulo TD y simula los resultados de planes hipotéticos. Incluye una base de datos con información sobre el espacio, el tiempo, entidades, eventos, estados del mundo y leyes de la naturaleza. Estas leyes se pueden representar por fórmulas o con reglas SI-ENTONCES, las cuales especifican qué sucede en ciertas circunstancias, y se usan para poder simular las acciones y razonar sobre ello.

El módulo de descomposición de tareas (TD) incluye tres subniveles. El primero, llamado *asignador de trabajos*, divide espacialmente la tarea en subsistemas diferentes, por ejemplo entre el subsistema de la cámara y el subsistema de manipulación. El segundo subnivel, llamado *planificador*, se encarga del aspecto temporal de esa descomposición, fraccionando cada tarea en una secuencia de subtareas. Finalmente el tercer subnivel, llamado *ejecutor*, proporciona la realimentación necesaria para que la acción siga el curso planeado, detectando contingencias y situaciones de emergencia si aparecen. Es el encargado final de ejectuar el plan. En conjunto, el módulo TD incluye conocimiento de las tareas, incluyendo el objetivo de cada una de ellas, la información que se requiere para llevarla a cabo, y cómo implementar su consecución, bien sea como reglas que lanzan procedimientos precompilados, bien como planificadores específicos.

Estos tres ingredientes son los pilares de la tercera generación, RCS-3, que recibe el sobrenombre de NASREM. Unos trabajos posteriores dieron lugar al modelo de referencia RCS-4, que incluye un cuarto elemento principal: el juicio de valor (*Value Judgement*, VJ) [Albus, 1996]. Este componente se utiliza para evaluar costes y beneficios de los planes potenciales, y para evaluar el atractivo y la

[3] http://www.soartech.com

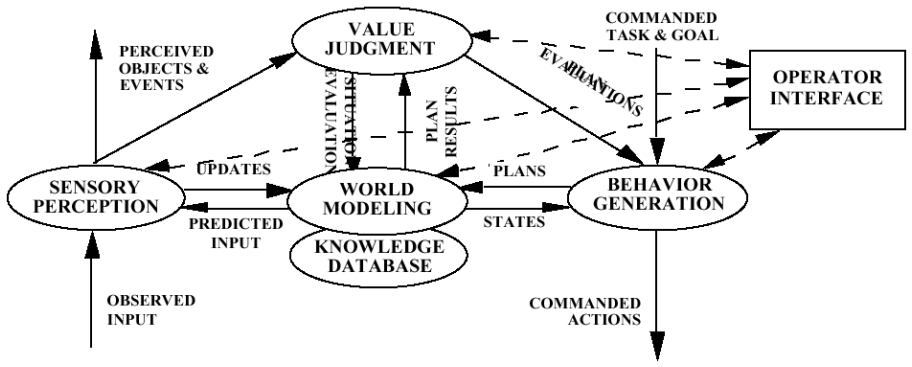


Figura 2.5: *Componentes principales en cada capa del modelo RCS de Albus*

incertidumbre de los objetos y eventos en el mundo. Incluye criterios de decisión, y sus variables de estado sesgan la toma de decisiones sobre qué curso de acción elegir.

El sistema global, mostrado en la imagen 2.6, es capaz de generar comportamiento inteligente y se articula como una jerarquía de capas, organizadas en niveles de abstracción. Cada capa contiene los elementos principales antes mencionados, como ilustra la figura 2.5. Cada capa tiene su funcionalidad, que es utilizada por niveles superiores, y su propio ritmo. El planificador en cada nivel tiene un determinado horizonte temporal, e incluso su resolución en el tiempo. Del mismo modo cada capa tiene su alcance y resolución espaciales. Como norma general los niveles más bajos suelen ser más rápidos y más locales que los superiores.

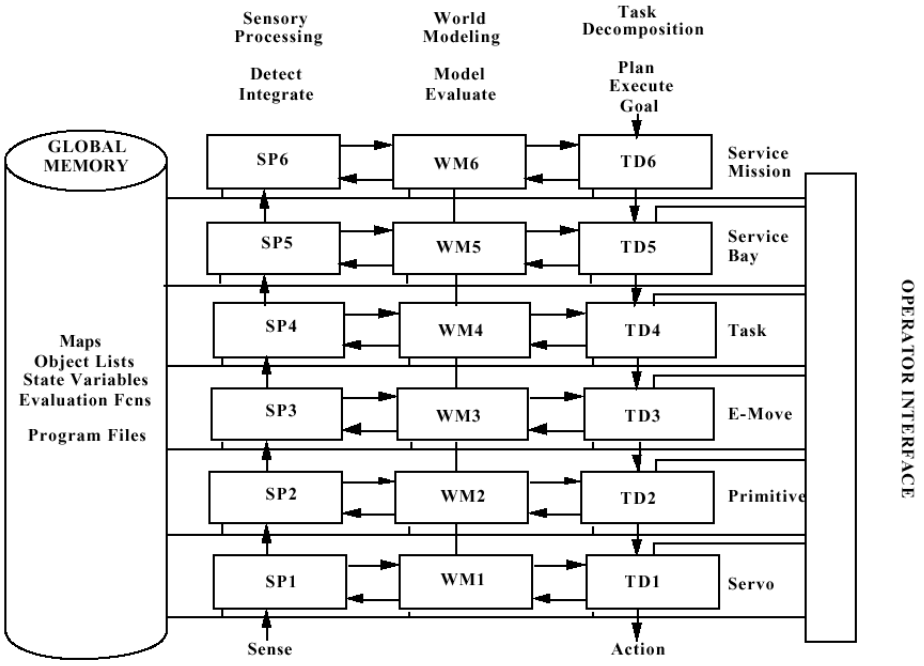


Figura 2.6: *Arquitectura NASREM (RCS-3)*

La interfaz de una capa con su vecina superior es doble. Hay un flujo ascendente de información sensorial, en el cual el módulo SP de cada capa elabora entidades y eventos que además de actualizar el modelo del mundo de esa capa, sirven como datos de entrada al SP de nivel superior, que elabora otro modelo más abstracto. De esta manera se elabora un modelo del mundo que tiene estructura jerárquica, y que está distribuido en los modelos de las diferentes capas. Hay un segundo flujo de comandos, que es descendente. Los comandos elaborados por el TD de un nivel, sirven de objetivos para la capa inferior, que los desarrollará en nuevos comandos más concretos que propagará al nivel

inmediatamente debajo suyo.

2.2.5. Ideas principales

Una vez revisados varios ejemplos representativos que hemos adscrito dentro de este enfoque vamos a detallar las características comunes principales. Todas ellas conforman los fundamentos identificativos de este paradigma que hemos denominado *deliberativo*.

Planificación

En este contexto entendemos por planificación el proceso de manipulación simbólica que produce una secuencia de acciones, llamado *plan*, para llevar a cabo un propósito. Esta interpretación cuadra con la asunción subyacente de que el comportamiento es una secuencia de acciones, que es quizá la idea central del paradigma deliberativo. El cálculo del plan es una deliberación finalista, orientada a alcanzar el objetivo. Este encadenamiento de actuaciones con una finalidad es una característica del comportamiento racional, introducida por la planificación en el comportamiento de los robots deliberativos.

El plan se calcula antes de empezar cualquier actuación, es decir, se anticipa en el tiempo a la acción real, a su ejecución. Esta anticipación en el tiempo requiere el conocimiento previo de las consecuencias de las acciones posibles, es decir, que el resultado de las actuaciones sea predecible y se conozca de antemano. Lateralmente esto implica que también necesita cierta estabilidad o predecibilidad en el entorno. Si éste es muy dinámico entonces no tiene mucho sentido planificar con demasiada antelación. Los planes tendrán una vida útil corta, pues los datos sobre los que se fundamentan pueden dejar de ser válidos muy pronto, debido al alto dinamismo del entorno.

La noción de *estado* como situación específica aparece íntimamente ligada con la idea de planificación. El robot en cada momento se encuentra en cierto estado, y para cambiar de estado el robot puede ejecutar unos operadores o actuaciones. El efecto de los actuadores sobre cada estado se conoce de antemano. En este contexto la planificación consiste en calcular la secuencia de operadores que llevan desde el estado actual hasta cierto estado objetivo. Se puede entender también como un proceso de búsqueda en el espacio de secuencias de operadores, tal y como hace SOAR.

El campo de la planificación es muy amplio, y aquí sólo hemos hecho un repaso somero, con finalidad ilustrativa, a algunas técnicas utilizadas como motor del comportamiento inteligente, hoy día anticuadas. Hemos omitido comentarios más amplios sobre otras técnicas de Inteligencia Artificial como los sistemas de producción, la resolución de problemas (*General Problem Solver*), análisis de medios-fines, técnicas de búsqueda como el descenso en dirección del gradiente (*hill climbing*), programación dinámica, planificadores con restricciones, etc. También hemos dejado fuera ejemplos como el planificador Prodigy [Haigh y Veloso, 1996] que hace énfasis en el aprendizaje, o los agentes BDI (*Beliefs, Desires, Intentions*) [Rao y Georgeff, 1995] [Fischer *et al.*, 1994] que razonan e interactúan de modo racional aplicando la lógica a sus modelos mentales de creencias, deseos e intenciones de los agentes.

Gran parte de los planificadores operan sobre representaciones en lógica proposicional, y son capaces de encadenar antecedentes y consecuentes de sus sistemas de producción. Otro grupo interesante de planificadores utilizados en robótica móvil son los planificadores espaciales, específicos para tareas de navegación [Giralt *et al.*, 1983] [Nilsson, 1969]. En este sentido sólo hemos mencionado los utilizados en Shakey y Hilare, sin profundizar en otros posteriores como los diagramas de Voronoi [Takahashi y Schilling, 1989] o el espacio de configuracion de Lozano-Pérez [Lozano-Pérez, 1983].

Modelo simbólico del mundo

La planificación, entendida como manipulación simbólica, requiere una representación explícita del estado del mundo (*modelo descriptivo*) para tomar decisiones en base a ella. También requiere almacenar y usar el conocimiento acerca de las consecuencias de las actuaciones posibles (*modelo operativo*). Estas representaciones conforman el modelo del mundo, que puede tener diversos formatos: declarativo, procedimental, en forma de reglas, marcos, geométrico, topológico, etc.

Desde el punto de vista de percepción este paradigma necesita representar simbólicamente el entorno del robot, construir un modelo del mundo, para poder deliberar sobre él. Este modelo debe contener todos los aspectos relevantes del entorno para el comportamiento. Los formatos en los que ese modelo aparece son múltiples. Como el comportamiento más relevante de los robots móviles es la navegación, los modelos empleados por muchos de los primeros robots son mapas de ocupación, que incluyen los obstáculos de entorno, ya sea con forma de rejilla, con segmentos, grafos de nodos, etc. tal y como hemos visto en Shakey y Hilare. De modo complementario el estado del mundo también se representa en un conjunto de proposiciones lógicas, como en SOAR y STRIPS [Fikes y Nilsson, 1971].

Además de la faceta descriptiva sobre el estado actual del entorno del robot, el modelo del mundo también suele almacenar conocimiento sobre el efecto que las acciones del robot pueden tener en el mundo. Esta información suele utilizarse en la deliberación para planificar las acciones correctas. Esta representación del conocimiento procede de los escenarios típicos de Inteligencia Artificial. Muchas alternativas han surgido en ese contexto, que luego se han aplicado al caso particular de la robótica, como los sistemas de producción, los marcos y la lógica de proposicional. Cabe resaltar que el uso de la lógica de predicados como modelo operativo del mundo no es exclusivo de las arquitecturas deliberativas. Por ejemplo, como veremos posteriormente Pattie Maes [Maes, 1989b] utiliza este razonamiento simbólico, aunque no llega a hacerse explícito ningún plan. Sólo cuando el encadenamiento de pasos lógicos conlleva la secuencia de actuaciones intermedias, calculadas de antemano, hasta llegar a un resultado final entonces si se está haciendo un uso de la lógica en el sentido deliberativo clásico.

En los trabajos más antiguos, tanto el modelo descriptivo como el operativo eran construidos por el diseñador y se ponían a disposición del robot para que pudiera utilizar ese conocimiento del entorno, que esencialmente era estático. Las posteriores aproximaciones incidieron en la necesidad de que el propio robot fuera capaz de construir esos modelos, al menos en su parte descriptiva, para aumentar el grado de autonomía. Así se les dotó con la capacidad de generar ese modelo y actualizarlo desde la información sensorial.

En cuanto a su naturaleza, el modelo del mundo empleado en este paradigma es típicamente objetivo, en él hay símbolos u objetos con propiedades que le son intrínsecas, como la posición y el tamaño. La representación establece una biyección entre los objetos del mundo y sus homólogos en el modelo. Las características modeladas son propias de cada objeto, independientes del observador y sus intereses. Esto contrasta, como luego veremos con la percepción deíctica o funcional que propugnan otros enfoques.

Además la percepción es usualmente monolítica, en el sentido de que toda la información se funde en un modelo *único* del mundo, centralizado, que está disponible para todas las posibles deliberaciones que se hagan sobre él. Incluso en sistemas aparentemente distribuídos como RCS, en capas, aparece la figura de una única memoria global, a la que pueden acceder todos los niveles.

Sensar-modelar-planificar-actuar

En cuanto a su inserción en robots reales, el factor común de los enfoques deliberativos es que ordenan en secuencia las tres primitivas señaladas por R.Murphy [Murphy, 2000]: sensar, planificar y actuar. De hecho plantean el comportamiento como una sucesión de acciones, y su generación como la ejecución iterativa, en bucle infinito, de la secuencia *Sensar-Modelar-Planificar-Actuar* (SMPA). Tal y como ilustra la figura 2.7, en la primera etapa se sensa del entorno y en la segunda la información proveniente de los sensores se funde en un único modelo del mundo, actualizándolo. Este modelo es completo y contiene la percepción del sistema, que se articula en estas dos primeras fases. Sobre ese modelo se planifica la mejor acción teniendo en cuenta todos los aspectos y consecuencias. Finalmente, en la parte genuina de actuación, el plan se ejecuta comandando los actuadores reales.

Atendiendo a este bucle fundamental de los sistemas deliberativos, las arquitecturas deliberativas presentan una descomposición funcional homóloga. Por lo tanto se dividen en módulos de percepcion, módulos de planificación y de ejecución (monitorización). Hay un único flujo de control en el cual se invocan sucesivamente las funciones especializadas en recoger los datos sensoriales, fundirlos en el modelo, elaborar el plan y ejecutarlo.

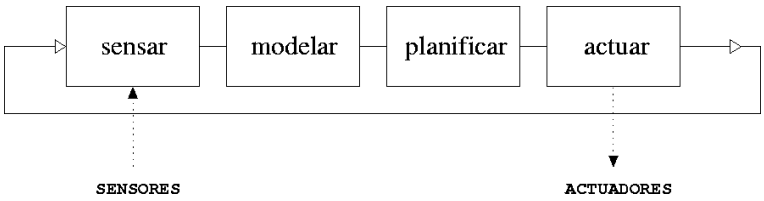


Figura 2.7: *Bucle infinito de control ejecutado en secuencia*

Sistemas jerárquicos

La apuesta de los sistemas deliberativos para aumentar complejidad en el comportamiento ha sido la jerarquía. Con ella se persigue descomponer esos comportamientos complejos en problemas más sencillos, siguiendo una estrategia de divide y vencerás. Esta jerarquía introduce varios niveles de abstracción en la resolución de problemas y en la generación del comportamiento.

En primer lugar hay jerarquía en la planificación, específicamente en la descomposición de tareas en subtareas más pequeñas y abordables. El hecho de construir un plan puede llevar implícito la descomposición de una tarea en sub-tareas que conducen a ella. Esta descomposición puede considerarse como una esencia del enfoque deliberativo [Simmons, 1994]. Implícitamente se asume que la solución a un problema complejo consiste en la subdivisión de éste en tareas más sencillas y fáciles de abordar. Las subtareas se convierten a su vez en subobjetivos que requieren una nueva planificación. Típicamente su retorno en tiempo de ejecución al módulo encargado de la ejecución del plan indica si la ejecución de la subtarea ha ido bien o ha ido mal.

En segundo lugar hay jerarquía en el modelo del mundo, en concreto en lo que Meystel denomina representación multiresolucional. Los niveles más bajos de la jerarquía procesan la información más local y más volátil. Los niveles superiores tienen un alcance espacial mayor y un horizonte temporal mayor.

También es frecuente dentro de este enfoque la división en módulos especialistas: motriz, visión, manipulador, etc. Por ejemplo, en Hilare [Giralt *et al.*, 1983] hay un planificador general en el nivel superior, que encarga tareas a los distintos módulos especializados de decisión: navegación y control de movimiento, comunicación y lenguaje natural, manipulador, modelado de objetos y comprensión de la escena. Otro ejemplo es el subnivel de asignación de trabajos en cada módulo de RCS [Albus, 1993], que realiza una descomposición *espacial* de tareas, en este mismo sentido. Si bien esta división tiene su lógica, también puede dificultar la interrelación entre unas partes y otras. Cada módulo suele tener su interfaz, generalmente en forma de función, para que otros puedan utilizar sus habilidades. Por ejemplo, el módulo de visión se considera separadamente del módulo motriz o de navegación, lo cual complica fenómenos que requieren una más estrecha colaboración entre ambas capacidades. Un caso particular es la percepción visual activa, la cual conlleva no sólo el análisis de las imágenes sino coordinarlas con movimientos específicos.

2.2.6. Limitaciones de los sistemas deliberativos

Como critican Agre y Chapman [Agre y Chapman, 1987], la planificación presenta explosión combinatoria en cuanto el problema a manejar crece en complejidad. Esto dificulta la aplicación de estas arquitecturas a comportamientos complicados, con muchas variantes, porque las posibilidades a contemplar se disparan y se hacen intratables. Otra limitación en este sentido es el problema del marco: cuando se tienen bases extensas de conocimiento es difícil determinar con agilidad qué partes son útiles para la situación y objetivos actuales. Por ello no escalan bien a situaciones reales donde hay miles de proposiciones a representar.

Adicionalmente la hipótesis de mundo cerrado, que es una asunción típica en los primeros planificadores en robots móviles, choca frontalmente con el escenario habitual de un mundo abierto en el que el propio robot se desenvuelve, al igual que otros agentes, a su libre albedrío. La idea de una representación en lógica proposicional actualizada exclusivamente por las acciones del propio robot

es inviable cuando otros agentes pueden provocar cambios impredecibles en el entorno, los cuales el robot debe necesariamente tener en cuenta.

En general el paradigma deliberativo no maneja bien la incertidumbre, necesita modelos completos. Esto hace difícil su acoplamiento con sensores reales que típicamente pueden dar medidas erróneas o inciertas sobre la realidad. Del mismo modo, no soporta la incertidumbre en el comportamiento de todos los participantes en la realidad, requiere tenerlo modelado. Por ello estas arquitecturas no funcionan bien en entornos muy dinámicos, donde hacer planes completos desde la situación actual hasta el estado objetivo deja de tener sentido.

Los sistemas deliberativos son inherentemente monolíticos y centralizados. Respecto del control, el cálculo de la mejor acción se centraliza en el planificador, que tiene en cuenta toda la información necesaria. Esa información necesaria debe estar contenida en el modelo del mundo, que es el lugar central donde la parte perceptiva integra todos los datos sensoriales. Esta centralización supone un cuello de botella en tiempo de ejecución. El flujo de información ha de pasar secuencialmente por todos los pasos. Esto introduce demoras entre la sensación de cierto evento y la actuación de respuesta. De esta manera todo el tiempo que lleva modelar completamente cierto objeto percibido, o el tiempo que lleva planificar, se propagan en retardos a la respuesta global, con lo que se pierde reactividad. Esto es especialmente grave en el caso de la planificación, que suele demandar mucho cálculo y el tiempo de deliberación suele no estar acotado.

Como hemos visto, las arquitecturas deliberativas se descomponen en diferentes funciones que se ejecutan en secuencia. Esta descomposición funcional dificulta la depuración en el ciclo de desarrollo: si una parte no está hecha, no hay comportamiento observable global. No se puede actuar sin un plan y no se puede planificar hasta que se tiene un modelo más o menos preciso del entorno. Hasta que no se tienen desarrollados todos los módulos del bucle SMPA el sistema completo no ofrece *ninguna* funcionalidad.

En resumen, estas arquitecturas requieren un entorno estable y conocimiento completo de él para tomar decisiones, pero han sido incapaces de generar comportamientos suficientemente ágiles en entornos dinámicos y con incertidumbre. De hecho, las desventajas señaladas se reflejan en la escasez de resultados prácticos en robots reales, sobre todo en escenarios dinámicos, donde el entorno es menos predecible. Pese a su buen funcionamiento en problemas acotados no degradan bien con la incertidumbre, que por otra parte es una constante en los robots y aplicaciones reales.

2.3. Sistemas reactivos: asociaciones situación-acción

A finales de los años 80, era notable la falta de flexibilidad de los robots reales conseguidos hasta la fecha, guiados por el enfoque deliberativo. Quizá motivados por esto, varios investigadores comenzaron a replantearse el modo de generar comportamiento, y el uso de los planes, que era la pieza central del paradigma deliberativo [Payton, 1990] [Agre y Chapman, 1990] [Firby, 1987]. Por ejemplo, Firby [Firby, 1987] refleja la necesidad de monitorizar la ejecución de los planes continuamente, debido a que hay eventos en el entorno del robot que no se pueden anticipar. Bien sean fallos de actuación o bien debidos simplemente al dinamismo del entorno, el robot debe reaccionar ante esos eventos.

Fruto de este replanteamiento nace el *enfoque reactivo*. Esta escuela reactiva, quizá liderada por los trabajos de Rodney Brooks [Brooks, 1986] [Brooks, 1991b] provocó un giro drástico en cuanto al modo de generar comportamiento. El nuevo enfoque hace hincapié en una ligazón más directa entre los sensores y los actuadores, sin pasar por las etapas intermedias que utilizan los robots deliberativos. De esta manera la reacción a los eventos era mucho más rápida. Uno de los pasos intermedios que se consideran innecesarios es el modelo del mundo, y otro el manejo de símbolos. En este sentido el enfoque reactivo es subsimbólico, y argumenta que no es necesaria la representación simbólica, ni el razonamiento sobre símbolos para generar comportamiento. En términos de Murphy [Murphy, 1998], este planteamiento reduce las primitivas esenciales a sensar y actuar, que están directamente ligadas, y deja a un lado la de planificar.

Si epistemológicamente el paradigma deliberativo entronca con el cognitivismo clásico y la introspección, entonces el enfoque reactivo entronca con el conductismo, y de modo más específico con el conexionismo. De hecho su hipótesis fundamental es que el comportamiento se puede generar

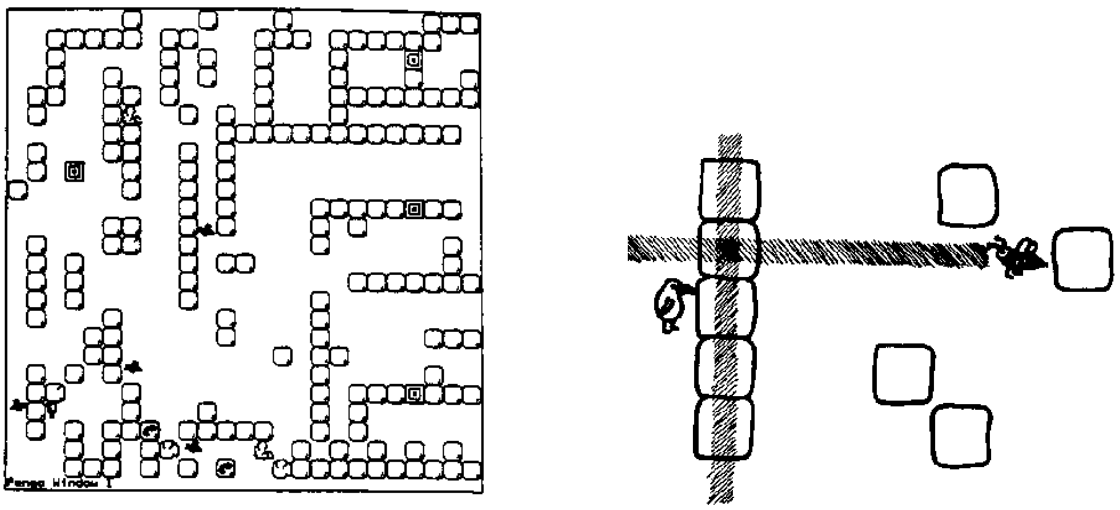
como una amalgama de reflejos, que conectan los datos sensoriales con los valores a los actuadores.

De todos los aportes de la escuela reactiva hemos separado la parte relacionada con los niveles de competencia y la subsunción, que se describe en la sección 2.4 dedicada a las arquitecturas basadas en comportamientos. En esta sección nos centramos la parte genuinamente reactiva, que gira alrededor de la rápida conexión entre sensores y actuadores. Esta separación viene argumentada también en [Mataric, 1992a], y se fundamenta en que aunque nacieron íntimamente ligadas, conceptualmente son ideas diferentes. Por ejemplo, hay otros trabajos que caen dentro de las asociaciones situación-acción sin provenir históricamente de los trabajos de Brooks, como los autómatas finitos. Además considerando así la componente reactiva, se puede identificar su presencia como unidad tanto en las arquitecturas basadas en comportamientos, como en gran parte de las arquitecturas híbridas.

La teoría de control clásica se puede ver como un sistema reactivo, donde el propio controlador materializa la asociación situación-acción. Esta analogía es válida principalmente si el controlador no incorpora memoria, si en él no influyen estados anteriores. Como veremos, recíprocamente se puede decir que la tabla de asociaciones implementa una superficie de control. A pesar de estas semejanzas hemos mantenido separadas las secciones 2.1 y 2.3 porque históricamente proceden de disciplinas diferentes y los sistemas reactivos no tienen detrás el potente aparato matemático, con herramientas de análisis y diseño sistemático, que tiene el control clásico.

2.3.1. Pengi juega a Pengo

Philip Agre y David Chapman plantean en [Agre y Chapman, 1987] y [Agre y Chapman, 1990] su *teoría de la improvisación continua* para generar comportamiento. Como demostración escribieron un programa que era capaz de jugar con éxito a un videojuego, Pengo, en el que un pingüino deambula por un mundo bidimensional de bloques donde hay abejas que debe evitar, según se aprecia en la figura 2.8. Los bloques se pueden desplazar empujándolos y con ellos se puede aplastar abejas y construir una figura de bloques, que es el objetivo del juego. Este entorno es dinámico, demanda rápidas actuaciones y es suficientemente complejo como para no poder abordarlo con técnicas deliberativas clásicas.



(a) (b)

Figura 2.8: *Pantalla de Pengo [Agre y Chapman, 1987] (a) y rutina para encontrar el bloque a empujar (b).*

El pingüino se puede considerar como un robot móvil en un entorno visual simulado. La arquitectura de control que gobierna su comportamiento está diseñada como unas sencillas reglas que enlazan ciertas situaciones con la acción adecuada. Las reglas se disparan dependiendo de la situación concreta en la que está el pingüino. De este modo el comportamiento surge de la interacción de esas reglas con el mundo, en lugar de emanar de una implementación procedimental basada en secuencia de acciones. Al no tener un plan preconcebido el comportamiento es mucho más flexible, puede adaptarse con facilidad a oportunidades y contingencias en el entorno. Si la situación cambia,

simplemente se disparan otras reglas. En caso de que se activen varias reglas conflictivas, un arbitraje explícito sensible a la situación determina cual de ellas prevalece.

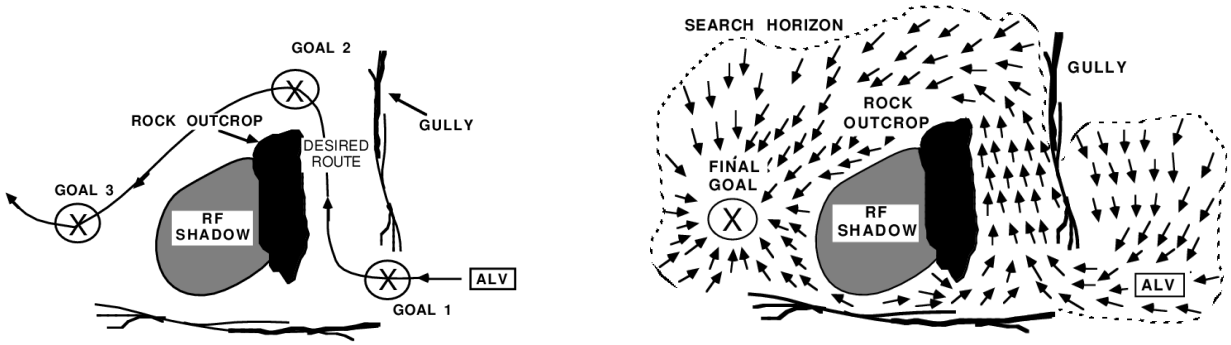
La descripción de la situación se construye de manera instantánea por unas rutinas perceptivas visuales que manejan entidades indexico-funcionales. Estas entidades reflejan hechos relevantes para el pingüino, por ejemplo `la-abeja-al-otro-lado-del-bloque-proximo`, `el-bloque-que-estoy--empujando`, `la-abeja-que-me-persigue`, etc. Son *indexadas* porque dependen de la situación actual, y sólo reflejan los alrededores del robot. En cada situación interesan unos datos y otros no, la situación indexa qué datos son relevantes de todos los disponibles. De este modo se materializa una suerte de atención, que ignora la mayor parte de los datos sobre la realidad. Además son *funcionales* porque tienen relación directa con los intereses del robot, más que con los objetos intrínsecamente. Es pues una percepción más subjetiva que objetiva. Antes que percibir la `abeja-23`, se busca caracterizar `la--abeja-que-me-persigue`, ya sea para huir de ella o para empujar un bloque con idea de aplastarla.

El buen resultado práctico de esta arquitectura pone en entredicho la necesidad de planes para generar comportamiento. De hecho el concepto de plan es cuestionado por la escuela reactiva, y sufrirá diferentes reformulaciones que tratan de flexibilizar su uso manteniéndolos como garantía de finalidad en las acciones.

2.3.2. Planes internalizados

En esa misma línea, David Payton cuestiona en [Payton, 1990] la naturaleza y el uso que se hace de los planes en el paradigma deliberativo. Mientras que el uso clásico prescribe el plan como programas de acción que deben ser ejecutados en secuencia, él propone el uso de planes como *recursos para la actuación*. Argumenta que al abreviar el plan a un curso explícito de acción, se pierde información que puede ser de utilidad.

El ejemplo prototípico es el comportamiento `ir-a-punto`. El plan clásico (figura 2.9(a)) consiste en una lista de puntos intermedios. Si por cuestiones coyunturales el robot sobrepasa cierta zona en su avance hacia el destino, alejándose del camino especificado, entonces puede interesar saltarse ese punto intermedio y avanzar directamente al destino final. Sin embargo el plan clásico no refleja este hecho, fuerza a alcanzar el punto intermedio, aunque ya no sea interesante. Todas las posibles rutas que no sean la óptima y no estén explícitamente representadas se pierden. Además si el alejamiento de la ruta dictada es significativo se fuerza una nueva replanificación, que suele ser costosa en tiempo.



(a) (b)

Figura 2.9: *Plan clásico (a) y plan internalizado (b): en cada posición almacena la actuación adecuada.*

En el plan internalizado (figura 2.9(b)), el vehículo identifica su posición y eso sirve como entrada a una tabla que tiene asociado una acción correspondiente a cada ubicación [Payton *et al.*, 1990]. El plan internalizado tiene aquí la forma de campo vectorial, que apunta en cada posición la dirección que debe tomarse si se quiere llegar al punto destino. El uso de este plan es absolutamente reactivo, muy rápido. Basta saber la posición, que el plan internalizado le tiene asociada una actuación adecuada. En este plan internalizado no hay compromiso con los objetivos parciales, con los puntos intermedios. Si por cuestiones coyunturales el robot se aleja de cierta trayectoria óptima, entonces allá donde esté el plan

le ofrece una sugerencia de acción. Además este tipo de planes minimiza la necesidad de replanificar, por mucho que se separe de la trayectoria óptima no hay que replanificar de nuevo.

El plan internalizado establece una tabla de situación-acción que se rellena en tiempo de ejecución en función del objetivo que se persiga. Por ejemplo, para alcanzar otro destino diferente desde el mismo punto, la tabla será distinta. Se utiliza un planificador para rellenar ese plan internalizado, pero no en el sentido clásico. Este planificador es un algoritmo de propagación en una rejilla de costes. Cada posición tiene asociado el coste de atravesarla. Recorriendo esa rejilla se construye otra que almacena el mínimo coste para viajar desde esa posición al destino. Es como si la propagación avanzara más rápida cuando atraviesa celdas de bajo coste, y más lenta en zonas menos deseadas. La planificación termina cuando destino y punto inicial están enlazados a través de esta rejilla. El camino óptimo en esta nueva rejilla viene marcado como aquel que maximiza el gradiente, es decir, el que en cada celdilla apunta a la vecina con el menor coste acumulado.

Este algoritmo permite incorporar la presencia de obstáculos imprevistos con una pequeña replanificación local. Esto supone una ventaja frente a los planes clásicos, que replanifican de modo completo, requiriendo mayor tiempo de cómputo. Además no ofrece mínimos locales, como principal ventaja frente a métodos basados en superposición de campos.

Este uso de los planes interiorizados es similar al de los planes como comunicación [Agre y Chapman, 1990] [Payton, 1990] en el sentido que ofrecen sugerencias de actuación, consejos para la acción. Sin embargo los planes como comunicación suelen ser más ambiguos y requieren un mayor esfuerzo de interpretación a la hora de usarlos.

2.3.3. Tabla rasa

Un modo eficiente de representar asociaciones situación-acción es una tabla. Para generar comportamiento con esta aproximación la tabla determina la salida adecuada correspondiente a cada una de las situaciones posibles. Cada situación plausible aparece como una entrada en la tabla y se caracteriza por las entradas sensoriales en ese momento junto con el valor de ciertas variables internas. Para cada una de ellas la tabla tiene como salida tanto las actuaciones adecuadas como el cambio en esas variables internas si procede. Esta idea sugiere el siguiente funcionamiento en bucle infinito: periódicamente se chequean los sensores y las variables internas, se empareja la situación actual con la más parecida en la tabla y se ejecutan las acciones correspondientes que dicta la tabla, incluyendo tanto actuaciones como modificación correspondiente del estado interno. La ejecución se centra en leer todas las variables de entrada (percepción) y seguir las indicaciones de la tabla (actuación).

Aunque no supone una arquitectura reconocida per se, defendida en varias publicaciones, sí supone un primera aproximación, inocente, a la creación de comportamiento. Nos parece interesante porque quizá expresa el paradigma reactivo llevado al límite. No hay que calcular nada en tiempo de ejecución, basta leer la recomendación de la tabla. Esto le da mucha velocidad al comportamiento del sistema que puede responder rápidamente a los cambios en el entorno.

Desde el punto de vista del control clásico, construir la tabla supone definir una *superficie de control* punto por punto, dando los valores para todos los puntos posibles del espacio de entrada. Tiene toda la flexibilidad puesto que puede implementar cualquier superficie de control, pero su construcción y su almacenamiento son difíciles de implementar. De hecho, un problema abierto es la construcción de esa tabla. El diseño del sistema consistirá en identificar las variables significativas para la acción, anticipar *todas* las posibles situaciones imaginables del robot y codificar en la tabla la actuación conveniente para cada una de ellas. Se puede utilizar conocimiento ad hoc, heurístico, ensayo y error, técnicas de aprendizaje, sistemas de reglas, etc. Cualquiera que haya sido su generación la tabla no es más que una técnica para expresar asociaciones situación-acción, con independencia de cómo éstas se hayan obtenido.

La extensión exhaustiva de todos los posibles casos hace inviable esta aproximación para comportamientos de cierta complejidad. A veces es difícil cubrir *todos* los casos imaginables, todos los que se pueden presentar y el sistema no tiene capacidad de improvisación: si aparece una situación no registrada no sabe cómo responder. Todo debe estar en la tabla, que debe ser exhaustiva. Para comportarse bien necesita un rediseño, incoporar esa nueva situación y la respuesta adecuada a la tabla.

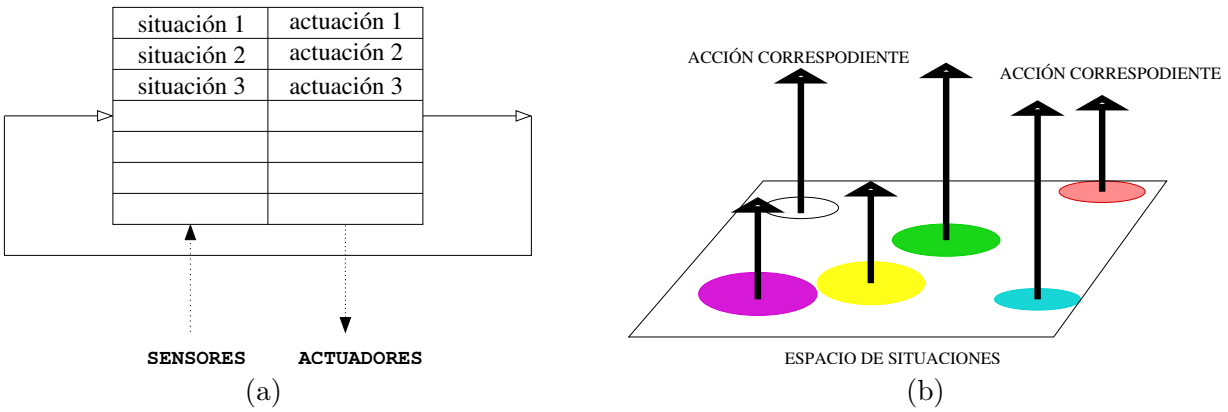


Figura 2.10: *Tabla de situación-acción: para cada punto del espacio de situaciones hay una actuación recomendada.*

Además la naturaleza continua de los sensores y actuadores es otro inconveniente. Para simplificar la confección de la tabla se puede ampliar el lenguaje con el que se especifican situaciones, utilizando intervalos, rangos o símbolos comodín (cualquier valor). Otra técnica aplicable consiste en introducir variables intermedias que resumen la información, reducen el número de variables significativas de la situación y permiten ignorar al resto a la hora de especificar situaciones. Sin embargo estas técnicas no resuelven la limitación inherente, sólo la palían. Sea como fuere la tabla debe contemplar *todos* los casos posibles, y su construcción es difícil.

Dentro del planteamiento que hicimos en el capítulo 1, esta arquitectura supone una estructura plana que resuelve la ambigüedad de qué hacer en cada momento, es decir, la selección de acción. Es una inmensa tabla homogénea, sin niveles diferenciados, con un único nivel de abstracción. Si tenemos varios actuadores la salida de la tabla tiene forma de vector. Desde la óptica de percepción asume que todos los estímulos están disponibles para ser leídos en todo momento, ya sean señales de los sensores o elaboraciones construídas desde ellas. El entorno decide qué valores toman ciertas variables. Por ello el entorno influye por lo tanto explícitamente en el recorrido por el espacio de estados. Los objetivos también influyen, están implícitos en las actuaciones elegidas para tal o cual situación.

Es un sistema centralizado en la tabla, y tiene una velocidad característica, determinada por la frecuencia con la que aquella se consulta. Sólo hay una escala de tiempo explícita, sólo hay un bucle. Pueden darse comportamientos a largo plazo o incluso cíclicos, pero eso no esta explicitamente en la tabla.

Es un enfoque de caja negra, porque permite imitar el comportamiento de un sistema sin necesidad de indagar en su estructura interna. Para construirla basta conocer un repertorio suficientemente amplio de pares entrada-salida del comportamiento del sistema. Es importante resaltar que aunque exteriormente se pueda construir una tabla que imite así el comportamiento, por ejemplo, de un animal, esto no significa que el mecanismo que realmente opera en el interior del animal para desencadenar esos comportamientos sea exactamente éste.

Los sistemas de reglas más sencillos son otra forma similar de expresar esas tablas. La diferencia principal con los sistemas de producción empleados en el paradigma deliberativo es que aquí no se permite el encadenamiento de reglas. Cada una de las entradas de la tabla se puede ver como una regla explícita de control, donde la descripción de la situación conforma el antecedente y la acción recomendada el consecuente. La descripción de la situación define la región de aplicación del espacio de variables de entrada donde el consecuente es válido (figura 2.10). En este marco aparecen dos problemas típicos que el diseño debe evitar: el *solape de control*, para las situaciones en las que varias reglas son aplicables y el *vacío de control*, para situaciones que no casan con ninguna de las entradas. El solape de varias reglas se puede resolver mezclando de algún modo sus consecuentes, de manera que la salida recomendada es una combinación de las acciones dada por cada una de ellas.

2.3.4. Autómata finito de estados

Un enfoque más amplio que la sencilla tabla de situación-acción es el de los *autómatas de estados finitos*. En esta aproximación se crean un conjunto de estados virtuales distintos. En principio en cada estado se podría ejecutar cualquier acción de las posibles, cada una de las cuales hace transitar al sistema a otro estado, conformando la matriz de transición del sistema. Esta matriz contiene toda la dinámica del sistema, y el autómata concreto elige una actuación determinada en cada estado.

La principal diferencia con respecto a las tablas de situación-acción es la introducción del *estado*. Este concepto permite enriquecer las respuestas del sistema, que ahora pueden dejar de ser reflejos automáticos a la situación, y pasan a incluir factores internos que se traducen en estados diferentes, y por ello provocan reacciones diferentes ante la misma situación externa. En la tabla de la sección 2.3.3 no hay el concepto de transición entre estados, ni determinista ni probabilística. Desde el punto de vista arquitectónico el estado marca un contexto, tanto de actuación como de percepción. Además de contener una actuación pertinente, cada estado tiene asociadas ciertas observaciones, ciertos eventos que ese contexto conviene detectar.

Otra extensión fundamental de los autómatas con respecto a las tablas de situación-acción es la posibilidad de reprogramación del sistema: sobre un determinado sistema, cuya dinámica se ha capturado en una matriz de transición, se puede rehacer el autómata dependiendo del objetivo. Es decir, se puede cambiar la actuación asociada a cada estado, provocando comportamientos diferentes. Para el cálculo de la acción adecuada en cada estado se pueden utilizar diferentes técnicas de planificación como la programación dinámica (iteración de políticas, iteración de estados), los procesos de decisión POMDP (*Partially Observable Makov Decision Process* sobre modelos de Markov), etc.. En general estos algoritmos suelen ser costosos computacionalmente, de modo que para sistemas de tamaño pequeño puede hacerse manualmente.

Un trabajo relevante en esta línea son los Sistemas de Eventos Discretos (DES). Este formalismo desarrollado por Ramadge y Wonhan [Ramadge y Wonham, 1989] nació en el contexto de control supervisor de sistemas discretos y permite explicitar en ese ámbito cuestiones como si el sistema es controlable o no. La transición entre estados se provoca por *eventos*, que son cambios cualitativos en el entorno o en la evolución de la tarea. Estos eventos se clasifican en controlables e incontrolables. Los primeros se corresponden con las observaciones sensoriales, que no se pueden evitar, y los segundos con los comandos de actuación, que sí se pueden elegir. El objetivo se especifica como cierto estado destino al que se quiere llegar desde el estado actual, un estado más dentro de todos los posibles. Se dice que un sistema es controlable si existe un conjunto de eventos controlables cuya aplicación garantiza que el sistema llega al estado objetivo. Como el entorno también introduce eventos no controlables, influye por lo tanto de modo explícito en el recorrido por el espacio de estados.

DES fue aplicado por Kosecka [Kosecká y Bajcsy, 1994; Kosecká y Bogoni, 1994] para describir y generar los comportamientos de navegación guiada por visión y de manipulación de su agente móvil. En ellos el comportamiento se caracteriza como un DES en el cual el *estado* se corresponde con cierta situación en la evolución de una tarea. Por ejemplo en la figura 2.11(a) se describe el autómata con cuatro estados que genera el comportamiento `seguir-trayectoria-sorteando--posibles-obstáculos` como un autómata con cuatro estados. Eventos relevantes en esta aplicación son `obstáculo-detectado` (*obstacle detected*), `camino-libre` (*clear path*), `ruta-completada` (*path completed*), etc. Es importante señalar que el estado no sólo marca la acción a ejecutar, sino también las estrategias perceptivas útiles, los eventos a detectar. En este sentido facilita cierto mecanismo de atención.

Otro trabajo importante en esta línea son los *autómatas situados* de Rosenschein y Kaelbling [Rosenschein y Kaelbling, 1995]. En esta teoría se hace hincapié en la interacción continua del robot con el entorno, que se rebautiza como *transducción*. El robot materializa esa interacción como una correspondencia permanente entre sus entradas (sensores) y sus salidas (actuadores), en la cual aparece el estado interno como intermediario necesario. Normalmente interesa que el sistema tome la acción B cuando el entorno se encuentra en la situación A. Como el comportamiento sólo puede depender del estado interno del robot, una primera aproximación de ese estado A puede ser el estado interno $\widetilde{A}$. El robot tomará la acción B cuando el robot se encuentre en el estado $\widetilde{A}$. Para que esta simplificación funcione correctamente A y $\widetilde{A}$ deben estar altamente correlacionados. El robot debe interiorizar la

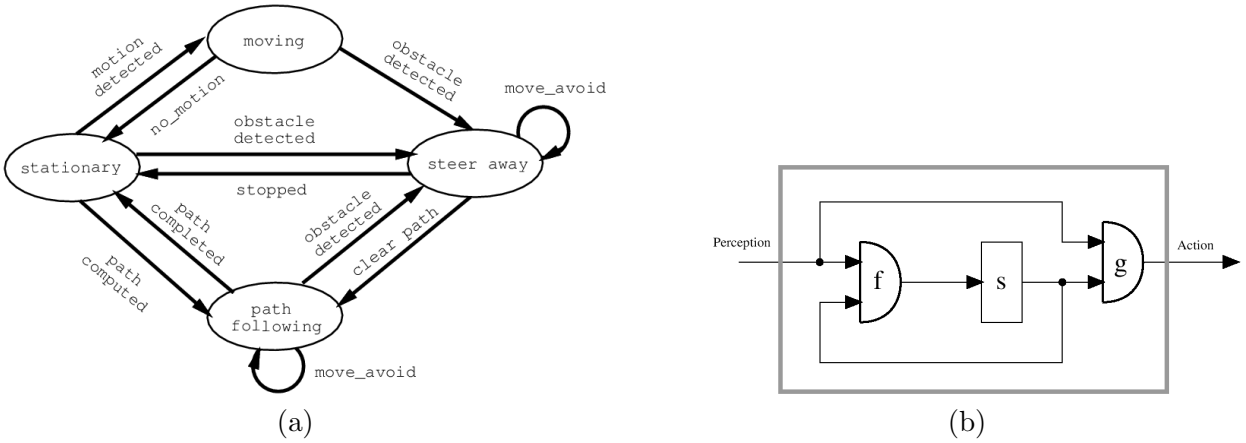


Figura 2.11: *Sorteo de obstáculos como DES (a) y bloque unitario del autómata situado de Rosenschein y Kaelbling (b)*

situación del entorno, de ahí el adjetivo *situado*.

En la figura 2.11(b) se esquematiza el modelo genérico de la máquina de estado finito que proponen. La función de entrada f se utiliza para actualizar el estado interno, que se denomina s y toma la forma universal de un *vector de estado*. Esa actualización puede depender del estado anterior así como de las percepciones presentes. La actuación global del sistema se calcula como el resultado de una función de salida g, que tiene en cuenta tanto las percepciones como el estado actuales. Este modelo engloba las tablas de situación-acción cuando se anula el estado interno, en terminología de Rosenschein, una *actuación pura*. También engloba, llamándolos *percepción pura*, a los autómatas que exclusivamente elaboran percepción, como aquellos sin función de salida que utilizan la función de entrada para actualizar cierto estado interno.

Esta propuesta no reniega de la construcción y el manejo de símbolos [Rosenschein y Kaelbling, 1995], pero busca un funcionamiento reactivo, muy distinto de la deliberativa clásica. De hecho propone un algoritmo simbólico (Gapps) para programar el autómata de modo que consiga cierto objetivo. Y propone otro algoritmo simbólico (Ruler) para reflejar los estados relevantes y su construcción desde una representación en forma de asertos del entorno. Aunque el diseño y síntesis de los autómatas conlleva tiempo y manejo de símbolos, en tiempo de ejecución son muy rápidos, dotando de agilidad al comportamiento del sistema.

Frente a la percepción de los sistemas deliberativos, la teoría de los autómatas situados considera que la información sensorial tiene una cierta validez temporal después de la cual caduca irremediablemente. Además, al favorecer cierto procesamiento de la información, en forma de autómatas con estado, es más versátil que una simple tabla sensación-actuación. Por ejemplo, cuando los datos sensoriales no son suficientemente informativos, las entradas para la acción se pueden derivar acumulando información parcial en el tiempo en algún estado interno. Este trabajo ahonda en los fundamentos teóricos de los sistemas reactivos modelados como autómatas, e identifica algunas de sus dificultades como el manejo de incertidumbre real y en la interacción entre percepción y actuación (por ejemplo cuando un objetivo en sí es conseguir información).

Cuando el autómata se extiende para reflejar actuaciones inciertas, la matriz de transición se define como un conjunto de probabilidades de transición. Esto es lo que se conoce como *modelo de Markov*. Si además el estado no es directamente observable entonces estamos ante un *modelo de Markov parcialmente observable*. En ellos las observaciones no indican taxativamente el estado del sistema, sino que se encuentran relacionadas con él a través de una dependencia probabilística. Es decir, en cada estado ciertas observaciones son más probables que otras, y viceversa, ante cierta observación unos estados son más verosímiles que otros. En estos casos el estado no se conoce con exactitud, sino con una cierta probabilidad. Ello no impide recomendar una actuación. Por ejemplo, eligiendo la actuación que acumula más probabilidad, aunque en él contribuyan varios posibles estados entre los cuales no se puede discernir el exacto.

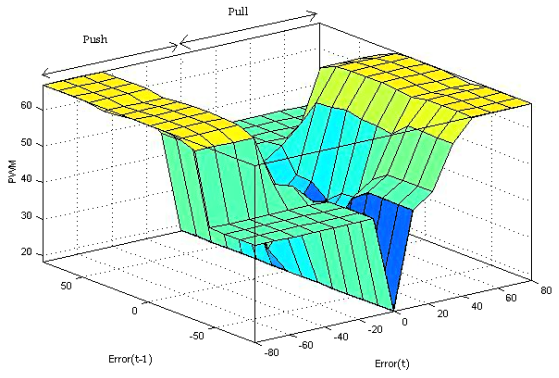
Una muestra concreta de Cadenas de Markov parcialmente observables es el algoritmo de navegación hacia objetivo del robot Xavier [Simmons y Koening, 1995]. En él se discretiza el espacio y se identifica cada posición posible como un estado del sistema, asociando una acción óptima a cada estado. Las acciones posibles también se discretizan, en giros de 90 grados y desplazamientos de 1 metro. La matriz de probabilidades de transición es relativamente sencilla pues un movimiento de avance tiende a cambiar a la posición vecina situada en frente, un comando de giro de 90 grados lleva al estado situado en la misma posición pero con diferente orientación. El sistema es un modelo de Markov porque la actuación puede fallar. Por ejemplo, si se comanda un giro, el estado real puede no haber cambiado, quizá por un atasco en el motor o quizá por el deslizamiento de las ruedas. En este sistema se mantiene una probabilidad asociada a cada uno de los estados, que se va actualizando en función de las observaciones sensoriales acumuladas, siguiendo la regla de Bayes. Como se identifican estados con posiciones, esto equivale a resolver el problema de localización. La programación del autómata empleando procesos de decisión de cadenas de Markov (POMDP) y técnicas ad hoc de planificación se pueden consultar en [Koening *et al.*, 1996].

2.3.5. Control borroso

Una aplicación de la lógica borrosa es el control, en concreto los controladores borrosos [Sugeno, 1985] [Saffiotti, 1997]. Estos controladores funcionan en base a unas reglas lingüísticas, con antecedentes y consecuentes expresados en lógica borrosa. Por ejemplo, `SI temperatura=mucho-frio Y calefacción=débil ENTONCES calefacción=fuerte`, donde `mucho-frio` es un conjunto borroso al que pertenecen en mayor o menor medida los distintos valores numéricos de la temperatura en grados. Tanto el antecedente como el consecuente admiten un valor de verdad gradual, siendo este último el fruto de la inferencia expresada en la regla. La salida borrosa de las distintas reglas aplicables se combinan dependiendo de su peso, y finalmente se destila un valor numérico que se aplica a los actuadores.

error(t-1) / error(t)	**NL**	**NS**	**Z**	**PS**	**PL**
NL	VW	M	VN	N	W
NS	W	N	VN	N	W
Z	W	N	VN	N	W
PS	W	M	VN	M	W
PL	W	M	VN	W	VW

(a)



(b)

Figura 2.12: *Superficie de control con reglas borrosas*

No hemos encuadrado el control borroso en la sección de control 2.1 porque históricamente no procede de la teoría clásica de sistemas. En su lugar hemos decidido incluirlos bajo el enfoque reactivo, pues las reglas borrosas dibujan de modo muy directo una asociación entre las entradas y las salidas del controlador. Las reglas borrosas se pueden ver como una generalización de las tablas simples, donde las situaciones se expresan como variables de entrada borrosas, y la superficie de control cubre todo el espacio de situaciones con una malla continua, fruto de las reglas lingüísticas. Por ejemplo, la figura 2.12(b) muestra la asociación situación-acción que se consigue con unas reglas en un espacio de situaciones determinado por dos variables de entrada (`error(t)` y `error(t-1)`), y un único valor de actuación como salida (señal `PWM`). En vez de definir la supercifie de control punto por punto se define con zonas o parches, cada una de los cuales corresponde a una combinación de las variables borrosas de entrada. Las fronteras entre los casos son más suaves que con una tabla discreta y varias reglas son aplicables simultáneamente con distinta intensidad.

De hecho, en muchas ocasiones el diseño de un controlador borroso consiste en rellenar una tabla

como la de la figura 2.12(a), de manera que se cubre todo el espacio de posibles situaciones para las variables de entrada consideradas. Las variables lingüísticas cuentan con la ventaja de que abarcan rangos continuos con un número finito de valores, de manera que las tablas quedan más compactas.

Los sistemas de control borrosos han demostrado ser útiles para problemas donde los controladores tradicionales fracasan, como es el control de sistemas no lineales o sistemas muy complejos. En general, debido su carácter *borroso*, estos controladores asimilan mejor la incertidumbre que los clásicos. Además al tener una formulación lingüística estas reglas se pueden extraer consultando a un experto humano si procede. Este tipo de controladores se ha empleado para desarrollar comportamientos atómicos [Reyes *et al.*, 1999] [Uribe *et al.*, 1995], y como piezas en arquitecturas más completas [Aguirre *et al.*, 2001] [Saffiotti y Wasik, 2003]. En [Sugeno, 1999] Michio Sugeno describe toda una arquitectura compuesta por muchos controladores borrosos, organizados en jerarquía. En esta sección no contemplamos este ejemplo, que será descrito más adelante, porque engloba cuestiones arquitectónicas más complejas, como la composición de comportamientos. En el paradigma reactivo únicamente consideramos los controladores borrosos unitarios, como otro modo de establecer asociaciones rápidas entre situaciones y acciones.

2.3.6. Ideas principales

El paradigma reactivo surgió como alternativa a los sistemas deliberativos simbólicos, que a mediados de los años 80 habían conseguido sólo resultados modestos sobre robots físicos. El nuevo paradigma defiende un modo de generar comportamiento basado en un acoplamiento directo entre situaciones y acciones. Esta asociación se almacena de algún modo dentro del sistema. El funcionamiento principal es un rápido bucle que chequea los valores sensoriales, busca en la situación que le corresponde y ejecuta la acción recomendada por la asociación. Lo que implica que no hay que esperar a tener el entorno completamente modelado para emitir una actuación ventajosa.

Las asociaciones situación-acción permiten reflejar conocimiento ad-hoc, heurístico, con independencia de cómo éstas se hayan obtenido. Entre las diferentes aproximaciones descritas en las secciones previas hay diferencias en cómo se almacenan y construyen las asociaciones: en una tabla rasa, con unas reglas borrosas, manualmente, con un programador de autómatas, etc. Sin embargo todas repiten el mismo esquema de funcionamiento: periodicamente leer los sensores y consultar la asociación para generar la acción oportuna. Este modo de operar confiere gran agilidad al comportamiento ante cambios en el entorno, que ya no tiene las etapas intermedias de modelado y planificación. Si la situación cambia, el sistema reacciona de inmediato, simplemente encuentra otra entrada en la asociación.

No hay manipulación de símbolos

Una de las aportaciones de este enfoque es su cuestionamiento de la necesidad de símbolos para generar comportamiento. La percepción en los sistemas reactivos no tiene que generar y mantener una representación simbólica, en correspondencia biyectiva con el mundo. En palabras de Brooks *el mundo es su mejor modelo* [Brooks, 1991c], su mejor representación, y el robot accede a ella a través de sus sensores. Es una percepción subsimbólica, en la cual las situaciones se describen como los valores de los sensores o sencillas transformaciones de éstos a través de variables intermedias. No se describen en términos simbólicos, sino muy pegadas a los sensores del robot, lo cual facilita la implementación de este enfoque en robots reales.

Robots reales con cuerpo

Otra característica relevante de este paradigma es la importancia que se da a los desarrollos sobre robots reales, como contraposición al paradigma deliberativo, que mantenía el procesamiento en un plano lógico y simbólico, un tanto alejado de la realidad física. La arquitectura de control reactiva ha de funcionar en un cuerpo robótico, con sensores y actuadores reales (*embodiment*) situados en un entorno real.

Acción situada

Otra aportación significativa es la interpretación del comportamiento como patrones de interacción continua con el entorno, en vez de una secuencia de acciones. Bajo esta concepción, el entorno cobra un papel relevante en el comportamiento, como contexto activo del robot. En este sentido a los enfoques reactivos también se les conoce como *teorías de acción situada*. El robot está inmerso en la situación actual, real (*situatedness*). Para el robot reactivo sólo existe el presente, y su preocupación es cómo actuar aquí y ahora. Por ello, en cuanto a las escalas temporales, los sistemas reactivos no tienen un horizonte futuro de anticipación, que sí es típico en los sistemas deliberativos. Además apenas tienen pasado, no necesitan memoria. Si acaso alguna variable interna que mantenga un pequeño estado.

La revolución reactiva supuso el replanteamiento de muchas asunciones troncales del paradigma deliberativo. Por ejemplo, la necesidad de símbolos, la idea de plan y su uso, incluso su propia necesidad, etc. No es de extrañar que la controversia haya sido muy fuerte [Vera y Simon, 1993], tanto en el plano filosófico, como en robótica. Por supuesto también en el psicológico, pues debajo de estos dos paradigmas subyacían sendas teorías de la inteligencia humana.

Englobados en el paradigma reactivo hemos descrito también otras técnicas nacidas en otros campos y en diferentes contextos históricos, como los controladores borrosos o los autómatas de estados finitos. La razón fundamental es que a pesar de su distinto origen, cuadran perfectamente con las líneas principales que argumenta la escuela reactiva.

2.3.7. Limitaciones

Además de las ventajas argumentadas y el éxito de este paradigma consiguiendo comportamientos relativamente complicados sobre robots reales, este enfoque también presenta numerosos inconvenientes. Por ejemplo, requiere la anticipación a tiempo de diseño de *todas* las situaciones en las que se va a encontrar el robot y el cálculo de la respuesta adecuada para todas ellas. La ejecución en un sistema reactivo es rápida y ágil, pero desde el punto de vista de diseño supone la traslación de todos los problemas de selección de acción a la etapa de construcción del sistema. Una vez calculada la acción oportuna, en tiempo de ejecución sólo hay consultar la asociada con la situación actual.

Una limitación de los sistemas reactivos puros es que no saben abordar situaciones nuevas. Si aparece una situación no contemplada en tiempo de diseño el sistema no sabe qué hacer. El aprendizaje conlleva un nuevo cálculo de la tabla de asociaciones. Esto contrasta con los sistemas deliberativos que enfrentados a una situación novedosa, son capaces de utilizar su deliberación para abordarla con éxito. Por ejemplo, a un sistema deliberativo con el comportamiento `ir-a-punto` se le puede comandar un destino nunca antes visitado. El robot cuenta con las herramientas para llegar. Por contra los sistemas reactivos no tienen objetivos explícitos, se mantienen implícitos en las asociaciones, y en este sentido no aceptan nuevos objetivos, salvo que se redefinan las asociaciones.

Otra crítica que se achaca a los sistemas reactivos puros es que no tienen anticipación temporal. Por ejemplo, como no tienen otra escala que el presente, hasta que no ven el obstáculo con sus sensores no reaccionan. No son capaces de anticipar la maniobra, pues no tienen una representación interna suya que les permita empezar a esquivarlo antes incluso de haberlo percibido con sus sensores.

Una desventaja de los enfoques basados en tablas es que resultan inadecuados en niveles bajos con variables continuas, donde quizá sea más conveniente una relación funcional directa, como un controlador clásico.

La principal limitación que adolecen los sistemas reactivos puros es que no escalan bien con la complejidad creciente. Su enfoque puede ser bueno para un sistema con pocos estados, en cuyo caso la actuación conveniente para cada estado se puede calcular incluso a mano. Sin embargo, cuando el robot se va a enfrentar a gran número de variables de entrada o un abanico amplio de condiciones sensoriales, entonces este mecanismo se vuelve inabordable por la explosión combinatoria en el número de estados y la necesidad de abarcarlos todos explícitamente. La tabla quedaría grandísima, imposible de manejar y calcular de antemano la respuesta óptima a cada posible situación. Por ejemplo, a nadie se le ocurre hacer un sistema de seguimiento visual teniendo como variables de entrada directamente los 200x200 pixels de la imagen de la cámara, con sus 256 valores posibles, de manera reactiva pura. Consistiría en anticipar todas las posibles imágenes que la cámara puede ver y codificar para cada una

de ellas el valor correspondiente de desplazamiento de la cámara. Para hacernos una idea, en la tabla habría $256^{40000} \sim 10^{96000}$ posibles estados.

Por ello apenas aparecen como técnica para programar un sistema completo que incluya un repertorio amplio de comportamientos observables distintos. Sin embargo, es frecuente su uso para programar comportamientos individuales, que forman parte, como unidades, de los arquitecturas basadas en comportamientos y de las arquitecturas híbridas.

2.4. Arquitecturas basadas en comportamientos

Los *sistemas basados en comportamientos* (SBC), surgieron dentro de la escuela reactiva, y su exponente más reconocido lo constituyen los trabajos de Rodney Brooks [Brooks, 1986]. Como distingue Mataric [Mataric, 1992a] en hemos separado los aspectos genuinamente reactivos, descritos en la anterior sección, de las cuestiones de distribución y emergencia de comportamientos, que abordamos en ésta.

La característica principal de los SBC es que abogan por la distribución del control y la percepción en varias unidades que funcionan en paralelo, llamadas de diversas maneras: niveles de competencia, comportamientos, esquemas, agentes, controladores, etc. Esta descomposición por actividades contrasta con la descomposición funcional y el control monolítico típicos de las arquitecturas deliberativas. La idea subyacente es que el comportamiento complejo de un sistema es una propiedad emergente que surge de las interacciones de los componentes básicos entre sí y con el entorno.

Cada componente es un bucle rápido, reactivo, desde los sensores a los actuadores, que tiene un objetivo propio. En general cada componente unitario implementa las reacciones adecuadas ante ciertos estímulos accediendo directamente a los datos sensoriales que necesita y emitiendo control a los actuadores. Por ello, además de suponer una distribución del control, este paradigma también propone la distribución de la percepción. En este sentido no necesita representación centralizada del entorno para generar comportamiento, ni modelos simbólicos. La información necesaria está directamente en las lecturas de los sensores.

Al surgir dentro del enfoque reactivo este paradigma comparte el énfasis en robots reales con cuerpo físico y en la interacción continua con el entorno (robots *situados*). De hecho, estos sistemas consiguen alta reactividad gracias a esa interacción incesante con el entorno sin necesidad de planificación ni de modelos del mundo. Estos sistemas basados en comportamientos suponen un salto sobre los sistemas reactivos puros, en busca de conductas más ricas y variadas. Como los sistemas de asociaciones situación-acción no escalan bien en complejidad, en este nuevo paradigma se trata de combinar varios de ellos para conseguir comportamientos finales más complejos.

Las arquitecturas basadas en comportamientos se diferencian unas de otras en el modo en que interactúan los diferentes comportamientos y en el método elegido para coordinarlos. Desde la jerarquía y la activación por prioridades, hasta la anarquía y la activación por un autómata finito de estados que hace las veces de árbitro. En esta sección describiremos con más detalle dos trabajos fundacionales, de Brooks y Arkin, que ya contienen las ideas principales de esta escuela. Hemos incluido también en esta sección dos arquitecturas más, la red de comportamientos de Maes [Maes, 1990] y la arquitectura borrosa de Sugeno [Sugeno, 1999]. Ambas se apoyan también en la distribución en comportamientos básicos y proponen sendas soluciones a la coordinación. Hemos desplazado otras aproximaciones diferentes al problema de la coordinación, como la arquitectura DAMN, a la sección de arquitecturas híbridas, pues nacieron con vocación mayor de coordinar las tendencias reactivas y deliberativas.

2.4.1. Arquitectura de subsunción de Brooks

En 1986 Rodney Brooks propuso una descomposición estratificada del comportamiento en *niveles de competencia* [Brooks, 1986; Brooks, 1987], como ilustra la figura 2.13. El control no reside en un único centro de decisión sino que el sistema se compone de varios niveles de competencia ejecutándose en paralelo, y de la interacción entre todos ellos sale la acción que se ejecuta en cada momento. Los niveles funcionan concurrentemente y el comportamiento global observable emerge de la ejecución simultánea de todos ellos.

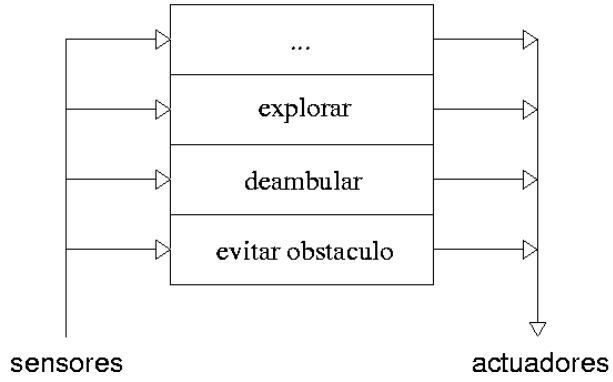


Figura 2.13: *Descomposición por actividades del comportamiento: niveles de competencia*

Un nivel de competencia es una especificación informal para una clase de comportamientos de un robot. Cada nivel se implementa como una red de autómatas de estados finitos, que tienen canales de comunicación de bajo ancho de banda para intercambiar señales y pequeñas variables entre ellos. Los niveles inferiores proporcionan comportamientos básicos como el sorteo de obstáculos, la navegación deambulante, etc. Comportamientos más sofisticados se generan añadiendo niveles adicionales sobre los ya existentes.

Los niveles superiores pueden suprimir las salidas de los inferiores y suplantarles sus entradas, como ilustra la figura 2.14(a). Esto se conoce como *subsunción* y da nombre a la arquitectura. En la figura 2.14(b) se muestra un ejemplo con dos niveles indicados por las elipses discontinuas. El nivel de competencia 0 materializa la capacidad de evitar el choque con obstáculos que se acercan al robot. Para ello conecta varios autómatas; en función de las lecturas de los sensores sonar (autómata *sonar*) indicando proximidad de obstáculos en varias direcciones elabora una fuerza repulsiva (*feelforce*) en dirección opuesta a los obstáculos cercanos. Esa referencia se envía a los motores (*motor*). El nivel de competencia 1 ofrece la capacidad de deambular sin chocar con obstáculos. Para ello conecta un autómata que genera direcciones aleatorias (*wander*) con otro que combina la dirección opuesta a los obstáculos con la dirección deseada de avance (*avoid*). Este nivel suprime la salida del primer nivel a los motores, y la suplanta con la suya. Otras capacidades como la navegación guiada por visión se pueden añadir posteriormente con niveles de competencia adicionales sobre los ya existentes [Brooks, 1986].

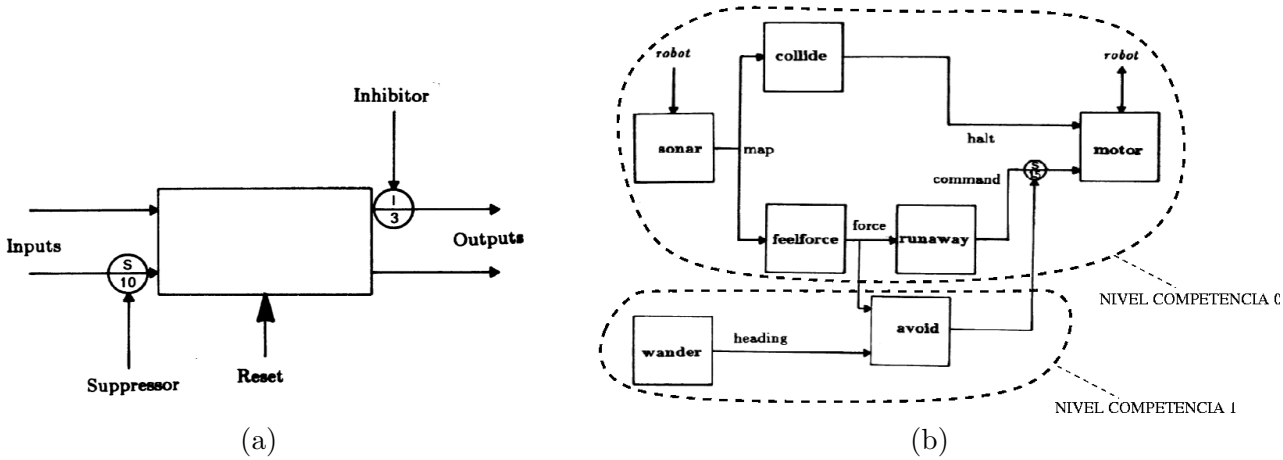


Figura 2.14: *Subsunción como supresión de salidas y suplantación de entradas (a). Dos niveles de navegación (b).*

Esta arquitectura desemboca en una red de autómatas, ordenados en una suerte de jerarquía, dado que los niveles superiores pueden suprimir las salidas de los inferiores. Como mecanismo de selección de acción se utilizan prioridades fijas, implícitas en la red de autómatas. Al ser una red fija, también lo

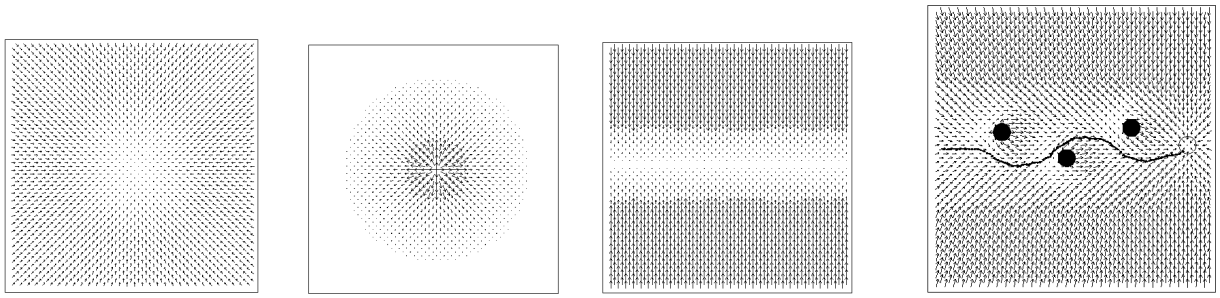


Figura 2.15: *Combinación de varios esquemas*

son estas prioridades, que establecen un arbitraje estático entre todos los comportamientos. Además no hay mezcla, no hay fusión de varias tendencias a la hora de elegir la actuación más ventajosa en cada momento.

La arquitectura de Brooks presenta una percepción reactiva, muy orientada a la tarea y también distribuida. En clara ruptura con las ideas deliberativas, no necesita un modelo completo y centralizado del mundo[Brooks, 1991c], ni manipulaciones simbólicas [Brooks, 1991b] para generar comportamiento. De hecho presenta una percepción muy próxima a los sensores, sin símbolos. En este sentido los objetivos del robot tampoco están representados simbólicamente, más bien están implícitos en la red de autómatas que decide su funcionamiento.

Desde el punto de vista de desarrollo del sistema, esta arquitectura plantea un desarrollo incremental en el que se tiene algún comportamiento real desde el primero momento. No hay que esperar a tener toda la arquitectura terminada para tener cierta funcionalidad. Ya con la primera capa el robot hace algo. El diseño es incremental, incorporando nuevos niveles de competencia sobre los ya existentes. Sólo cuando una capa se ha probado exhaustivamente entonces se añade otra. Además el diseño de una capa es independiente del funcionamiento de las superiores, y debe ser terminado completamente antes de empezar con las otras.

Siguiendo la arquitectura de subsunción se han conseguido demostraciones prácticas impresionantes, que muestran gran eficiencia en tareas de bajo nivel como la recolección de latas, la navegación local, etc. Por ejemplo, el robot Herbert[4][Brooks *et al.*, 1988], que recogía latas de refresco vacías y las llevaba a una papelera, o el robot Toto [Mataric, 1992b], que elaboraba mapas topológicos del entorno.

2.4.2. Esquemas de Arkin: AuRA

Próximos a la neurofisiología surge la teoría de esquemas, en la que destacan los trabajos de Arbib [Arbib y Liaw, 1995; Corbacho y Arbib, 1995] y Arkin [Arkin, 1989b]. Los *esquemas* son pequeñas unidades de ejecución que pueden ser *perceptivos* si elaboran representación, o *motores* si generan actuación. Los esquemas se interconectan entre sí, de manera que las salidas de los esquemas perceptivos suelen ser entradas para los esquemas motores. Una arquitectura representativa basada en esquemas es AuRA (Autonomous Robot Architecture) [Arkin, 1989b].

En esta arquitectura es normal la mezcla de salidas de varios esquemas motores. Por ejemplo, la salida de un esquema motor de navegación en [Arkin, 1989b] es un vector con las velocidades de giro y tracción deseadas. El comportamiento global de navegación se obtiene con la combinación de varios esquemas: `evita-obstáculos-móviles`, `evita-obstáculos-estáticos`, `permanece-en-el-camino` y `avanza-hacia-el-objetivo`. En este caso cada esquema se implementa como un campo vectorial que entrega un vector de fuerza en cada punto del espacio, tal y como ilustra la figura 2.15. La figura 2.15(a) contiene el campo vectorial del esquema que acerca el robot al objetivo. En cada punto recomienda la orientación que le lleva al punto de destino. La figura 2.15(b) contiene el campo vectorial del esquema `evita-obstáculos-estáticos`, que para cada punto recomienda un movimiento de alejamiento, como

[4]http://www.ai.mit.edu/projects/mobile-robots/veterans.html

si el obstáculo ejerciera una fuerza de repulsión sobre el robot. El sistema instancia uno de estos esquemas por cada obstáculo que detecta. La figura 2.15(c) muestra el campo vectorial del esquema `permanece-en-el-camino`, que ejerce una fuerza sobre el robot tratando de centrarlo en el pasillo. El vector final de movimiento que realmente se comanda a los motores es la *superposición* de todos esos campos, su suma vectorial.

Esta variante de los campos de potencial es la solución de Arkin al problema de la selección de acción y la coordinación entre los distintos esquemas motores. Es esta combinación la que consigue un comportamiento global correcto, que navega por el pasillo hacia el destino evitando los obstáculos del camino, tal y como ilustra la figura 2.15(d).

Una diferencia con respecto a los mapas internalizados de Payton es que estos campos de potencial no necesitan explicitarse en todos los puntos de antemano. Sólo se calculan en el punto actual en el que se encuentre el robot, y su cálculo puede venir dado por fórmulas sencillas como $\frac{S-d}{S-R}$ para el caso de`evita-obstáculos-estáticos`[5] o $\frac{d}{W/2}$ para el esquema `permanece-en-el-camino`[6] [Arkin, 1989b]. De este modo en cada iteración sólo se hace un conjunto de rápidos cómputos, que no retardan en exceso la respuesta.

La percepción en esta arquitectura también es distribuida, precisamente en los distintos esquemas perceptivos. Además tiene la característica de ser *orientada a la tarea*, y no hacia la reconstrucción del entorno. En este sentido recalca que no todos los datos sensoriales deben ser procesados, sólo aquellos pertinentes para la tarea que el robot tiene entre manos. En AuRA, cada esquema motor incorpora un esquema perceptivo para proporcionar la información sensorial necesaria [Arkin, 1989b]. Al subyugar los esquemas perceptivos a los esquemas motores se obtiene un mecanismo que enfoca la atención del robot, de manera que concentra el procesamiento en la elaboración de los datos útiles para la tarea actual. Además esta concepción contempla la fusión sensorial e incluso la elaboración de *perceptos*, representaciones locales a comportamientos individuales [Arkin, 1995].

Una revisión posterior, del propio Arkin, incluye una respuesta al problema de la secuenciación de varios comportamientos complejos. Para ello se usa un autómata de estados finitos, en el cual cada estado se corresponde con la activación de ciertos esquemas y se definen eventos clave que disparan el salto entre estados [Arkin y Balch, 1997]. También con las arquitecturas basadas en esquemas se ha conseguido replicar en robots algunos comportamientos de animales como la rana [Corbacho y Arbib, 1995] o la mantis religiosa [Ali y Arkin, 1998].

2.4.3. Red de comportamientos de Maes

La arquitectura de Pattie Maes genera el comportamiento como acción combinada de un conjunto de comportamientos básicos [Maes, 1989b]. Los módulos se entrelazan en una red, como la mostrada en la figura 2.16, de precondiciones y postcondiciones. En esa red de módulos cada cual tiene su propia competencia, y todos interactúan de manera que el comportamiento emergente de esa interacción es el deseable.

Cada módulo o comportamiento básico tiene unas precondiciones, que expresan las condiciones requeridas por el módulo para poderse ejecutar. Además almacena unas postcondiciones, que expresan las consecuencias en el mundo de la acción que ejecuta el módulo, en forma de proposiciones que serán ciertas cuando el módulo se ejecute y proposiciones que serán falsas cuando el módulo se ejecute, muy similares a las `add-list` y `delete-list` utilizadas por STRIPS. Los módulos cuyas postcondiciones hacen cierta alguna precondición de otro comportamiento son *predecesores* suyos, del mismo modo que para aquellos éste es un *sucesor* suyo. Cada módulo está conectado con sus sucesores y con sus predecesores. Adicionalmente también se conecta con sus *detractores*, aquellos otros cuyas postcondiciones hacen falsa alguna precondición suya, y a los módulos de los que él es detractor. En la figura 2.16 se aprecia la red del sistema, las flechas nacen de un módulo hacia sus predecesores, y los enlaces con un círculo apuntan hacia sus detractores.

Para coordinar todos esos módulos Maes plantea el problema de la selección de acción como una dinámica de energía que se propaga a través de la red siguiendo unos patrones y acaba concentrándose

[5] donde R es el radio del obstáculo, d la distancia al centro del obstáculo desde el punto sobre el que estamos calculando el campo, y S es el radio de cierta esfera de influencia a partir de la cual consideramos nulo el campo

[6] donde d es la distancia al centro del pasillo, y W la anchura del propio pasillo

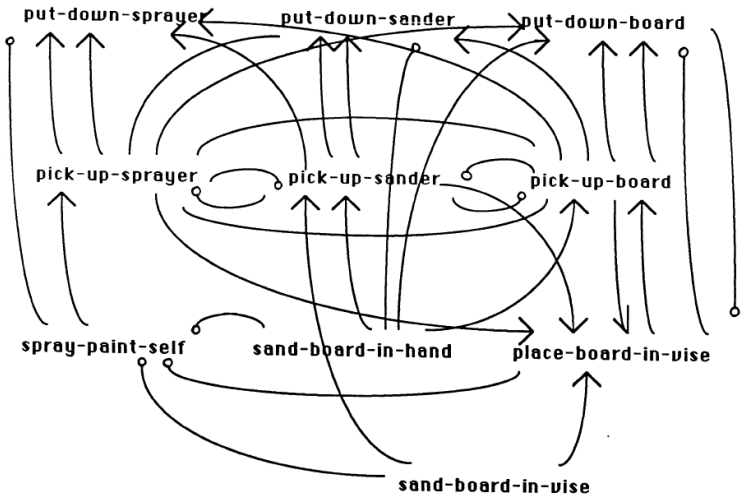


Figura 2.16: *Red de comportamientos de Pattie Maes*

en el módulo más adecuado para la situación del entorno y los objetivos actuales [Maes, 1989a; Maes, 1989b; Maes, 1990]. En cada iteración de control el estado del entorno inyecta energía a los módulos cuyas precondiciones casan con la situación actual. Los objetivos por conseguir también inyectan energía a los módulos que los incluyen en sus postcondiciones. Los objetivos conseguidos la detraen de los módulos cuyas postcondiciones los niegan, tratando de mantenerse. Además de estas inyecciones externas, se establece una dinámica similar de propagación de energías entre la propia red de módulos. Por un lado, los módulos no ejecutables propagan parte de su activación a sus predecesores, y los módulos ejecutables a sus sucesores. Adicionalmente los módulos ejecutables detraen energía de sus detractores. Después de un tiempo de estabilización de esta dinámica, se elige el módulo que acumula más energía de activación como aquel que toma el control en el siguiente instante.

Esta técnica de selección de acción acaba eligiendo una actuación que supone un compromiso entre la orientación necesaria hacia los objetivos y su relevancia en la situación actual, pues de ambos parte inicialmente la energía de activación. Esta dinámica se repite periódicamente cada vez que el sistema va a decidir una actuación en la situación actual. La transmisión de energía en la red se puede modular con ciertos parámetros, cuya sintonía puede hacer que el comportamiento global sea más reactivo o más orientado hacia objetivos. Maes construyó la red para un robot con dos tareas, que comprobó en un simulador: pintar un tablón y pintarse a sí mismo [Maes, 1989b].

Una característica relevante de esta arquitectura es que incluye objetivos de modo explícito, en lógica proposicional. Esto la hace diferente de la mayoría de los SBC, que manejan objetivos implícitos, grabados a fuego en el código de cada comportamiento. Además, a pesar de carecer de módulos deliberativos, la dinámica de energías ofrece una cierta capacidad de planificación, de anticipación de consecuencias y encadenamiento hacia los objetivos. Por ejemplo, los comportamientos que consiguen un objetivo pero no son directamente ejecutables propagan su energía a sus predecesores, tratando de que hagan ciertas sus precondiciones en el siguiente instante. Si estos no son ejecutables, nuevamente se propagará a sus propios predecesores, estableciendo de este modo una suerte de cadena, cuya ejecución ordenada, lleva a la consecución del objetivo. Sin embargo, este orden se consigue sin utilizar planificación explícita, gracias al chequeo reactivo de precondiciones y la propagación de energías de activación.

Esa anticipación temporal se consigue sin sacrificar a cambio la reactividad. Esta técnica de selección de acción es muy flexible: si los objetivos del sistema se cambian en algún momento, la red adaptará su comportamiento a los nuevos objetivos. Igualmente ocurre con cambios repentinos en el entorno, que varían los caminos de la energía inyectada en la red.

Una de las críticas principales a esta arquitectura es que omite completamente la parte perceptiva, que asume resuelta y expresada como un conjunto de hechos en lógica proposicional. En ningún momento se describe cómo se atribuyen los valores de verdadero o falso a las precondiciones de los

módulos y este punto es fundamental en la percepción de cambios en el entorno. No se plantea cómo se elabora esa percepción, ni que unos estímulos puedan requerir más tiempo de elaboración de otros. De este modo se puede decir que la selección de acción sí es reactiva y flexible, pero no podemos decir que lo sea el sistema global, puesto que no se ha descrito el modo de mantener esa representación necesaria.

Además el hecho de considerar que las precondiciones toman valores binarios es una simplificación excesiva sobre la realidad del mundo tal y como señala Tyrrell [Tyrrell, 1993a]. En este sentido trabajos posteriores [Dorer, 1999] generalizan esa representación a situaciones continuas. Otra limitación es que no ofrece percepción selectiva. Es necesario percibir los estados de todas las precondiciones de todos los módulos del sistema, ya que la inyección de energía por parte del estado del mundo a la red depende de esta percepción.

2.4.4. Arquitectura borrosa de Sugeno

Hemos incluido dentro del paradigma basado en comportamientos la arquitectura borrosa de Michio Sugeno [Sugeno, 1999], que extiende el uso de la lógica borrosa a todos los aspectos de arquitectura. Esta arquitectura es la extensión natural de los controladores borrosos simples para tareas más complejas. Aunque surge en un ámbito distinto, coincide en la línea de distribución de control que propugnan los SBC. En general la tecnología borrosa se ha mostrado útil en robótica, tanto en percepción, por su capacidad de representar incertidumbre, como en el diseño de controladores o para combinar recomendaciones de actuación, etc. El trabajo [Saffiotti, 1997] de Saffiotti supone una buena revisión de estos usos.

La arquitectura de Sugeno está muy ligada la aplicación concreta en la que nació: controlar el movimiento de un helicóptero autónomo. La arquitectura ofrece gran reactividad, manteniendo un tiempo de ciclo muy pequeño, lo que le permite afrontar con éxito el efecto impredecible del viento. El helicóptero tiene principalmente cuatro grados de libertad: inclinación vertical, lateral (*roll angle*), longitudinal (*pitch angle*) y pedales de cola, que se varían manipulando los ángulos del rotor principal del helicóptero, de sus hojas (*collective pitch*) y de las hojas del rotor de cola (*antitorque pedals*). Sugeno propone una arquitectura de control jerárquica, que en su primer nivel tiene una batería de controladores borrosos sobre los cuatro grados de libertad, tal y como ilustra la figura 2.17 (*fuzzy control modules*). Estos grados de libertad están muy relacionados entre sí, de manera que un cambio en un actuador puede hacer variar la situación en varios de ellos. Por ello se incluye un manejador de controladores (*fuzzy control management*) que se encarga de ajustar parámetros de interacción y compensar explícitamente el acoplamiento entre los distintos controladores.

Por encima de este nivel se tienen controladores borrosos que materializan los modos básicos de vuelo (*basic flight modes*), modulando las referencias para los controladores inferiores. Por ejemplo, el despegue, el aterrizaje, el avance hacia delante, el cernido, los giros, etc son modos básicos. Cada uno de estos controladores se ha desarrollado y probado exhaustivamente por separado. Estos controladores son los ladrillos con los que se componen movimientos compuestos. De hecho un mismo movimiento básico puede participar en la generación de varios movimientos compuestos.

La arquitectura ofrece la fusión de modos básicos o su alternancia temporal, como ilustra la parte superior de la figura 2.17. Por ejemplo, el comportamiento volar en línea recta (tridimensional) se consigue fusionando dos modos básicos: el ascenso y el avance longitudinal. Por supuesto esta combinación se realiza utilizando lógica borrosa, que es el leit motiv de la arquitectura. Esta composición borrosa de comportamientos es, junto con la organización jerárquica, la característica fundamental de esta arquitectura, y una solución a la selección de acción cuando hay que coordinar los muchos controladores, borrosos a su vez, del sistema.

La percepción en esta arquitectura se plantea como un conjunto de variables físicas (valores de los giróscopos, altímetros, tacómetros, etc) sobre los cuales se crea un repertorio de etiquetas lingüísticas. Esas variables están siempre disponibles, y el tratamiento más complicado a que son sometidas es su *borrosificación*, pasando su valor continuo a un valor borroso de pertenencia a cada una de las etiquetas.

Otra arquitectura relevante que incorpora la lógica borrosa como herramienta de coordinación entre diferentes comportamientos es la arquitectura de Saphira [Konolige y Myers, 1998]. Como veremos

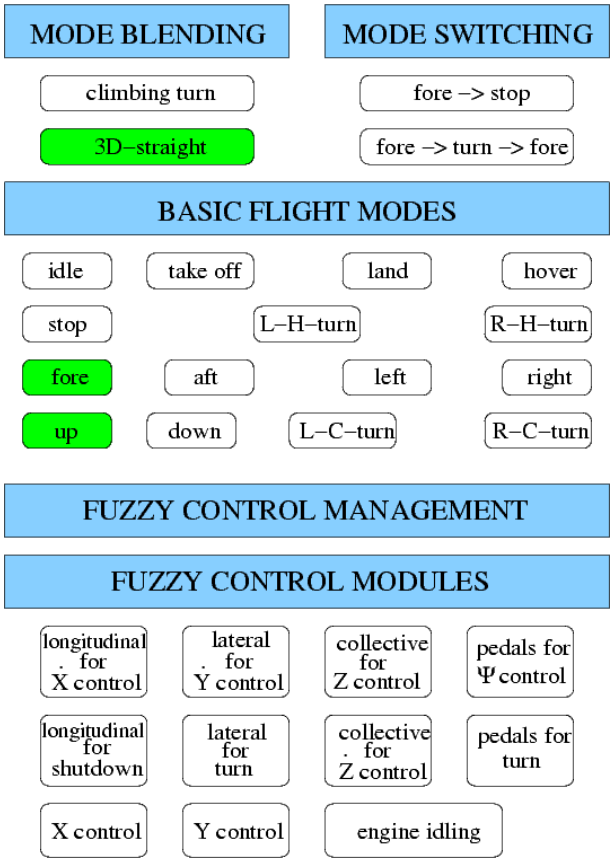


Figura 2.17: *Arquitectura borrosa de Sugeno para varios modos de vuelo*

posteriormente, Saphira elabora un modelo simbólico del entorno e introduce elementos de planificación en sus niveles superiores, por lo que la describiremos con más detalle dentro del paradigma híbrido.

2.4.5. Ideas principales

Una vez presentadas cuatro arquitecturas concretas que hemos encuadrado dentro del paradigma basado en comportamientos, repasaremos ahora los principios que hemos identificado subyacentes a todas ellas. De este modo hemos identificado las aportaciones principales de esta corriente desde nuestra óptica. Otros análisis de las ideas fundamentales de esta escuela se pueden encontrar en [Brooks, 1991b; Mataric, 1992a; Arkin, 1995].

Distribución de comportamientos y coordinación de acción

El concepto fundamental de los SBC es la distribución en comportamientos individuales, llámense estos niveles de competencia, esquemas o controladores. Esta distribución introduce el problema de coordinar los comportamientos, principalmente a la hora de decidir cuál es la acción a ejecutar en cada momento. El robot tiene varios objetivos que satisfacer a la vez (por ejemplo, mantenerse alejado de los obstáculos pero avanzar simultaneamente en la dirección deseada), y en los SBC se distribuye esa colección de objetivos parciales en pequeñas unidades que velan por la consecución de cada uno de ellos. Así, cada comportamiento unitario tiene su propio objetivo, que puede entrar en contradicción con algún otro. El mecanismo de coordinación es el responsable armonizarlos para satisfacerlos en el mayor grado posible. ¿Cómo se calcula la actuación final en este escenario? Básicamente hay dos paradigmas para resolver este problema de selección de acción en estas arquitecturas: el arbitraje y la fusión de comandos.

El arbitraje establece una competición por el control entre todas las unidades de comportamiento y sólo la ganadora determina la actuación final atendiendo exlusivamente a sus intereses (*Winner takes*

all). Las prioridades fijas [Brooks, 1986], las redes de activación de Maes [Maes, 1989a] y el arbitraje basado en estados [Arkin y Balch, 1997] pertenecen a esta familia.

Las técnicas de fusión de comandos mezclan las recomendaciones de las unidades de comportamiento relevantes en una recomendación global que tiene en cuenta las preferencias de todas ellas. Las aproximaciones significativas dentro de esta línea son la superposición [Arkin, 1989b], la mezcla borrosa [Saffiotti, 1997] y las votaciones de Payton [Payton, 1990] o de Pirjanian [Pirjanian *et al.*, 1998]. Una buena recapitulación de las distintas técnicas de selección de acción aparece en [Pirjanian, 1997a; Pirjanian, 1997b]. Cabe resaltar el trabajo de Paolo Pirjanian [Pirjanian, 2000], el cual resalta la equivalencia entre el problema de selección de acción y la teoría de decisión multiobjetivo. En este contexto se plantea como un problema de optimización vectorial, en el que se definen criterios de optimalidad (en el sentido de Pareto) y que alcanzan soluciones de compromiso entre los distintos objetivos del robot.

En general, no está claro cuando conviene más una coordinación arbitrada o una que fusione, es un debate aún abierto. El arbitraje adolece de no considerar varios comportamientos simultáneamente, por lo que no concluye actuaciones de compromiso convenientes a varios objetivos a la vez, aunque estas puedan existir. Por ejemplo, un robot puede bordear un obstáculo por sus dos lados, por la izquierda beneficia además a otro objetivo del robot. Sin embargo, decide irse por la derecha porque no se exploran posibilidades de acción que vengan bien a otros objetivos. A su vez, en escenarios con alto dinamismo donde está en juego la propia supervivencia del robot, el arbitraje concentra la acción en el objetivo prioritario mejor que otras opciones fusionadoras, tal y como señala Cañamero [Cañamero *et al.*, 2002].

En cualquier caso la coordinación es una cuestión complicada y las comparativas realizadas [Tyrrell, 1993b; Cañamero *et al.*, 2002] entre diferentes arquitecturas son muy difíciles, debido principalmente a la heterogeneidad de los sistemas y a la ausencia de un patrón de medida universalmente admitido. En la sección dedicada a los sistemas híbridos veremos algún otro mecanismo de coordinación, que entonces se emplea para combinar las tendencias deliberativas y reactivas del sistema.

Percepción subsimbólica orientada a tarea

La distribución típica de este paradigma no es sólo del control, también lo es de la percepción. En este sentido, en los SBC no hay una representación centralizada del mundo. En su lugar cada comportamiento elabora su propia representación, de manera que la percepción del sistema aparece fragmentada en pequeñas unidades que se actualizan independientemente.

Además la percepción en este paradigma es más orientada a tarea que reconstructora del entorno u objetiva. Está muy pegada a los sensores, las lecturas sensoriales o sencillas transformaciones a partir de ellos. Se percibe para actuar, y se buscan invariantes sensoriales para la actuación correcta en base a ellos, evitando al máximo la necesidad de modelar simbólicamente la realidad.

Componentes reactivos

En cuanto a cada comportamiento individual, el paradigma propone sistemas reactivos en continua interacción con el entorno. Por lo tanto, comparte con los sistemas reactivos descritos en la sección 2.3 el énfasis en la acción situada, el comportamiento como patrones de interacción con el entorno, y la importancia de trabajar con robots reales, con cuerpo físico. En este sentido estas arquitecturas han dado lugar a multitud de trabajos con robots reales en funcionamiento, capaces de reaccionar de modo seguro y eficaz ante cambios en el entorno.

Unos comportamientos pueden modular a otros, por ejemplo a través de la subsunción o enviando referencias de control, como en la arquitectura borrosa, pero típicamente no hay comunicación simbólica entre ellos. De hecho, la comunicación entre comportamientos se realiza fundamentalmente a través del mundo. Cada comportamiento tiene sus sensores y actuadores, de manera que uno puede provocar cambios en la realidad que el otro percibirá a través de sus sensores. De esta manera el diseño queda más limpio, pues minimiza las interacciones internas.

En esta misma línea, como los sistemas reactivos puros de la sección 2.3, los SBC no tienen representación explícita de los objetivos. Éstos se encuentran implícitos en la red de autómatas, en la

jerarquía de controladores o en el conjunto de esquemas. Están grabados a fuego en el código de los comportamientos unitarios y por ello son difícilmente modificables. Además esta elipsis corta de raíz cualquier manipulación simbólica con los objetivos. Una notable salvedad es la red de comportamientos de Maes [Maes, 1989a], que sí tiene explícitos los objetivos, aunque no haga una deliberación clásica con ellos.

2.4.6. Limitaciones

Pese a los éxitos conseguidos con robots reales bajo este paradigma, con el paso del tiempo se han ido apreciando limitaciones inherentes a este enfoque. En primer lugar, no han escalado a comportamientos de complejidad humana, tal y como preveía Brooks [Brooks *et al.*, 1998]. Los problemas de navegación, o de seguimiento se han resuelto correctamente con este enfoque, pero otras tareas que requieren cierta anticipación no se han conseguido. Además, pese al diseño ortogonal entre capas, no se ha subido en muchos niveles de competencia. La complejidad en los niveles superiores, por encima de cuatro, empieza a ser inmanejable. A medida que crece el número de comportamientos básicos, la tarea de gestionar las relaciones entre ellos y de diseñar un buen sistema de arbitraje se hace muy complicada. La introducción de un nuevo comportamiento a partir de los ya existentes no es sistemática, es más bien un arte difícil. Por ejemplo, la complejidad de la red de Maes [Maes, 1990] crece rápidamente cuando se añaden módulos nuevos, y su inclusión puede modificar el ajuste conseguido entre los comportamientos anteriores.

La ausencia de representación simbólica ha coartado las tareas que se pueden abordar con éxito en este paradigma. Por ejemplo, hay situaciones que requieren una cierta anticipación o el manejo de información que no proviene directamente de los sensores. En otros casos los estímulos relevantes son multimodales, es decir, no están contenidos en ningún sensor aislado, y necesitan de una combinación no trivial de información proveniente de varios sensores.

Al prohibir la representación simbólica, la información disponible es básicamente el conjunto de lecturas sensoriales, y por tanto están limitadas a su rango de observación. Este alcance es eminentemente local, tanto en tiempo como en espacio. Esta información local es toda la que se tiene para decidir la actuación, lo cual le confiere cierta miopía a esas actuaciones. Esto contrasta con el enfoque deliberativo que puede razonar sobre elementos que no son directamente accesibles por los sensores. Por ejemplo, puede almacenar la posición de cierto obstáculo, que ahora está fuera del alcance sensorial, pero que puede afectar a las decisiones de movimiento.

En el plano temporal los SBC son sistemas sin capacidad de visión a largo plazo, funcionan únicamente en el instante actual. Esta falta de anticipación, de horizonte temporal, puede en ciertos casos presentar comportamientos cíclicos. Por ejemplo, los mínimos locales en el campo vectorial de Arkin [Arkin, 1989b] son los puntos de equilibrio indeseados entre la querencia hacia el objetivo y la repulsión ante los obstáculos. Sin algún mecanismo para detectarlos y evitarlos, el robot entra en un bucle que bloquea su progreso hacia el objetivo. Por ejemplo, los obstáculos en forma de U detrás de los cuales está el objetivo presentan un grave problema para estos sistemas porque son un callejón sin salida. Se han desarrollado mecanismos ad hoc que palían estos casos, por ejemplo, con algún temporizador monitorizando progreso hacia el objetivo [Borenstein y Koren, 1989; Boumaza y Louchet, 2001].

Una crítica que los investigadores proclives a los sistemas deliberativos achacan a los SBC, y a las arquitecturas reactivas en general, es la dificultad de especificar objetivos [Maes, 1990]. En este sentido para realizar comportamientos similares hay que recodificar parte del sistema. Esto contrasta con los planificadores deliberativos, que son capaces de elaborar un plan de acción para objetivos similares, de modo automático. Adicionalmente hace que estos sistemas sean difícilmente predecibles.

2.5. Arquitecturas híbridas

A la luz de las limitaciones observadas en las escuelas reactiva [Mataric, 2002] y deliberativa [Brooks, 1991b] han surgido muchas aproximaciones híbridas, que tratan de combinar las ventajas de ambos en una única arquitectura. Por ejemplo, aunar las habilidades de anticipación y orientación a

objetivos que proporciona la planificación con la flexibilidad frente a imprevistos que proporciona la reactividad.

Las arquitecturas deliberativas ofrecen una alta orientación hacia los objetivos, pero fallan en la reacción a tiempo ante cambios imprevistos en el entorno. Además son frágiles frente a incertidumbres, pues necesitan modelar el mundo antes de empezar a actuar. Si bien el modelo reactivo supera esta falta de agilidad también adolece de otras deficiencias, como las dificultades para incorporar objetivos explícitos, la ausencia de horizonte temporal o que no escala bien cuando se quieren desarrollar comportamientos complejos. Por otro lado, las arquitecturas deliberativas puras complementan bien estas limitaciones: son capaces de actuar con anticipación temporal y de resolver problemas complejos mediante la descomposición de éstos en otros más sencillos. Esta misma capacidad evita que los navegadores deliberativos se vean atrapados en callejones sin salida.

Las arquitecturas híbridas son sin duda las más extendidas en la actualidad. Es difícil encontrar un robot autónomo que no tenga algunas componentes reactivas y cierta deliberación. De hecho hay un extenso consenso en que la componente reactiva es el modo correcto de realizar el control de bajo nivel, quizá sustentado en su éxito práctico.

Las diferencias dentro de las arquitecturas híbridas aparecen fundamentalmente en cómo introducir la parte deliberativa sin fisuras y sin romper el funcionamiento reactivo. Así, hay sistemas jerárquicos que utilizan deliberación simbólica como motor principal, pero tienen como acciones primitivas la activación de ciertas rutinas reactivas básicas. Este matiz descendente aparece en arquitecturas concretas como Saphira [Konolige y Myers, 1998] o TCA [Simmons, 1994]. Otros enfoques híbridos tienen un marcado matiz ascendente como los RAP de Firby [Firby, 1992] que construyen desde la parte reactiva pequeñas unidades de actuación que incorporan cierta replanificación, o la arquitectura DAMN [Rosenblatt y Pyton, 1989] que introduce unos *comportamientos deliberativos* en un sistema reactivo, al mismo nivel que los genuinamente reactivos.

Tanto las características principales de los sistemas deliberativos (orientación a objetivos, planificación, descomposición de tareas en subtareas y el modelado), como las ofrecidas por las arquitecturas basadas en comportamientos (reactividad, orientación al entorno) son recomendables para un robot real. Un robot en el mundo real necesita poseer ambos tipos de capacidades: necesita ser reactivo para poder trabajar en entornos altamente dinámicos, pero también necesita ser capaz de realizar tareas complejas que pueden requerir cierta planificación. Ya Maes identificó estas dos características como deseables en el comportamiento del sistema [Maes, 1989b]. De hecho, en su sistema se ponen en contraposición, y el comportamiento global se puede balancear hacia un extremo u otro sintonizando ciertos parámetros, tal y como vimos en la sección 2.4.

En cuanto a las tres primitivas señaladas por R.Murphy [Murphy, 2000] para el comportamiento (sensar, planificar y actuar), los enfoques híbridos las organizan como dos bloques interrelacionados. El primero se encarga de la planificación, e interactúa con el segundo, que ejecuta constantemente asociaciones entre sensar y actuar. Así pues, su organización sería: planificar, sensar-actuar.

En esta sección describiremos cuatro ejemplos concretos de arquitecturas híbridas: TCA de Simmons, arquitectura 3T de Bonasso, Saphira de Konolige y DAMN de Rosenblatt. Hemos elegido éstas porque barren un amplio abanico de posibilidades y son muy representativas. Finalmente, extraemos las características comunes, que identifican al paradigma híbrido dentro de su heterogeneidad, y señalamos algunas limitaciones y críticas que se le achacan.

2.5.1. TCA de Simmons

El *control estructurado* [Simmons, 1994] es la propuesta de Reid Simmons para combinar en una misma arquitectura aspectos deliberativos y aspectos reactivos. La implementación de esta idea es la arquitectura TCA (*Task Control Architecture*), que propone una descomposición del sistema en módulos concurrentes que intercambian mensajes entre sí. Unos módulos utilizan la funcionalidad de otros a través de paso de mensajes que necesitan ser tratados. Cada uno de ellos emite ciertos mensajes y tiene registrados manejadores para los mensajes que es capaz de procesar. TCA imbrica la planificación y la ejecución. Además introduce explícitamente la monitorización, y con ella la planificación de percepción (una suerte de atención).

Dentro de TCA los mensajes que puede emitir un módulo son: (1) informativo (*Information*

Message), para comunicar cierta información a quien le pueda interesar, tipo multidifusión; (2) petición (*Query Message*), para preguntar cierta información, el receptor elabora la respuesta y se la devuelve; (3) objetivo (*Goal Message*), para introducir un nuevo objetivo al sistema; (4) comando (*Command Message*), para solicitar cierta actuación, enviar comando a los actuadores; (5) monitor (*Monitor Message*), para programar cierta comprobación de condiciones y (6) excepción (*Exception Message*), cuando ha detectado cierta emergencia. Tal y como muestra la figura 2.18(b), hay un módulo central que se encarga de coordinar los distintos módulos del sistema encaminando los mensajes hacia aquellos que son capaces de manejarlos.

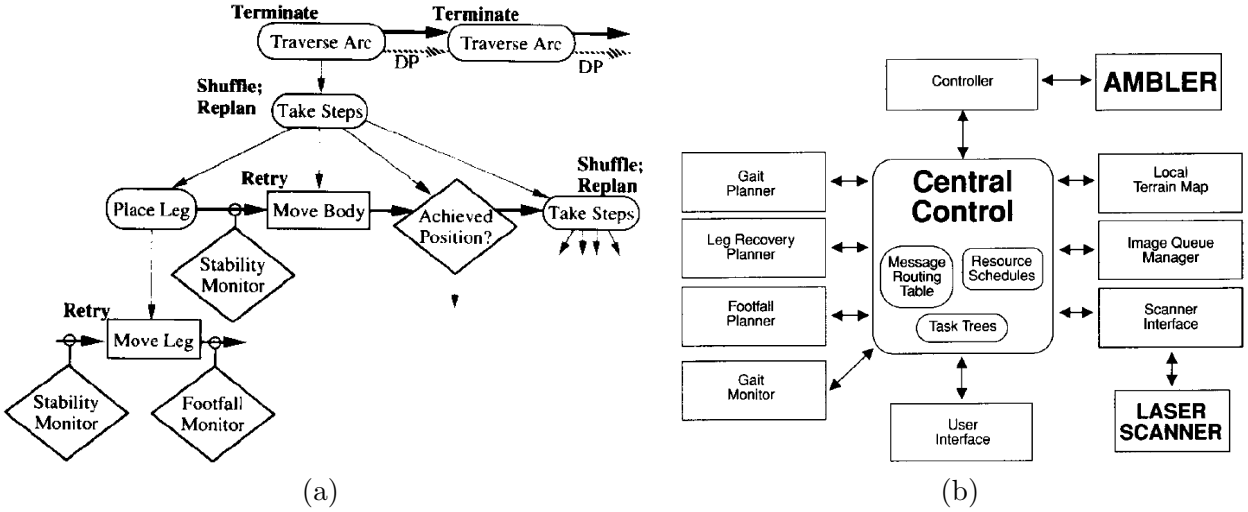


Figura 2.18: *Árbol de tareas en cierto instante para el robot Ambler (a) y la arquitectura de control como colección de módulos (b).*

El intercambio de mensajes entre la red de módulos va construyendo y modificando un *árbol de tareas* dinámico, como el mostrado en la figura 2.18(a). Este árbol está constituido por nodos que pueden ser de tres tipos, en correspondencia con el envío de mensajes homólogos: nodo objetivo (rectángulos redondeados en la figura 2.18(a)), nodo comando (rectángulo) y nodo monitor (rombo). Los nodos de objetivos se pueden expandir recursivamente en más subárboles, hasta la profundidad que sea necesaria, mientras que los comandos y los monitores son nodos terminales. El árbol de tareas representa el plan actual de acción y percepción que tiene el robot para desenvolverse por su entorno. Este árbol se construye a medida que los módulos emiten mensajes que provocan la inserción de nuevos nodos. Cuando el mensaje es tratado, su nodo correspondiente se elimina del árbol. El flujo de mensajes marca el estado actual del árbol y éste determina tanto qué módulos gobiernan la actuación en el instante presente como cuáles lo harán previsiblemente en un futuro inmediato según qué condiciones.

Dentro del espíritu deliberativo, TCA proporciona los mecanismos para implementar la descomposición de tareas en subtareas y secuencias de comportamientos (nodos de objetivos, comandos). De modo general, permite materializar restricciones temporales entre los distintos módulos, bien sea para secuenciar actuaciones, percepciones y planificaciones, como para que puedan realizarse en paralelo; y bucles de realimentación percepción-actuación hasta que se cumple tal o cual condición objetivo. Esto se consigue con la entrega condicionada de mensajes, que pueden llevar asociadas restricciones de tiempo. La descomposición en sí de cierta tarea (*goal message*) en subtareas corre a cargo del módulo correspondiente, no la resuelve TCA. Sin embargo, la arquitectura obliga a que esa descomposición tenga la forma de nuevos mensajes TCA, es decir, nuevos mensajes de objetivo, nuevos monitores o comandos, ya sean con o sin restricciones temporales.

Dentro del espíritu reactivo, la arquitectura proporciona herramientas para monitorizar la ejecución (nodos monitor), percepción orientada a la tarea y un mecanismo de excepciones (mensajes de excepción). Los monitores chequean una condición por deseo explícito de cierto módulo, que ya tiene previsto que esa situación puede darse. Por ejemplo, cierta situación sensorial o si la tarea en curso ha terminado correctamente. Hay de varios tipos: los que se invocan una sola vez, los que se invocan

periódicamente y los que se activan ante tal o cual evento. El monitor puede provocar la inserción de nuevos nodos en el árbol como reacción a la condición que chequean. Los monitores permiten realizar bucles de realimentación de alto nivel como una secuencia de tres pasos por iteración: un mensaje de percepción, un comando y un monitor que comprueba si se ha satisfecho la condición objetivo final. Si no se ha satisfecho se expande un nodo de objetivo que representa otra iteración del bucle.

Otra herramienta reactiva distinta son las excepciones, que surgen cuando un módulo percibe cierta condición de emergencia y decide emitir un mensaje de excepción. El mecanismo de TCA permite asociar a cada nodo varios manejadores de expeciones. Cuando un mensaje de este tipo surge al procesar cierto nodo entonces se sube hacia arriba en la jerarquía del árbol de tareas buscando un manejador para él. El primero que se encuentre tratará de procesarlo, materializando la respuesta del sistema a tal eventualidad. Si no es capaz de resolverlo vuelve a emitir el mensaje hacia más arriba.

Esta arquitectura ha sido utilizada en multitud de robots, entre los que destacan Ambler y Xavier [Simmons *et al.*, 1997b]. Ambler es un robot de exteriores con patas, diseñado para caminar por terrenos escabrosos. Xavier es un robot de interiores con ruedas capaz de entregar correo en entornos de oficina. Hay un módulo que planifica sus movimientos desde un mapa métrico y topológico del entorno, conociendo la posición actual del robot, la cual también se estima [Simmons y Koening, 1995; Koening *et al.*, 1996]. Este planificador hace que el robot efectivamente tienda a ir hasta el destino. Hay un módulo reactivo que elabora los comandos finales que se envían a los motores. Existen varias versiones de este módulo, que ha ido mejorándose con el tiempo [Simmons, 1996; Ko y Simmons, 1998; Benayas *et al.*, 2002]. Cada una de ellas acepta consignas que vienen del módulo planificador anterior, pero también tiene en cuenta los obstáculos del entorno. En el funcionamiento normal no hay obstáculos demasiado cerca y el robot sigue la consigna del planificador, lo que le permite llegar a su punto destino aunque esté relativamente lejos. No obstante, si aparece un obstáculo próximo, entonces el módulo reactivo obvia la consigna de alto nivel y el objetivo prioritario es no chocar contra el obstáculo. A la hora de elegir entre los distintos comandos alternativos se considera que el robot no se desvíe mucho de la consigna y el mantenimiento de distancias de seguridad.

La representación del entorno necesaria en este módulo reactivo es la última imagen sónar del entorno. Un monitor chequea si se ha desviado mucho de la consigna, por si es necesario replanificar. Otro monitor chequea si se ha llegado al punto destino.

2.5.2. RAP de Firby y la arquitectura 3T

La necesidad de replanificar ante la ocurrencia de algún imprevisto se hizo patente ante las limitaciones de los sistemas deliberativos clásicos. Con el fin de agilizar estos sistemas sin renunciar a las ventajas de la planificación aparecieron conceptos como la *planificación reactiva*, la monitorización continua de los planes, o el intercalado de planificación y ejecución.

Otra muestra relevante de sistema híbrido es el trabajo de James Firby [Firby, 1987]. Su planificación continua huye de un plan completo de gran tamaño y se centra en un plan fraccionado en pequeñas unidades que incluyen su replanificación y su monitorización. Estas unidades se llaman *paquetes de acción reactiva* (*Reactive Action Packages*, RAP), que consiguen cierta meta y para ello tienen preprogramadas varias recetas que llevan al objetivo en pasos discretos. Estas recetas son secuencias o redes de tareas parcialmente ordenadas. Cada tarea puede ser una acción primitiva o bien la invocación de otro RAP, lo cual abre la posibilidad de tenerr una jerarquía.

Además estos módulos RAP incluyen una monitorización continua sobre si han conseguido el objetivo o no (*goal check*), e incluso si son aplicables o no (*validity check*). Cada unidad elije una receta de actuación y monitoriza su progreso. En caso de que falle, ella misma elige otra de las que tiene y prueba con ella. Esta capacidad de replanificación a bajo nivel le otorga flexibilidad al sistema. Además no consume mucho tiempo pues estrictamente no recalcula ningún plan, sino que ejecuta otra alternativa que ya estaba precompilada. Si todas las alternativas que tiene preprogramadas fallan entonces reporta fallo global de ese RAP.

El sistema global original se articula como un conjunto de RAPs activos, que se extraen de una biblioteca de RAPS, y se almacenan en una cola de ejecución. Un intérprete maneja esa cola para determinar en todo momento cual de ellos realmente toma el control, haciendo las veces de árbitro. Los trabajos posteriores han mejorado el sistema para tratar con procesos continuos [Firby, 1992;

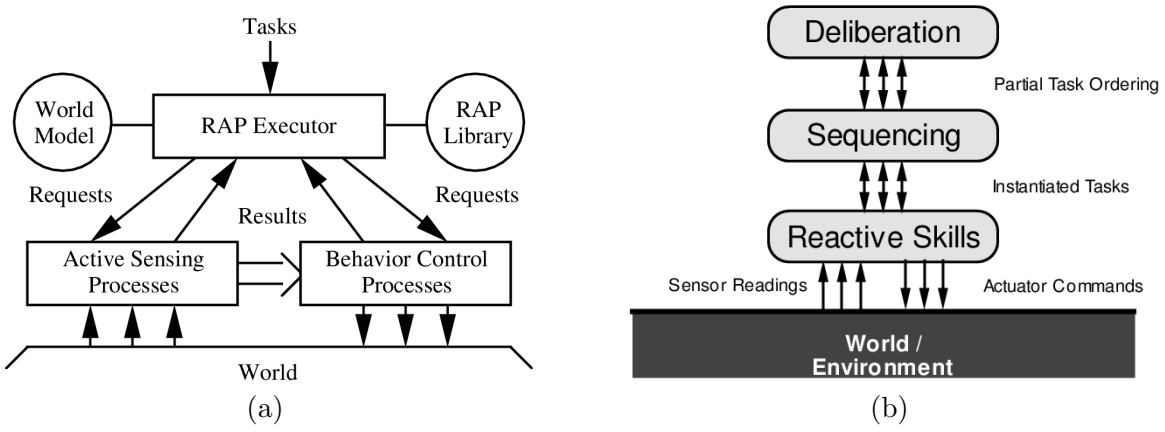


Figura 2.19: *RAP de Firby (a) y arquitectura 3T (b)*

Firby, 1994]. En ellas se ha establecido los RAP como unidades simbólicas sobre las que por un lado se puede planificar, y por otro lado controlan procesos continuos en la realidad. El sistema en este caso incluye una biblioteca de muchos RAPs que se toman como bloques unitarios, cuya activación decide un planificador de alto nivel. Tal y como señala la figura 2.19(a) el planificador entrega una tarea al ejecutor RAP, que echa mano de las unidades que tiene en su biblioteca de RAPs para materializar las tareas.

Esta línea ha desembocado en las arquitecturas de tres capas como la 3T [Bonasso *et al.*, 1997], que distribuye el control en una jerarquía fija de 3 niveles de abstracción, los cuales funcionan de modo concurrente y asíncrono. En ella han confluido varios trabajos como el de Firby con RAP, el de Gat con ATLANTIS [Gat, 1992] y otros. En esencia se tiene una capa deliberativa pura, una capa reactiva pura y otra intermedia que sirve de nexo de unión entre ambas.

Tal y como aparece ilustrado en la figura 2.19(b), la capa superior incluye deliberación sobre representaciones simbólicas de estado, por ejemplo en lógica proposicional. En ella hay un planificador que delibera en profundidad sobre metas, recursos y restricciones temporales, y genera planes compuestos de tareas. La capa intermedia, llamada *secuenciador*, recibe dichas tareas y tiene una biblioteca de recetas que describen cómo ejecutarlas o conseguirlas. Tiene la capacidad de manejar situaciones rutinarias correctamente, pero no de organizar nuevas secuencias o considerar las implicaciones de ciertas acciones. Este funcionamiento se ha materializando empleando directamente el ejecutor RAP de Firby. Para conseguir las tareas, activa y desactiva conjuntos de *habilidades*.

Esas *habilidades* constituyen el nivel reactivo. Cada una de ellas es una rutina continua que consigue o mantiene cierto objetivo en cierto contexto. Cada habilidad se representa explícitamente en el sistema por una especificación de entrada/salida, una función que va a ejecutar continuamente y rutinas de inicialización, activación y desactivación. Su función genuina es la que se ejecuta constantemente para recalcular su salida desde sus entradas actuales. Ofrecen un funcionamiento continuo, transformando el estado inicial en el deseado con un control continuo de motores o generando una interpretación continua de los sensores. La comunicación interna de estas habilidades con el secuenciador se produce a través de habilidades especiales llamadas *eventos*. Los *eventos* señalizan por ejemplo, que algún conjunto de habilidades completó su trabajo, cuál falló, o que el estado del mundo ha cambiado. Esta información puede ser muy importante para sincronizar el despliegue del plan con su desarrollo incremental.

Este sistema combina de modo asíncrono las tres capas. De este modo la parte deliberativa, que típicamente consume mucho tiempo, no bloquea la respuesta del sistema. En su lugar va modulando a las inferiores, para orientarlas hacia los objetivos, con lo cual el sistema no se atasca. Además la replanificación rápida y monitorización incluidas en RAP dotan de mayor flexibilidad al sistema.

La descomposición que propone 3T se ha implementado en muchos robots de la NASA. En la misma línea un trabajo similar es el de Volpe [Volpe *et al.*, 2001], que plantea una arquitectura de sólo dos capas llamada CLARAty. Tiene una capa superior de decisión, que engloba al planificador y al secuenciador de 3T, y otra funcional que se enfrenta a los sensores y actuadores reales. Esta

arquitectura argumenta la falta de fronteras nítidas entre las capas de 3T y propugna un acercamiento del planificador a la capa reactiva, de modo que tiene acceso más directo a información en ese nivel. Además, introduce la idea de varios niveles de granularidad en cada capa, con lo cual materializa distintos horizontes de planificación en la capa superior.

2.5.3. Arquitectura DAMN de Rosenblatt

La principal crítica a la subsunción de Brooks es la imposibilidad de alcanzar soluciones de compromiso entre varios comportamientos, lo cual es extensible a todos los métodos de coordinación arbitrados en los que hay un único ganador. La funcionalidad de los niveles de competencia perdedores en esa competición no se aprovecha en absoluto en esa iteración. Ahondando un poco en esas limitaciones Rosenblatt y Payton proponen un nuevo método de selección de acción cuando hay múltiples comportamientos [Rosenblatt y Pyton, 1989]. Este nuevo método obliga a cada comportamiento a explicitar sus preferencias sobre todas las posibles acciones. Finalmente, utilizando una votación ponderada entre todos los comportamientos, aquella acción que reune más votos es la que finalmente se comanda a los actuadores.

Cuando los comportamientos tienen estas salidas, que expresan predilección o no por todas las posibilidades de actuación, se llaman *de grano fino*. Estos contrastan con las salidas normales, de una recomendación para la acción según ese comportamiento. El inconveniente de esta salida única es homólogo a la pérdida de información señalada por Agre [Agre y Chapman, 1990] y Payton [Payton, 1990] en los planes en cuanto abstracciones con un único curso de actuación. La nueva concepción de Rosenblatt soluciona esa limitación pues mantiene una información más completa del comportamiento. Por ejemplo permite señalar preferencias de actuación positivas o prescriptivas, como negativas o proscriptivas.

Además este mecanismo facilita la adopción de actuaciones de compromiso, lo cual resulta muy útil. Por ejemplo, la figura 2.20(b) corresponde a la fusión de dos comportamientos. El primero (arriba) trata de mover el volante de un coche autónomo para sortear obstáculos, y el segundo (abajo) de girarlo para mantener al auto centrado en la carretera. En la zona intermedia se tienen discretizadas las posibles acciones: girar el volante suave o completamente hacia la izquierda o la derecha y mantenerlo recto. Los círculos rellenos expresan apetencia por la actuación, mayor a mayor tamaño. Los círculos huecos expresan aversión, igualmente proporcional al tamaño. De este modo el comportamiento que sortea obstáculos ha detectado uno en el medio, por lo que expresa su negativa a mantener el volante recto, y su predilección por los giros, mejor cuanto más pronunciado, con indiferencia de si se realiza por la izquierda o por la derecha. El comportamiento que centra al robot en la carretera recomienda un giro suave a la izquierda, quizá por la proximidad del coche al borde derecho de la carretera. La actuación final es un giro firme a la izquierda que satisface en mayor medida las preferencias combinadas.

Esta técnica se materializó en la arquitectura DAMN (Distributed Architecture for Mobile Navigation), descrita en la tesis de Rosenblatt [Rosenblatt, 1997], que se centra en el problema de la navegación. En ella el control se distribuye en un grupo de comportamientos y un árbitro de comandos. Rosenblatt reconoce la necesidad del razonamiento deliberativo, así como de reactividad frente al entorno para que un sistema pueda satisfacer sus objetivos, y aboga por combinarlos en su arquitectura en el espacio de comandos. En este sentido los posibles planes tienen interpretación muy próxima a la de los planes internalizados de Payton [Payton, 1990].

La tendencia hacia los objetivos se traduce en ciertos comportamientos *deliberativos*, tal y como muestra la figura 2.20(a). Del mismo modo también existen comportamientos estrictamente reactivos. Más que imponer una jerarquía de niveles, DAMN tiene una estructura plana en la cual los módulos participan en igualdad de condiciones, únicamente sesgados por los pesos de los votos.

Cada módulo utiliza las representaciones internas simbólicas que le convengan para realizar bien su labor. Un módulo de DAMN puede ser desde reactivo puro hasta implicar el manejo y manipulación de mapas. De hecho, en la arquitectura se incluyen procesos perceptivos que fusionan los datos sensoriales en un modelo centralizado del mundo sobre el cual se realizan los razonamientos geométricos típicos del problema de navegación. Esto le diferencia enormemente de los sistemas basados en comportamientos.

Las votaciones son asíncronas, de manera que los módulos no han de estar sincronizados ni ejecutándose al mismo ritmo. El árbitro reevalúa varias veces por segundo cual es la mejor acción a

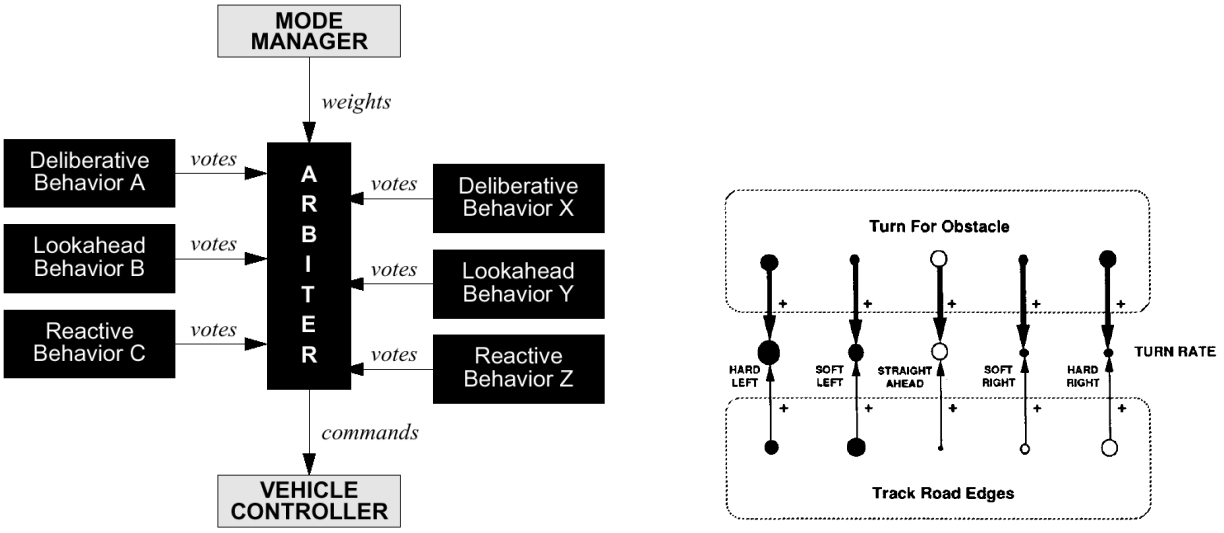


Figura 2.20: *Diagrama de la arquitectura DAMN (a) y ejemplo de fusión de comandos en ella (b).*

ejecutar en ese momento. Este árbitro es el punto donde se centralizan las preferencias de los módulos de la arquitectura, que por lo demás es perfectamente distribuida.

La actividad secuencial se consigue con el manejador de modo (*mode manager*) que varía los pesos de los votos asignados a cada módulo, controlando de este modo la influencia de cada uno de ellos en el comportamiento final.

Dentro de un SBC esta técnica de fusión de comandos se puede usar como mecanismo de coordinación entre diferentes comportamientos reactivos. Sin embargo, el uso que resaltaremos aquí es la combinación de la tendencia deliberativa de un sistema, orientada a objetivo, con la tendencia reactiva, orientada a entorno. Lo hemos incluido aquí y no en los sistemas basados en comportamientos porque Rosenblatt admite la necesidad de deliberación y crea un modelo explícito del mundo.

El ejemplo típico de combinación entre la tendencia reactiva y la deliberativa es el que hemos visto en la figura 2.20(b). Por una parte el robot quiere acercarse al destino siguiendo la carretera (tendencia deliberativa), por otra si hay un obstáculo cercano que no estaba contemplado en el mapa sobre el que se planificó la trayectoria el robot debe evitarlo y para ello lleva un comportamiento reactivo que evita el choque. Con la arquitectura conexionista de Rosenblatt se consigue un movimiento de compromiso entre ambas tendencias, sorteándo el obstáculo por el lado más útil para el acercamiento al destino.

El mecanismo de selección de acción de esta arquitectura guarda bastante paralelismo con otras técnicas de coordinación que fusionan comandos como la lógica borrosa y otras técnicas de votación. De hecho en [Yen y Pfluger, 1995] se realiza una extensión de DAMN insertándola en un marco borroso. El método de votaciones también ha sido explorado por Paolo Pirjanian [Pirjanian *et al.*, 1998], aunque éste analiza nuevas variantes a la hora de asignar distinto peso a los distintos comportamientos y exigir diferentes umbrales a los ganadores.

2.5.4. Arquitectura Saphira de Konolige

Una arquitectura muy exitosa en robots de interiores es Saphira [Konolige y Myers, 1998], creada por Kurt Konolige en SRI International. La arquitectura software asociada ha sido integrada y difundida en la plataforma de los robots comercializados por Activmedia. Recientemente se ha reescrito en C++ con el nombre de ARIA bajo licencia GPL. La orientación cliente-servidor de esta arquitectura software ha facilitado su uso en robots tan distintos como Flakey del SRI, los modelos Kephera de K-Team, los B21 de iRobot o los Pioneer de ActivMedia.

En cuanto a la percepción, Saphira construye y actualiza constantemente una representación interna llamada *espacio perceptivo local*, en la cual fusiona información sobre el entorno próximo del robot. Este modelo está compuesto por perceptos, llamados *artifactos*, que representan internamente

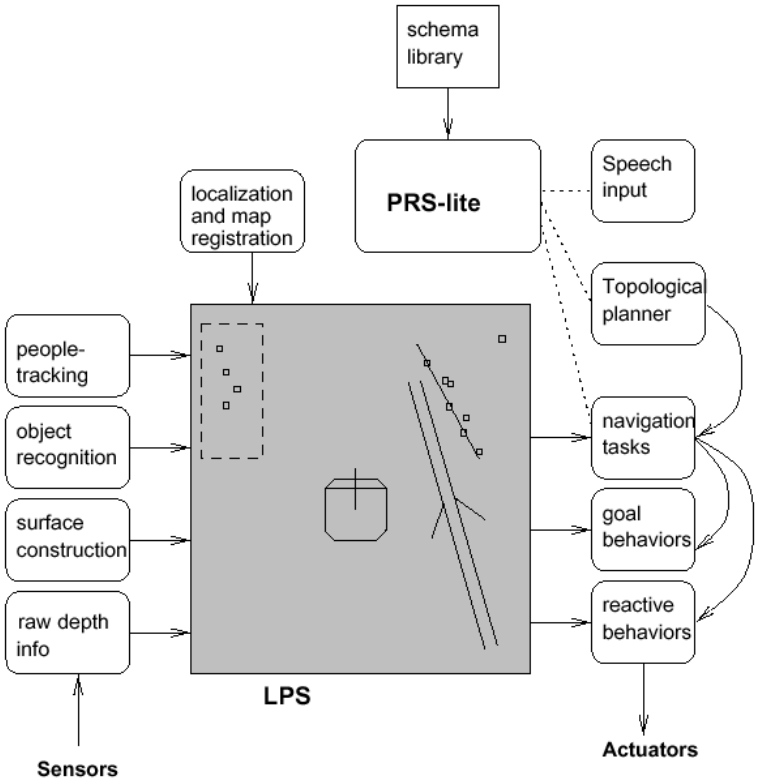


Figura 2.21: *Arquitectura Saphira*

algún objeto real y que están en permanente correspondencia con los objetos homólogos de la realidad. Estos perceptos también pueden representar alguna configuración útil para guiar un comportamiento o algún objetivo instanciado. Varios procesos concurrentes mantienen ese modelo coherente, con sus símbolos *anclados* en el mundo real (*perceptual anchoring*), tal y como muestra la figura 2.21.

Este espacio perceptivo tiene estructura. Sobre los datos sensoriales crudos se construyen las *características*, que son descripciones más elaboradas y abstractas. Por ejemplo, los datos de proximidad de los sensores pueden conformar un segmento recto. Además, estas características pueden agruparse en hipótesis de objetos, que finalmente se integran en lo que Konolige llama *artefactos* como una pared, un pasillo, etc.

Una de las ventajas de mantener representación interna es la persistencia. Por ejemplo, se puede seguir a una persona aunque desaparezca temporalmente de los sensores porque dobla una esquina, y se puede seguir una pared aunque en cierto momento no se aprecie por el vano de alguna puerta. Otro argumento a favor de esta representación simbólica explícita es que simplifica la generación del comportamientos porque desacopla la parte del control de la parte de interpretar sensaciones ruidosas.

En cuanto al control, Saphira presenta dos niveles. En el nivel inferior dispone de unos comportamientos básicos escritos como controladores borrosos, es decir, como una colección de reglas borrosas con antecedentes y consecuentes. Estos comportamientos materializan las reacciones básicas, bien a los estímulos del entorno, bien a los objetivos instanciados (por ejemplo dirección de avance). Además, se engarzan con el espacio perceptivo local, de manera que los antecedentes de sus reglas hacen referencia a información almacenada en ese espacio, que se traduce a variables borrosas. La actuación efectiva se calcula promediando de modo borroso las salidas individuales de todos los comportamientos. Ese promedio es una combinación lineal, donde los pesos de cada comportamiento dependen de su prioridad, que es fija, y su adecuación a la situación actual, que se calcula con una reglas de contexto (*context-dependent-blending*).

La figura 2.22 muesta un ejemplo de la combinación de dos comportamientos, en la que se ve como el cambio de contexto va variando el peso real de cada uno de ellos en la determinación de la actuación efectiva. Esta fusión permite tener en cuenta la tendencia hacia los objetivos incluso durante las maniobras reactivas.

Figura 2.22: *Arquitectura Saphira*

En el nivel superior Saphira se encarga de relacionar los comportamientos básicos con la consecución de metas intermedias y objetivos del robot. En este nivel se determina la activación y coordinación de los diferentes comportamientos básicos. El planificador PRS-Lite se encarga de ello, permitiendo la descomposición jerárquica de tareas en subtareas y ofreciendo un rico repertorio de mecanismos para especificar secuencias, despliegues condicionales de planes, ordenación de subobjetivos y de actuaciones.

PRS-Lite permite definir los *esquemas de actividad*, que son las recetas para conseguir cierto objetivo declarado. Estas recetas separan la planificación de la ejecución y las fraccionan en pequeñas unidades, flexibilizando su imbricación mutua. Mientras que la ejecución significa la activación de cierto comportamiento básico, la planificación conlleva la modificación de una estructura de intenciones en la que el sistema almacena el desarrollo y las relaciones entre los subobjetivos de la actividad. Esta estructura tiene forma de grafo, de manera que ciertos nodos se expanden jerárquicamente en otro subgrafo.

Las primitivas básicas de este nivel superior son la ejecución y la planificación de algún subobjetivo, pero el sistema también cuenta con primitivas para chequear condiciones o detener un subobjetivo hasta que se presente cierta situación. El despliegue de un esquema de actividad permite la activación de varias subobjetivos en paralelo, y va combinando el desarrollo de estas primitivas de modo condicional, en secuencia estricta, como nodos en paralelo, etc. Esta combinación va conformando la estructura de intenciones.

2.5.5. Ideas principales

El paradigma híbrido, dentro de su heterogenidad, combina los principios de la escuela deliberativa y de la reactiva, tratando de complementarlos y conseguir de esta manera comportamientos más robustos y complejos. La parte reactiva se ha mantenido muy próxima a los sensores y actuadores reales, dotando al sistema de flexibilidad. Sin embargo, esta componente se no escala hasta completar todo el sistema, como en los sistemas basados en comportamientos, sino que tiene por encima una capa deliberativa que modula su funcionamiento. Esta parte deliberativa incluye la representación explícita de objetivos y la planificación como motor final del comportamiento. Por ejemplo, [García-Alegre *et al.*, 1993] incluye un planificador para el nivel abstracto y unos comportamientos reactivos borrosos para tratar con los actuadores reales. En el mismo sentido, Chatila propone una arquitectura en [Chatila, 1995] con dos niveles diferenciados, el decisional y el funcional, correspondientes a la parte deliberativa y reactiva respectivamente.

La separación en parte deliberativa y reactiva atiende también a un principio de ingeniería software, que tiende a agrupar las cosas similares juntas. Así, a la hora de incorporar una nueva tarea o componente en una arquitectura híbrida hay que considerar si maneja información numérica que varía continuamente, o si la información que utiliza tiene una dinámica más lenta y es más abstracta.

En cierto modo el paradigma híbrido mejora la ejecución de planes típica de los sistemas deliberativos. La cual deja ahora de ser una ejecución ciega, en lazo abierto, fruto de una excesiva confianza en los modelos internos y en la secuencia planificada. La planificación a largo plazo es útil cuando la mayoría de las variables que intervienen están bajo nuestro control o son predecibles (por eso ha mostrado sus mayores éxitos en entornos estáticos), que no suele ser el caso en escenarios reales. En los sistemas híbridos se contempla que el mundo puede no seguir el curso planeado. Por ejemplo, las propias actuaciones del robot pueden no tener el resultado previsto, o el entorno puede cambiar impredeciblemente debido a la actuación de otro agente. Por ello la ejecución se deja en manos de

algún comportamiento reactivo que es capaz de responder con flexibilidad ante desviaciones ligeras, y cuya ejecución continua materializa el subobjetivo deseado. La planificación se mantiene en la parte deliberativa del sistema, mientras que la ejecución se encarga a la parte reactiva.

Además de esta separación entre planificacion y ejecución, ambas tienden a intercalarse en el tiempo [Haigh y Veloso, 1996]. Esta imbricación elimina los retardos típicos necesarios para construir grandes planes, y permite no realizar esfuerzos computacionales inútiles en grandes planes que pueden dejar de ser válidos porque la realidad cambió inopinadamente. Tanto la planificación como la ejecución se fraccionan en unidades o pasos, de manera que su progreso se puede monitorizar explícitamente. Con ello se logra una imbricación más armoniosa y efectiva. Un buen ejemplo de esta idea es la arquitectura TCA de Simmons.

La inclusión del nivel reactivo permite subir el mínimo grado de abstracción que los planificadores manejan. El nivel de detalle que la parte deliberativa debe considerar puede subir, justo hasta donde la parte reactiva puede digerir, entender y materializar. Si en los sistemas clásicos bajaban hasta los actuadores, en los híbridos pueden limitarse a la activación de tales o cuales comportamientos reactivos y a sus parámetros. La parte reactiva ya se encarga de traducir eso a acciones concretas de los actuadores. Un ejemplo que ilustra los diferentes interfaces entre capas y sus diferentes niveles de abstracción es la arquitectura SSS de Connell [Connell, 1992]. En ella se tienen tres capas: la simbólica, la de subsunción y la de control. La capa simbólica se limita a activar comportamientos y parametrizarlos. La capa de subsunción entrega unas consignas de referencia a la capa de control que realmente manipula los actuadores para conseguirlas.

En cuanto a la percepción, los sistemas híbridos presentan de nuevo características tanto de la escuela deliberativa como de la reactiva. En general, cada capa tiene su propia representación, de manera que la parte perceptiva es de nuevo mixta. Por un lado, admiten modelos simbólicos del entorno, como en Saphira, o necesitan de modelos centralizados del mundo para poder deliberar sobre ellos, como en 3T. Por otro lado, la parte reactiva tiene distribuida la percepción en cada comportamiento, muy orientada a tarea. Además, esta percepción suele estar muy pegada a los datos sensoriales y ser no simbólica.

Una variante la constituyen los comportamientos básicos en Saphira que reaccionan ante ciertos estímulos simbólicos anclados en la realidad, o incluso ante otros símbolos de consumo interno como los artefactos. Para aumentar en autonomía la parte deliberativa suele incluir sus módulos constructores de representación, por ejemplo los constructores de mapas, o incluso apoyarse en la parte perceptiva del nivel reactivo.

Una familia de arquitecturas híbridas que no hemos descrito son las basadas en pizarra. En ellas, la pizarra se utiliza como punto central para la comunicación entre diversos módulos. Por ejemplo, la arquitectura ABC^2 de Matellán [Matellán y Borrajo, 1998] utiliza la pizarra y permite la cooperación entre distintos agentes híbridos. A grosso modo la arquitectura consta de dos capas, en la inferior se acumulan comportamientos básicos que llama *habilidades*. En la capa superior tiene una *agenda* que almacena las acciones pendientes de ejecución. La mera aparición de un acto en la agenda supone la motivación del robot para ejecutarlo, para conseguir el objetivo de ese acto. Los actos ejecutivos consisten en lanzar una habilidad. Unos heurísticos, programados como reglas borrosas, examinan esa agenda y determinan la activación de uno u otro comportamiento básico, eligiendo en cada ciclo de control cuál es la acción que debe ejecutarse.

2.5.6. Limitaciones

Los sistemas híbridos son los de mayor éxito en la actualidad. Este paradigma pretende ampliar el rango de aplicaciones automatizables, superando las limitaciones propias del enfoque reactivo y deliberativo. Su propuesta es la combinación de las ventajas de ambos en la misma arquitectura. Típicamente se ofrece un compromiso modulable, que se apoya más en la parte deliberativa o en la reactiva dependiendo de la aplicación. Uno de los riesgos de esta combinación es ajustar correctamente ese balance y evitar con ello las limitaciones de ambos enfoques.

Mientras que los sistemas deliberativos y los basados en comportamientos llevan detrás toda una teoría sobre el comportamiento inteligente, los sistemas híbridos son más una solución ad hoc de un problema que una teoría subyacente de la inteligencia. Más que una escuela propiamente dicha,

parece un corta y pega ingenieril de soluciones parciales. No presenta una solución homogénea al comportamiento. Esta heterogeneidad que puede dificultar el desarrollo de las aplicaciones.

2.6. Modelos biológicos

Los sistemas reactivos puros y los basados en comportamientos tienen fuertes enlaces con la biología [Arkin, 1998]. Un ejemplo son los trabajos de Arbib sobre esquemas, los cuales se han apoyado en ciertos estudios neurofisiológicos. En ellos se armonizan los modelos computacionales propuestos con los experimentos neuronales sobre ranas reales, y con la separación funcional y fisiológica encontrada en su tejido nervioso [Arbib y Liaw, 1995; Corbacho y Arbib, 1995]. Además de estas relaciones, en los últimos años ha habido un creciente número de trabajos robóticos que buscan inspiración en la biología [Tyrrell, 1993b; Duchon *et al.*, 1998; Kuc y Barshan, 1992; Mataric, 1991; Nehmzow, 1995], entendida en sentido laxo. Consideramos dentro de ella a campos tan variados como la neurofisiología, la ecología, psicología ecológica y en particular la etología. Precisamente dedicaremos esta sección a describir la influencia de esta última, dado que su aportación a la robótica nos parece más intensa y prometedora que la del resto.

La *etología* es la ciencia que estudia el comportamiento de los animales. La robótica actual busca nuevas ideas en el estudio del comportamiento animal, quizá por las limitaciones encontradas en las técnicas desarrolladas hasta el momento, genuinamente robóticas, para generar comportamientos artificiales complejos. Con las arquitecturas actuales la comunidad robótica no es capaz de dar un salto cualitativo en complejidad y calidad en cuanto al comportamiento autónomo. En contraste, los animales consiguen comportamientos enormemente complejos y satisfactorios en entornos dinámicos y reales, donde la robótica actual fracasa. Al fin y al cabo, un animal perfectamente adaptado a su entorno supone un modelo a seguir, pues sus mecanismos causales de comportamiento generan todo un repertorio de conductas que le permite sobrevivir. Por ejemplo, Mallot hace en [Mallot y Franz, 1999] un repaso extenso de mecanismos de navegación desarrollados por animales. Con él se replantea las técnicas de movimiento utilizadas en robots y la información necesaria para sustentarlas.

En este sentido se muestran útiles las arquitecturas generales propuestas por etólogos para explicar algunas conductas animales [Lorenz, 1978] y las claves perceptivas que se han identificado en ciertos comportamientos [Gibson, 1977]. Por ejemplo, los trabajos de Baerends [Baerends, 1976], Tinbergen, Lorenz, etc. han buscado modelos que reflejen los mecanismos reales de generación de comportamiento, modelos que abandonan explicaciones finalistas y buscan las causas materiales de las conductas. Este caracter causal, etiológico, hace muy útil a la etología, pues es esta perspectiva la que interesa a la robótica para poder replicar estos mecanismos. Esta relación entre robótica y etología es ventajosa para ambas disciplinas, manteniendo cada una su idiosincrasia. Por un lado, la robótica recoge los modelos y el conocimiento acumulado por etólogos, como nuevas vías para superar sus propias limitaciones. Por otro lado, los etólogos se encuentran por primera vez con la posibilidad de simular y validar sus teorías usando robots.

Hay gran analogía entre la robótica y la etología, pues ambas abordan el problema del comportamiento. Un análisis sobre este paralelismo se pueden encontrar en [McFarland y Bösser, 1993] y [Hallam y Hayes, 1992], que presentan unos puntos comunes como sensores, actuadores, objetivos, navegación, etc. y sus características genuinas en cada disciplina. Amén del estudio del comportamiento como gran coincidencia, hay matices diferenciadores entre ellas. Por ejemplo, la etología lo aborda desde una perspectiva descriptiva y analítica, centrándose en los mecanismos ya existentes en los animales. En cambio la robótica lo aborda desde una perspectiva sintética, tratando de aprovechar esos modelos para implementarlos sobre el *cuerpo* de los robots, con sus sensores artificiales y sus peculiares actuadores.

Los mecanismos generadores de comportamiento en los animales son responsables de la supervivencia de la especie, y por ello su utilidad está más o menos garantizada. El proceso evolutivo ha ido diseñandolos por ensayo y error, programando filogenéticamente las conductas de los animales que existen en la actualidad, filtrando los animales que no reaccionaban correctamente ante estímulos significativos o no desarrollaban los comportamientos adecuados. En este sentido los programadores de robots nos encontramos en el mismo lugar que la evolución. Con la diferencia de que la evolución

ha ido programando de modo incremental, construyendo sobre los mecanismos ya existentes, mientras que nosotros podemos diseñar desde cero el interior del robot.

Actualmente los comportamientos animales mejor comprendidos son los innatos, por lo que gran parte del corpus etológico sirve de referencia para esta tesis, que explícitamente deja fuera el aprendizaje. Por ello en esta sección nos centraremos en los modelos más o menos detallados que se han desarrollado para explicar cómo se generan, activan y coordinan los comportamientos instintivos. Ellos son característicos de cada especie, innatos a todos sus individuos y sin la parte aprendida. El peso de estos comportamientos en los animales sencillos como peces y pájaros es prácticamente total. Incluso los animales superiores como los mamíferos que presentan ciertas componentes aprendidas en sus conductas también tienen una parte innata fundamental.

Si pensamos que la inteligencia tipo insecto ya se conseguía en robots basados en comportamientos, básicamente como una colección organizada de reflejos, entonces con los estudios etológicos sobre el comportamiento instintivo se apunta a replicar en los robots la flexibilidad de animales más complejos como peces, perros o aves [Arkin *et al.*, 2003; Corbacho y Arbib, 1995; Blumberg, 1997; Arkin *et al.*, 2000]. En estos animales la teoría de los reflejos se queda corta para explicar otros fenómenos y deben explorarse nuevas hipótesis. Quizá estos estudios etológicos suponen un paso más en el largo camino de la comprensión de sistemas más complejos, como el comportamiento humano, donde las partes aprendidas tienen más influencia.

A lo largo de muchos años la investigación etológica ha ido acumulando un vocabulario que resulta adecuado para describir comportamientos y sus mecanismos subyacentes. Por ejemplo, a principios del siglo XX Craig [Vogel y Angermann, 1974] estableció la distinción entre comportamientos apetitivos y comportamientos consumatorios, que ha resultado de gran utilidad. La idea de homeostasis y mantenimiento de ciertas variables en un rango aceptable.

En esta sección haremos una descripción de los trabajos de Nikola Tinbergen y Konrad Lorenz, dos etólogos que sentaron las bases de nuestra comprensión sobre el comportamiento instintivo en los animales. Son trabajos antiguos pero fundacionales. Dado el carácter causal de sus hipótesis, éstas sirven de inspiración y refrescante punto de vista como mecanismos para generar comportamiento en robots. También veremos otros trabajos significativos como el de Ludlow y Tyrrell, que han tenido una influencia clara en varios mecanismos de selección de acción empleados en robótica.

2.6.1. Tinbergen

Uno de los etólogos más relevantes, cofundador de la disciplina junto con Lorenz, Heinroth y Craig, es Nikolaas Tinbergen. En lo que se refiere a mecanismos generadores de comportamiento Tinbergen argumentó la organización jerárquica de los sistemas nerviosos [Tinbergen, 1950]. Fruto de esta idea central hipotetizó una arquitectura para generar el comportamiento, compuesta de nodos organizados en jerarquía, que se muestra en las figuras 2.23 y 2.24. La arquitectura responde a ciertas observaciones en conductas animales que claramente se escapan a una explicación con la teoría de reflejos. Aparece descrita con más detalle y numerosos ejemplos experimentales en [Tinbergen, 1987].

Como principales argumentos Tinbergen recabó evidencias etológicas y neurofisiológicas que apoyaban su hipótesis. Entre las evidencias neurofisiológicas señala la diferenciación fisiológica de las células nerviosas, y que ciertas pautas motoras completas se provocan estimulando eléctricamente el hipotálamo, como si se activara un centro nervioso, de cierto nivel, de su jerarquía.

Una primera evidencia etológica que justifica esta arquitectura es que el comportamiento depende tanto de estímulos internos como de externos. Esto rompe con la teoría reactiva pura, de asociación situación-acción, descrita en anteriormente pues la misma situación externa provoca un distinto comportamiento, dependiendo de la motivación interna [Arkin *et al.*, 2003; Cañamero *et al.*, 2002; Lorenz, 1978]. Por ejemplo, el comportamiento de lucha o de nidificación en el pez espinoso macho no sólo depende de los estímulos externos adecuados como presencia de un adversario, o la visión de una zona vegetal y temperatura cálida del agua. También depende de la activación del instinto reproductor, que es un factor interno.

Ordenados y conectados en la jerarquía, los centros inferiores de la arquitectura generan respuestas motoras ante ciertos estímulos, mientras que los centros superiores generan predisposiciones hacia ciertos comportamientos y no otros. Por ejemplo, cuando un halcón peregrino se va de caza, tiene cierta

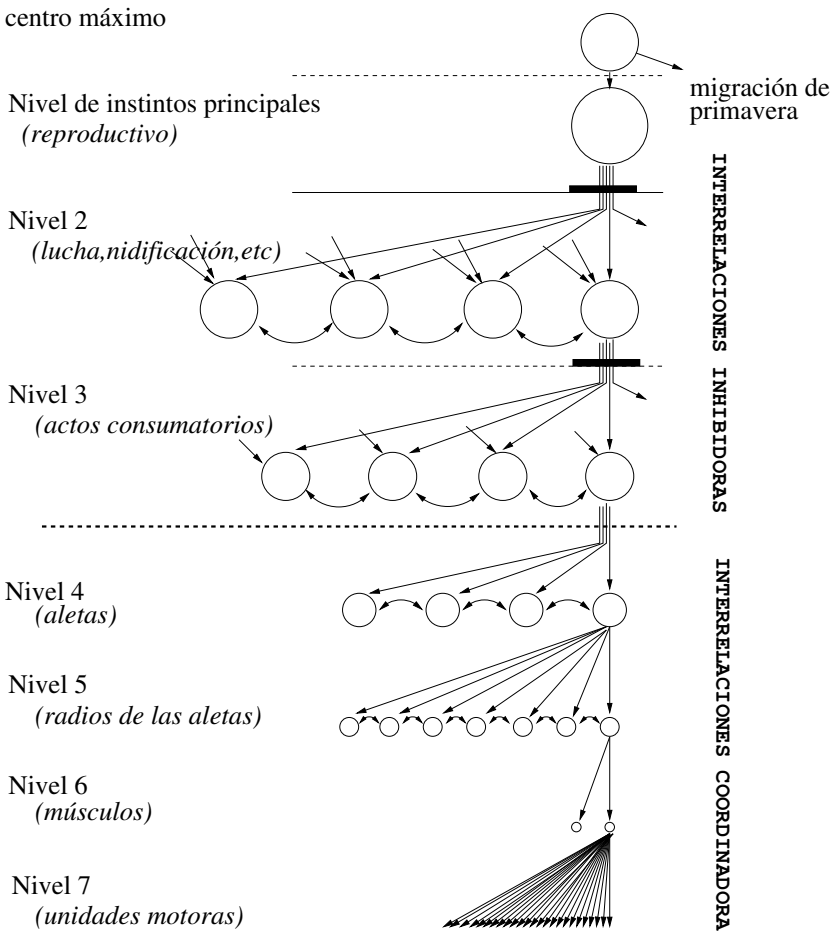


Figura 2.23: *Arquitectura de Tinbergen.*

predisposición a varios comportamientos concretos: caza-conejo, caza-paloma, etc. que requieren de pautas motoras muy diferentes. En este sentido, el halcón no se va a poner a construir un nido, cortejar o dormir, porque no está predispuesto para ello en ese momento. Tiene más querencia a desarrollar unas conductas que otras.

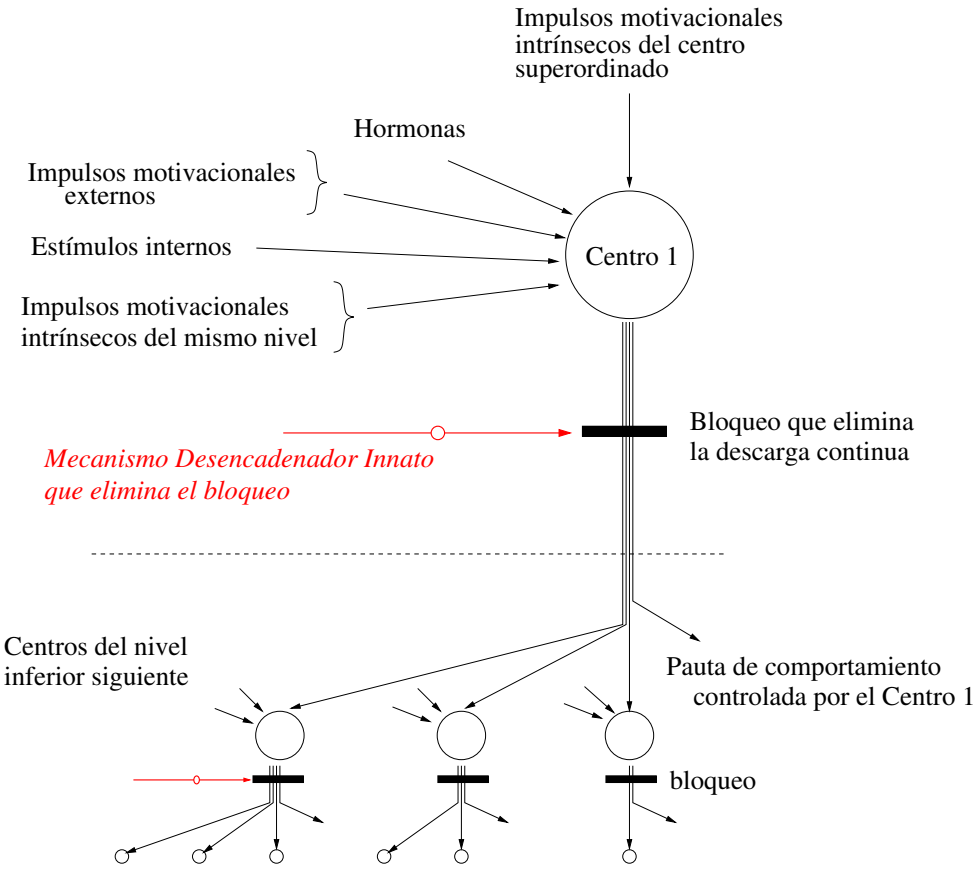


Figura 2.24: *Detalle de un nodo, con estímulos externos, motivación interna y mecanismo desencadenante innato.*

Siguiendo con el ejemplo, parece que la generación de comportamientos obedece a la liberación de un bloqueo más que a un patrón reflejo, de situación acción. Por ejemplo, la presencia de un conejo o una bandada de palomas, dispara todo un repertorio de pautas motoras especializadas. En este sentido, Tinbergen habla de *mecanismo desencadentante innato* al fenómeno que precipita la conducta posterior, y de *estímulos clave* como la llave que libera la motivación interna existente y que desencadena el desarrollo ulterior. Este funcionamiento es coherente con la observación de que la aparición de la paloma, por ejemplo, no provoca el comportamiento consumatorio de cazarla directamente, sino otros comportamientos apetitivos relacionados, como son el acecho, los ataques ficticios para dispersar la bandada y seleccionar la presa, etc..

Así pues, Tinbergen propone una arquitectura jerárquica de nodos donde los superiores almacenan motivación interna, que va fluyendo hacia determinados nodos inferiores. Esta propagación hacia abajo, que es donde están los nodos encargados de los comportamientos consumatorios finales, va fluyendo y conformándose a medida que aparecen estímulos clave que liberan su propagación. La selección de acción viene marcada por los nodos que consiguen activarse, es decir, aquellos cuyo estímulo clave aparece y reunen suficiente motivación interna. Los nodos que no reúnen motivación o cuyo estímulo clave no aparece, permanecen latentes sin provocar actuación o propagación de su motivación. Tinbergen habla incluso de mecanismos de inhibición lateral entre centros del mismo nivel, en clara sintonía con la competencia por el control que se establece otros sistemas robóticos.

Uno de los trabajos más modernos en robótica que recoge la influencia de Tinbergen en la arquitectura de control es el de Ronald Arkin [Arkin *et al.*, 2003] con los perritos robóticos de Sony. En él se tiene una jerarquía de comportamientos y espacios de estado que determinan cuál de los

comportamientos hijo toma realmente el control en cada instante, materializando la inhibición lateral que menciona Tinbergen.

2.6.2. Lorenz

Otro padre fundador de la etología es Konrad Lorenz, que estudió durante muchos años el comportamiento animal e identificó la existencia de pautas motoras características de ciertas especies animales, llegando a la conclusión de que eran rasgos tan identificativos de la especie como lo son el color, tamaño y otras características morfológicas. Básicamente, estas pautas se presentan incluso en individuos aislados de otros de su misma especie, así que su existencia debe figurar en algún lugar de su cadena genética. Por tanto también están determinadas por los genes, quizá *programadas* y conformadas por la evolución de la especie en su interacción con el entorno. Estos comportamientos se denominan instintivos o innatos.

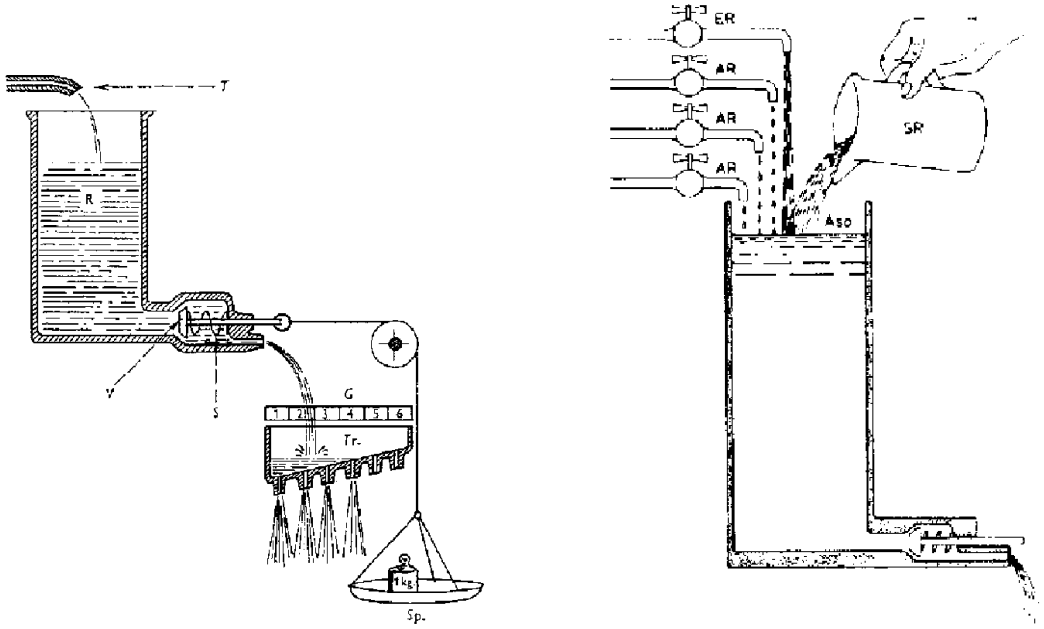
Un ejemplo clarificador lo suponen los ánsares comunes reincorporando un huevo que por cualquier circunstancia está fuera del nido. En esta tesitura todos las ánsares estiran el cuello desde el nido para dejar el huevo debajo de su pico y después lo arrastran hacia el nido. Esta maniobra falla con frecuencia debido al tamaño y forma del huevo. ¿Por qué el ganso no estira el ala y recoge con ella el huevo, que sería una maniobra mucho más sencilla y efectiva? ¿Por qué no lo hace si es capaz de identificar el huevo y tiene los actuadores necesarios (el ala)? Básicamente porque no puede actuar de otro modo, su sistema nervioso tiene grabado a fuego esa pauta de comportamiento y no otra.

Uno de los principales aportes de Lorenz es la superación de los reflejos como único mecanismo generador de conductas [Lorenz, 1978]. Por ejemplo, revelando la influencia de la motivación interna en los comportamientos. Para corroborar esta influencia se apoyó en la gradación de los comportamientos instintivos, y en la proporcionalidad encontrada por Alfred Seitz entre la fuerza de los estímulos ofrecida y la intensidad del movimiento desencadenado. Lorenz apreció que una estimulación externa débil y una fuerte disposición interna puede originar la misma pauta motora que una disposición interna débil y una fuerte estimulación externa. Esto remarcaba el peso de la motivación interna en el desarrollo de las conductas. Otro argumento en contra de la teoría de reflejos es que una estimulación excita a una secuencia de movimientos entera, no a un movimiento único.

Lorenz distingue al menos dos funciones fisiológicas distintas: el mecanismo desencadenante y la pauta motora desencadenada. Gracias al primero se reconocen de modo innato situaciones ventajosas para la especie, y gracias al segundo se sigue una pauta conductual finalista. A grandes rasgos esta distinción coincide con nuestro planteamiento de que la arquitectura del robot debe dar respuesta a la percepción y la actuación, para generar comportamiento.

Lorenz compartía a grosso modo el modelo jerárquico esbozado por Tinbergen. El trabajo de Lorenz ahondó en los mecanismos desencadenantes innatos y en la influencia de la parte perceptiva. Como hemos visto, la influencia de la disposición interna rompe la teoría de reflejos como mecanismo generador del comportamiento. Además la dinámica de esa motivación interna no es sencilla. Lorenz hipotetizó un modelo hidráulico para reflejar su funcionamiento, cuyo esquema aparece en la figura 2.25(a). Cada centro nervioso va acumulando la disposición a ejecutar cierto comportamiento instintivo, que se denomina *potencial específico de acción*. Sólo cuando esa motivación supera cierto umbral se lleva cabo realmente.

Como fuentes de ese potencial se tienen los diferentes estímulos externos o incluso disposición interna proveniente de otros centros superiores. Suyo es el principio de *suma heterogénea de estímulos*, ilustrado en la figura 2.25(b). Este principio replantea por completo el papel de la percepción en el comportamiento, que se limita a captar cierto conjunto de estímulos significativos y no a reconstruir el estado del mundo. Estos estímulos tienen su papel como motivadores de los comportamiento y/o como mecanismo desencadenador. Cada situación se describe como los estímulos presentes en ella, a modo del cuadro que se compone. Por ejemplo, en el comportamiento de apertura de pico de las crías de gaviota cuando viene el progenitor para alimentarlas intervienen muchos factores separados: la forma de pico que observa la cría, el color amarillo y la cabeza blanca, la forma cabeza, una mancha roja en el pico, etc.. Varios experimentos comprobaron que hay estímulos artificiales que disparan el comportamiento con más frecuencia e intensidad que sus homólogos naturales. Éstos son los denominados *superestímulos*.



(a) (b)

Figura 2.25: *Modelo psicohidráulico inicial de Lorenz (a), con estímulos desencadenantes y otros recargadores de disposición, y modelo mejorado (b).*

En cuanto al mecanismo desencadenante innato, tan relacionado con el problema de la selección de acción, Lorenz también reconoció la existencia de estímulos clave, que eliminan el umbral y provocan el desarrollo real de la pauta motriz. Una aportación suya es el reconocimiento de que ese umbral es dinámico. Por ejemplo, cuando se tiene mucha motivación interna el umbral puede bajar lo suficiente como para disparar la acción, incluso en ausencia del estímulo clave. Esto es lo que se llama *actividad en vacío*. Otra incorporación al sistema de motivación hipotetizado es el mecanismo de la fatiga específica, que no es cansancio físico, sino reluctancia a esta actividad en concreto. Esta fatiga aparece cuando se presentan muy repetidamente los estímulos adecuados para provocar cierto comportamiento. Esto origina una saturación, un cierto vaciado de motivación interna que impide desencadenar esa pauta motora durante un tiempo.

En [Vogel y Angermann, 1974] hay una buena introducción genérica a las ideas fundacionales de la etología moderna como rama de la biología y en [Lorenz, 1978] encontramos una recapitulación muy completa de los fundamentos de la etología, realizada por del propio Lorenz.

2.6.3. Ludlow

Un trabajo etológico significativo por su aplicabilidad en robótica es el realizado por [Ludlow, 1976]. En él describe el modelo utilizado para simular el sistema nervioso de un pequeño insecto, capaz de explicar su comportamiento. Este sistema incluye conceptos como la inhibición lateral y dinámica de energías que se han mostrado útiles dentro de la selección de acción. De hecho, el mecanismos de selección de acción que utiliza [Maes, 1990] en su arquitectura guarda bastantes semejanzas con el hipotetizado por Ludlow. Una de las principales ventajas de este modelo es que es computacionalmente materializable.

El mecanismo concreto se visualiza en la figura 2.26 y consta de cuatro nodos que gobiernan la ejecución de otras tantas pautas motoras. En el modelo no hay jerarquía explícita y todos los nodos están conectados entre sí. Para determinar el comportamiento que realmente se realiza se establece una dinámica de activaciones entre ellos, que elige a un único ganador. En esa dinámica cada nodo resta activación a los otros, de manera proporcional a su propia activación. Tal y como explica Ludlow, si ese factor de proporcionalidad es mayor que uno, entonces el sistema sólo es estable con un único centro activado, precisamente el que mayor impulso absoluto recibe. Con ello materializa la inhibición

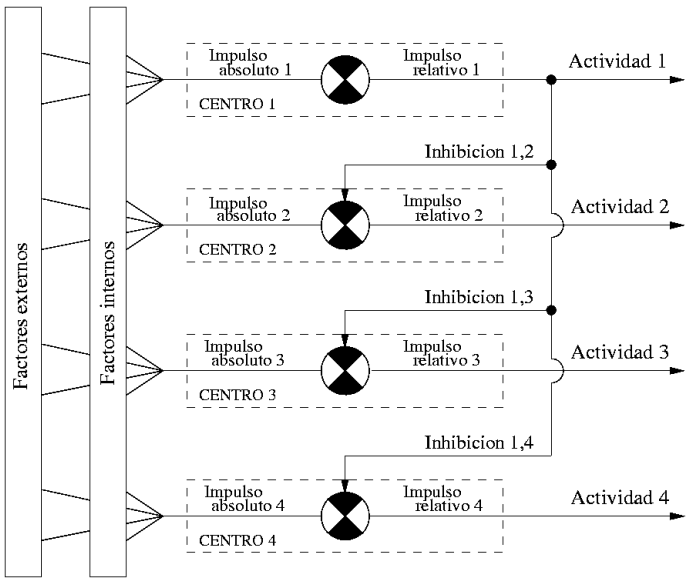


Figura 2.26: *Modelo de inhibición entre centros de Ludlow*

lateral que ya propugnaba Tinbergen [Tinbergen, 1950] y que en robótica es típica de las técnicas de arbitraje en la selección de acción (*winner takes all*).

La dinámica de activaciones también incorpora mecanismos de fatiga que permiten la alternancia entre centros en escenarios donde los estímulos presentes no varía sobremanera. Además presenta histéresis para evitar la hiperalternancia[7].

2.6.4. Toby Tyrrell

Toby Tyrrell presenta en su tesis doctoral [Tyrrell, 1993a] una comparativa muy completa de varios mecanismos de selección de acción en un mismo escenario. En concreto, Tyrrell estudia las redes de activación de Maes [Maes, 1990], los sistemas jerárquicos al estilo Tinbergen [Tinbergen, 1950] o Baerends y las técnicas de fusión de comandos como la de Rosenblatt y Payton [Rosenblatt y Pyton, 1989].

Para ello programa las distintas alternativas como generadores de comportamiento en un animal que vive en un entorno simulado, en el que hay alimentos de distintos tipos, predadores, vegetación en la que ocultarse, madrigueras, etc. En este entorno ecológico simulado [Tyrrell, 1992; Tyrrell, 1993b] el animal tiene que luchar por su supervivencia satisfaciendo varias necesidades como obtener suficiente comida, escapar de los predadores, evitar lugares peligrosos, asearse, aparearse y dormir en un lugar seguro. La satisfacción de estas necesidades se reflejan en el valor de ciertas variables internas con su propia dinámica, que conviene mantener dentro de unos márgenes de funcionamiento. El entorno ofrece una variedad muy grande de estímulos, objetivos múltiples, necesidades periódicas, necesidades urgentes, etc. por ello sirve de referencia para probar en profundidad las distintas técnicas de selección de acción.

Además de las evidencias recogidas en el entorno simulado, Tyrrell encuentra argumentos etológicos para proponer que la selección de acción basada en combinación de preferencias es la más adecuada. Critica las debilidades del mecanismo de Maes [Tyrrell, 1994] y propugna el uso de jerarquías como mejor alternativa. Pero no como estructuras de decisión jerárquica con una decisión por nivel, tipo Tinbergen, sino como lo que él llama *jerarquías de flujo libre* (*free flow hierarchies*), al estilo de Rosenblatt. El principal argumento es que las decisiones en los sistemas estrictamente jerárquicos no son capaces de combinar diferentes necesidades, y en las decisiones de un nivel se pierde información que se podría incorporar en la decisión global mejorando su conveniencia.

[7] el ejemplo más sencillo para explicar este fenómeno es del refrán del burro con una gavilla de paja a su izquierda y otra a su derecha. Cuando se había decidido a comer de uno de ellos, le parecía mejor la otra y cambiaba. Así, el burro acabó muriéndose de hambre porque nunca llegaba a comer.

2.6.5. Ideas principales

Los aportes de la etología a la generación de comportamiento complejo en robots son potencialmente enormes. Las conductas de animales sencillos, como los insectos, ya se ha conseguido imitar con robots, con mayor o menos éxito, utilizando un SBC para crear una cierta colección organizada de reflejos. Las arquitecturas descritas por los principales etólogos abordan el comportamiento de animales más complejos, como pájaros, peces o mamíferos inferiores. Estas conductas tan flexibles y eficientes no han sido aún conseguidas en robots autónomos, y supondrían un salto cualitativo en el rendimiento conseguido hasta el momento.

La etología ha presentado muchos conceptos útiles a la hora de estudiar y generar comportamientos: conductas apetitivas y consumatorias, suma aditiva de estímulos, estímulos clave, comportamientos de desplazamiento, motivación interna, etc.

Quizá el principal aporte hasta ahora haya sido la superación de la teoría de reflejos como mecanismo generador de comportamiento. Esta teoría subyace a todos los sistemas reactivos y basados en comportamientos. Por ejemplo, la secuencia de conductas se puede ver como unos reflejos encadenados, en la cual la ejecución del anterior provoca los estímulos necesarios para activar el siguiente, y cuyo resultado final es la consecución de cierto objetivo. Hay ejemplos sencillos en el mundo animal que cuadran perfectamente con este enfoque, como el comportamiento de captura de la notonecta [Vogel y Angermann, 1974], separado en distintas fases, cada una de las cuales presenta unos estímulos relevantes.

Sin entrar en conductas aprendidas, gran parte de los comportamientos instintivos no se pueden explicar con sencillas asociaciones situación-acción, son más complejos que simples reflejos, encadenados o no. Por ello se amplió el concepto de actividad instintiva para que incluyera los comportamientos reflejos, llamados taxias, además de los denominados movimientos instintivos. Mientras que las taxias requieren de la presencia durante todo el trayecto del estímulo externo, los movimientos instintivos tienen una coordinación hereditaria y son desencadenados por estímulos específicos que pueden desaparecer una vez el movimiento ha comenzado.

En esta línea los estudios fundacionales de etología introducen la disposición interna como factor decisivo en el comportamiento observable, lo cual es un aporte novedoso muy significativo. La motivación interna participa en el juego de la selección de acción con una dinámica compleja. Dentro de esa dinámica la idea de jerarquía de centros nerviosos, con motivación interna y mecanismos desencadenantes innatos, supone un nuevo enfoque al problema de la selección de acción. Otros trabajos relevantes que manejan motivaciones, aunque no en el sentido de Tinbergen o Lorenz, son los de Cañamero [Cañamero *et al.*, 2002], Timberlake [Timberlake, 2000] y Blumberg [Blumberg, 1994].

Un concepto fundamental surgido en la etología es la organización jerárquica del sistema nervioso que controla las conductas animales. Esta jerarquía se entiende como una predisposición hacia ciertos comportamientos, que están latentes, esperando a que se perciba cierto estímulo clave para manifestarse. Ese estímulo desinhibe esa predisposición, libera las barreras que retenían bloqueado al comportamiento. Sobre este concepto profundizaremos en el capítulo 3, porque es un eje central de la arquitectura propuesta y la herramienta para ordenar la complejidad del sistema de control.

Otro aporte importante procedente de la etología es la renovación del papel de la percepción en el comportamiento. Ésta cobra un papel fundamental puesto que además de intervenir en el desarrollo del comportamiento (por ejemplo en las taxias) también influye en la motivación interna y el mecanismo de selección de acción. La percepción se entiende como la identificación de ciertos estímulos relevantes de la situación, en términos de Gibson *affordances* [Gibson, 1977]. Tanto el principio de suma heterogénea de estímulos de Lorenz, como la propia teoría de *affordances* de Gibson constituyen aportaciones originales en este sentido.

En los últimos años la comunidad robótica ha prestado mayor atención a la etología como fuente de inspiración directa, y son muchos los trabajos que se han arrogado la influencia biológica en su arquitectura de control. Más allá de las modas, las contribuciones de fondo que esta disciplina puede traer a la generación de comportamiento artificial son muy prometedoras.

2.6.6. Limitaciones

La principal crítica que se puede achacar a las arquitecturas etológicas es que están lejos de su implementaciones en programas. Por ejemplo, la terminología que utilizan y las descripciones de conductas que ofrecen necesitan un trabajo arduo de interpretación y adaptación al entorno robótico. Esta limitación exige interdisciplinariedad al ingeniero robótico y un esfuerzo de comprensión e interpretación, porque las teorías etológicas explican mecanismos más o menos complicados, pero alejados de la formalización que facilite su materialización en programas. Esta dificultad resulta ineludible porque son teorías que han nacido en otro ámbito totalmente diferente al de la robótica genuina e incluso al de la informática.

En ese mismo sentido, se sitúan en un nivel de análisis abstracto. Los estímulos que resultan útiles para explicar la conducta animal pueden ser difíciles, si no imposible, de percibir y elaborar por un robot real con los sensores disponibles en la actualidad. Por ejemplo, la percepción de formas, tamaños y parecidos que parece emplear el sistema visual de algunos animales pueden ser computacionalmente irrealizables en el estado actual de la visión computacional o con la potencia de cálculo disponible.

En la actualidad el número de trabajos robóticos con influencia de la etología está creciendo significativamente. Quizá las dificultades que hemos identificado justifican la relativa escasez de ejemplos hasta hace pocos años.

Capítulo 3

Jerarquía dinámica de esquemas

> *Al ritmo al que aumenta la capacidad de los ordenadores, doblándose cada año aproximadamente, dentro de dos o tres décadas contaremos con suficiente potencia como para realizar el trabajo de un gran sistema nervioso. Ahora bien, disponer de esa potencia es tan solo la mitad del problema. La otra mitad es organizar correctamente los mecanismos internos del soporte lógico, o software. Por supuesto, aún no sabemos cómo crear una inteligencia análoga a la humana.*
> Hans Moravec – Beyond Human, 2001

En el capítulo anterior hemos repasado una colección representativa de arquitecturas concretas, organizándolas en familias o paradigmas e identificando las ideas principales de cada una de ellas. En este capítulo describiremos nuestra propuesta para generar comportamiento artificial autónomo, explicando los principios generales adoptados y los mecanismos creados para resolver problemas tales como la atención, la selección de acción o la composición de comportamientos. La perspectiva de este capítulo es eminentemente teórica y especulativa, y con ella se argumentan los posicionamientos principales de la propuesta. En los capítulos 4 y 5 adoptaremos un punto de vista más pragmático y describiremos una implementación concreta realizada de los mecanismos desarrollados aquí, así como su utilización en robots reales para generar algunos comportamientos de ejemplo.

Nuestra arquitectura recibe el nombre de *Jerarquía Dinámica de Esquemas* (JDE) y está profundamente enraizada en las ideas de Arbib [Arbib y Liaw, 1995] y los trabajos de Arkin [Arkin, 1989b]. A la hora de diseñarla hemos mantenido en todo momento la óptica causal, etiológica, no teleológica ni antropomórfica, puesto que lo que se persigue es causar comportamiento, originarlo. La arquitectura JDE parte del planteamiento inicial que contempla el comportamiento como la *combinación de percepción y control* en la misma plataforma. Además, cada uno de estos subproblemas lo divide y cuantiza en pequeñas porciones, que se resuelven por separado. Así *la unidad básica del comportamiento se denomina esquema*, como expondremos posteriormente, y tanto el control como la percepción necesarios para el comportamiento se distribuyen en una colección de esquemas. Hasta aquí nada nuevo respecto a algunas propuestas existentes en la literatura. Como hipótesis genuina se propone en esta tesis que esa colección de esquemas se puede *estructurar en una jerarquía*, la cual varía con el tiempo dependiendo del comportamiento concreto a exhibir. Además, se argumenta que ello facilita la composición de nuevos comportamientos a partir de los existentes, y proporciona una generación relativemente simple de fenómenos como la atención, la percepción contextuada y la selección de acción.

El capítulo está organizado empezando en la sección 3.1 con una descripción profunda de lo que llamamos esquemas. En ella hacemos hincapié tanto en los origenes históricos del término como en la interpretación precisa que le damos para nuestra tesis. En la sección 3.2 realizamos un barrido completo y descriptivo de la arquitectura en su conjunto, introduciendo la idea de jerarquía. Posteriormente, en las secciones 3.3 y 3.4, profundizamos en los detalles del control y de la percepción respectivamente. En cuanto al control explicaremos con detalle el mecanismo propuesto de selección de acción y sus propiedades. Ahondaremos también en la ausencia de niveles preestablecidos y la reconfiguración dinámica del sistema. En cuanto a la percepción veremos que es orientada a tarea, estructurada, fusionadora y contextuada. Finalmente, en la sección 3.5 resumimos formalmente los principios

fundamentales de JDE y discutimos comparativamente nuestra propuesta, reconociendo influencias y parecidos con otras arquitecturas anteriores, señalando también las diferencias, los aportes novedosos y los problemas que se intentan resolver con ellos.

3.1. Esquemas

El encapsulamiento de funcionalidad en pequeñas unidades que luego pueden ser reutilizadas ha sido una constante desde los primeros intentos de comprender el comportamiento. A lo largo de estos años se han ideado diferentes unidades: módulos, habilidades, agentes, esquemas, comportamientos básicos, etc. cada una con sus matices y peculiaridades. En concreto, la aparición de los esquemas como parte explicativa del comportamiento surge en el campo de la fisiología y neurofisiología [Arbib, 1981]. Su instalación en el campo de la robótica ha recibido el apoyo de muchos investigadores, entre los que se podría destacar a Michael Arbib y Ronald Arkin[1].

Por ejemplo, Arbib y sus colaboradores han estudiado el problema de la coordinación visuomotora. En esa línea han creado un modelo computacional, *rana computatrix*, capaz de engendrar los comportamientos de caza de moscas y fuga ante predadores [Arbib y Liaw, 1995], que se dan en las ranas reales. Incluso han generado el comportamiento de acercamiento y sorteo de una valla para capturar una presa al otro lado [Corbacho y Arbib, 1995]. Utilizan los esquemas como unidad adaptativa básica del comportamiento, sin perder de vista su implementación neuronal, y combinan varios de ellos para generar la conducta observable [Arbib, 1981; Arbib, 1989]. "Los esquemas perceptivos encierran el procesamiento que permite a un sistema reconocer un cierto dominio de interacción [por ejemplo predador, mosca, valla]. Proporcionan una estimación del estado del entorno y una representación de objetivos. [...] El estado interno también se actualiza con el conocimiento del estado de ejecución de los planes actuales conseguido por los esquemas motores. Los esquemas motores son una especie de sistemas de control, que se distinguen porque pueden combinarse para formar programas de control coordinado, los cuales regulan la entrada y salida secuencial de patrones de coactivación que tienen mecanismos de parametrización desde los esquemas perceptivos" [Arbib, 1989; Corbacho y Arbib, 1995]. Desde el punto de vista externo estos esquemas se manifiestan en determinados patrones de acción. Interiormente su implementación neuronal realiza un mapeo permanente de ciertas entradas a ciertas salidas.

La existencia de esquemas en los seres humanos ha recogido evidencias neurofisiológicas y psicológicas. Computacionalmente, una de sus ventajas es que son susceptibles de procesamiento distribuido, tal y como señala el propio Arkin [Arkin, 1989a]. De hecho, uno de los primeros escenarios en los que se aplicaron esquemas fue la visión [Arbib y H., 1987], donde los esquemas ofrecían potencialmente una aceleración del análisis al distribuir el procesamiento. Por ejemplo, en [Draper *et al.*, 1989] se utiliza una colección de esquemas para realizar de modo distribuido el procesamiento de imágenes de entornos naturales. "Una instancia de un esquema es una copia ejecutable del esquema, la cual corre como un proceso separado, con su propio estado"[Draper *et al.*, 1989]. Cada esquema es especialista en reconocer una determinada clase de objetos, y para ello tiene un conocimiento específico. En este sistema los esquemas funcionan en paralelo, comunicándose a través de una pizarra, y en ella van construyendo un conjunto de hipótesis perceptivas semántica y espacialmente compatibles. La salida que uno vuelca a la pizarra puede ser la entrada de otros, lo cual permite la coordinación y la construcción distribuida de cierta estructura.

Dentro de JDE definimos *esquema* como un flujo de ejecución independiente con un objetivo; un flujo que es modulable, iterativo y que puede ser activado o desactivado a voluntad. Hay *esquemas perceptivos* y *esquemas motores o de actuación*. Los esquemas perceptivos producen piezas de información que pueden ser leídas por otros esquemas. Estos datos pueden ser observaciones sensoriales o estímulos relevantes en el entorno actual, y son la entrada para los esquemas motores. Los esquemas de actuación acceden a esos datos y generan sus salidas, las cuales son comandos a los motores o las señales de activación para otros esquemas de nivel inferior (nuevamente perceptivos o motores) y sus parámetros de modulación, tal y como se ilustra en la figura 3.1.

[1] `http://www.cc.gatech.edu/aimosaic/faculty/arkin/`

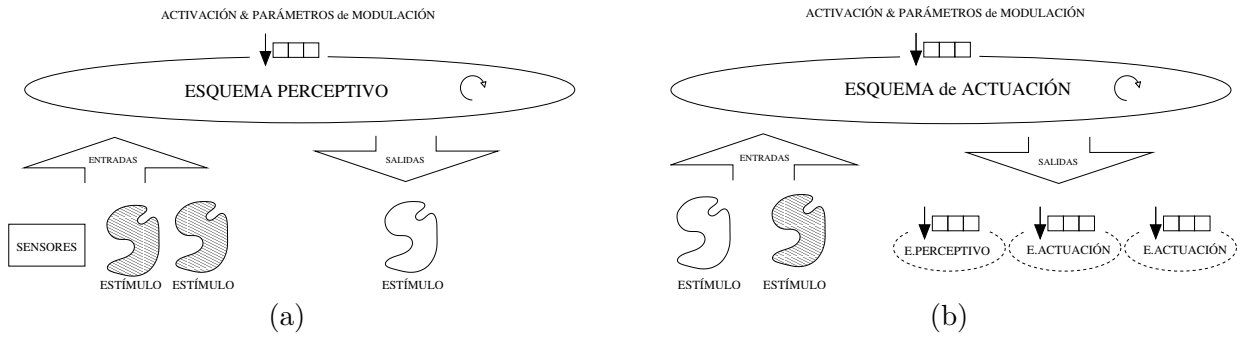


Figura 3.1: *Patrón típico de un esquema perceptivo (a) y de un esquema motor (b) en JDE*

Los esquemas JDE son por definición *modulables*. Siguiendo el segundo principio de Arbib [Arbib y Liaw, 1995], referente a la evolución y la modulación, los esquemas pueden aceptar varios parámetros de entrada que modulan su propio funcionamiento haciendo que se comporte de diferentes maneras. Además todos los esquemas en JDE son procesos *iterativos*, realizan su misión en iteraciones que se ejecutan periódicamente. De hecho, el periodo de esas iteraciones es un parámetro principal de modulación de cada esquema, permitiendo que se ejecute muy frecuentemente o con menor cadencia. Los controladores digitales se pueden ver como un ejemplo de este paradigma, pues ellos entregan una acción correctora cada ciclo de control. Los esquemas son además *suspendibles*, de manera que pueden ser desactivados al final de cada iteración, y en ese caso no producirán ninguna salida hasta que sean activados nuevamente.

A los esquemas se les asocia cierto *estado*[2], que ayuda a regular su coordinación, como veremos más adelante. Un esquema perceptivo puede estar en dos estados DORMIDO o ACTIVO. Cuando se encuentra ACTIVO, el esquema está actualizando las variables correspondientes al estímulo del cual se encarga. Cuando está DORMIDO las variables en sí existen, pero están sin actualizar, posiblemente desfasadas. El cambio de DORMIDO a ACTIVO o viceversa lo determinan los esquemas de nivel superior. Para los esquemas motores las cosas son un poco más elaboradas, pueden estar en cuatro estados: DORMIDO, ALERTA, PREPARADO y ACTIVO. Esto es debido a que cada esquema motor puede tener asociadas unas precondiciones y competir por el control con otros, según veremos en el apartado dedicado a la selección de acción.

Con JDE el sistema completo está formado por una colección de esquemas, siguiendo el primer principio de Arbib sobre computación cooperativa de esquemas [Arbib y Liaw, 1995]: "Las funciones del comportamiento (perceptivo-motor) y la acción inteligente de animales y robots situados en el mundo se pueden expresar como una red de esquemas o instancias de esquemas que interactúan".

El empleo de este tipo de esquemas tiene dos implicaciones importantes: primero, la separación entre la parte perceptiva y la parte de actuación; y segundo, una fragmentación de ambas en pequeñas unidades que reciben el nombre de esquemas. La separación permite simplificar el diseño, porque la percepción y el control son dos problemas diferentes, relacionados pero distintos. Como ambos son complejos y distintos se resuelven en zonas del código separadas, eso facilita las cosas. La fragmentación en unidades pequeñas facilita la reutilización y permite acotar mejor la complejidad de cada uno de los subproblemas que aborda, haciéndolos manejables en una estrategia de divide y vencerás. Además esta separación permite dar cuerpo a cada esquema en un procesador distinto, posibilitando una implementación distribuida.

Esta división del sistema entre percepción y control está suficientemente probada en la literatura. Por ejemplo, Konolige en su arquitectura Saphira [Konolige y Myers, 1998] presenta una separación similar. En ella hay varias rutinas perceptivas creando representación del entorno, lo que él llama *espacio perceptivo local*, y en paralelo tiene unos comportamientos. Esta distribución y lo que él llama *artefactos* desacoplan el problema del control de los problemas de interpretar datos sensoriales ruidosos. De modo muy similar Saffiotti [Saffiotti y Wasik, 2003] tiene un módulo de anclaje perceptivo (*Perceptual Anchoring Module*) que estima la posición de obstáculos cercanos, y sobre

[2]Estado en el sentido de los autómatas, es decir, que pasa de uno a otro

esa representación opera el módulo de comportamientos. También Budenske [Budenske y Gini, 1994; Budenske y Gini, 1997] fragmenta acción y percepción en nodos, que en este caso reciben el nombre de actuadores y sensores lógicos (*Logical Actuators* y *Logical Sensors*). En cuanto a la utilización en concreto de esquemas para esta distribución, una larga colección de trabajos robóticos en los que se han utilizado avalan su utilidad [Arkin, 1989b; Arkin y MacKenzie, 1994; Arkin, 1995; Arkin y Balch, 1997; Ali y Arkin, 1998; Arkin *et al.*, 2001; Stoytchev y Arkin, 2001].

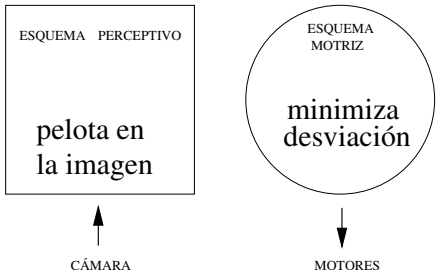


Figura 3.2: *Comportamiento sigue pelota como activación simultánea de dos esquemas, uno perceptivo y otro motor*

Como veremos en el capítulo 5, también se ha usado esta división en [Gómez *et al.*, 2003; Lobato, 2003; Gómez, 2002; Martín, 2002]. Por ejemplo, en [Gómez *et al.*, 2003] se describen los comportamientos `sigue-pelota` y `sigue-pared` que se han conseguido como la conjunción de dos esquemas JDE. Tal y como muestra la figura 3.2 el comportamiento sigue pelota consta de un esquema perceptivo, mostrado con forma cuadrada, y un esquema motor, mostrado como un círculo (seguiremos esta notación a lo largo de toda la tesis: cuadrados para percepción y círculos para actuación). El perceptivo se encarga de buscar la pelota en la imagen y caracterizar su posición en ella, y el motor materializa un control proporcional que mueve el robot tratando de centrar la pelota en la imagen. Si la pelota aparece desviada a la izquierda, el esquema tratará de girar el robot hacia la izquierda, y de modo simétrico para desviaciones a la derecha. Igualmente, si la pelota aparece en la parte superior de la imagen, el esquema aumentará la velocidad de avance del robot porque la pelota está relativamente lejos. Si por el contrario aparece en la parte inferior significa que está demasiado cerca del robot, y el esquema hará retroceder al robot. Este esquema utiliza un control en velocidad sobre los motores de las ruedas y emplea la desviación de la posición central de la pelota respecto del centro de la imagen como error a minimizar. La ejecución simultánea de estos dos esquemas permite al robot generar la conducta observable de seguimiento de la pelota.

Figura 3.3: *Sigue pelota con comportamiento apetitivo de búsqueda*

Con estos dos esquemas, si no hay pelota alguna en la imagen entonces el esquema de control mantiene detenido los motores. Para ver la flexibilidad de esta descomposición, siguiendo con el sistema de ejemplo, una incorporación razonable podría ser un tercer esquema, también de control, que se activa cuando no aparece ninguna pelota en la imagen, y que se encargaría de mover el robot tratando de buscar las pelotas en su vecindad. Por ejemplo, girando el robot sobre sí mismo para barrer todos los alrededores. Tal y como muestra la figura 3.3, este esquema materializaría una conducta apetitiva, pues fomenta la aparición de la pelota en la imagen, que es el estímulo necesario para el comportamiento

fundamental de seguimiento. La pelota sería el estímulo clave que activa al esquema de seguimiento. En la figura 3.4 se añaden dos nuevos esquemas que enriquecen el comportamiento global con la capacidad de sortear obstáculos. Uno perceptivo para detectarlos y otro de control para sortearlos.

Figura 3.4: *Sigue pelota complementado con sorteo de obstáculos*

Tal y como vimos en la sección 2.4.2, otro ejemplo lo constituye el comportamiento `sigue-pasillo--evitando-obstáculos`, que Arkin [Arkin, 1989b] consigue como la superposición de varios esquemas motores, y la activación simultánea de varios esquemas perceptivos. Según muestra la figura 2.15 hay un esquema por cada obstáculo, que se encarga de interiorizar la presencia de los diferentes obstáculos y materializa la repulsión hacia él. Además hay un esquema que implementa la querencia hacia el punto objetivo y otro que tiende a mantener al robot en el pasillo. En ese ejemplo hay combinación de la salida de los esquemas motores, lo cual diferencia los esquemas de Arkin de los esquemas de JDE. Mientras en JDE hay activaciones exclusivas, Arkin plantea la activación simultánea y la combinación lineal de las salidas activas.

3.2. Jerarquía dinámica de esquemas

Una vez presentada nuestra unidad básica del comportamiento, el esquema, hay muchas opciones para su ensamblaje en un sistema completo. En este apartado daremos una visión global a la arquitectura JDE, que utiliza la jerarquía para regular el modo en el que se combinan los esquemas, y veremos también la manera en que éstos interactúan en ella y cómo unos utilizan la funcionalidad de otros para conseguir la conducta observable. Presentaremos aquí los mecanismos de percepción y actuación que ofrece esta jerarquía como principio organizador, aunque los detalles de cada uno de ellos los ampliaremos en las secciones 3.3 y 3.4.

Como vimos en capítulo 2 las arquitecturas basadas en comportamientos nacieron hacia finales de los años 80, siendo los esquemas de Arkin y las redes de autómatas de estado finito de Brooks las instancias más conocidas que se engloban dentro de este paradigma. Para conseguir comportamientos crecientemente complejos Arkin abogaba por máquinas de estado finito que seleccionan en cada instante los esquemas activos de una colección de esquemas disponibles [Arkin y MacKenzie, 1994]. Por su parte Brooks abogaba por los niveles de competencia y un apilamiento creciente de nuevos niveles para desarrollar funcionalidad más elaborada, apoyándose en los ya construidos.

Después de unos años de rotundo éxito práctico de los sistemas basados en comportamientos para conductas relativamente sencillas (pero no conseguidas hasta entonces) como la navegación reactiva, su aplicación a comportamientos más complicados no ha tenido tanto refrendo experimental. Los primeros robots de Brooks como Herbert, Toto, Genghis, etc. supusieron un avance más significativo que sus más recientes hermanos como Cog y Kismet. Se ha hecho evidente que no escala bien a comportamientos complicados. Si bien la identificación de comportamientos básicos como unidades es conveniente, el factor clave para conseguir con este paradigma conductas más complejas es el modo de combinarlos, de gestionar las interacciones entre ellos, sobre todo cuando hay un número ascendiente de éstos. Para abordar esa complejidad creciente muchos investigadores proponen incluir los sistemas basados en comportamientos como nivel básico, y por encima suyo componentes deliberativos clásicos. Tal y como vimos en el capítulo 2 son los sistemas híbridos, por ejemplo, las arquitecturas de tres niveles. Nuestra propuesta para crecer en complejidad no va por esos caminos, nos inclinamos por la jerarquía de esquemas, sin perder la esencia de los sistemas basados en comportamientos y recogiendo la influencia

de los modelos etológicos. En la literatura existen otras propuestas para abordar comportamientos más complejos e incluso otras interpretaciones de la idea de jerarquía (Meystel, Albus, Sugeno) con las que iremos contrastando la nuestra.

3.2.1. Jerarquía como predisposición

La tesis fundamental que proponemos en este trabajo es que se puede generar comportamiento autónomo complejo con una jerarquía dinámica de esquemas como los descritos en la sección anterior. Tal y como vimos anteriormente, cada esquema de JDE tiene un objetivo propio, realiza alguna funcionalidad específica en la que es experto (bien sea en control, bien sea en percepción). La jerarquía aparece por el hecho de que los esquemas pueden aprovechar la funcionalidad de otros para materializar la suya propia, y en JDE el modo que un esquema tiene de aprovechar la funcionalidad de otro es precisamente su activación y modulación.

En JDE la salida de un esquema motor situado en los niveles más bajos de la jerarquía suelen ser los comandos directos que envía a algún actuador. Pero la salida también puede ser la activación y modulación de otros esquemas (esquemas hijos), los cuales en su ejecución, mientras persiguen o mantienen sus propios objetivos, están ayudando a que el que los activó (esquema padre) persiga o mantenga los suyos. Cuando un esquema es activado trata de conseguir su propio objetivo de algún modo, como una secuencia de acciones o como respuestas a ciertos estímulos del entorno, que son finalmente llevadas a cabo por los del nivel adyacente inferior. La activación del esquema inferior es la unidad de acción en ese nivel, es el vocabulario en el que se expresan sus actuaciones. Para flexibilizar esta subcontratación de funcionalidad el esquema superior puede sesgar ligeramente el comportamiento de los inferiores para adaptarlo más a su propio objetivo. En general, los esquemas ofrecen una funcionalidad específica, pero de manera flexible pues admite modulación a través de ciertos parámetros.

Este patrón de activación puede repetirse recursivamente, de modo que aparecen varios niveles de esquemas donde los de bajo nivel son despertados y modulados por los del nivel superior. Las activaciones en cadena van conformando una jerarquía de esquemas específica para generar ese comportamiento global en particular. La colección de esquemas se organiza en jerarquía para materializar cierta conducta. Esta jerarquía se reconstruye y modifica dinámicamente, según cambie el comportamiento a desarrollar o las condiciones por las que un esquema padre activa a cierto esquema hijo y no a otros.

La jerarquía que proponemos no es la clásica de activación directa, en la que en padre pone a ejecutar a un hijo durante cierto tiempo para que realice cierta misión, mientras él espera el resultado. En vez de que la misión del hijo sea un paso en el plan secuencial del padre, se entiende la jerarquía como una coactivación que expresa predisposición. En JDE un padre puede activar a varios hijos a la vez, porque no es una puesta en ejecución en la cual los hijos emitan directamente sus comandos a los actuadores, sino un situar en alerta. La activación final se deja en manos del entorno y de la competición con otros hermanos.

La forma en que se implementa ese proceso de activación y selección es usando cuatro estados: DORMIDO, ALERTA, PREPARADO y ACTIVO. El paso de DORMIDO a ALERTA lo determina la preactivación del padre. El paso de ALERTA a PREPARADO lo determinan las precondiciones del hijo, que él mismo evalúa periódicamente. El paso de PREPARADO a ACTIVO se pelea en una competición por el control entre los hermanos preparados en ese nivel.

En cada instante hay varios esquemas en ALERTA por cada nivel, ejecutándose concurrentemente, pero sólo uno de ellos es activado por la percepción del entorno. Cuando ningún esquema o más de uno quiere ser activado, entonces el esquema de nivel superior se invoca para que arbitre cual de ellos toma realmente el control. Por ejemplo, la figura 3.5 muestra una instantánea de la arquitectura JDE en funcionamiento. Los esquemas motores en ALERTA aparecen como círculos con bordes continuos. Sería el caso de los esquemas 5, 6 y 7. Varios de estos pueden ser compatibles con la situación perceptiva del entorno, pasando a PREPARADO, pero sólo uno de ellos ganará la competición por el control en ese nivel y pasará a ACTIVO. Los esquemas en ACTIVO se muestran en la figura 3.5 como círculos rellenos, como el 1, el 6 y el 15.

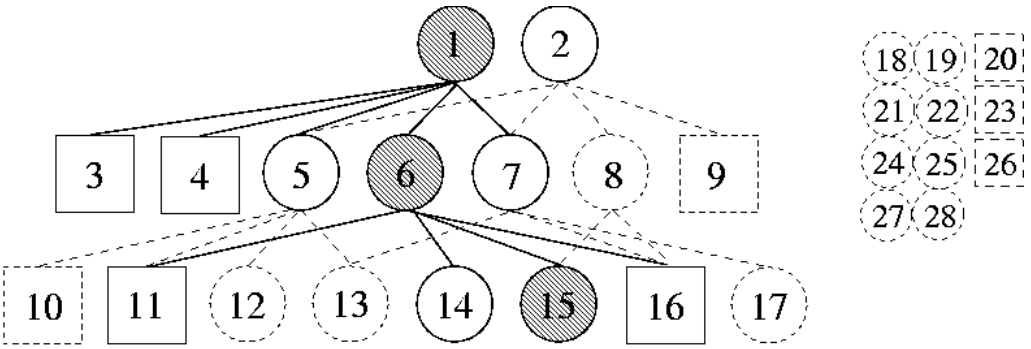


Figura 3.5: *Jerarquía de esquemas y batería de esquemas en* DORMIDO.

Debido a este juego de estados la activación desde el padre consiste en un incremento de la predisposición a ciertas actuaciones y de la sensibilidad a ciertos estímulos. Éste es el fruto de concebir el comportamiento como un conjunto de respuestas a ciertos estímulos relevantes más que como una secuencia de acciones ejecutadas siguiendo una pauta fija.

El padre activa a los esquemas perceptivos que buscan los estímulos relevantes para realizar su misión y a los esquemas de actuación que generan las respuestas convenientes cuando esos estímulos aparecen realmente. Si un esquema motor ACTIVO necesita cierta información para conseguir su propio objetivo entonces activa los esquemas perceptivos relevantes (cajas cuadradas en la figura 3.5) para recoger, buscar, construir y actualizar esa información. Además despertará a un conjunto de esquemas motores de bajo nivel (círculos) que pueden ser útiles para su objetivo porque materializan las respuestas adecuadas a los estímulos del entorno. Les modula para que se comporten de acuerdo a sus propios objetivos, y los pone en estado ALERTA. Despierta no sólo a los convenientes para la situación actual, sino también a todos los esquemas motores inferiores que pueden tratar alguna situación plausible. Por ejemplo, el esquema motor 6 de la figura 3.5 es el ganador de la competición de control en ese nivel. Él despierta a los esquemas perceptivos 11, 16, y a los esquemas motores 14 y 15, a los que configura sus parámetros de modulación. Esto explica que en JDE la correspondencia suela ser de un padre a varios hijos, en incluso que el padre pueda activar a diferentes hijos (o modularlos de modo diferente) en distintas etapas de su propia ejecución.

Esta idea de predisposición ya aparece en las propuestas de jerarquía aparecidas en etología. Además esta interpretación es compatible con que se pongan en alerta varios esquemas que realizan la misma función, y dependiendo de la situación del entorno, sólo se activará el más adecuado a ese contexto. Tal y como señala Tinbergen para ejemplificar su jerarquía [Tinbergen, 1950], el halcón que sale de caza tiene preactivados, predispuestos los módulos de `cazar-conejos`, `cazar-palomas`, etc. mientras sobrevuela su territorio de caza. Todos ellos satisfacen la finalidad de alimentarlo, se activará realmente uno u otro dependiendo de la presa que encuentre. Es lo que Timberlake [Timberlake, 2000] llama *variabilidad restringida*, y que tiene perfecta cabida en JDE. Esta capacidad de tener varias alternativas predispuestas de modo simultáneo contrasta con las alternativas que tiene RAP de Firby [Firby, 1987; Firby, 1992; Firby, 1994] (y por ello que hereda la arquitectura híbrida 3T [Bonasso *et al.*, 1997]), que se ensayan una detrás de otra, pero sólo cuando la anterior ha fracasado.

Desde el punto de vista de los hijos, un esquema en estado activo sólo puede tener un único padre en un instante dado. Es decir, un hijo no puede tener varios padres en el mismo instante. Sí puede ocurrir que sea activado por distintos padres en diferentes momentos de tiempo. También puede suceder que se activen simultáneamente instancias diferentes del mismo esquema, probablemente con modulaciones distintas y en niveles diferentes. Esta posibilidad es un ejemplo de reutilización de esquemas.

Los esquemas que un momento dado no se usan para la tarea en curso descansan en una batería de esquemas, suspendidos en estado DORMIDO, pero preparados para la activación en cualquier momento. Éstos aparecen como cuadrados y círculos discontinuos en el lateral derecho de la figura 3.5 (esquemas 8, 9, 10, 12, 13, 17, 18, etc.).

Las jerarquías se construyen y modifican dinámicamente y son específicas de cada comportamiento. Si las condiciones del entorno o los objetivos del robot varían, puede ocurrir que entre los hermanos

que están ALERTA en cierto nivel cambie la relación de cuáles de ellos están preparados para afrontar la nueva situación. Esto altera quién es el ganador en la competición por el control en ese nivel y fuerza una reconfiguración de la jerarquía desde ese nivel hacia abajo. Todos los que estaban activos por debajo se desactivan y el nuevo árbol se establece a partir del nuevo ganador. Por ejemplo, los esquemas 10, 12 y 13 de la figura 3.5 están a un sólo paso de la activación, pues serán despertados si el esquema 5 pasa a ACTIVO en su nivel.

Del mismo modo, si la situación del entorno o los objetivos se alteran, cierto esquema en determinado nivel de la jerarquía actual puede decidir modificar los parámetros de sus hijos actuales, o incluso cambiar de hijos. De nuevo, esto fuerza una reconfiguración de la jerarquía desde ese esquema hacia abajo, pues los nuevos hijos al ejecutarse activarán a otros hijos suyos. Las reconfiguraciones en JDE suelen ser rápidas puesto que el arbitraje y las decisiones de cada esquema de control se toman periódicamente, a un ritmo suficientemente vivaz.

En JDE el comportamiento se considera globalmente, no sólo la parte de actuación, sino también la parte de percepción. Los esquemas perceptivos se encargan de elaborar estímulos relevantes, que se emplean bien en la ejecución de los esquemas de actuación, bien para determinar su activación. La jerarquía incluye la percepción como parte subsidiaria de los esquemas de actuación, y gracias a ello se facilita la coordinación, esto es, se da un contexto a la percepción. Para cada comportamiento hay que ligar los esquemas perceptivos a los de actuación, con ello se resuelve la coordinación percepción--actuación. Por ejemplo, la coordinación visuomotora en el movimiento. La jerarquía determina qué percibir y cuándo.

Además ese contexto de percepción permite, como veremos posteriormente, replicar fenómenos como la atención o la interpretación. Además la jerarquía organiza la percepción y se contempla la posibilidad de subjerarquías formadas por esquemas exclusivamente perceptivos, donde la información que elaboran los de nivel inferior sirve como base para elaborar el estímulo conceptualmente superior. Gracias a ello permite dotar de estructura a ciertos estímulos.

Tal y como muestra la figura 3.5 hay reutilización de esquemas perceptivos puesto que el mismo esquema perceptivo puede asociarse a distintos esquemas motores. En realidad son distintas instancias del mismo esquema. Por ejemplo, el esquema 11 puede dar percepción al esquema motor 6 y al 5, o el 16 al 6, 7 y 8. Puede haber estímulos específicos, como el que elabora el 10, que sólo le interesa al esquema motor 5, pero en general la reutilización es la norma en percepción, para estímulos comunes. También puede haber reutilización de esquemas motores.

El sistema de control completo consiste en varios esquemas en ACTIVO ejecutándose a diferentes velocidades, cada uno de los cuales tiene sus propios objetivos locales y su percepción. Cada esquema se puede ver como un bucle de control que funciona a cierto ritmo (al necesario para su tarea) sobre sus datos de entrada, y que está o no ACTIVO en cada intervalo temporal. El sistema global se puede ver como una ejecución simultánea de varios controladores, de modo similar a la jerarquía de controladores borrosos de Sugeno [Sugeno, 1999]. Los esquemas superiores activan y modulan a los inferiores, dando coherencia a todo el sistema, que procede así a conseguir la misión global. Este modo de acumular esquemas no ralentiza el funcionamiento en absoluto, pues los inferiores siempre están funcionando en un rápido bucle de realimentación con el entorno. Así pues, esta jerarquía no añade retardos que hagan perder la reactividad[3].

La idea central subyacente en JDE es la creencia en la composición de comportamientos [Aguirre *et al.*, 1998]. Composición en dos sentidos: temporal y en abstracción conceptual. En sentido temporal con JDE se argumenta que *una gran variedad de tareas motoras se puede describir y conseguir en términos de secuenciación de varios esquemas*, exactamente como enuncia el primer principio de Arbib [Arbib y Liaw, 1995]. En sentido conceptual, JDE apuesta por la ejecución simultánea de varios niveles de abstracción, cada uno de los cuales se materializa en otro nivel inferior y puede en cada momento materializarse en otro diferente de bajo nivel para adaptarse a las distintas condiciones del entorno.

Este planteamiento separa la abstracción conceptual de la abstracción temporal, que son indisociables en otras unidades de comportamiento como las unidades de acción [Nilsson, 1969] o

[3]El número de esquemas que se pueden acumular sin aumentar peligrosamente el tiempo de respuesta depende de la velocidad del procesador (o procesadores) en los que se implementen los esquemas

los RAP [Firby, 1987]. En JDE los elementos de muy alto nivel de abstracción se pueden traducir en esquemas tan rápidos como se necesite. Se asciende en nivel de abstracción obligando a que las salidas de los esquemas de alto nivel sean exclusivamente activaciones de los esquemas de nivel inferior y su correspondiente modulación. Todos ellos funcionan en paralelo y toman decisiones en cada instante, cada uno en su nivel de abstracción. Un nivel más alto se distingue de otro inferior porque utiliza variables de estímulos que semánticamente son más abstractas, no por tener diferente velocidad.

De hecho, en JDE todo es presente, como en los sistemas reactivos, no hay largo plazo de modo explícito. En este sentido sigue la teoría de la actividad improvisada que describe Agre [Agre y Chapman, 1990]. Puede haber consideraciones de largo plazo interiormente a cierto esquema, o de modo implícito, pero debe cristalizarse en alguna actuación presente. El largo plazo se instancia en el corto plazo como una predisposición en el instante actual hacia ciertos comportamientos. Esto contrasta con otras arquitecturas donde las decisiones de largo o medio plazo se cristalizan en un plan, y las de corto plazo (basadas en los datos sensoriales) son muy volátiles. En ellas hay tratamiento explícito de diferentes horizontes temporales.

La influencia de planteamientos etológicos en esta propuesta es muy grande. Por ejemplo, guarda muchas similitudes con la jerarquía que propone Tinbergen [Tinbergen, 1950] para explicar la generación de comportamientos instintivos. Los esquemas de control son análogos a sus centros nerviosos. Como hemos señalado anteriormente, otra similitud es la utilización de la jerarquía para generar predisposición de ciertos comportamientos cuando se activa cierto instinto de nivel superior. Curiosamente las propuestas modernas de Arkin [Arkin *et al.*, 2003] también se apoyan en esta idea de jerarquía, donde se admite además la influencia que han ejercido en su sistema las ideas nacidas en la etología. Como valor añadido sobre el trabajo de Tinbergen y Lorenz, JDE explicita también la organización de la percepción, tal y como hemos visto.

JDE además permite la implementación directa de conceptos como los *estímulos clave* o los *comportamientos apetitivos*, cuya definición nació en el campo de la etología. Por ejemplo, el *comportamiento apetitivo* o de apetencia lo acuñó el biólogo Craig en 1926 y se utiliza para designar a aquella conducta que no satisface directamente ninguna necesidad del animal, sino que busca la situación desencadenante de la acción final, del *comportamiento consumatorio*, que es el que realmente sacia la necesidad interna [Vogel y Angermann, 1974; Lorenz, 1978]. Una muestra de conducta consumatoria podría ser comer una presa, mientras que buscarla sería apetitiva.

Otro concepto etológico materializable en JDE podría ser los *estímulos clave*, que se definen como aquellos que disparan la activación de cierto patrón de comportamiento. En JDE se puede implementar de modo sencillo un esquema perceptivo que busca en la realidad el estímulo que desencadena el paso de ALERTA a PREPARADO de cierto esquema motor, promoviendo su activación. En general cada esquema motor tiene su *estímulo clave* [Corbacho y Arbib, 1995], y reacciona ante él cuando está presente.

3.2.2. Configuraciones típicas

El escenario típico de esta arquitectura es un procesador (o varios) ejecutando una colección de esquemas simultáneamente, todos los esquemas están despiertos en cierto instante (en ACTIVO, PREPARADO o ALERTA). Usualmente hay 20 o 30 esquemas ejecutándose en paralelo. Esto choca con el escenario históricamente habitual de un único proceso ejecutando en una máquina, donde la limitación de cómputo determinaba la organización conceptual. Ahora que el desarrollo creciente de la microelectrónica ha facilitado la aparición de procesadores cada vez más rápidos nos enfrentamos directamente al problema de la organización del sistema. La misma cuestión subyacía antes, pero la limitación de potencia ya imponía por sí misma criterios de organización para conseguir algún comportamiento real de modo eficiente. Por supuesto, siempre habrá aproximaciones que necesiten más capacidad de procesamiento para generar comportamiento. Cualquier enfoque de fuerza bruta lo hace. Pero hoy en día podemos decir que la potencia de cálculo no es el único factor determinante para conseguir comportamientos complejos, también lo es la organización del sistema generador.

Dentro de JDE se pueden identificar distintos patrones de uso, que muestran distintas configuraciones frecuentes entre padres e hijos. En la figura 3.6 se muestran algunas de ellas. Por ejemplo, en la figura 3.6(a) se observa una configuración en la que un esquema motor activa los perceptivos que necesita, en este caso 3 estímulos, y él mismo envía comandos a los actuadores para

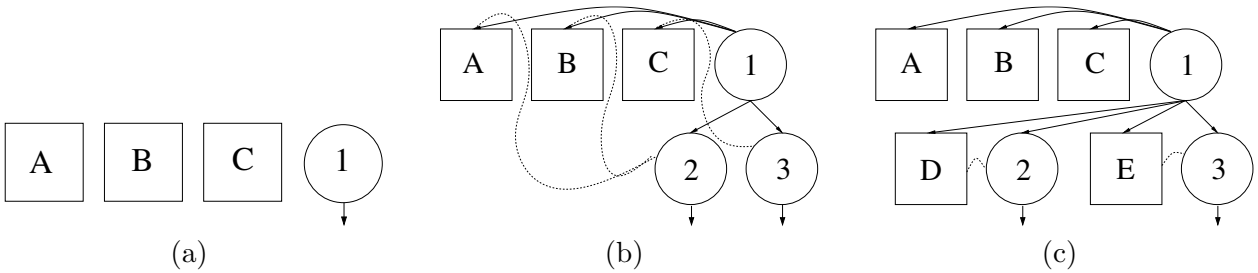


Figura 3.6: *Configuraciones típicas*

generar el comportamiento. La figura 3.6(b) ilustra un paso más avanzado, donde el mismo esquema motor 1 materializa su conducta indirectamente, activando y modulando otros dos esquemas motores, 2 y 3, que son los que envían comandos a los actuadores en este caso. Hay que destacar que no lo harán simultáneamente, sino según dicte la competición por el control que se establece entre ellos. Tanto para resolver esa competición como para determinar los comandos adecuados, los hijos motores consultan la información de los esquemas perceptivos activados por el padre: el esquema 2, consulta la de A y B, y el esquema 3 la de C, según muestran las líneas discontinuas. Una configuración más elaborada aún es la que se observa en la figura 3.6(c). Si en la anterior los hijos motores no activan ningún otro esquema, en ésta otra activan sendos esquemas perceptivos. En este caso, el esquema 2 activa al D, y el 3 al E.

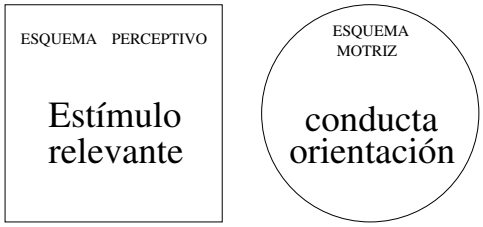


Figura 3.7: *Materialización de una taxia con JDE.*

Un tipo de comportamientos muy frecuente en la naturaleza son las *taxias*, que se definen como "movimiento en respuesta a un excitante exterior". Un ejemplo sencillo es el movimiento del los girasoles, persiguiendo con su giro al sol para recibir más luz. Otro ejemplo es el seguimiento que hacen las ranas alineándose con una mosca antes de lanzar la lengua para capturarla. En estas conductas el propio movimiento sigue el de cierto estímulo exterior. Tal y como muestra la figura 3.7, dentro de JDE se pueden implementar las taxias como un par de esquemas: uno perceptivo que interioriza el estímulo externo, y otro de actuación que materializa el seguimiento. En el capítulo 5 describiremos con detalle algunas taxias conseguidas desde información visual, como el comportamiento `sigue-pelota`, que también hemos presentado en la figura 3.2.

Otro tipo útil de comportamientos son las secuencias, como sucesión de pasos que tienen un objetivo final conjunto. Dentro de JDE las secuencias de comportamientos se pueden conseguir utilizando un esquema motor codificado como una máquina de estado finito. Cada estado corresponde con un paso en la secuencia, y provoca que un conjunto diferente de esquemas inferiores sean despertados. Además activa a los esquemas perceptivos necesarios para detectar los eventos que disparan ese cambio de su estado interno. En este caso se pueden generar secuencias fijas, patrones invariables, desencadenadas por algo, pero una vez activadas, actúan prácticamente en lazo abierto. Estas secuencias fijas son típicas de algunos movimientos instintivos de animales, por ejemplo el de recogida de un huevo con el cuello dentro de los ánsares [Lorenz, 1978; Tinbergen, 1987]. No se puede saltar ningún estado intermedio. El simple paso del tiempo puede provocar el paso de un estado a otro. Este uso ilustra que el mismo esquema padre puede activar a varios hijos diferentes en distintas fases (estados internos) de su propia ejecución.

Un modo diferente de materializar secuencias de comportamientos en JDE se muestra con el

Figura 3.8: *Reflejos encadenados con JDE.*

conjunto de esquemas de la figura 3.8. El esquema perceptivo A percibe el estímulo que activa la ejecución del esquema motor 1, y que le da la información necesaria para su funcionamiento normal. Del mismo modo se encuentran emparejados el esquema perceptivo B con el motor 2, el perceptivo C con el motor 3. Supongamos que el esquema motor 1 se encuentra funcionando y actúa de comportamiento apetitivo que aumenta la probabilidad de que aparezca el estímulo B, lo cual dispara el esquema motor 2. Éste a su vez materializa la conducta apetitiva que fomenta que aparezca el estímulo C, el cual activa el comportamiento consumatorio final 3.

Este modo de generar la secuencia es más flexible que el anterior, pues permite saltar pasos. Esta secuencia flexible supone una gran ventaja con respecto a los sistemas deliberativos que utilizan planes como curso explícito (y único) de la acción. Tal y como señala Payton en [Payton, 1990], los planes clásicos se atascan en el compromiso con metas intermedias, que en ciertas situaciones dejan de ser convenientes y deberían poderse saltar. Por ejemplo, si estando activos A-B-C-1 surge el estímulo C, entonces directamente se activa el esquema motor 3, no siendo necesario el paso previo por el esquema motor 2. Este mismo fenómeno se ha observado en la naturaleza, como comportamiento de caza de la notonecta [Vogel y Angermann, 1974], ya mencionado en el capítulo 2. Éste y otros ejemplos típicos de la *teoría de reflejos encadenados* se pueden explicar dentro de JDE con secuencias como la de la figura 3.8.

A su vez esta materialización también explica la variabilidad restringida de Timberlake. Imaginemos que existiera un cuarto par, formado por el esquema perceptivo D y el esquema motor 4, los cuales provocan la aparición del estímulo C-3. Pensemos también que pudiera surgir el estímulo D durante la ejecución de A-1, con la misma probabilidad que B-2. De esta manera habría dos alternativas que cumplen la función de segundo paso en la secuencia. Otra manera de implementar esta secuencia sería meter B-2 y D-4 como posibles hijos de un único paso funcional.

Además, el funcionamiento de JDE no dificulta la activación de otros esquemas. El padre de los seis esquemas de la figura 3.8 fija un contexto acotado, y busca los estímulos que a él le atañen, pero no impide la búsqueda de los estímulos relevantes para otros esquemas. En paralelo pueden estar ejecutando los esquemas perceptivos que detectan la aparición de otros estímulos significativos, los cuales pueden dar pie a la activación de otros esquemas más convenientes entonces.

3.3. Control

Una vez delineadas las características generales de la arquitectura JDE en su conjunto, ahora describiremos con mayor minuciosidad los mecanismos que incluye. En esta sección ampliaremos la parte de control y en la sección siguiente la parte de percepción. No presentaremos ninguna técnica de control concreta, sino cómo se integran diferentes técnicas y conductas en un único sistema. En terminología de Brooks [Brooks, 1991a], no entramos aquí en el nivel *micro* del comportamiento, sino en el nivel *macro*, en la organización de sus unidades, contemplando las implicaciones que tiene la arquitectura propuesta.

Hay que resaltar que lo único que regula la arquitectura es el modo en que los esquemas se interconectan y ensamblan en un todo. JDE regla el modo de conectar los esquemas entre sí y como unos aprovechan la funcionalidad de otros. La arquitectura normaliza el interfaz de los esquemas, pero no su funcionamiento interno. Así los esquemas pueden implementarse con muchas técnicas diferentes. Por ejemplo, los esquemas motores se pueden realizar como sencillas reglas ad hoc sobre los datos sensoriales crudos, como tablas fijas de entradas-salidas, como controladores borrosos basados en

reglas, como un proceso que discretiza el espacio de alternativas y elige aquella que maximiza cierta función objetivo, como planificadores, como máquinas de estado finito, con redes neuronales, etc. El único requisito que impone la arquitectura es que el esquema sea iterativo, suspendible y si procede, modulable. La iteratividad implica que ha de entregar periodicamente sus salidas.

De hecho, la opción de incluir un planificador deliberativo dentro de un esquema es perfectamente válida pues los esquemas de actuación pueden guardar estado interno. Por ejemplo, pueden almacenar internamente un plan simbólico al estilo clásico. En este sentido JDE admite razonamiento simbólico, manejo de símbolos, y eso la distingue frente a las aproximaciones reactivas puras, o subsimbólicas. Incluso en el caso de implementarlo como un planificador, JDE obliga a que el esquema entregue una recomendación de actuación en cada iteración. Esta imposición es similar a la que obliga Rosenblatt [Rosenblatt y Pyton, 1989], en su caso para poder combinarlos adecuadamente con otras tendencias reactivas. Con ella se fuerza a que el plan se considere como un recurso más, similar a los planes internalizados de Payton [Payton, 1990]. En el caso de JDE no es para combinar diferentes tendencias, es para integrar el esquema en la arquitectura, para materializar su efecto. El formato en el cual unos esquemas utilizan a otros es uniforme (activación y modulación en el tiempo), siempre el mismo, sean los esquemas de naturaleza reactiva, deliberativa, etc. No se prohibe la planificación, pero se obliga a que sean recursos para la acción. Con o sin horizonte temporal, todos los esquemas deben proponer algo concreto para ahora.

En cuanto a la descomposición de tareas en subtareas, en JDE cada esquema es responsable de una tarea y la soluciona a cierto nivel de abstracción, con iteraciones periódicas. Su ejecución permanente en un periodo de tiempo tiende a conseguir su objetivo o mantenerlo, bien enviando comandos a los actuadores, bien activando y modulando otros esquemas de nivel de abstracción menor. Además, el esquema es responsable de todo lo relacionado con su comportamiento, de manera que además de activar la percepción necesaria para su ejecución típica, también debe detectar las sorpresas, contingencias y oportunidades que puedan surgir, y tratarlas convenientemente.

3.3.1. Selección de acción

Como vimos en la sección anterior, los esquemas motores tienen asociado estados para ayudar en su coordinación: DORMIDO, ALERTA, PREPARADO y ACTIVO. Por defecto un esquema entra en la arquitectura en estado DORMIDO, es decir, sin consumir potencia de cálculo alguna. Antes de que un esquema motor pueda tomar el control real debe dar tres saltos en su estado de activación. El paso de DORMIDO a ALERTA lo determina el padre, que considera conveniente preactivar a ese hijo porque puede llegar a ejecutar parte de su propio comportamiento si se dan las condiciones oportunas.

En estado de ALERTA significa que el esquema está despierto, consumiendo tiempo de computador para chequear periódicamente si se dan sus precondiciones. El esquema puede estar infinitamente en ese estado latente, inhibido, a la espera de que la situación le sea favorable. Estas precondiciones evitan que un esquema se active cuando la situación es muy diferente de aquella para la que ha sido programado, con ellas se asegura que la acción se ejecuta en un contexto en el que resulta conveniente. En lugar de unas precondiciones clásicas, discretas (se-cumplen-las-precondiciones, no-se-cumplen), en JDE se utiliza un sistema más flexible, descrito en la figura 3.9.

Está inspirado en los *mecanismos desencadenantes innatos* de etología [Lorenz, 1978]. En el esquema se almacena una cierta energía de activación que procede de los distintos estímulos externos, estímulos y motivación internos que hacen conveniente que ese esquema tome el control en la situación de ese instante. En la figura 3.9 se representa con el nivel mayor o menor de la zona rayada. Si esa energía es superior a cierto umbral, entonces el esquema considera que está en una situación favorable y pasa a estado PREPARADO, un avance más persiguiendo tomar el control del robot. Ese *umbral de activación* se representa en la figura 3.9 con la línea horizontal discontinua. Hay que resaltar que no hay propagación de esa energía de activación de un instante a otro, en cada uno se reevalúa completamente, no hay memoria. Como luego veremos, esto posibilita una rápida configuración.

Este planteamiento contempla la gradualidad en los estímulos y en cómo éstos afectan a la activación de los patrones de comportamiento. Por ello permite explicar fenómenos como la suma aditiva de estímulos, muy conocidos en la literatura etológica[4]. También incluye la posibilidad de

[4]Como luego detallaremos en la sección 3.4.1, un ejemplo de esta suma se presenta en el comportamiento de apertura

materializar fenómenos más complicados como el descenso del umbral ante la insatisfacción de cierta necesidad (que el esquema alivia) y la actividad en vacío. Además casa perfectamente con una descripción de la situación consistente en una cierta colección de estímulos, cada uno de ellos materializado en la salida de algún esquema perceptivo. Ese es justo el planteamiento de la percepción que hemos presentado en la sección anterior.

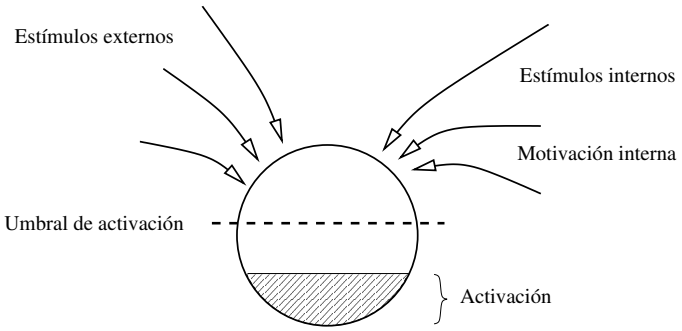


Figura 3.9: *Suma aditiva de estímulos, umbral de activación.*

Cuando un esquema pasa a PREPARADO trata de ganar la competición por el control contra otros esquemas motores PREPARADOS en el mismo nivel, y así promocionar a estado ACTIVO. Sólo los esquemas ACTIVOS toman el control de la actuación en ese nivel, que como hemos visto consiste en entregar las señales de activación (flechas negras en la figura 3.1) y los parámetros de modulación (hilera de cajas en la figura 3.1) a los esquemas de nivel inferior.

Esta asignación de estados a los esquemas que pueden ejecutarse para gobernar su coordinación es similar a la que se utiliza en la planificación de procesos de los sistemas operativos. En ellos hay que repartir el microprocesador entre los distintos procesos que quieren ejecutar, con el requisito de que en un instante sólo uno de ellos tiene el control del microprocesador. El sistema operativo atribuye a cada proceso un estado: PREPARADO, PENDIENTE DE ENTRADA-SALIDA, DETENIDO, DORMIDO, etc y los mueve de una cola a otra. Hay varias políticas para coordinarlos: basados en prioridades, *round-robin*, etc.

¿En qué influye el padre? tiene una influencia directa en cuanto a que decide preactivar a unos hijos y no a otros. Conviene destacar que la modulación paterna a través de parámetros puede no sólo sesgar el desarrollo del comportamiento del hijo, sino que también puede alterar las condiciones de su desencadenamiento, cambiar la desinhibición del hijo, su activación. Además en JDE el padre tiene funciones de arbitraje entre los hijos en PREPARADO. De hecho, distintos arbitrajes sobre los mismos esquemas básicos dan lugar a diferentes comportamientos observables.

El padre tiene cierto conocimiento sobre los hijos: qué hacen y cuándo se activan, aunque sea un conocimiento aproximado. En general, en JDE un esquema de control no necesita saber la percepción requerida para la ejecución de sus hijos, ya se encargarán ellos de activar los esquemas perceptivos que necesiten. No obstante, siempre despertará a sus esquemas motores hijo y a los esquemas perceptivos necesarios para resolver su activación.

En cuanto a esa activación, el padre modulará a sus hijos para que tiendan a activarse en situaciones diferentes. Para ello cuenta con la ayuda de los estímulos clave asociados a los esquemas motores hijos. Dicho de otro modo, en todo momento los esquemas perceptivos en ACTIVO dibujan un subespacio perceptivo (*subespacio de atención*) que corresponde a todos los posibles valores de los estímulos relevantes para cada nivel de los existentes (se muestra como el conjunto de areas blancas de la figura 3.10). Este subespacio de atención es un subconjunto de todos los posibles estímulos, porque no incluye los estímulos producidos por los esquemas perceptivos que están en DORMIDO (los cuales serían el área sombreada de la figura 3.10). La modulación del padre en los parámetros que afectan

del pico de la cría de gaviota [Lorenz, 1978]. Que el polluelo abra o no el pico depende de la combinación de varios estímulos: que vea una cabeza blanca redondeada, que vea un pico amarillo, que aparezca una mancha roja en ese pico, etc. Cada uno por separado no es suficientemente discriminante, pero la combinación de varios de ellos sí lo es. No es necesario que estén presentes todos, pues entonces sería demasiado discriminante.

al desencadenamiento de sus hijos trata de particionar este subespacio de atención en varias *regiones de activación* disjuntas. Estas regiones se definen como las áreas donde las precondiciones de un esquema motor hijo se satisfacen, y se representan en la figura 3.10 encerradas en líneas continuas. Una determinada situación corresponde con un punto en dicho subespacio, y puede caer en la región de activación de un determinado esquema motor u otro. Sólo el correspondiente esquema motor será activado, por lo tanto es la propia situación la que activa un único esquema por nivel entre los que están ALERTA. Éste mecanismo es un arbitraje de grano grueso, basado en regiones de activación.

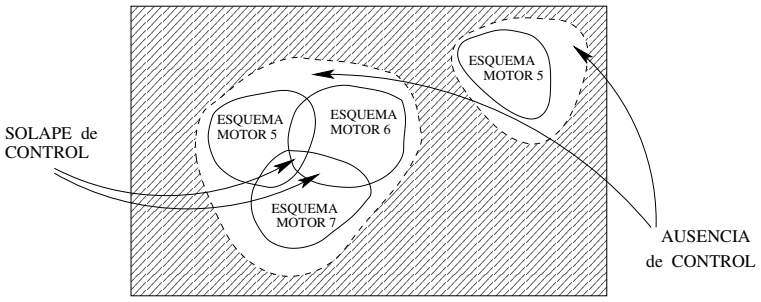


Figura 3.10: *Espacio perceptivo, subespacio de atención y regiones de activación.*

A pesar del arbitraje de grano grueso pueden aparecer situaciones donde más de un esquema motor por nivel satisface sus precondiciones. Esta situación se denomina *solape de control*, y se corresponde en la figura 3.10 con el solapamiento geométrico de las regiones de activación. También pueden presentarse situaciones donde ningún hijo se encuentra PREPARADO para la activación, es decir, la situación no está cubierta por ninguna región de activación y se produce lo que llamamos *vacío de control*. En ambos casos es el padre el que debe resolver el conflicto. Para ello el padre puede cambiar la modulación de sus hijos, o sencillamente arbitrar un ganador en ese nivel de la competición por el control, forzado o no. Bien sea eligiendo a dedo un ganador entre los hijos PREPARADOS, bien sea forzando a alguno de ellos a tomar el control cuando no hay ninguno preparado y todos permanecen inhibidos en ALERTA.

Es destacable que son los esquemas hijo quienes detectan estos fallos de control. Cada uno de ellos es iterativo y funciona asíncronamente con respecto a sus hermanos. En cada iteración comprueba su propia energía de activación y el estado de sus hermanos. Si él mismo se encuentra PREPARADO buscará si hay más hermanos en el mismo estado, en cuyo caso invoca al padre porque ha detectado un solape de control. Si él mismo se mantiene ALERTA porque no se dan sus precondiciones, busca entre sus hermanos algún otro que sí esté preparado. Si no lo encuentra entonces invoca al padre porque ha detectado un vacío de control. No hay problema en que todos los hermanos hagan estas comprobaciones periódicamente. Cuando surja un solape, el propio esquema que acaba de estrenar el estado PREPARADO lo detectará. Cuando surja un vacío, el propio esquema que deja de estar en ACTIVO lo detectará.

El mecanismo de selección de acción de JDE se apoya en dos pilares. Por un lado, en la modulación de los hijos para que sus regiones de activación sean disjuntas, y por otro lado, en un arbitraje explícito del padre cuando surgen fallos de control, bien sean solapes o vacíos. Tal y como está planteado, este mecanismo de selección de acción ofrece varias ventajas como su distribución, agilidad y escalabilidad.

Por su propia naturaleza es un algoritmo distribuido, por lo que no es necesaria la existencia de un árbitro central, como en DAMN [Langer *et al.*, 1994]. El sistema es homogéneo, sólamente existen padres e hijos. Son los hijos los que detectan la necesidad de arbitraje explícito, ejecutando su parte del algoritmo. Es la ejecución distribuida simultánea entre hijos y padre la que evita la necesidad de un nodo especial que centralice su coordinación. En caso de que detecten algún fallo de control, es el propio padre, el cual conoce el contexto del problema mejor que nadie, quien arbitrará una coordinación adecuada. El padre conoce el contexto para tal arbitraje, y por lo tanto puede hacerlo muy específico y ajustado para la tarea en cuestión.

Hay una competición por nivel, que tiene lugar cada vez que algún hijo realiza una iteración. Al realizarse en cada iteración, y en todos los hijos en ALERTA o PREPARADOS, cualquier cambio en la situación que recomiende la activación de otros esquemas se detecta enseguida. Esto permite

una rápida reconfiguración si la situación cambiara. Por lo tanto, la selección de acción responde rápidamente a los cambios en el entorno, y el mecanismo es ágil. Además es eficiente porque sólo se esfuerza en coordinar explícitamente a los hijos cuando hay dificultades. Mientras la coordinación de grano grueso, que es implícita (sin coste computacional), consigue que sólo un hijo esté PREPARADO, no hay que resolver ningún problema, lo cual hace muy eficiente al algoritmo.

Además, es un algoritmo que se escala bien cuando la complejidad del sistema crece. Normalmente para abordar comportamientos más complicados la cantidad de esquemas existentes se incrementará. Sin embargo, el número de hermanos preactivos compitiendo en un nivel nunca será muy grande, para mantener la complejidad acotada en cada contexto. Por ello no es de esperar que crezca exponencialmente la dificultad de modular las regiones de activación para que sean disjuntas, y los solapes nunca serán de muchos hijos a la vez.

La selección de acción presentada ofrece una doble orientación, por un lado orientada a la consecución de los objetivos (alerta), y por otro lado orientada a la situación del entorno (activación final). Esta doble orientación incorpora los principios que Maes señala como convenientes de todo mecanismo de selección de acción [Maes, 1989b; Maes, 1990]. Supone un compromiso entre la finalidad, u orientación a los objetivos, que viene de arriba a abajo en la jerarquía, y la tendencia a reaccionar ante la situación del entorno.

En JDE, si un esquema fue despertado por un padre, es porque éste pensó que el esquema era bueno para conseguir los objetivos del padre en alguna situación. La activación de padres a hijos que hemos visto es simplemente una predisposición, y conlleva ya cierta orientación a los objetivos. Sólo los esquemas despertados por el nivel superior están habilitados para ganar el control, pero es finalmente la situación percibida la que elije uno entre ellos como ganador para ese nivel. Dicho de otra forma, el nivel superior propone un conjunto de alternativas y las condiciones sensoriales disponen, dentro de ese subconjunto, la más conveniente. El ganador reúne por lo tanto una doble motivación: orientada a tarea y orientada a la situación. Se puede pensar que los esquemas que carecen de alguna de ellas no acumulan suficiente motivación para su activación y permanecen silentes, en ALERTA o DORMIDO.

En el mecanismo propuesto no hay inhibición lateral explícita entre hermanos, en el sentido de Ludlow [Ludlow, 1976], porque la activación no interviene directamente en el proceso de competición. Se calcula primero la activación, para pasar al estado PREPARADO y *después* se compite con los hermanos en esa misma disposición. Se han separado para simplificar la coordinación y reutilización en varios contextos del mismo esquema. La activación es inherente a cada esquema, no depende de los hermanos, mientras que en el algoritmo de Ludlow sí. Es interesante resaltar que sí hay inhibición lateral a posteriori, puesto que aquél que gana frena a todos sus hermanos.

Otra opción para materializar la selección de acción hubiera sido elegir como ganador directamente el hermano con una activación más intensa. Sin embargo, esto dificultaría la reutilización del mismo esquema en diferentes contextos. El número que refleja la activación de un esquema es algo inherente a él, con su propia escala. No depende de qué otros esquemas están compitiendo con él por el control, que vienen fijados por el contexto. Si ajustáramos los valores numéricos para la competición entre un subconjunto de esquemas, en cierto contexto, ese ajuste previsiblemente no valdría cuando el esquema se utiliza en otro contexto, contra otros hermanos.

El hecho de que la selección de acción en JDE sea excluyente (*winner-takes-all*) y sólo pueda haber un único ganador por nivel, fuerza a que la arquitectura sea esencialmente monoobjetivo en cada instante. Esto dificulta la combinación de preferencias y las soluciones de compromiso, porque impone que en cada instante, en cada nivel de abstracción, se persiga un único objetivo: el del esquema ganador.

Sin embargo, es usual que el robot tenga muchos objetivos que querrá satisfacer con su comportamiento. Por ejemplo, en sistemas con cierta influencia etológica, los objetivos se explicitan como el mantenimiento en un rango de tolerancia de cierto conjunto de variables internas como el hambre, sed, temperatura, etc. [Cañamero *et al.*, 2002] (control homeostático). ¿Cómo se satisfacen varios objetivos con JDE? Con la alternancia temporal, ahondando en la secuenciación de esquemas. La idea es que los diferentes objetivos, a veces incompatibles, influyan en qué esquemas resultan activos en cada momento. Por ejemplo, los valores de estas variables internas deben tener influencia en la selección de acción, lo cual permite alternar los esquemas activos a lo largo del tiempo para conseguir

todos los objetivos. De modo similar a Tyrrell [Tyrrell, 1994], el recurso a repartir es el tiempo que se está gobernando el cuerpo del robot. Su animal simulado está durmiendo en un cierto periodo, en otro comiendo, etc.

Otra manera de combinar varios objetivos consiste en introducir distintos criterios dentro del mismo esquema. Cuando ese esquema gane su competición por el control, será su propio código el que establezca las soluciones de compromiso entre los criterios que explícitamente tiene en cuenta.

3.3.2. Modulación continua para usar funcionalidad

Una de las apuestas de nuestra arquitectura es la interfaz normalizada de los esquemas empleando activación y modulación. Éste es el modo en que unos esquemas utilizan la funcionalidad de otros en JDE. Esta interfaz es similar a la empleada por Gat en su arquitectura ATLANTIS [Gat, 1992], y al uso que hacen las capas deliberativas de la capa reactiva en las arquitecturas híbridas. Y no es el único modo de utilizar la funcionalidad que se ha empleado en robótica a lo largo de su historia. Por ejemplo, la *abstracción procedimental* es típica de los sistemas deliberativos; la *subsunción* es típica de las arquitecturas basadas en comportamientos de Brooks y su escuela.

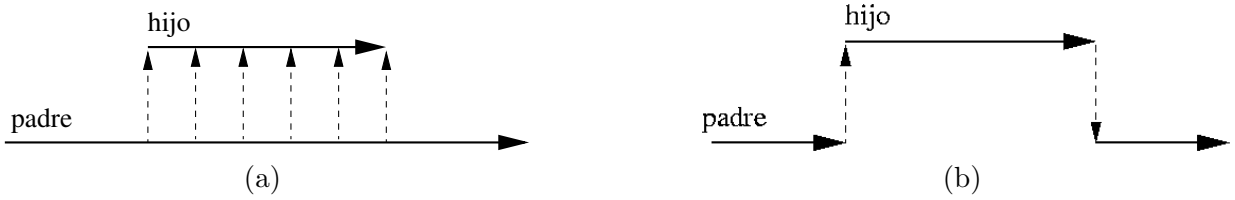


Figura 3.11: *Modulación continua (a) frente a abstracción funcional (b).*

En la *abstracción procedimental* (o funcional) el cliente de la funcionalidad invoca al llamado, para que éste materialice su funcionalidad ejecutándose, y espera un retorno, que es el error o éxito en esa ejecución. Tal y como muestra la figura 3.11(b), mientras ese procedimiento del llamado se está ejecutando, el llamante está bloqueado, esperando, y por lo tanto no es sensible a los cambios en el entorno, las contingencias, las oportunidades, etc. que ocurran en ese momento. El llamante pone los parámetros en el momento de la llamada y luego espera a que el llamado retorne el control. Esto quizá es una herencia del pasado, pues históricamente los primeros en generar comportamientos eran programadores, a los que el concepto de subrutina y librería era muy familiar. Esta abstracción responde perfectamente a las funciones de programación: existe un único flujo de control, se llama a las funciones con unos parámetros y éstas devuelven resultados tras ejecutarse.

Esta abstracción funcional tiene dos consecuencias nocivas. En primer lugar, si en ese momento hay alguna sorpresa que conviene detectar al llamante, obliga a que el llamado se dé cuenta de ella, pues el llamante va a estar bloqueado mientras tanto. Esto complica el código de monitorización: si el llamado puede tener varios llamantes, es decir, muchos usuarios de sus servicios, el llamado debe ser consciente de las contingencias que les atañen a todos ellos, pues debe estar pendiente de ellas cuando le llamen. En segundo lugar, el llamante sólo modula al llamado en un instante de tiempo, el de la llamada. Es pues una modulación puntual, tal y como ilustra la única flecha ascendente en la figura 3.11(b). Como menciona Brooks en [Brooks, 1991c], la abstracción es un arma peligrosa.

Un segundo modo de utilizar funcionalidad es la subsunción. Como vimos en el capitulo 2, el cliente de la funcionalidad, que suele ser un autómata en un nivel de competencia superior, puede suplantar las entradas y suprimir las salidas del autómata que quiere utilizar. La suplantación de entradas se puede ver como un cambio de parámetros, similar al empleado en JDE, sin más que considerar la entrada del autómata como un parámetro más. En general, las entradas tienen un carácter más vivaz que los parámetros de modulación, y por ello con la suplantación de entradas de la subsunción estamos obligando al cliente (el módulo que utiliza la funcionalidad del otro) a ir tan deprisa como el ritmo de variación que esperara el reutilizado (el módulo que está siendo reusado) en sus entradas. Por otro lado, la supresión de salidas permite al autómata cliente reimplementar la funcionalidad equivalente a la del llamado, pero sesgada con sus propios objetivos. No hay reutilización en sentido estricto. Como

veremos posteriormente en la sección 3.5, la necesidad consustancial en subsunción de anular lo que no nos interesa dificulta el crecimiento de la arquitectura.

La abstracción de esquemas empleada en JDE ofrece una interacción más continua que la abstracción funcional. Esto es así porque la modulación continua, en lugar de puntual, permite un funcionamiento más flexible. En JDE el esquema llamante y el esquema llamado funcionan en paralelo, y el llamante modula continuamente al llamado, a su ritmo de iteraciones. Esto se aprecia en las flechas verticales ascendentes de la figura 3.11(a). Manteniendo la activación, los parámetros de modulación se pueden cambiar en todo momento. Esto tiene en general el coste de una sobrecarga de cómputo, pero al ser fácilmente paralelizable, se puede superar ese cuello de botella. Esta continuidad permite adaptar la modulación a las nuevas circunstancias, bien del entorno, bien de los propios objetivos del cliente. Esta flexibilidad es posible porque tal y como hemos utilizado la abstracción en JDE, la hemos despojado de cualquier horizonte temporal, sólo es conceptual. Por lo tanto, puede cambiar de un instante al siguiente.

3.3.3. Monitorización distribuída continua y reconfiguración

Otro aporte relevante de JDE es la monitorización distribuida. Cuando un esquema padre utiliza un esquema hijo, no se queda esperando a que termine, sigue funcionando en paralelo. El padre debe monitorizar si su hijo consigue o no su objetivo. Así, la monitorización del conjunto de esquemas despiertos en cierto nivel se sitúa en el padre, que tiene el contexto perfecto para interpretar los fallos o éxitos de sus hijos y responder en consecuencia. Si tal hijo no consigue su objetivo, el padre sabe qué es lo que debe hacer. El código de monitorización queda fragmentado y distribuido en los diferentes padres, que son responsables de la monitorización en su contexto. De este modo, la monitorización sigue la misma organización jerárquica que la activación.

El uso de flujos independientes con modulación continua en JDE facilita enormemente la monitorización. Como llamante y llamado siguen ejecutando la monitorización es más sencilla: en cada iteración el padre monitoriza el desarrollo de su hijo. De este modo no se carga a los módulos reactivos con la responsabilidad de explorar las excepciones o condiciones que le interesan a los de nivel superior. Por el contrario, en el caso de la abstracción procedimental, se carga a los módulos inferiores con la responsabilidad de detectar las sorpresas de los superiores, porque éstos se quedan bloqueados cuando invocan a aquellos. En ese caso, como son muchos módulos diferentes los que pueden invocar al inferior, el código de monitorización de todos ellos se acumula en el llamado, el cual debe detectar los eventos significativos. Queda pues lejos de cada llamante, que es quien realmente sabe el contexto de su monitorización.

En JDE no hay comunicación explícita de retorno, ha-ido-bien o ha-ido-mal, entre el hijo y el padre. El padre debe poder percibir por él mismo, con sus propios esquemas perceptivos, el éxito o fracaso de su hijo. Es una comunicación indirecta, a través del mundo: ya se dará cuenta el padre de que el hijo ha terminado, ha cumplido su labor. En este sentido JDE comparte con Brooks la *comunicación a través del mundo* entre los distintos esquemas, incluso de padres a hijos. La única comunicación interna permitida corresponde a la activación, modulación y lectura de datos (de los perceptivos).

No sólo hay que monitorizar continuamente el resultado de las acciones para saber si han tenido éxito o fracaso, también hay que estar pendiente del mundo por si surgen oportunidades o contingencias que no habíamos tenido en cuenta. Las oportunidades y las contingencias son entendidas en el sentido de Agre y Chapman [Agre y Chapman, 1990], como razones para abortar o revisar un plan. Quizá lo más didáctico es mencionar su mismo ejemplo para explicar las diferencias entre ambas: "cuando decides ir a la cocina a por un bolígrafo que necesitas, encontrarte con la puerta de la cocina cerrada es una contingencia y encontrarte con un bolígrafo en la mesa del salón en el que estás es una oportunidad". En general las contingencias se detectan como la insatisfacción de algunas precondiciones o como el fallo de cierta parte del plan. Las oportunidades suelen ser más difíciles de detectar puesto que obstruyen menos, no impiden la acción.

Ciertos sistemas deliberativos sólo replanifican cuando no puede ejecutar el plan según lo previsto. Este planteamiento detecta bien las contingencias, pero no las oportunidades, pues éstas no se reflejan en un incumplimiento del plan actual. Si esa oportunidad surge después de suministrar el plan, se pierde. La replanificación cuando el plan actual no puede llevarse a cabo hace que en la escuela

deliberativa sólo se reaccione ante las contingencias.

Una buena monitorización debe poder reaccionar no sólo antes las contingencias, sino también aprovechar las posibles oportunidades que surgan. La monitorización continua permite aprovechar tanto las contingencias como las oportunidades, aunque nada es automático, hay que programar al sistema para que las busque y las aproveche. En este sentido un esquema de actuación debe activar no sólo a los esquemas de percepción necesarios para su ejecución típica, sino también a aquellos que le permiten detectar esas contingencias u oportunidades que puedan aparecer. De manera similar a los esquemas perceptivos en [Arkin y Balch, 1997], que se activan para detectar los eventos que provocan el cambio de estado, pero que no afectan a la ejecución de los esquemas motores actuales. Esto conlleva un mayor coste computacional porque hay muchas cosas funcionando en paralelo. No obstante este coste se ve acotado porque normalmente no todos los esquemas existentes se encuentran ejecutando, sólo un subconjunto reducido.

En cierto instante la arquitectura tiene la forma de cierta jerarquía de esquemas la cual genera cierto comportamiento y está persiguiendo ciertos objetivos. ¿Cómo responde el sistema a las sorpresas? ¿Cómo responde un esquema padre ante el fallo de un hijo suyo, la aparición de una contingencia, una oportunidad, o simplemente un cambio en la situación del entorno? Si los cambios necesarios son ligeros puede remodular a los hijos actuales, para cambios de mayor calado puede forzar la reconfiguración de la jerarquía desde ese nivel. Visto desde una perspectiva global, si la situación cambia mínimamente los esquemas activos actuales pueden manejarla y simplemente generan comandos motores o modulación a sus hijos ligeramente distinta. Si la situación varía un poco más, quizá algún esquema activo de cierto nivel ya no es apropiado y el propio entorno activa otro esquema motor en ese nivel que estaba preparado para reaccionar a ese cambio. Esta gradación en niveles es similar a la granularidad de actuaciones que menciona Laird [Laird y Rosenbloom, 1990] para el sistema de producción SOAR.

Por ejemplo, en la figura 3.5 se muestra el estado de la jerarquía en cierto momento. Hay muchos esquemas activos simultáneamente: uno motor por cada nivel, y varios perceptivos. Sin embargo, hay muchos más esquemas despiertos, ejecutando código de CPU: todos aquellos que están ALERTA pero no se dan sus condiciones de activación, todos aquellos PREPARADOS que no han ganado la competición por el control en su nivel. Todos ellos marcan las posibles configuraciones internas a las que se puede saltar desde la configuración actual. Dependiendo de si el cambio surge en un nivel más alto o más bajo de la jerarquía actual, la reconfiguración será de mayor o menor calado. Si ocurre en un nivel superior, todos los descendientes del esquema que deja de estar activo se inhabilitan, pasando a reconfigurarse y activarse los descendientes del nuevo esquema ganador a ese nivel.

En el caso de cambios muy significativos, éstos pueden provocar un fallo de control en cierto nivel, o que el padre decida cambiar a sus hijos (dormir hijos inútiles, despertar a otros que ahora son relevantes), o propagar el conflicto hacia su propio padre, hacia arriba en la jerarquía. En este punto JDE es similar a TCA [Simmons, 1994], donde hay una recogida jerárquica de las excepciones, que van subiendo hasta que alguien las trata. Esto permite que la misma excepción se trate de modo diferente en diferentes contextos.

La reconfiguración puede añadir niveles a la jerarquía o reducir su número. Los niveles no son estáticos sino orientados a la tarea, como en los árboles de tareas de TCA [Simmons, 1994]. Los esquemas no pertenecen a ningún nivel en particular. Pueden situarse en diferentes niveles, probablemente con otros parámetros, para desempeñar otro comportamiento global. De esta manera los esquemas pueden reutilizarse en diferentes niveles dependiendo del comportamiento observable deseado. En [Budenske y Gini, 1997] también se presenta la reconfiguración dinámica. En cuanto a la velocidad de reconfiguración, el mecanismo de selección de acción de JDE dota al sistema de gran reactividad. Las energías de activación no se acumulan de una iteración a otra, sino que se recalculan en cada instante. De este modo, la jerarquía puede reconfigurarse en cada nivel a la velocidad de iteración de los esquemas de ese nivel.

Los cambios que se realicen han de diseñarse para que funcionen correctamente y por ejemplo, no se produzcan oscilaciones. Como señalaremos posteriormente en la sección 3.5.6, una desventaja de este planteamiento de la monitorización y reconfiguración es que requiere la anticipación de las posibles sorpresas, contingencias y oportunidades, en primer lugar para activar los esquemas perceptivos que las detecten. También deben prepararse las respuestas adecuadas ante ellas. En este sentido podemos

tomar el mismo principio que TCA: programar el caso regular, y a medida que aparecen caso extraños, incorporar en la arquitectura los esquemas necesarios para detectarlos (percepción) y resolverlos (actuación), en los contextos en los que puedan aparecer y deban ser tratados. Además hay que señalar que en tiempo de ejecución no hay reconocimiento de que un plan ha fallado, ni se identifican explícitamente como tales contingencias y oportunidades, simplemente aparecen nuevos estímulos en la situación ante los cuales se reacciona.

3.3.4. Ejemplo: comportamiento ir-a-punto

Describiremos ahora un caso específico para ilustrar mejor los mecanismos presentados en esta sección. En concreto explicaremos brevemente cómo se puede generar el comportamiento `ir-a-punto` con una jerarquía de tres niveles, dejando los detalles para la sección 5.4.1. La conducta observable consiste en que el robot alcance un cierto punto de destino en su entorno local y sortee a los obstáculos que puedan aparecer en su camino. Tal y como muestra la figura 3.12(a), el comportamiento ha sido generado con una arquitectura de tres niveles. El esquema motor superior activa dos esquemas perceptivos y tres motores como hijos: `para`, `sigue-pared` y `avanza`. El ganador del control entre ellos activa y modula un esquema hijo, `control-en-velocidad` que envía los comandos de movimiento directamente a los motores. En cuanto a los esquemas perceptivos, se utilizan varios que recogen las medidas sensoriales, como el `sónares` que da información de proximidad a objetos, el `odómetros` que entrega información de los cuenta vueltas. Cada esquema tiene su propio ritmo, su tiempo de ciclo.

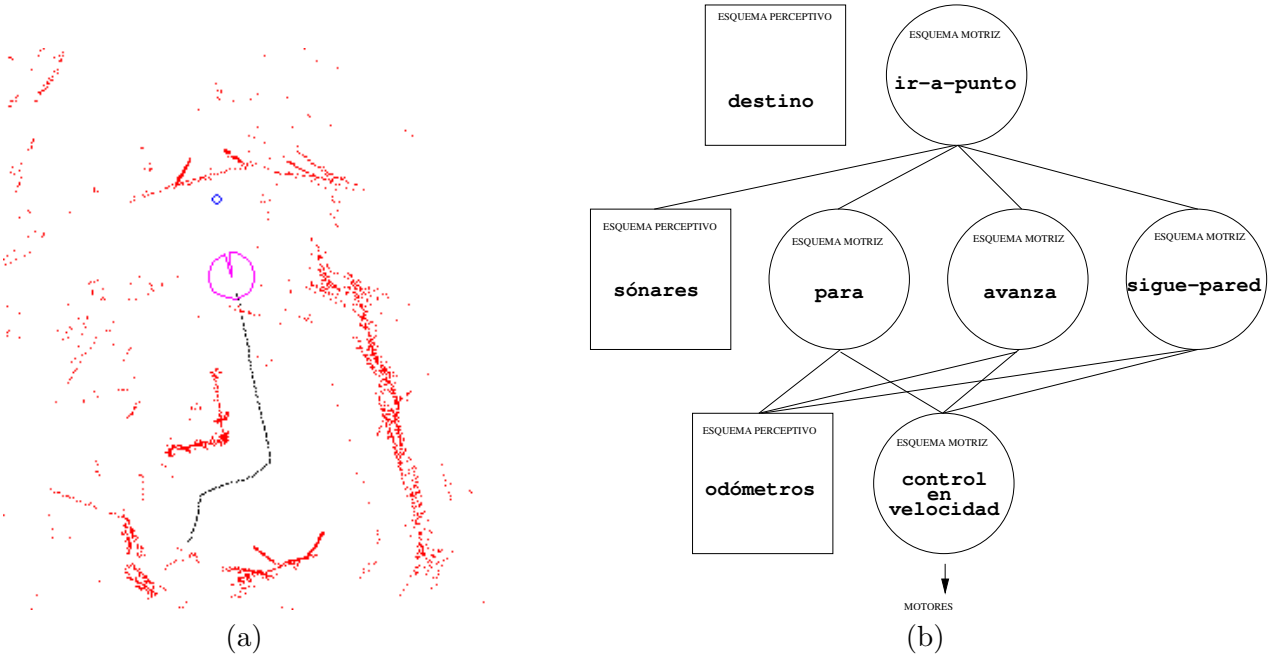


Figura 3.12: *Trayectoria descrita por el comportamiento* `ir-a-punto` *(a) y jerarquía de esquemas que lo genera (b).*

El nivel más bajo de esta jerarquía tiene un único esquema motor `control-en-velocidad`, que ofrece dos parámetros fundamentales v (velocidad de tracción) y ω (velocidad de giro) para que otros esquemas lo usen a través de la modulación. La activación de este esquema no depende de ninguna condición sensorial y basta con que un padre lo sitúe en ALERTA para que promocione a PREPARADO. No obstante para su ejecución lleva asociado un esquema perceptivo, `odómetros` que proporciona las lecturas de los odómetros acoplados a los motores, y que le permite establecer el control en velocidad con rápida realimentación PID.

El esquema `para`, detiene al robot si hay algún obstáculo próximo. Este esquema se activa cuando hay un obstáculo más cerca de cierta distancia umbral, `distancia-seguridad`. Éste sería su estímulo clave, que resulta modulable porque variando ese parámetro se cambian los escenarios en los que se satisfacen las precondiciones del esquema. En cuanto a su acción, es siempre la misma: frenar al máximo

los motores y la ejecuta activando al agente `control-en-velocidad` y modulándolo con velocidades nulas de giro y tracción. Suele asociarse con un esquema perceptivo que detecte la proximidad de obstáculos, en cualquier dirección. También se disponde de un esquema `sónares` que se encarga precisamente de recoger las medidas de los sensores de ultrasonido al ritmo al que vienen.

El esquema `sigue-pared` acepta los datos de proximidad de obstáculos y mueve al robot paralelo al obstáculo más cercano. Reacciona al obstáculo sólo si está más cerca que cierta `distancia--susceptible`. El comportamiento genuino de este esquema es seguir la pared, sin embargo, en este contexto contribuirá al de `ir-a-punto`. Esto supone un ejemplo de reutilización de un esquema motor, ya que usándolo con otra región de activación y otros hermanos puede ayudar a generar otro comportamiento global completamente diferente. Su salida se expresa como la activación y modulación del esquema `control-en-velocidad`. En cuanto a sus entradas, necesita la información de distancia a los obstáculos, que en éste caso se la proporciona el esquema `sónares`. Este esquema motor ha sido implementado como un controlador borroso, que entrega una salida en cada iteración. Para contornear el obstáculo selecciona el sensor que entrega la distancia más corta y gira el robot hacia la perpendicular al radio central que sale de ese sensor. De los dos sentidos posibles de la perpendicular escoge el más próximo a la orientación actual del robot.

El esquema `avanza` desplaza al robot en cierta orientación. Su activación depende de la existencia de algún obstáculo próximo en la orientación objetivo. Si lo hay, el esquema se inhibe, no pasa a estado PREPARADO. El padre modula la activación de este esquema fijando la orientación objetivo y la distancia de seguridad admisible en esa dirección. El vacío en la dirección objetivo sería el estímulo clave asociado a la activación del esquema `avanza`. En cuanto a su ejecución, si no hay obstáculos en esa dirección y está alineado el robot avanzará, más rápido si el camino está más despejado. Si se encuentra desalineado con esa orientación, girará hasta aproximarse a esa dirección. Por lo tanto, necesita la información de los obstáculos a su alrededor tanto para activarse como para ejecutarse, y se la proporciona el esquema `sónares`. Como se verá en el capítulo 5, éste esquema ha sido implementado como un controlador borroso, que en cada iteración busca hueco para avanzar en la dirección adecuada. Las reglas borrosas aceleran la rotación correctora proporcionalmente al error de orientación. La velocidad de tracción disminuye en la medida que lo hace la distancia al obstáculo en la dirección objetivo, y es nula para errores angulares elevados.

Una vez descritos los tres esquemas motores que aparecen en el segundo nivel de la figura 3.12(a), explicaremos ahora el esquema `ir-a-punto`, que utiliza como hijos a los anteriores. En la implementación actual, recibe directamente la activación desde el operador humano, que señala un punto objetivo pinchando con el ratón en una interfaz gráfica. La distancia a ese punto objetivo insatisfecho es el estímulo clave de este esquema. Cuando esa distancia caiga por debajo de cierto umbral, `distancia-a-objetivo`, el propio esquema se inhibe (durmiendo a todos sus hijos) y el objetivo se considera logrado. Tanto la posición del objetivo como ese umbral de distancia se consideran parámetros del esquema. En cuanto a la información de entrada necesaria, este esquema sólo necesita saber su posición relativa respecto al punto destino, y se la proporciona un esquema perceptivo bautizado como `destino`, que se ejecuta 5 veces por segundo. No hay ninguna corrección de los datos odométricos, que son susceptibles de acumular error. Como el punto de destino es cercano (navegación local), asumimos que el error que se puede acumular es acotado.

Este esquema materializa su objetivo despertando a los tres esquemas anteriores y modulándolos para que su activación sea más o menos disjunta. Al esquema `para` le fija la `distancia-de-seguridad` en 20cm. Al esquema `sigue-pared` le hace sensible sólo a obstáculos más próximos que 100cm. Al esquema `avanza` le establece la orientación a seguir como aquella que conduce al punto objetivo, y la distancia de seguridad admisible en esa dirección a 100 cm. A grosso modo se consigue que las activaciones sean disjuntas y la propia situación sensorial elige cual de los tres esquemas hijo gana la competición por el control y pasa a ACTIVO en cada momento. Como vimos anteriormente, el padre nunca fuerza la activación total de uno de sus hijos, simplemente despierta a estos tres de la colección de esquemas disponibles, y parametriza los estímulos clave a los que responden. Para que uno de los tres esquemas hijo se active realmente su estímulo clave (obstáculo muy cercano, obstáculo cercano a bordear y espacio vacío por el que avanzar), debe aparecer en la situación. El resto de los esquemas existentes permanecen DORMIDOS aunque su estímulo clave aparezca, porque ningún padre

los despertó. De este modo la activación tiene la doble orientación que comentamos anteriormente.

La activación de los esquemas `avanza` y `sigue-pared` es más o menos disjunta, dado que uno reacciona ante el espacio vacío mayor de 100 cm en la dirección de avance y el otro ante un obstáculo más cercano a esos 100 cm. Pueden producirse situaciones puntuales con vacíos de control o con solapes, en las que el padre `ir-a-punto` se invoca para un arbitraje explícito. Si el esquema `avanza` colisiona con el `para` entonces el padre inhibe a éste último, asumiendo que se ha encontrado un obstáculo muy próximo, pero que no está en la dirección de avance. Si el esquema `sigue-pared` trata de activarse a la vez que `para` entonces se inhibe al éste último con idea de sortearlo. Si ninguno de los tres hijos se encuentra PREPARADO, el padre fuerza la activación de `para` como acción por defecto en espera de que la situación en el entorno cambie y seleccione a otro esquema.

En la figura 3.12(a) se muestra la trayectoria seguida en una ejecución típica de este comportamiento. En ella el punto objetivo aparece como un pequeño círculo en la parte superior. A efectos de visualización se incluye la nube de puntos como una acumulación de medidas sónar, que da idea del perfil de los obstáculos que rodeaban al robot en su movimiento. Al comienzo de su trayectoria no hay ningún obstáculo cercano, ni siquiera en la dirección hacia el punto objetivo, por ello se activa el esquema `avanza`. Aunque no se aprecia en la figura, el robot estaba apuntando incialmente hacia abajo, por lo que `avanza` implementa un giro de reorientación antes de comenzar el desplazamiento. Una vez que el robot ha avanzado un tramo, de repente un objeto irrumpe en su trayectoria y el esquema `sigue-pared` gana la competición de control mientras el obstáculo interfiere el camino hacia el objetivo. Una vez superado, el esquema `avanza` retoma el control y conduce hacia el destino.

Es interesante resaltar que ninguno de los tres esquemas del segundo nivel utiliza los *odómetros*. Todos son reactivos y basan sus decisiones sobre la información instantánea que supone la última lectura sónar, como la mostrada en la figura 3.13(a). La poca fiabilidad de los sónares puede acarrear una menor calidad de sus decisiones.

Dos de los esquemas motores utilizados en este comportamiento se han implementado como controladores borrosos, pero la arquitectura permite cualquier otra implementación. De hecho, en JDE la elección del método más apropiado de control se deja al gusto del diseñador, siempre y cuando se respete en cada esquema el patrón iterativo, suspendible y modulable. En este sentido, el comportamiento `ir-a-punto` se puede plantear de otros muchos modos. Por ejemplo, la parte perceptiva se puede resolver con un único esquema que se encargue de dibujar una descripción completa de la ocupación en el entorno local del robot, tal y como se hizo en [Lobato, 2003]. Si no se necesita estructura, entonces una rejilla de ocupación es suficiente. También se puede plantear como una colección de varios esquemas perceptivos, cada uno de los cuales está buscando obstáculos reales sobre los que apoyarse. Por ejemplo, así se hizo en la segmentación con EEM [Cañas y García-Alegre, 1999a] de la rejilla de ocupación. Cada uno de los esquemas puede encontrar o no el estímulo real que busca. Típicamente se lanzan más esquemas de los obstáculos reales, y habrá varios esquemas que no logran encontrar evidencias sobre la que sustentar su estímulo.

En cuanto al lado de la actuación, ésta se puede organizar como un único esquema de actuación que lanza un planificador clásico sobre la descripción completa del entorno local. El tiempo de reacción breve estaría asegurado, al tener el espacio sobre el que se lanza unas dimensiones pequeñas. Esta implementación tendría énfasis deliberativo, en el sentido de manipulación simbólica. Sin embargo, no comporta ninguna abstracción temporal y su ejecución es prácticamente instantánea, como una reacción. Además resuelve el problema de elegir por qué lado se sortea el obstáculo intermedio (¿por la derecha del obstáculo o por su izquierda en la figura 3.12(a)?), eligiendo aquel que favorezca el progreso al destino.

Otra opción es organizar la actuación con énfasis reactivo, como un esquema que implementa fusión entre las reacciones de repulsión a cada uno de los obstáculos percibidos y la atracción hacia el objetivo. Esta implementación resalta el hecho de que dentro del esquema se puede tener en cuenta la información procedente de varios esquemas perceptivos a la vez, y materializar la fusión de comandos intraesquema. Un esquema puede ser un conjunto de reglas y fusionar la salida de todas ellas.

Como hemos visto esta conducta `ir-a-punto` podría haberse generado de otros modos, incluso muchos de ellos más eficientes que el realizado (un controlador ad hoc que tuviera como entradas los

sensores y como salidas los actuadores directamente). Sin embargo, este modo de resolverlo facilita su integración en un sistema que, además de éste, ofrece otros comportamientos más. Además, facilita la reutilización de partes de este comportamiento.

3.4. Percepción

Una vez vista la arquitectura en su conjunto en la sección 3.2, y sus detalles más relacionados con la parte de actuación en la sección anterior, en este apartado profundizaremos en los mecanismos y las facilidades que ofrece JDE para organizar la percepción del sistema. No describimos técnicas concretas para percibir algunos estímulos a partir de ciertos sensores, sino la organización de la percepción, su inserción en la arquitectura y su imbricación con el control. En el capítulo 5 describiremos algunas técnicas perceptivas concretas, las cuales se encuadran en el marco teórico que detallamos a continuación.

Como mencionaremos en la tabla 3.5.1, uno de los principios básicos de JDE establece que el objetivo fundamental de la percepción es proporcionar información al sistema para tomar decisiones. Como hemos visto en la sección anterior, las decisiones en JDE se han fragmentado y se han organizado de modo jerárquico, es natural por lo tanto que la información sobre las que éstas se toman también siga esa fragmentación y esa ordenación. La unidad básica sigue siendo el esquema, en este caso perceptivo. La percepción se divide en pequeñas unidades que son, por su propia naturaleza, iterativas, activables, suspendibles e incluso se modulan, de un modo similar a como operan los esquemas de actuación. Veremos que esta fragmentación y su organización en jerarquías confiere a la arquitectura mecanismos perceptivos como la atención, la interpretación y los estímulos estructurados.

Proponemos que la percepción esté distribuida en esquemas perceptivos, cada uno de los cuales puede producir distintas *piezas* de información. A cada una de esas piezas de información la llamaremos *estímulo*, pudiendo ser los datos sensoriales crudos que el esquema recoge o una información más elaborada acerca del entorno o del propio robot que el esquema construye (por ejemplo, un mapa, una puerta, etc.). Además de estímulos específicos para la actuación de cierto esquema motor, o para su activación, pueden existir estímulos más genéricos que sean utilizados simultáneamente por varios esquemas, o en diferentes instantes por esquemas distintos. Esta posibilidad de reutilización es una de las razones a favor de incluir el procesamiento de los estímulos en sendos esquema perceptivos, y en contra de incluirlo en el código del esquema motor que lo necesita, pues haría más difícil su reutilización.

Normalmente cada esquema elabora un estímulo. Se pueden agrupar varios estímulos dentro del mismo esquema perceptivo, para que se actualicen al mismo ritmo y por el mismo flujo de ejecución, cuando hay criterios funcionales o semánticos que invitan a ello (por ejemplo, que se consulten siempre de modo conjunto). No obstante, para facilitar la reutilización de unos y no de otros, lo común es separar cada estímulo en un esquema.

La Real Academia define estímulo como un *agente físico, químico, mecánico, etc., que desencadena una reacción funcional en un organismo*[Española, 1992]. Para nosotros será la unidad operativa de información, la representación interna, de alguna característica o hecho relevante de la realidad. Por ejemplo, para el comportamiento `sigue-pelota`, la propia pelota será un estímulo, aunque también lo será cualquier obstáculo con el que deba evitarse el choque.

Los estímulos se materializan en variables que todos los otros esquemas pueden leer. Las variables son visibles por todos los esquemas que lo necesiten, con independencia del nivel en qué estén. Los estímulos se pueden ver como variables internas ancladas a la realidad de modo dinámico. El anclaje a la realidad (*anchoring*), tal y como lo define Coradeschi [Coradeschi y Saffiotti, 2001], lo materializa el propio esquema, que asegura una correspondencia entre la variable y la realidad. Ese anclaje es dinámico, si la realidad cambia, la variable interna debe reflejarlo. Cuando un esquema perceptivo no está ACTIVO sus variables existen, y en ese sentido pueden ser leídas, pero no están actualizadas y por ello contienen valores espúreos. Cuando alguien active ese esquema, las variables volverán a tener valores con sentido. Hay una sutil diferencia entre variables y estímulos. Mientras que las primeras son entes pasivos, los segundos son activos, en el sentido de que incluyen el flujo de ejecución que busca la información pertinente en la realidad y la incorpora a las variables internas. El estímulo y el

esquema perceptivo están indisociablemente unidos. El esquema busca al estímulo real en el mundo y cuando está presente lo describe actualizando periódicamente ciertas variables internas. El esquema implementa el cómo se construye el estímulo.

La información útil para cierta tarea se puede expresar como una colección de estímulos. El sistema global tiene muchas tareas que ejecutar, y como hemos visto, se estructuran en una jerarquía dinámica de esquemas motores que las llevan a cabo. Homólogamente la colección de estímulos de todas esas tareas se organiza, de nuevo, en jerarquía. Los esquemas perceptivos se asocian a los esquemas motores para los cuales proporcionan información y se activan a la par que ellos.

También en percepción JDE normaliza el interfaz entre los distintos esquemas y establece la jerarquía como principio organizador. En cuanto a su interfaz, los esquemas perceptivos se activan y desactivan a voluntad, como sus homólogos motores. Al estar distribuido en esquemas que se activan o no, el sistema perceptivo en JDE se puede entender como una colección dinámica de estímulos relevantes. En cada momento cada uno de ellos puede buscarse o no. También admiten modulación a través de parámetros que sesga su propio funcionamiento. Por ejemplo, el esquema `pelota`, que interioriza cualquier objeto esférico en la imagen, puede tener un parámetro que sea el color de la pelota que interesa en cada momento, ignorando el resto.

En cuanto a la organización jerárquica, en cada nivel se activará un subconjunto de esquemas perceptivos. Esto marca un contexto de atención porque se activan esos esquemas y no otros, de manera que cualquier otro estímulo, aunque esté presente en la realidad, será ignorado por el robot. Los activos buscarán su correspondiente estímulo real, que puede estar presente o no. Por lo tanto la jerarquía en percepción se puede interpretar como una predisposición ante ciertos estímulos. El contexto marca la predisposición sensorial a detectar éstos y no otros, una sensibilidad mayor hacia estos y no hacia otros. Por ejemplo, la percepción de cualquier estímulo que pueda comerse se agudiza cuando se está hambriento.

Así es como hemos materializado el tercer principio de Arbib [Arbib y Liaw, 1995]: "una multiplicidad de diferentes representaciones (representaciones parciales retinotópicas, conocimiento abstracto de objetos del mundo o más abstracto de planificación) deben ser enlazadas en un todo integrado". Este planteamiento de la percepción reconoce que todos los eventos o estímulos que queremos tener en cuenta en el comportamiento requieren un esfuerzo computacional para detectarlos y percibirlos. Han de buscarse, y ese procesamiento se sitúa en el correspondiente esquema perceptivo.

En cuanto a la naturaleza de los estímulos, la arquitectura no se posiciona, simplemente normaliza que se respete la interfaz de esquema, con comunicación entre ellos a través de variables, y regula la activación de las unidades perceptivas, asociándolas a las de actuación en su jerarquía. En este sentido dentro de un estímulo puede haber símbolos, en el sentido más deliberativo del término, y por ello nos distanciamos de Brooks y los reactivos puros. En contraste, las lecturas sensoriales también pueden ser estímulos, en JDE no se impone que el estímulo tenga necesariamente cierta abstracción o significado humano. Igualmente dentro de un estímulo cabe una representación objetiva de un objeto real, como `mesa-15`, `obstáculo-19` o incluso un mapa de las cercanías. Aunque también caben conceptos funcionales, más subjetivos o deícticos [Agre y Chapman, 1987] como `la-abeja--que-me-persigue`. En cualquier caso debe especificarse cómo se construye tal o cual estímulo. En general, al elaborarse desde los sensores del robot, la percepción tiende a ser subjetiva y deíctica con los estímulos clave que disparan la activación de los esquemas motores, y como veremos posteriormente, más gestáltica que reconstructura.

En JDE el concepto *estímulo* no es una descripción funcional, es una descripción formal. En este sentido hemos separado su semántica, entendiendo como tal las cosas que se pueden hacer con él, los comportamientos que desencadena o en los que interviene. En contraste con la definición que da la Real Academia, al estímulo-JDE le despojamos del desencadenamiento directo de la reacción, que queda para la selección de acción. Con ello se posibilita que al mismo estímulo se le puedan asociar distintas reacciones, dependiendo del contexto en que se trate. El estímulo sigue siendo utilitarista y tiene un carácter subsidiario con la acción, pero no hay una fusión completa con la acción, como sí la encontramos en las *potencialidades* (*affordances*) de Gibson [Gibson, 1977]. Gibson argumenta que se puede percibir *el significado* de las cosas directamente, entendiendo por tal las acciones que se pueden hacer con ello. En JDE los esquemas perceptivos existen per se, sin significado asociado,

su semántica se fija cuando se activa en cierto contexto y con ciertos esquemas motores (y no otros) esperando sus datos. Ese sería el momento de su anclaje semántico, de acuerdo con la terminología de Horswill[5] (*grounding*) [Horswill, 1997].

Cada estímulo tiene su propia dinámica que hereda su esquema perceptivo. Cada esquema perceptivo tiene su propio ritmo. Esta velocidad depende de la latencia (tiempo en detectar su presencia) y la velocidad de cambio del estímulo que se desea percibir. Los estímulos que cambian rápidamente tendrán asociados esquemas perceptivos que ejecutan su iteración en intervalos pequeños, muy frecuentes. En esa iteración va encerrada la actualización y el mantenimiento de la validez del estímulo interiorizado. Del mismo modo, los eventos que conviene detectar a la mayor brevedad posible desde que ocurren tendrán asociados esquemas rápidos que los buscan.

En este sentido las variables contienen el estado actual del mundo, filtrado por lo que interesa en ese momento. Es una percepción fresca, que no se proyecta en el futuro, pero admite que la situación presente puede cambiar. Como veremos en el capítulo 5 y en el anexo A, esto obliga a un replanteamiento de las técnicas clásicas de fusión sensorial, para adaptarlas a los escenarios dinámicos. Además de esa localidad temporal, le daremos un matiz espacialmente local, por cuanto que incideremos en estímulos del entorno próximo al robot. Nuestro objetivo concreto de la percepción es el mantenimiento de una descripción rica y dinámica del entorno próximo, de modo parecido al espacio perceptivo local de Konolige [Konolige y Myers, 1998]. Más que una imposición de la arquitectura ésto es un uso concreto que se hace de ella. Aunque los esquemas perceptivos pueden mantener estado, no hemos tratado el problema de la memoria a largo plazo, ni el empleo de mapas estructurales. Cuestiones como qué cosas ingresar en esa memoria a largo plazo y cuáles olvidar, quedan por lo tanto fuera de esta tesis.

Este posicionamiento contrasta con la clásica construcción de mapas, que ya implica un largo plazo y memoria. En otras arquitecturas la representación del entorno se ha dividido típicamente en información estática por un lado, como los mapas que almacenan la estructura del entorno para navegar, y por otro lado, una representación local basada en las lecturas instantáneas de los sensores. La mayoría de los sistemas híbridos parten de esta división de la información en cuanto a su dinamismo y alcance. De hecho, la parte deliberativa suele operar sobre la información más lenta, y la reactiva sobre la otra, como en Xavier [Simmons *et al.*, 1997b; Koening *et al.*, 1996; Simmons y Koening, 1995].

3.4.1. Imbricación entre percepción y actuación

La existencia de los esquemas que simplemente elaboran información explicita un reconocimiento de que la percepción es una parte relevante en el comportamiento. En los últimos años la preocupación por incluir percepción en la arquitectura ha sido cada vez mayor: Budenske con sus sensores lógicos [Budenske y Gini, 1997], Wasson [Wasson *et al.*, 1999] con memoria perceptiva para integrar visión activa en una arquitectura, Horswill [Horswill, 1997] con su anclaje de primitivas perceptivas, etc.. Esta tendencia contrasta con otros enfoques que asumen la percepción resuelta y no se preocupan de ella, que se concentran en la descomposición de tareas en subtareas, etc, es decir, sólo en el lado de actuación. Por ejemplo, Maes [Maes, 1990] parte de una percepción del mundo expresada en la lógica de predicados, pero que no se preocupa en generar, mantener, o anclar en el mundo real; ella se concentra en los mecanismos de selección de acción asumiendo que la percepción tal y como ella la necesita, está resuelta.

La incorporación explícita de la percepción a la arquitectura del robot es una característica de JDE. De hecho, consideramos que la percepción es clave para generar buenos comportamientos, y que con la percepción adecuada el control puede ser relativamente sencillo. Por ejemplo, las rutinas de percepción en Pengi [Agre y Chapman, 1987] son relativamente complejas, mientras que las reglas de control son sencillas. Así, percibir bien es más de la mitad del problema del comportamiento.

La percepción tiene pues un carácter subsidiario, orientado a la acción: un esquema perceptivo existe porque al menos un esquema motor necesita, tarde o temprano, la información que elabora. Esta asociación recae en el esquema de actuación, el cual tiene apuntados los esquemas perceptivos

[5] Él interpreta anclaje como la asociación a cada rol de sus descriptores sensoriales asociados

que necesita, es decir, de qué información dependen sus decisiones. Los esquemas perceptivos, más que estar orientados a la reconstrucción genérica del entorno, están orientados a satisfacer a los esquemas de actuación directamente. Este planteamiento coincide con los enfoques reactivos y basados en comportamientos, los cuales se distancian de la percepción reconstructora y objetiva típica de los sistemas deliberativos.

Para desarrollar un comportamiento con JDE el diseñador debe pensar qué información necesita para que el robot pueda tomar él sólo las decisiones de actuación convenientes para esa conducta. En el lado de actuación hay que responder a preguntas como: ¿Cuáles son las decisiones correctas?¿Cómo se toman desde la información perceptiva?. En el lado perceptivo: ¿Qué información se necesita para desarrollar ese comportamiento? ¿Cuándo es conveniente? ¿Cómo se genera esa información desde los datos sensoriales? Finalmente los esquemas perceptivos deben encuadrarse en un contexto, hilvanarse con el(los) esquema(s) de actuación pertinente(s).

En JDE los esquemas motores toman sus decisiones sobre información producida por esquemas perceptivos. Así, cada esquema motor suele activar los esquemas perceptivos que necesita para desarrollar su labor. Ahí se halla parte de la coordinación sensorimotora. Un esquema de actuación puede necesitar a los perceptivos para decidir qué esquemas hijo activar, para decidir cómo modular a sus hijos, o para monitorizar el progreso de sus hijos. Es decir, los esquemas perceptivos proporcionan a los esquemas motores información de activación, de ejecución y de monitorización. En cuanto a la ejecución, hay modulación de los esquemas perceptivos hacia los esquemas de actuación, como menciona Arbib [Arbib, 1989]. Esta modulación no es a través de los parámetros sino a través de las variables compartidas, que unos escriben y otros leen para materializar su ejecución de un modo u otro. Por ejemplo, la percepción de las jambas influye en el desarrollo del comportamiento `atraviesa--puertas`, modulando continuamente el movimiento del robot para alinearse con la puerta.

La percepción también juega un papel importante en la activación de unos esquemas y no de otros. De hecho desempeña un rol fundamental en el mecanismo de selección de acción, como hemos visto. El padre activa a los esquemas perceptivos que permiten decidir cuál de sus hijos resulta más adecuado a la situación, por ejemplo, los que buscan los estímulos clave de los hijos. Para la activación de un hijo el efecto de los esquemas perceptivos se puede acumular y que no pase a PREPARADO si no se presentan varios de estos estímulos. De este modo se flexibiliza la definición de *situación favorable* para el desencadenamiento de tal esquema motor. Tal y como muestra la figura 3.9, la activación puede depender de la presencia, en mayor o menor grado, de un conjunto de estímulos, no de uno en concreto. Por ejemplo, un primer estímulo externo A puede hacer subir la activación como máximo en 3 unidades, un segundo estímulo B, en 5, y un tercero C, en 9. Si el umbral de activación se establece en 10 unidades entonces con que se presenten C y B en cierta medida el esquema motor asociado pasará a estado PREPARADO; también lo hará si aparecen A y C. Sin embargo, no promocionará en situaciones donde exclusivamente se presenten A y B, o alguno de ellos a solas. En ninguno de estos casos la activación supera el valor umbral.

Este mecanismo permite generar fenómenos ampliamente estudiados y confirmados en el campo etológico como la *suma aditiva de estímulos*. Además confiere, a la percepción un carácter gestáltico: no es necesario reconstruir espacialmente todo el entorno, basta con detectar cierta configuración de estímulos para identificar una situación favorable, cierta composición de cuadro.

Por ejemplo, el comportamiento de apertura de pico en las crías de gaviota cuando están en el nido depende fundamentalmente de sus observaciones visuales [Lorenz, 1978]. Para que abran el pico en busca de alimento deben apreciar una forma blanca redondeada (la cabeza de la madre), de la cual sobresale un pico amarillo, y en el pico una mancha roja cercana a la punta. No es necesario reconstruir geométricamente el espacio desde visión, si ese patrón visual se presenta, entonces se dispara la apertura del pico y el polluelo se pone a piar con todas sus fuerzas esperando comida. Haciendo pruebas con simulacros se ha comprobado que el grado en que aparecen unos estímulos puede sustituir la carencia de otros. Por ejemplo, si el pico no es lo suficientemente largo, pero tiene una mancha muy pronunciada, se produce el mismo efecto que si el pico es perfecto pero la mancha es naranja en vez de roja y no está situada en la punta. Es la combinación de estos estímulos, y no la presencia de uno solo, la que dispara el comportamiento. De este modo se consigue una identificación de la situación favorable muy robusta y flexible. Además se mantiene su poder de discriminación: es

posible que alguno de esos estímulos aislado aparezca por azar (por ejemplo, una cabeza blanca que en vez de la madre sea de un predador), pero la combinación de varios es más improbable que se presente en situaciones que no sean las convenientes para el comportamiento. Mucho menos si hemos lanzado la sensibilidad a esos estímulos sólo en el contexto adecuado.

En cuanto a monitorización, en la sección anterior vimos que el padre necesita monitorizar el desarrollo de la funcionalidad de sus hijos, y para ello debe activar los esquemas perceptivos pertinentes. Ellos abren los caminos por los cuales puede avanzar la reconfiguración de toda la jerarquía en el instante siguiente. Dependiendo de lo que realmente suceda se irá por un camino u otro, pero los esquemas perceptivos deben estar despiertos para detectar las desviaciones posibles respecto del funcionamiento esperado o incluso el curso normal de la ejecución (por ejemplo, cuando un hijo consigue su objetivo).

Otra de las interrelaciones que abre JDE entre percepción y actuación es la percepción activa [Bajcsy, 1988]. Es decir, la posibilidad de que en ciertos momentos las acciones del robot tengan por objetivo facilitar cierta percepción. Uno de los principios de JDE es que se percibe para actuar, en la percepción activa se contempla que para percibir una información relevante es conveniente actuar de cierta forma. Esto podría interpretarse como que esa actuación está subordinada a la percepción. No obstante, la percepción conseguida siempre será subsidiaria de un esquema de actuación del que depende.

Un modo de materializarla dentro de JDE consistiría en que un esquema perceptivo tenga como hijos a uno o varios esquemas de motores para actuar de modo que favorezca la percepción de lo que interesa a ese esquema perceptivo.

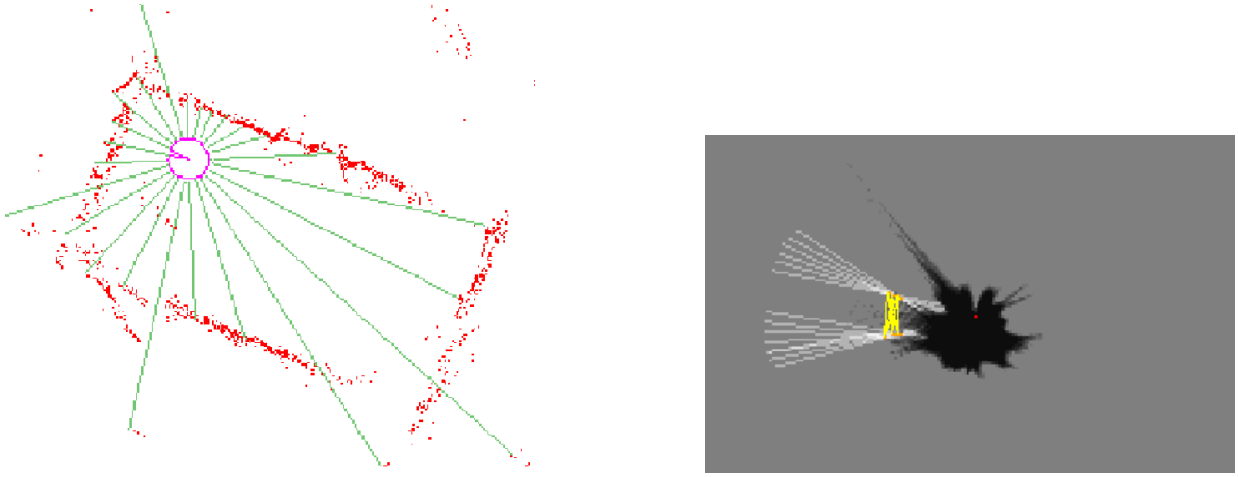
Como recalcaremos en el capítulo 6, la percepción activa simplemente está esbozada, no se ha profundizado en ella. Queda abierto definir con qué otros esquemas compiten esos hijos de actuación para garantizar la consistencia. Un idea tentativa sería hacerlos competir con los hermanos de actuación de su esquema perceptivo padre (sus tíos).

3.4.2. Fusión sensorial y percepción estructurada

En los sistemas reactivos la percepción se reduce al nivel del sensor, evitando el uso de cualquier abstracción, y se busca asociar las consignas de los actuadores a los valores sensoriales. Por el contrario, en JDE admitimos representación simbólica, el estímulo como estación intermedia entre los sensores y los actuadores. Esta indirección permite más riqueza, pues los estímulos pueden no estar explícitamente en los sensores, necesitar cierta elaboración. Puede haber estímulos simbólicos si resultan convenientes para el comportamiento entre manos, aunque deben estar anclados, con claros algoritmos de construcción desde los datos sensoriales u otros estímulos de nivel inferior al suyo. Este posicionamiento parece tener un paralelismo en la naturaleza, donde los sensores específicos atados a cierto patrón motor son típicos de animales inferiores. Por contra, los animales que exhiben comportamientos más complejos y evolucionados suelen utilizar sensores genéricos como la vista o el oído y procesar esa información de modo específico. Esto les permite reutilizar el mismo sensor para una gran variedad de comportamientos, mientras que el uso de sensores específicos para una tarea dificulta su reutilización.

En general, los estímulos en JDE necesitan elaboración, y ese tratamiento lo realizan los esquemas perceptivos. Por ejemplo, definimos un *estímulo complejo* como aquel que no queda completamente definido y caracterizado desde la lectura de un único sensor, es decir aquel en el cual no hay una correspondencia *biunívoca* entre ese estímulo y los datos instantáneos de un sensor. La instantánea sensorial por sí sola casa con muchas explicaciones, y no se puede determinar a cuál de ellas corresponde realmente. Una instantánea sensorial por sí sola no es concluyente sobre la existencia de tal o cual estímulo. La detección de un estímulo complejo necesita indefectiblemente de la fusión sensorial, de la integración de medidas, bien sean éstas del mismo sensor a lo largo del tiempo (integración temporal), bien sean de sensores de naturaleza muy distinta pero que aportan información sobre el mismo estímulo (integración multimodal).

La fusión sensorial permite acumular indicios, evidencias parciales, y ello resulta crucial en la identificación de estímulos complejos. Esto no se arregla con un sensor más preciso. Un ejemplo podría ser el estímulo `pared`: la última instantánea sónar se ve afectada por la existencia de una pared próxima



(a) (b)

Figura 3.13: *Instantánea sonar (a) y detección de la puerta acumulando evidencia (b).*

o no, pero desde esa instantánea es imposible distinguir si se trata de una pared o de cualquier otro obstáculo. Será la acumulación de varias lecturas, y el alineamiento de las posiciones ocupadas lo que permitirá concluir que el obstáculo concreto es una pared. Otro ejemplo lo constituye el estímulo `jamba de puerta`: en la figura 3.13(b) hemos visualizado una rejilla que acumula las evidencias de jamba, conseguidas tanto desde los sensores sónares, como del análisis de imágenes. A vista de pájaro, las zonas claras de la figura corresponden a las posibles proyecciones que explican los bordes verticales apreciados en la imagen, similares a los de la figura 3.14. Es la acumulación de proyecciones en la misma posición la que permite asegurar que realmente existe allí una jamba [Cañas *et al.*, 2001]. Veremos una descripción más detallada de este ejemplo en la sección 5.2. Este estímulo `jamba` ayuda en la detección de puertas, que resulta muy útil para navegación en interiores, y que como también señala Budenske [Budenske y Gini, 1994] es muy complicado de detectar con los sónares.

La fusión sensorial a la que dejan hueco los estímulos ofrece otras ventajas más como la corrección de errores sensoriales. Un robot autónomo percibe el estado de su entorno a través de sus sensores, que tienen inevitablemente cierta incertidumbre. La última lectura de todos sus sensores le proporciona una *instantánea* sobre el estado de sus alrededores. Por ejemplo, los rayos de la figura 3.13(a) muestran la medida momentánea de los sensores de ultrasonido, conformando una instantánea con información sobre el estado actual de sus alrededores. Los puntos rojos son los extremos de esos rayos que se han acumulado a lo largo de cierto intervalo. Como se puede apreciar, algunas lecturas instantáneas no se corresponden fielmente con el perfil de la habitación. Sin las fusión de las lecturas siempre tendremos esta instantánea sensorial, continuamente refrescada (en el robot concreto de los experimentos, cada sensor realiza 3 medidas por segundo). Debido a su simplicidad (en el robot concreto de los experimetnos, 16 valores enteros) y a su vivacidad, esta representación ha sido utilizada en muchos casos para construir comportamientos reactivos sobre ella, por ejemplo, el sorteo de obstáculos.

Ahora bien, si utilizamos una rejilla para acumular en ella múltiples lecturas del sonar [Matía y Jiménez, 1998; Peignot *et al.*, 1998] se pueden compensar las lecturas erróneas con las medidas correctas, que serán previsiblemente más numerosas. Además, se puede recordar la información de zonas próximas o zonas que de repente quedan ocluídas por un obstáculo intermedio. Con ello la fusión también ayuda a depurar errores sensoriales y perfilar mejor el contorno de los obstáculos. Esta compensación es especialmente adecuada para los sensores sónares porque sus medidas son propensas a ruidos. Esto también se puede solucionar con sensores más precisos.

La existencia de varios niveles de esquemas perceptivos permite la abstracción y los estímulos compuestos: los esquemas perceptivos pueden tener como entrada la salida de otros esquemas perceptivos. Es decir, se tiene literalmente una jerarquía de estímulos, y por lo tanto una percepción estructurada. *Los estímulos de alto nivel están compuestos básicamente por otros estímulos de menor nivel de abstracción.* Así la estructura conceptual de los estímulos puede tener su reflejo en la jerarquía

de esquemas que lo construye. Por ejemplo, el estímulo `puerta` descrito en [Cañas *et al.*, 2001] se compone de dos estímulos `jamba` que satisfacen ciertas condiciones, como estar a cierta distancia razonable.

La idea de percepción estructurada ya ha aparecido con otros formatos en la literatura. El espacio perceptivo local de Konolige [Konolige y Myers, 1998] también tiene una estructura: el estímulo `pasillo` se apoya en dos `segmentos de pared` paralelos. También existen funciones que sacan variables borrosas desde la representación geométrica del espacio perceptivo local, y esas variables son las que aparecen en los antecedentes de las reglas de control de los comportamientos básicos. En este sentido, la estructura se puede ver como un caso de fusión sensorial, pues un estímulo depende de varios *subestímulos* a los cuales integra. También la representación multirresolucional de Meystel [Meystel, 1998] presume de estructura, aunque en este caso es la escala temporal y espacial lo que separa un nivel preceptivo de otro.

3.4.3. Percepción situada: atención e interpretación

Una vez que hemos descrito la discretización de la percepción en esquemas, su planteamiento jerárquico e imbricación con la actuación, vamos a explicar ahora dos fenómenos exclusivamente perceptivos que posibilita este enfoque: la atención y la interpretación. Los dos son posibles gracias al caracter jerárquico de la arquitectura, donde cada nivel de la jerarquía marca un contexto, tanto de actuación como de percepción.

La atención se materializa por el hecho de que en cada nivel el esquema padre determina a cuáles hijos perceptivos activa y a cuáles hijos motores despierta. El hecho de que ese esquema padre esté activo marca ya un contexto, porque para lograr su activación es necesario satisfacer la doble orientación a objetivos y al entorno que vimos anteriormente. En ese contexto sólo le interesará detectar ciertos estímulos y por ello activará a voluntad los esquemas perceptivos que los buscan y elaboran, y no a otros. Esto ya supone un sesgo, pues no se detecta todo lo percibible, sólo lo que es relevante en ese escenario. De esta manera se filtra una gran cantidad de datos inútiles procedentes de los sensores, y hace al sistema más eficiente, porque no se dedica capacidad de cómputo alguna a los estímulos que no son interesantes en el contexto actual. Por ejemplo, en la figura 3.10 los estímulos que elaborarían todos los esquemas perceptivos del repertorio marcan el espacio perceptivo completo. En ese momento sólo interesan las zonas blancas y sólo esos esquemas se activan. Las zonas sombreadas corresponden a los que elaborían el resto de los esquemas perceptivos, que permanecen dormidos. En otro contexto, otro esquema activará otros esquemas perceptivos diferentes. Al igual que la acción, en JDE la percepción también es contextuada y situada.

Como sólo se percibe aquello que se busca, han de anticiparse todas esas sorpresas previsibles, y activar no sólo los estímulos necesarios para la ejecución normal, sino también todos aquellos que pueden presentarse en ese escenario y ante los que habría que responder. Uno de los riesgos es hacer este filtro perceptivo demasiado severo, impidiendo la detección de desviaciones o sorpresas plausibles.

Es importante resaltar que la atención no es un problema de tener un sensor más o menos preciso. Si tuvieramos el supersensor, de precisión perfecta, dinámico y que nos proporcina una rejilla tridimensional de todo el mundo, aún no habríamos resuelto el problema de la atención. ¿Cómo se vincula esa representación perfecta con las tareas del robot? Esta pregunta sugiere que una percepción reconstructora puede ser necesaria pero no es una solución completa al problema de la atención, porque aún no está enlazada con los objetivos del robot. Otra pregunta diferente es cómo hacer uso de esa representación. Como hemos visto, la atención es el mecanismo que determina qué percibir y cuándo hacerlo.

La percepción contextuada que ofrece JDE permite otro fenómeno perceptivo: la *interpretación*. La interpretación se hace útil en estímulos complejos. Como hemos visto anteriormente, éstos estímulos necesitan la acumulación de evidencias para su detección. Una vez que se tiene suficiente evidencia de que realmente existe, entonces puede caracterizarse, es decir, concretar sus propiedades o parámetros, posiblemente dinámicos, como su tamaño o posición. Habría pues dos fases diferentes en la percepción, una de detección y otra de caracterización. Típicamente la detección suele estar relacionada con la activación de tal o cual esquema (por ejemplo el estímulo clave). Por el contrario, la caracterización suele influir en la ejecución del comportamiento, en su modulación de grano fino. Por ejemplo, la

Figura 3.14: *Interpretación de bordes verticales como jambas visuales*

posición del estímulo `pelota` modula el esquema motor de seguimiento, como veremos en la sección 5.3.

En la fase de detección pueden estar pugnando diferentes interpretaciones de los mismos datos sensoriales. Por ejemplo, utilizando la transformada de Hough [Hough, 1959; Illingworth y Kittler, 1988]. La confirmación en el tiempo de unas hipótesis y no de otras corrobora la detección o no de tal o cual estímulo [Cox y Leonard, 1994]. En la fase de caracterización ya se pueden interpretar las lecturas sensoriales de modo diferente al usual. Por ejemplo, una vez que sabemos que existe una pared en cierta posición, podemos entonces interpretar las lecturas sónares y filtrar las reflexiones especulares, porque podemos interpretar que cierto dato proviene de una reflexión especular. De igual modo, si ya sabemos que por cierta zona, más o menos, existe una puerta, entonces podemos interpretar los bordes verticales de la imagen que se proyectan en ese área (figura 3.14) como las jambas visuales de esa puerta. Éstas no son más que bordes verticales en la imagen, pero el hecho de saber que existe una puerta compatible con esa observación sensorial permite interpretar esos bordes como jambas visuales.

Esta asociación rápida entre observaciones sensoriales y estímulos complejos que proporciona la interpretación ofrece ventajas a la hora de generar comportamiento. Por ejemplo, en una conducta `atraviesa-puerta` se puede emitir el control directamente desde los bordes verticales de la imagen. De este modo no se pierde reactividad, e incluso se pueden establecer lazos rápidos de realimentación para estímulos complejos.

La interpretación anterior de la lectura sensorial se apoya en que la existencia del estímulo complejo se ha asegurado en la fase de detección. De modo más genérico, en JDE la interpretación se apoya en el contexto que marca la activación del esquema padre. Esa activación indica que el robot se encuentra en cierta coyuntura, marca un contexto tanto de actuación como de percepción, con lo cual las posibles interpretaciones de los datos sensoriales no son tantas, están acotadas. Ese contexto se puede utilizar para eliminar las interpretaciones poco probables de los datos sensoriales. Así, cuando se tienen evidencias que no son concluyentes, por el hecho de estar situados en cierto contexto, se pueden tomar como si lo fueran (incluso sin fase de detección). Una vez que estamos en tal contexto, los estímulos reales que pueden ocasionar el síntoma-A son sólo los estímulos B,C y D. No hay que discernir posibilidades con estímulos complejos E, F y G, sólo entre los plausibles en ese contexto.

En el caso extremo sólo se tiene una hipótesis de alto nivel, y si aparece una evidencia de nivel inferior (`borde vertical`), directamente ya se confirma la de alto nivel (`puerta`). Esto da pie a la percepción abductiva, pues el síntoma sensorial permite abducir su causa: si se percibe cierta característica, ello implica que existe determinado estímulo complejo. La relación causal fluye realmente en sentido contrario: si existe el estímulo entonces se presenta tal característica, que por sí sóla no es concluyente (pues puede estar causada por otros estímulos complejos). El contexto permite dar este salto e ignorar otras posibles interpretaciones del síntoma sensorial.

La percepción abductiva es mucho más rápida que la constructivista. Con ella se acelera la percepción de estímulos complejos, que se abducen desde las lecturas sensoriales por el hecho de estar

en cierto contexto. Por ejemplo, en el comportamiento `sigue-pared` [Gómez *et al.*, 2003], el esquema perceptivo abduce la pared y su posición desde los bordes de la mancha de color suelo. Además, no es necesaria una reconstrucción geométrica de la pared para emitir el control, las consignas a los motores se establecen directamente desde los bordes visuales de la mancha del color suelo. Las hipótesis que se lanzan en la percepción abductiva tienen que ver con el contexto de percepción en que nos hallemos. Además, la interpretación ayuda a digerir la intertidumbre sensorial, como en el ejemplo de la reflexión especular de los sónares ya mencionado.

3.5. Discusión

Una vez que hemos descrito la arquitectura en todas sus facetas, esta sección es eminentemente comparativa. Primeramente resumimos los puntos relevantes de la arquitectura JDE en la sección 3.5.1, para despues contrastarlos con otras alternativas existentes en la literatura. En esa comparativa discutiremos cuestiones tales como: ¿por qué utilizar al esquema como unidad de comportamiento?, ¿por qué utilizar la jerarquía para crecer en complejidad? ¿qué otras interpretaciones hay de jerarquía? Esta confrontación servirá como una evaluación y puesta en valor de la arquitectura propuesta. Finalizamos el capítulo mencionando las limitaciones que apreciamos en la arquitectura.

3.5.1. Principios de JDE

La arquitectura JDE descrita se apoya en tres planteamientos básicos de partida. El primero es que el comportamiento autónomo se puede generar combinando percepción y control en una misma plataforma. La actuación se encarga de tomar decisiones de control acordes con la situación y los objetivos. La percepción se encarga de proporcionar información para tomar esas decisiones. Ambos se identifican como dos subproblemas interrelacionados que se resuelven por separado. El segundo es su cuantización en esquemas, en pequeñas unidades que recuerdan la idea seminal de Minsky [Minksy, 1986] sobre la sociedad de la mente, y se alinean con las arquitecturas basadas en comportamientos. El tercero es la organización de éstos en jerarquía: los esquemas se agrupan en niveles de abstracción, que no son fijos, sino dependientes de la tarea en curso. La salida de un esquema es la activación y modulación de otros esquemas inferiores, de este modo se consigue la composición de comportamientos y que el sistema se estructure como un árbol de esquemas en el que unos activan a otros.

La arquitectura propuesta queda definida por unos principios axiomáticos fundamentales que la caracterizan. Los hemos ido explicando a lo largo de este capítulo, y los recopilamos en la tabla 3.5.1. De estos principios fundamentales se pueden deducir un conjunto de corolarios significativos, en cuanto a propiedades ventajosas de la ordenación propuesta. Los hemos resumido en la tabla 3.5.1.

La apuesta fundamental de JDE para crecer en complejidad es la jerarquía de activaciones y modulación. En este sentido, propone un arbitraje distribuido para la coordinación entre los esquemas. Cada esquema motor marca un contexto de actuación al despertar a sus hijos para que alguno de ellos, y sólo de ellos, tome el control en cada instante. En esta competición por el control también interviene la situación motivacional y del entorno, determinando cuál de los hijos es el más adecuado para activarse finalmente. Además, cada padre monitoriza la ejecución de sus hijos, de manera que la monitorización también aparece distribuida en la jerarquía.

Otro aporte esencial de JDE es la inclusión explícita de los aspectos perceptivos en la arquitectura. La percepción se considera así orientada a la acción ya que le proporciona la información relevante para que tome sus decisiones. En este sentido, la coordinación entre percepción y acción se resuelve en cada esquema motor, que apunta los esquemas perceptivos que él necesita. Cada esquema perceptivo se encuentra atado a su correspondiente esquema motor, del cual depende su propia activación. Con ello se consigue concentrar la computación perceptiva en los estímulos que interesan en cada momento, materializando un mecanismo de atención.

PRINCIPIOS	
Esquemas	
1	La unidad básica del comportamiento es el *esquema*
2	Un esquema es un flujo de ejecución independiente con un objetivo
3	Es modulable, iterativo y puede ser activado o desactivado a voluntad
4	Hay *esquemas perceptivos* y *esquemas de actuación*
5	Un esquema de actuación puede estar DORMIDO, en ALERTA, PREPARADO o ACTIVO
6	Un esquema perceptivo puede estar DORMIDO o ACTIVO
7	Un esquema aprovecha la funcionalidad de otro despertándolo y modulándolo, mientras él sigue ejecutándose concurrentemente
Jerarquía	
8	Los esquemas se estructuran en jerarquía, distinguiendo entre padres e hijos
9	En un instante dado el sistema consiste en un conjunto de esquemas, organizados en jerarquía, ejecutándose simultáneamente
10	Las jerarquías se construyen y modifican dinámicamente, y son específicas de cada comportamiento
Actuación	
11	La salida de un esquema de actuación pueden ser comandos a los actuadores o la activación y modulación de esquemas hijos
12	Mientras los esquemas hijos persiguen o mantienen sus propios objetivos, están ayudando a que el que los activó persiga o mantenga los suyos
13	El padre pone en ACTIVO a los esquemas perceptivos que buscan los estímulos relevantes para realizar su misión y pone en ALERTA a los esquemas de actuación que generan las respuestas convenientes cuando esos estímulos aparecen realmente
14	Si la situación del entorno satisface las precondiciones de un esquema en ALERTA entonces pasa a PREPARADO
15	Entre los hijos de actuación en PREPARADO se establece una competición de control, de modo que en cada instante sólo hay un único ganador que pasa a ACTIVO
16	Sólo los esquemas en ACTIVO pueden activar a otros esquemas
17	El padre monitoriza iterativamente los resultados de sus hijos, y puede cambiar su modulación e incluso cambiar de hijos cuando le convenga
Percepción	
18	La información útil para un comportamiento se expresa como una colección de estímulos, que se organiza en jerarquía
19	Un *estímulo* es una pieza de información que al menos un esquema de actuación necesita para tomar sus decisiones
20	Los estímulos pueden ser simples lecturas sensoriales, transformaciones suyas más elaboradas, o depender de otros estímulos menos abstractos
21	Los esquemas perceptivos producen estímulos y los mantienen actualizados
22	Los estímulos se materializan en variables que otros esquemas pueden leer
23	Los esquemas perceptivos se asocian a los esquemas de actuación para los cuales proporcionan información (*coordinación percepción-actuación*)
24	Los esquemas perceptivos proporcionan información de activación, de ejecución y de monitorización a los esquemas de actuación

Tabla 3.1: *Principios fundamentales de JDE.*

COROLARIOS	
Actuación	
1	La selección de acción presenta una doble orientación hacia objetivos y hacia la situación actual: los hijos colaboran en el objetivo del padre; las condiciones del entorno eligen entre ellos al más conveniente según la situación
2	JDE simplifica la selección de acción, dividiéndola en varios niveles, dentro de cada cual está acotado el número de hermanos entre los que hay que elegir
3	Los esquemas que en un momento dado no se usan descansan suspendidos en estado DORMIDO, pero preparados para la activación en cualquier momento
Percepción	
4	La existencia de varios niveles de esquemas perceptivos permite la abstracción y los estímulos que dependen de otros más básicos (*percepción estructurada*)
5	La *atención* se materializa por el hecho de que en cada nivel el esquema padre determina a qué esquemas perceptivos activa y a cuáles no
6	Dentro de un esquema puede haber *fusión sensorial*, que es imprescindible para percibir los estímulos que no quedan completamente caracterizados desde la lectura instantánea de un único sensor
7	En JDE la percepción y la actuación son contextuadas y situadas. La activación del padre marca un contexto tanto de actuación como de percepción
8	El contexto de percepción acota las posibles *interpretaciones* de los datos sensoriales, permitiendo la *percepción abductiva*
9	El que un esquema perceptivo pueda tener como hijos a esquemas de actuación abre la puerta a la *percepción activa*
Jerarquía	
10	La existencia de jerarquía no añade retardos, porque los esquemas de bajo nivel funcionan en paralelo a los de alto nivel
11	Un nivel más alto se distingue de otro inferior porque utiliza estímulos semánticamente más abstractos, no por tener diferente frecuencia de iteraciones
12	En cada instante el sistema es monoobjetivo dentro de cada nivel de abstracción. Para satisfacer varios objetivos la jerarquía puede ir reconfigurándose
13	Los niveles de la jerarquía son variables, habrá más o menos niveles dependiendo de la complejidad del comportamiento actual

Tabla 3.2: *Corolarios relevantes de los principios de JDE.*

La percepción aparece distribuida en una colección de esquemas perceptivos activos, cuyos integrantes pueden cambiar dinámicamente. Se pueden utilizar lecturas sensoriales para montar sobre ellas esquemas motores. También se admiten símbolos, siempre que esten anclados dinámica y permanentemente a la realidad, siendo los que permiten crecer en abstracción.

En conjunto, JDE ofrece una arquitectura distribuida, homogénea y genérica para causar comportamiento. La percepción, actuación, monitorización y selección de acción estan distribuidas en los esquemas. La ausencia de un modelo centralizado supera el cuello de botella típico del paradigma deliberativo SMPA. Además, evita la necesidad de un modelo completo del entorno para empezar a actuar. Por otro lado, JDE ofrece una interfaz única entre niveles, que es la de cada esquema. Todos los esquemas interactúan igual: activándolos y modulándolos. Esta homogeneidad contrasta con la heterogeneidad de interfaces que ofrecen los sistemas híbridos, ya sea entre capa deliberativa y secuenciadora, o entre secuenciador y capa reactiva.

La arquitectura que aquí hemos presentado es genérica, de propósito general. Los experimentos con JDE del capítulo 5 describen comportamientos principalmente de navegación, pero no hay ninguna limitación en JDE que restrinja su aplicación a otro tipo de comportamientos como la manipulación u otros. En las décadas de los años 80 y 90, quizá el comportamiento principal que se le pedía a los robots era la navegación, y las arquitecturas estaban muy sesgadas por ese comportamiento concreto, que era prácticamente el único del robot (bastante se tenía con resolver ese problema). Por ello surgieron arquitecturas muy orientadas al problema de la navegación [Crowley, 1985; Meystel, 1987], pero que son difícilmente extrapolables a otros comportamientos. Por ejemplo, Meystel [Meystel, 1987] planteaba una división de la arquitectura en un planificador de misión que fija un destino global, un planificador de caminos que establece una ruta hacia él compuesta de puntos intermedios (navegación global) y un pilotaje (navegación local) para conseguir el punto intermedio siguiente. Sin embargo, esta organización no es genérica, no está claro que esta división sea aplicable a otras tareas como la manipulación o la coordinación visuomotora.

3.5.2. Unidades del comportamiento

Quizá el primer punto a discutir es la conveniencia de utilizar los esquemas como unidad primitiva del comportamiento. Vamos ahora a contrastar los esquemas con otros conceptos similares, que también han ejercido de unidades de comportamiento. De este modo ahondamos en descubrir qué caracteriza genuinamente a los esquemas frente a otras unidades.

Unidades de acción de Nilsson

Por ejemplo, en uno de los trabajos pioneros de robótica móvil [Nilsson, 1969] se definen las *unidades de acción* (*action units*) como la base del comportamiento. Las unidades de acción ejecutan una función en la que son especialistas, involucrando bien algún movimiento en el entorno, bien la recogida y análisis de datos sensoriales o incluso ambos. Por ejemplo, se tienen unidades de acción para los movimientos básicos (TURN, MOVE) y otra más para llevar al robot de un punto a otro cercano (LE) utilizando las anteriores. Sobre éstas se construye la acción que lleva al robot a recorrer una secuencia de puntos (EX), ya que las unidades pueden invocar a otras para realizar su propia función, incorporándolas a su propia implementación. Para ello cada una sabe el efecto de las otras acciones disponibles.

Las raíces teóricas de estas unidades de acción se asientan en la concepción del comportamiento como la planificación y la ejecución del plan. Estas unidades nacen ante la imposibilidad práctica de un único planificador que tenga en cuenta todos los detalles en secuencias muy largas. Para resolver esta imposibilidad se distribuye la planificación y ejecución en estas pequeñas unidades, cada una responsable de planificar y ejecutar una función especializada. Cada unidad de acción tiene su propio resolvedor de problemas y su propio modelo del mundo. Encontramos muchas diferencias entre las unidades de acción y los esquemas. Las primeras heredan la ejecución secuencial mientras que los segundos se ejecutan en paralelo. Además los esquemas no tienen abstracción temporal, mientras que la invocación de una unidad de acción conlleva al invocador la pérdida del control (de hecho el bloqueo) durante la ejecución de esa unidad.

RAP de Firby

La evolución natural de las unidades de acción podría considerarse que son los *paquetes de acción reactiva* RAP (*Reactive Action Packages*) de James Firby [Firby, 1992]. En ellos se intenta conciliar la naturaleza discreta de las acciones que necesita el planificador con los procesos continuos que operan realmente debajo de esa unidad simbólica. En este sentido, Firby incorpora la necesidad de monitorización dentro de cada RAP: cada uno de ellos realiza su misión cuando se la encargan y devuelve un indicador de éxito o de fracaso tras un cierto tiempo. Además del chequeo permanente del objetivo, en cada RAP se introducen alternativas de actuación para conseguir su meta. De manera que cada RAP tiene varias recetas de actuación, prueba con la primera y si falla prueba con la segunda, sin tirar la toalla. Sólo cuando se han agotado todas las alternativas y no se ha conseguido el objetivo, se señala el fracaso al llamante. Estas capacidades de replanificación a bajo nivel dotan al sistema de mayor robustez, sin complicar en absoluto al de arriba.

Los RAP como unidades de comportamiento intentan aproximar los planificadores clásicos a su uso en el mundo real, con sensores y actuadores reales que pueden desviarse de su funcionamiento ideal y tener incertidumbre. Los planificadores típicos de IA clásica necesitan unos operadores discretos, con pre y postcondiciones, sobre los cuales ellos pueden calcular la secuencia que lleva del estado inicial al final. En el mundo de bloques las acciones estaban idealizadas y sólo se tenían en cuenta sus consecuencias lógicas. Al aterrizar sobre el mundo real surgen las primitivas básicas de ejecución que encapsulan todos los problemas *de bajo nivel* y permiten al planificador seguir concentrado en sus efectos lógicos.

Con los RAP se abre la posibilidad de que su ejecución no consiga lo esperado y falle, el planificador puede considerarlo de antemano. Se cuenta para ello con que la propia primitiva señala su éxito o fracaso a quien la ha invocado, tras cierto tiempo. Estas primitivas tienen un resultado discreto: se ha conseguido la misión o no, el cual ellas mismas señalizan. Por el contrario, los esquemas no señalizan nada, y aunque tienen un objetivo implícito no hay comunicación interna, del esquema hijo al padre, acerca del éxito de la misión. Además los RAP tienen unos límites temporales claros, se sabe cuando empiezan y cuando acaban, aunque el llamante no tenga ningún control sobre ésto una vez que los ha invocado. Sin embargo, los límites temporales de los esquemas los determina completamente quien los usa, que decide cuándo activarlo y cuándo desactivarlo.

En la misma línea de los RAP se encuentran los *módulos especializados de procesamiento* SPM utilizados en Hilare [Giralt *et al.*, 1983], las tareas de TCA [Simmons, 1994] y más recientemente los objetos de CLARAty [Volpe *et al.*, 2001]. En concreto, en las tareas de TCA se incluye la monitorización explícita, el chequeo de condiciones que regulan el despliegue del plan materializando una planificación y una ejecución incrementales.

Habilidades en los sistemas híbridos

En gran parte de los sistemas híbridos la monitorización y la replanificación básica se han situado en el nivel intermedio, que se suele llamar secuenciador. En la parte reactiva se cuenta con unidades de comportamiento los cuales son procesos continuos que consiguen o mantienen un objetivo manejando los sensores y actuadores directamente. Por ejemplo, en la arquitectura 3T [Bonasso *et al.*, 1997] se utilizan las *habilidades situadas* (*skills*) y los *monitores de eventos* (*events*) como habilidades especiales. Los primeros serían análogos a los esquemas motores, y los segundos a los esquemas perceptivos. Ambos se conciben como un proceso activable y desactivable que realiza una transformación continua de sus entradas en sus salidas siguiendo cierta función. En este sentido las unidades de comportamiento son muy parecidas a los esquemas de JDE. Sin embargo, estas unidades no son completas, pues por sí solas no son suficientes, necesitan de dos capas superiores, la del secuenciador y la deliberativa, para que se genere comportamiento. En contraste, los esquemas JDE sí son completos, ya que en nuestra arquitectura sólo hay esquemas, y nada más que esquemas. En otros sistemas híbridos hay unidades similares como las *actividades primitivas* de ATLANTIS [Gat, 1992]. En todos estos casos el control sobre el momento de empezar a ejecutar y de acabar está fuera de las actividades primitivas, lo tienen las capas superiores.

Agentes

Otra unidad de conducta que ha aparecido en la literatura es el agente. Este término también se ha empleado en otros ámbitos como la inteligencia artificial distribuida y la programación, incluso para denominar así al propio robot como sistema completo [Maes, 1990]. Aquí nos referiremos a su uso como unidad básica para generar comportamiento, que parte de las ideas originales de Minksy en su sociedad de la mente [Minksy, 1986]. En ella argumenta que se puede contruir un sistema inteligente a partir de pequeñisimas partes, cada una de ellas sin inteligencia, a las cuales denomina agentes y que interactúan entre sí. Resulta útil en este sentido la definición de García-Alegre [García-Alegre y Recio, 1998]: "Cada agente se entiende como un proceso capaz de percepción, computación y actuación [...] que tiene como objetivo conseguir una meta o mantener cierto estado".

El esquema de JDE comparte con el agente la cualidad de poder activarse y desactivarse a voluntad, y que ambos tienen cierto ritmo propio de ejecución. El agente es autónomo en sí mismo, porque engloba tanto la percepción como la actuación necesarias para el comportamiento. El esquema por el contrario no es completo en este sentido, puesto que, por definición, se ocupa exclusivamente de una de éstas facetas: o bien percepción, o bien actuación, pero no de ambas a la vez. Un esquema por sí mismo no es autónomo, no desarrolla ningún comportamiento. Es la conjunción de varios esquemas la que puede generar una conducta observable.

3.5.3. Jerarquía para crecer en complejidad

Como antes mencionabamos la navegación ha sido históricamente el primer problema con el que se han enfrentado los robots móviles. A medida que se han ido desarrollando las técnicas que lo solventaban ha surgido la necesidad de aumentar el repertorio de comportamientos con nuevas funcionalidades. Un buen ejemplo de ello son los robots de entretenimiento, como el perrito Aibo de Sony [Arkin *et al.*, 2003], que ofrecen un amplio abanico de conductas. Cuando se enriquece el repertorio se complica la arquitectura, ya que debe integrar los mecanismos de generación de todos los comportamientos que puede exhibir el robot y resolver las relaciones entre ellos. Con ello se ha ido perfilando la organización del sistema como un problema en sí mismo y como un factor crítico a la hora de conseguir el sistema completo o no. Además, el continuo crecimiento de la potencia de cómputo disponible también ha ido borrando los requisitos que antes imponían ciertos principios de organización, como la existencia de un único proceso.

JDE aborda cuestiones genéricas de arquitectura, enfocando más el problema de la organización del sistema, que la solución a una conducta concreta. En esencia, JDE parte de los sistemas basados en comportamientos y surge como alternativa a los sistemas híbridos de tres capas para crecer en capacidad. Permite la utilización de partes reactivas y partes deliberativas, siempre que ambas cumplan el interfaz de esquema y entreguen una recomendación de acción en cada iteración. En lugar de encapsular la funcionalidad reactiva en una capa y adosarle una o dos capas deliberativas más para gobernarla, se plantea el uso recursivo de los esquemas, donde la salida de unos es la activación y modulación de otros, que a su vez pueden activar y modular a sus propios hijos, y así sucesivamente. De este modo se configura una jerarquía de esquemas para afrontar cierta tarea.

La fragmentación en esquemas permite la estrategia divide y vencerás, que simplifica la resolución de cada tarea. La jerarquía es el principio que se propone para abordar la complejidad creciente de organización del sistema. Una alternativa a la jerarquía es la de un único nivel donde conviven todos los esquemas, cada uno a su ritmo. Sin embargo, la jerarquización aborda la división en niveles de abstracción y permite así acotar la complejidad de los problemas en cada nivel: la monitorización, la selección de acción, la inserción de nuevos esquemas, etc. Por ejemplo, con la jerarquía se simplifica el mecanismo de selección de acción porque marca un contexto de actuación. Los niveles sirven para ordenar con qué otros agentes se compite con el control, y garantiza la consistencia. El que gana en un nivel superior condiciona quiénes competirán en los inferiores. Así la selección de acción no se plantea como una competición de todos contra todos, que sería muy complicada de ajustar, sino que la organiza en pequeñas competiciones parciales, una por cada nivel. Además en cada conjunto coyuntural de esquemas hermanos, no hay que resolver la selección de acción para todas las situaciones, sólo para aquellas que pueden aparecer en el contexto en el que el padre común está activo. Adicionalmente la

jerarquía también ofrece un contexto de percepción lo cual posibilita la atención e interpretación y facilita la coordinación entre percepción y acción.

Construcción incremental

Una de las ventajas que ofrece JDE es que la jerarquía propuesta simplifica la inserción de nuevos esquemas, y por ello hace a la arquitectura extensible. Normalmente se introducirán nuevos esquemas para implementar un nuevo comportamiento o mejorar uno existente. Añadir un nuevo esquema en JDE es bastante fácil: requiere definir sus parámetros, su función que ejecuta iterativamente y las relaciones de arbitraje con otros esquemas.

En cuanto al arbitraje no hay que reprogramar la coordinación de todos los demás esquemas existentes entre sí. Únicamente hay que estudiar la coordinación específica para la nueva tarea, o retocar las relaciones con sus hermanos en aquellos contextos en los que participará el nuevo esquema, *y sólo en ellos*. El resto sigue exactamente igual. Gracias a los contextos que proporciona la jerarquía se mantiene acotada la complejidad de añadir una nueva tarea. Esto contrasta, por ejemplo, con las redes de activación de Maes [Maes, 1989a], donde la inserción de un nuevo nodo implica modificar las relaciones de los ya existentes entre sí puesto que se incrementa el número de nodos que satisfacen las precondiciones o cuyas postcondiciones se relacionan con las precondiciones del nuevo, y eso afecta a la dinámica de propagación de energías. Esto obliga a un reajuste de los pesos para los comportamientos anteriores, amén del necesario para el nuevo comportamiento.

En cuanto a la implementación del nuevo esquema, JDE permite la reutilización de los ya existentes. Todos los esquemas ya presentes, codificados anteriormente, pueden reutilizarse como bloques básicos con los cuales construir el nuevo comportamiento. La reutilización se materializa siempre en el formato de activación y modulación, que es siempre el mismo en todos los niveles. Esta manera de reutilización recalca el sentido evolutivo de los comportamientos que apuntan los etólogos: los nuevos esquemas se incorporan sobre la base de los ya existentes. Desde la perspectiva animal se podría decir que esto condiciona el desarrollo filogenético del repertorio de comportamientos.

Una característica genuina de JDE es que permite la coexistencia de varias versiones del mismo esquema. Esto facilita que no haya que reprogramar los esquemas de nivel alto cuando se añaden versiones nuevas de esquemas de bajo nivel al conjunto. Los que ya lo utilizaban pueden seguir utilizando la versión antigua, que sigue disponible en la población de esquemas, u opcionalmente adaptarse para emplear la nueva.

La escalabilidad es una propiedad deseable de las arquitecturas, en el sentido que se pueda expandir para desarrollar comportamientos más complejos o un repertorio más extenso de ellos. Sin embargo, la escalabilidad es una propiedad muy difícil de conseguir. Muchos sistemas se han arrogado esta característica, aunque los experimentos reales hayan mostrado posteriormente las limitaciones de la expansión. Por ejemplo, los sistemas basados en comportamientos de Brooks [Brooks, 1986], que prometían una sencilla incrementalidad hasta capacidades humanas, subiendo en complejidad sin más que añadir de modo simple nuevos niveles de competencia a los ya existentes. Posteriormente se matizaría esta promesa [Brooks, 1991b], aunque siempre ha sido una aspiración de los sistemas basados en comportamientos (por ejemplo, el robot humanoide COG apuntaba hacia comportamientos humanos).

Subsunción de Brooks

Una vez que hemos discutido algunas ventajas sobre la jerarquía que proponemos con JDE, vamos ahora a comparar nuestra propuesta con otras interpretaciones de jerarquía que se han hecho en el campo de las arquitecturas.

JDE tiene varios puntos en común con la arquitectura de subsunción de Brooks, y en general con los agentes situados [Maes, 1990]. La principal coincidencia es la descomposición del sistema de control. En la arquitectura de subsunción *la computación se organiza como una red asíncrona de elementos computacionales activos en una red de topología fija de conexiones unidireccionales* [Brooks, 1991b]. Esta organización la comparte JDE, con la salvedad de que en JDE la topología es cambiante, la red se reconfigura dependiendo de la tarea en curso. Además en ambas se pueden reutilizar partes de la

implementación de un comportamiento para materializar uno nuevo diferente. Otra similitud más es que tanto JDE como Brooks escalan la complejidad con mayor número de niveles de abstracción y utilizan activación exclusiva, no utilizan la fusión de comandos.

Sin embargo hay diferencias fundamentales. Primero, la pieza fundamental del comportamiento para Brooks son los niveles de competencia, mientras que en JDE son los esquemas, que son de grano más fino. Los niveles de competencia elaboran funcionalidad apoyándose en sus inferiores, y una vez que está cerrado el diseño de un nivel no se puede retocar (podría afectar a los superiores). En JDE no hay niveles preestablecidos per se, se forman y de destruyen ad hoc en tiempo de ejecución según conviene para la tarea en curso. Al ser dinámicos y dependientes de la tarea no hay niveles como entidad propia, es meramente coyuntural, circunstancial.

Segundo, JDE difiere de la subsunción en que los esquemas no alimentan a los actuadores directamente, como sí lo hacen los niveles de competencia. Los esquemas superiores no dan salida directamente a los actuadores, en lugar de eso activan y parametrizan a los inferiores. Esta modulación continua a través de parámetros permite una reutilización más flexible que la subsunción.

La arquitectura de subsunción de Brooks es una jerarquía *sustractiva*: todos los autómatas están funcionando y luego los activos de alto nivel deben explícitamente subsumir (anular) las salidas de los de bajo nivel que no conviene se activen en esa situación. Por defecto están activos y se pueden anular. Por el contrario, JDE es *aditiva*: por defecto todos los esquemas están en DORMIDO, desactivados, y los esquemas superiores deben despertar explícitamente a los inferiores que sean relevantes para cierta situación. Normalmente los esquemas relevantes formarán un conjunto menor que los no-relevantes.

Este planteamiento facilita la inclusión de nuevos esquemas de bajo nivel. En subsunción, una vez que se ha programado un nivel, no se debe abrir más, pues si se añade algo nuevo en el bajo nivel, aquellos autómatas del alto nivel no anulan a estos nuevos del nivel inferior. Los de alto nivel tendrían que *recompilarse* para que anularan a estos nuevos también, si es que no los necesitan. Por el contrario, como ya hemos mencionado, con JDE las nuevas versiones de esquemas pueden coexistir con las antiguas de modo que los cambios en un esquema muy reutilizado no obligan a reprogramar todos los esquemas que usan su funcionalidad. Los nuevos utilizarán la nueva versión, y los antiguos siguen funcionando con la antigua versión (dado que la nueva versión por defecto está desactivada). Esto es parecido al uso de librerías dinámicas en programación. Con ello la incorporación y retoques de ciertos niveles queda muy abierta a modificaciones futuras, incorporando nuevos esquemas.

Ronald C. Arkin

Uno de los investigadores que más ha influido en la concepción de JDE ha sido Ronald C. Arkin. En sus orígenes planteó el uso de los esquemas en robótica[Arkin, 1989b], tanto perceptivos como motores, idea que sigue al pie de la letra nuestra propuesta. Así, dos puntos en común entre sus sistemas y JDE son la incorporación explícita de la percepción a la arquitectura a través de los esquemas perceptivos, y que la percepción sea orientada a la acción.

Una de las diferencias entre los esquemas de Arkin y los de JDE es que en sus trabajos iniciales se emplea la fusión de comandos como mecanismo básico de composición de comportamientos y coordinación [Arkin, 1989b]. De hecho, propone la superposición, la combinación lineal de las salidas de los esquemas activos para calcular el valor final que se envía a los actuadores. Las distintas combinaciones lineales generan nuevos comportamientos observables. Además, todos sus esquemas están al mismo nivel, no hay jerarquía, las salidas de los esquemas son directamente recomendaciones de actuación. Por el contrario, JDE utiliza la jerarquía, es decir, varios niveles, y dentro de cada uno de ellos una competición por el control en la que sólo puede haber un ganador (*winner takes all*, activación exclusiva). La coordinación se resuelve con la activación exclusiva de un esquema motor por nivel. Otra diferencia, corolario de la anterior es que en Arkin no hay modulación, en JDE sí.

La evolución de sus trabajos es reveladora y apunta nítidamente a la etología como fuente de inspiración para los programadores de robots. Inicialmente apostó por subir en complejidad utilizando un autómata de estados finito como árbitro [Arkin y MacKenzie, 1994], de modo que variara el repertorio de esquemas activos a lo largo del tiempo. Es decir, por la secuenciación de esquemas como método fundamental de coordinación. Otra de las alternativas que probó para crecer en capacidad fue la inserción de una capa de esquemas dentro de la arquitectura híbrida AuRA [Arkin, 1989a],

con una capa deliberativa encima. Finalmente, en sus trabajos más recientes [Arkin *et al.*, 2001; Arkin *et al.*, 2003] se ha acercado enormemente a planteamientos etológicos y jerárquicos para organizar el comportamiento y los esquemas con que se genera. Fruto de ello son sus colaboraciones con Sony para diseñar el sistema de conductas del perro robótico Aibo o el humanoide SDR.

Control jerárquico de Albus y Meystel

Según vimos en el capítulo 2, otras arquitecturas que también utilizan la jerarquía como principio organizador son las propuestas por Albus y Meystel. Un parecido de nuestra propuesta con ellas es que tanto RCS como JDE emplean varios niveles de abstracción para resolver las tareas, ambas utilizan la jerarquía para abordar la complejidad de modo incremental. En RCS los niveles superiores manejan términos lingüísticos y tienen un tiempo de respuesta más lento que los niveles inferiores. Además manipulan información más abstracta, lo que le permite tomar decisiones *de grano grueso*. Los niveles más bajos manipulan información altamente dinámica, como las lecturas crudas de los sensores, que tienen que ser procesadas en rápidos bucles de control para derivar los comandos de control. En JDE también se usan los niveles, y los más bajos suelen estar más próximas a los sensores y actuadores reales.

Otro parecido es la incorporación explícita de la percepción dentro de la arquitectura, como ampliaremos posteriormente. A pesar de estos parecidos, es más lo que separa JDE de estas arquitecturas de lo que les une. Por ejemplo, una diferencia fundamental es la rigidez de las capas. Mientras que la jerarquía es dinámica en JDE y los niveles son meramente coyunturales, en Albus son estructurales y la jerarquía es estática. De hecho, Albus asume la existencia de niveles per se y argumenta que siete niveles es el número óptimo para englobar cualquier tarea [Albus, 1993]. Esta afirmación, al igual que cualquier otra cantidad fija, nos parece demasiado pretenciosa. En JDE el número de niveles depende de la complejidad de la tarea a resolver, y puede crecer en tiempo de ejecución (como TCA con su árbol de tareas). Habrá tareas que se puedan realizar con uno o dos niveles y tareas para las cuales se necesiten siete u ocho. La posibilidad de reconfiguración proporciona gran flexibilidad a la hora de que la arquitectura se adapte a la complejidad requerida por la tarea. Los esquemas no pertenecen a un nivel per se. Mientras que en una jeraquización fija, estratificada, se obliga a utilizar todos los niveles.

Otra gran diferencia es que dentro de cada nivel Albus propone un mecanismo genérico de resolución de problemas, que se aplica a todos los niveles: procesamiento sensorial, representación del mundo, juicio de valor y descomposición de tareas. Por el contrario, en JDE cada esquema sabe hacer una cosa y es especialista en su tarea concreta. Cómo se hilvanan varios esquemas en cada nivel para resolver una tarea conjunta depende del comportamiento concreto y la arquitectura JDE no regula nada.

Además la descomposición que se propone en RCS de tareas en subtareas es *procedimental* (esperando éxito o fallo), mientras que en JDE la tarea se descompone en varios esquema funcionando en paralelo. En este sentido, en RCS se señala un curso específico de la acción, mientras que en JDE se manejan predisposiciones a alternativas de acción, como ya vimos. Otras discrepancias tienen que ver con el caracter instantáneo que rezuma en JDE y la escala de tiempos que se tiene en RCS, donde hay planificación que tiene en cuenta el futuro de modo explícito.

Adicionalmente, en la propuesta Meystel hay dos flujos de información generales: uno ascendente, que generaliza la información perceptiva relevante hacia arriba, y uno descendente, que particulariza las actuaciones de un nivel de abstracción a otro menor. Estos saltos de abstracción son obligados y hay exactamente tantos en percepción como en actuación. Como veremos, esto dificulta la coherencia de mantener una representación distribuida, y que la parte de actuación de un nivel pueda consultar la información que elabora otro nivel diferente. En JDE también se dan esos flujos, pero no están acoplados: en el tercer nivel de abstracción se pueden emplear estímulos extraidos directamente de los datos sensoriales, o incluso con 2 niveles de abstracción perceptiva para construirse.

Una variante de las jerarquías de Albus y Meystel es la arquitectura AMARA [García-Alegre *et al.*, 1993], que consiste en poblar cada una de las capas de agentes, manteniendo la división por niveles de abstracción. De hecho, "cada nivel corresponde a un conjunto de agentes que comparten un lenguaje común con sus interlocutores en niveles adyacentes" [García-Alegre *et al.*, 1993]. Los

esfuerzos por incluir en AMARA la activación y modulación [Cañas y García-Alegre, 1999b] sirvieron como embrión de la actual arquitectura JDE. La principal diferencia es que JDE tiene al esquema como unidad básica, mientras que AMARA utiliza agentes especialistas, que ya comentamos anteriormente. Además, AMARA respeta íntegramente el control jerárquico que comparte con Albus y Meystel.

Etología

El ascendiente de la etología sobre JDE es enorme. Como ya mencionamos en el capítulo 2, pensamos que esta disciplina tiene mucho que aportar a la robótica y que puede servir como fuente de inspiración. Su principal virtud es que han abordado el problema del comportamiento desde una óptica causal, buscando los mecanismos que causan la conducta. Una característica genuinade la etología es que cuenta con sujetos que realmente se comportan de cierta manera, los animales, y tiene por ello una vocación descriptiva y un entronque directo con la neurofisiología y la biología en sentido amplio. En contraste, la robótica parte de una página en blanco, el cuerpo de estudio es un ingenio artificial, con actuadores y sensores funcionalmente equivalente a los músculos, o los sentidos, pero totalmente diferentes.

La idea de jerarquía también se ha empleado en etología para describir el comportamiento, en particular el comportamiento innato en animales como los pájaros, que suponen un salto en complejidad respecto de los insectos (aún lejos del abismo de los mamíferos superiores). JDE le debe mucho a la jerarquía y a los mecanismos de selección de acción propuestos por los etólogos Lorenz [Lorenz, 1978] y Tinbergen [Tinbergen, 1950]. Comparte con ellos la división del sistema en nodos, que para Tinbergen son centros instintivos y para JDE esquemas, los cuales se integran en una estructura con forma de árbol invertido. En esa jerarquía el flujo de las activaciones fluye de arriba a abajo, entendidas éstas como predisposiciones.

Otra característica diferenciadora de JDE con respecto de sus ascendientes etológicos es la inclusión explícita de la percepción en la jerarquía. Mientras que la jerarquía en Tinbergen contempla fundamentalmente la coordinación entre nodos motores, en JDE se vertebra también la coordinación entre esquemas perceptivos y motores. El estudio de los estímulos, su naturaleza y el efecto en los mecanismos desencadenantes de los comportamientos ha sido ampliamente estudiado en la etología. También en neurofisiología se ha reconocido su papel modulador en la ejecución de los esquemas motores. Sin embargo, no hemos encontrado una formalización que aborde de manera sistemática la coordinación de la percepción con la acción, y su doble función en la activación y ejecución de los esquemas.

Jerarquía borrosa de Sugeno

Otra interpretación diferente de la idea de jerarquía para generar comportamientos cada vez más complejos es la jeraquía de controladores borrosos de Sugeno [Sugeno, 1999]. De modo análogo a JDE, las variables de salida del controlador en un nivel modulan a los controladores en el nivel inferior, pues se utilizan directamente como consignas de referencia para los controladores inferiores.

La jerarquía de Sugeno nace con vocación particular, para resolver la navegación autónoma en un helicóptero de aeromodelismo, lo que le hace difícilmente expandible o portable a otros escenarios. En contraste, JDE nace con vocación generalista. Esto se refleja de manera particular en los mecanismos propuestos para resolver la coordinación entre controladores o esquemas diferentes. En la arquitectura de Sugeno todos los controladores básicos están activos a la vez, lo cual implica una coordinación complicada. Para resolver posibles acoplamientos indeseados entre estos controladores se sitúan controladores ad hoc específicos en el nivel inmediatamente superior. Por el contrario, JDE propone un mecanismo genérico basado en contextos y arbitraje de la activación de los hijos.

La principal diferencia de JDE con la jerarquía borrosa de Sugeno es la capacidad de fusión que exhibe esta última, mientras que en la jerarquía de JDE la activación en cada nivel es excluyente. Por ejemplo, el comportamiento de ascenso frontal en diagonal se puede conseguir como la activación simultánea de los controladores `arriba` y `adelante` y la fusión borrosa de sus salidas en ese nivel, por debajo del cual funcionan los controladores básicos. Por contra, en JDE sólo puede haber un esquema realmente activo en cada nivel, un único ganador de la competición por el control.

Sistemas híbridos y fusión contextuada de Saffiotti

Otra alternativa interesante que también emplea fusión borrosa de comandos es la propuesta en la arquitectura híbrida de Saffiotti [Saffiotti y Wasik, 2003] y en Saphira [Konolige y Myers, 1998]. En ella los comportamientos reactivos se expresan como una colección de reglas borrosas de control. El nivel deliberativo puede activar selectivamente un conjunto de ellos, conformando así una colección de reglas borrosas. Cada comportamiento reactivo se acompaña de una prioridad y de unas metareglas que marcan en qué medida es aplicable para la situación actual, lo que Saffiotti y Konolige llaman *contexto de activación*. Finalmente se utiliza el formalismo borroso para combinar todas las reglas activas.

Saphira es una muestra de cómo los sistemas híbridos tratan de subir en complejidad frente a los sistemas basados en comportamientos. Por ejemplo, la activación selectiva de la capa reactiva que se usa en Saphira también se presenta en otras arquitecturas híbridas, como la de tres capas [Bonasso *et al.*, 1997]. De modo genérico, en los sistemas híbridos el nivel deliberativo (el planificador o el secuenciador) determina los controladores borrosos activos en el nivel reactivo. Esto imprime un sesgo finalista en las partes reactivas que se activan, puesto que la parte deliberativa activa cada comportamiento reactivo sólo cuando colabora a la consecución del objetivo. Esta activación selectiva es similar a la que se presenta en JDE cuando el padre decide activar a sus hijos y no a otros esquemas. Sin embargo, en las arquitecturas de tres capas sólo hay *un* contexto, es decir, dentro del único nivel reactivo hay una sopa de comportamientos reactivos y el nivel deliberativo elige entre ellos cuáles se activan y cuáles no, parametrizándolos ocasionalmente. Por el contrario, en JDE los contextos se manejan recursivamente, uno por cada nivel de abstracción. En cada nivel de la jerarquía se determina un contexto despertando a los esquemas correspondientes.

Los contextos, tanto el marcado por los niveles superiores como el que determina cuándo es aplicable un comportamiento (en el sentido de Saffiotti y Konolige), se pueden ver como un mecanismo para acotar la complejidad en los sistemas de reglas. En este sentido, una arquitectura posible de control es una gran base de reglas de producción. La analogía con JDE es clara: los contextos deciden qué subconjuntos se aplican en cada momento. Además los esquemas perceptivos serían los que proporcionan el valor de los antecedentes de las reglas, mientras que los de actuación serían la parte de consencuentes. La estructuración jerárquica de los contextos que se da en JDE simplifica su organización y, como ya hemos visto, facilita el crecimiento o modificación del sistema.

3.5.4. Percepción en la arquitectura

Una vez que hemos discutido la unidad de comportamiento de JDE y comparado la jerarquía propuesta con otras alternativas para crecer en complejidad, comentaremos ahora la tercera piedra angular de JDE: la inclusión de la percepción en la arquitectura.

En los sistemas más antiguos la percepción del entorno se formalizaba en lógica de predicados, como hechos lógicos. Este formato estaba lejos de los sensores reales, pero próximo a los sistemas de inteligencia artificial que manipulaban estos símbolos. Un ejemplo curioso no deliberativo que emplea esta percepción es la red de nodos de Maes [Maes, 1990]. En sistemas con mayor énfasis deliberativo era habitual el uso de un motor de inferencia, que deducía nuevos hechos desde unos iniciales. Típicamente los sistemas de deliberación que implementaban razonamiento sobre esta percepción necesitaban de la hipótesis de mundo cerrado. A medida que aumentaron los trabajos con robots reales se vio que esa hipótesis no era asumible. Un robot autónomo debe estar alerta continuamente, tener un sistema perceptivo que detecte los hechos relevantes que puedan surgir en el mundo, aunque no hayan sido originados por él mismo.

En los trabajos iniciales de Arkin la percepción se articula como un conjunto de esquemas perceptivos, todos al mismo nivel [Arkin, 1989b]. Tanto el esquema que percibe un obstáculo como el que refleja la existencia de un pasillo están a la misma altura. Es decir, se tiene una colección plana de estímulos. JDE también presenta una distribución de la percepción. A diferencia de los esquemas de Arkin, JDE ofrece percepción estructurada, en la cual un estímulo puede depender de otros de nivel inferior, lo que se refleja en una jerarquía de esquemas perceptivos homóloga.

Además, los esquemas perceptivos de Arkin están indisolublemente asociados a una recomendación

de actuación. Por ejemplo: alejarse de los obstáculos y acercarse al centro del pasillo. Por el contrario, como ya señalamos en la sección 3.4, en JDE el estímulo no tiene asociada ninguna acción por defecto. Ello facilita que el mismo estímulo pueda servir para actuaciones diferentes.

Una arquitectura que también ofrece percepción estructurada es Saphira [Konolige y Myers, 1998]. En ella varias rutinas perceptivas construyen de modo distribuido una representación estructurada del entorno, denominada espacio perceptivo local. En este espacio se tienen símbolos como `hueco-de-la-puerta` y `pared`, e incluso `pasillo`, que se compone a partir de dos primitivas `pared` que sean paralelas. Un parecido con JDE es la preocupación en Saphira por tener permanentemente anclados los símbolos en la situación real. En cuanto a las diferencias, la representación del entorno en JDE se construye y almacena de manera distribuida, mientras que en Saphira la construcción es distribuida, pero se centraliza en una estructura común, a modo de pizarra.

Manejo de símbolos

Como ya avanzamos en 3.4, en JDE pueden utilizarse símbolos abstractos. Un esquema perceptivo puede elaborarlos y actualizarlos, mientras otros esquemas pueden consultarlos, tanto si son motores como si son perceptivos. En este sentido, nos distanciamos de la escuela reactiva, que argumenta fuertemente en contra del uso de símbolos [Brooks, 1991c] y se limita a enganchar acciones sobre las lecturas directas de los sensores o transformaciones sencillas a partir de ellas. Un estímulo en JDE puede consistir en simples lecturas sensoriales subsimbólicas, pero no se obliga a que todos sean así, como ocurre en los reactivos.

En contraste con los sistemas deliberativos clásicos, los símbolos en JDE están muy anclados en el mundo real, con un esquema perceptivo que se encarga de mantener actualizada su correspondencia con la realidad. Este mantenimiento le confiere un dinamismo del que carecen muchos símbolos clásicos.

Además, en los sistemas deliberativos los símbolos suelen tener una naturaleza objetiva, en el sentido de que representan propiedades inherentes a los objetos del mundo, con independencia del observador. Por el contrario, los símbolos en JDE pueden tener un carácter subjetivo, en sintonía con la percepción deícitica y funcional que propone Agre [Agre y Chapman, 1987].

Percepción continua de los sistemas basados en comportamientos

Frente a la ceguera típica de los sistemas deliberativos, debida a un exceso de confianza en los modelos y a la hipótesis del mundo cerrado, los sistemas reactivos proponen una percepción siempre encendida. En este sentido incorporan la percepción funcionando permanentemente a la arquitectura, en su caso los sensores. Este mismo planteamiento heredan los sistemas basados en comportamientos, que de este modo son capaces de reaccionar ante cualquier evento, previsto o no. En este planteamiento no hay atención alguna, se percibe siempre todo lo perceptible. Tiene la ventaja de que nunca deja de percibir algo que ocurre en el mundo, puesto que siempre está buscándolo.

Sin embargo, este planteamiento de la percepción no es escalable. Es viable cuando el repertorio de comportamientos es escaso y se tienen sensores sencillos que proporcionan poco ancho de banda como los infrarrojos, los sónares, los táctiles, etc. cuyo procesamiento e interpretación son directos. Cuando el repertorio de comportamientos crece, se multiplica el número de estímulos perceptibles. Además, cuando los estímulos se apoyan en sensores más densos en datos, o cuando los estímulos demandan bastante computación (como los definidos sobre imágenes), entonces el coste computacional de tenerlo todo activo se dispara. En ese caso el sistema no puede estar permanentemente buscando y elaborando todos los estímulos posibles, ejecutando simultáneamente todos los algoritmos perceptivos porque el robot agotaría sus recursos computacionales.

Por ejemplo, en la arquitectura de subsunción de Brooks todos los sensores y los autómatas que procesan esa información están funcionando continuamente. Si sus datos no son necesarios en un momento dado, su salida simplemente será subsumida (suprimida) por un nivel superior [Brooks, 1986]. En contraste, JDE también incluye explícitamente la percepción en la arquitectura, pero maneja contextos en percepción de manera que si el estímulo de cierto esquema perceptor no se utiliza, entonces ese esquema está DORMIDO, sin consumir capacidad de cómputo alguna. Ésta es la capacidad de atención que comentamos anteriormente.

En cuanto a la reutilización del procesamiento perceptivo, en los trabajos de Brooks se reduce a pinchar en el hilo adecuado para capturar la salida de cierto autómata. Esto implica que hay que conocer el funcionamiento interno del nivel de competencia inferior que se quiere usar. El nivel se implementa como una red de autómatas interconectados, y hay que conocerla para pinchar en el hilo adecuado. Frente a eso, JDE ofrece un formato más homogéneo: sólo hay que saber las salidas del esquema y activarlo.

Jerarquía perceptiva de Meystel y Albus

Otro planteamiento explícito de la percepción es el sistema multirresolución en Meystel [Meystel, 1998]. En su jerarquía la percepción está distribuida por niveles, siguiendo unos principios de resolución (espacial y temporal) concomitantes a cada nivel. Típicamente los niveles inferiores tienen información volátil y local, prácticamente numérica, que puede evolucionar muy rápidamente. Los niveles superiores tienen información más abstracta, simbólica, que tiene una evolución más lenta y un mayor horizonte temporal. Esta descomposición en niveles se apoya en el hecho de que la unidad mínima de un nivel perceptivo se construye desde un conjunto de primitivas del nivel inferior. Para su mantenimiento y actualización, cada nivel tiene un proceso que se encarga de actualizar la representación del mundo en esa capa, únicamente desde las primitivas que elabora el nivel inmediatamente inferior. En cuanto a su coordinación con la acción, las partes que generan comportamiento en un nivel sólo pueden utilizar la información disponible en su nivel.

Esta organización de la percepción es muy ineficiente y problemática. Para empezar, la resolución temporal y de abstracción van de la mano, de manera que las decisiones abstractas no pueden tomarse, por la propia organización del sistema, sobre información rápida. ¿Qué pasa si un nodo elevado necesita información muy inmediata para tomar sus decisiones, por ejemplo, los datos sensoriales crudos? La jerarquía de Meystel y Albus se lo impide.

Por el contrario, en JDE un esquema conceptualmente abstracto puede depender para su funcionamiento de información muy rápida. Si un nodo de gran nivel de abstracción necesita datos sensoriales numéricos volátiles para tomar una decisión basta con que active el esquema perceptivo correspondiente, en el nivel en que ese esquema motor se encuentre. En JDE el criterio de organización de la información perceptiva no es absoluto, ni atiende a la naturaleza del estímulo, sino que está orientado a la tarea. Por lo tanto es más flexible que unos niveles definidos para todas las posibles tareas.

Otro de los problemas de la jerarquía perceptiva de Albus y Meystel es la dificultad para adscribir estímulos a los niveles, y su mantenimiento respetando la jerarquía. Esto genera el problema de mantener la consistencia entre las representaciones de un nivel y otro, y provoca grandes retrasos artificiales porque el flujo de actualización ha de subir por toda la jerarquía antes de actualizar los cambios en un nivel elevado. Un modo de saltárse esta rigidez es tener un sistema de pizarra al que todos los niveles tienen acceso, y volcar en ella toda la información perceptiva, ya sean lecturas sensoriales, o bien estímulos elaborados. Si se necesita de una estructura perceptiva, ésta puede aparecer en esa pizarra de modo independiente a los niveles de actuación (como ocurre en Saphira).

Otra discrepancia entre JDE y la percepción planteada en el control jerárquico de Albus es que en JDE la percepción es orientada a la tarea, e incluso tiende a ser subjetiva, funcional e indexada. Por el contrario, en Albus tiene un matiz de objetividad. La percepción busca mantener la correspondencia entre el modelo interno y el mundo externo, que se entienden son homólogos objeto a objeto.

Alternativas de atención

Un planteamiento complementario a la percepción continua es la percepción puntual. Conscientes del elevado coste computacional de elaborar algunos estímulos, ciertos trabajos no tienen al robot con los ojos permanentemente abiertos, sino que los abren en ciertas circunstancias. Por ejemplo, en la arquitectura TCA de Simmons [Simmons, 1994] la percepción no es continua y se incorpora al plan como una *acción* más, es decir, hay una orden explícita de percibir. Esto marca una cierta atención porque se busca percibir algo concreto, bien sea para la ejecución de un módulo, bien sea para su monitorización. Así por ejemplo, hay un mensaje `dame-imagen` porque a algún módulo le interesa esa

información en ese momento.

Otro ejemplo, muy antiguo, es el sistema perceptivo de Shakey [Nilsson, 1969]. Si bien se mantenía un mapa de ocupación del entorno obtenido desde visión, la cámara no se tenía activa permanentemente porque el procesamiento de la imagen era muy costoso. Una vez que se había construido el mapa se generaba una trayectoria y se activaban los sensores de contacto (cuyo procesamiento es muy rápido) para detectar choques. La reactualización del mapa desde visión se lanzaba cuando los sensores de contacto detectaban algún obstáculo y eso marcaba una inconsistencia entre la realidad y el último mapa generado.

Otra implementación de atención son los trabajos de Saffiotti [Saffiotti y Wasik, 2003], que asocia un valor de necesidad a cada variable a percibir. Esto le sesga en todo momento qué variables son las más necesarias. Esa necesidad no es constante, sino dinámica, dependiendo de la tarea en curso y la fecha de la última actualización. De este modo, por ejemplo, mantiene con una cámara direccional una representación actualizada de un entorno que no se puede cubrir con un solo vistazo de la cámara. Al tener una estimación de la necesidad de ciertos estímulos u otros, puede implementar políticas de percepción activa, moviendo la cámara a las zonas que ayudan a actualizar las variables más demandadas. Por ejemplo, alternando temporalmente entre contemplar la pelota y la portería contraria.

Otra posibilidad de organizar la percepción es imbricarla completamente con la actuación. Este planteamiento se da en los agentes de sistemas como [García-Alegre y Recio, 1998]. De esta manera se resuelve la atención, porque la parte perceptiva de un agente se activa al tiempo que lo hace el propio agente. Aunque esta opción es buena para resolver la coordinación entre percepción y actuación, dificulta la reutilización del procesamiento perceptivo, o incluso su compartición dentro del mismo comportamiento. Por ejemplo, los agentes de diferentes niveles pueden necesitar literalmente la misma información para tomar sus decisiones. Si la parte perceptiva está dentro de cada agente, entonces no queda más remedio que replicar el procesamiento en cada uno de ellos. Esta característica complica la escalabilidad de la arquitectura, que debe apoyarse en la reutilización de sus partes.

3.5.5. Selección de acción

El mecanismo de selección de acción de JDE es una herencia de los modelos psicohidráulicos de Lorenz [Lorenz, 1978]. Como vimos, plantea cuatro estados DORMIDO-ALERTA-PREPARADO-ACTIVO, y tres saltos necesarios desde el estado de reposo a la actividad, lo que garantiza la doble orientación finalista y situada de los esquemas activos. En primer lugar, el salto de DORMIDO a ALERTA lo provoca el esquema padre y marca la orientación del esquema hacia los objetivos. El salto de ALERTA a PREPARADO lo determina la coyuntura del entorno, y marca la idoneidad del esquema en la situación actual. Finalmente, el salto de PREPARADO a ACTIVO selecciona al más adecuado, y sólo será problemático cuando más de un esquema hermano esté PREPARADO. En ese caso puede solucionarse con un arbitraje explícito del padre para ese contexto.

En la literatura existen muchos mecanismos de selección de acción, cada uno con características genuinas. Paolo Pirjanian hace una buena recopilación en [Pirjanian, 1997b]. Por ejemplo, tal y como vimos en el capítulo 2, Maes también consigue con su red de activación la doble orientación finalista y situada. La energía de activación emana de los objetivos a conseguir y de la situación del entorno, propagándose a los nodos cuyo consecuente son los objetivos o aquellos cuyo antecedente casa con la situación actual. En cuanto a la propagación interna, se siguen unos criterios similares, donde unos nodos aparecen como precondiciones ($\approx$ situación actual) o postcondiciones ($\approx$ objetivos) de otros. Una desventaja de este mecanismo, que no tiene el de JDE, es que es dificil incorporar un nuevo comportamiento. Para ingresar un nuevo nodo hay que recomputar los parámetros de propagación de activación de toda la red, incluso para los *contextos* donde no interviene directamente. Este mismo defecto ocurre en la selección de acción de Ludlow [Ludlow, 1976], pues la incorporación de un nuevo nodo altera la dinámica de activaciones entre los ya existentes.

JDE también ha recogido de la etología el mecanismo de selección de acción, basado en combinar el efecto de varios estímulos, externos o internos, con un valor umbral. Así, la adición de motivación y la inhibición lateral dentro de los nodos del mismo nivel aparecen también en esas arquitecturas etológicas. En contraste, la dinámica de energías de activación propuesta en JDE no es tan compleja

como los estudios etológicos. Por ejemplo, no hay inercia ninguna en cuanto a la activación, con idea de poder reaccionar rápido en caso de que sea necesario. Por ello no hemos estudiado cómo materializar los mecanismos de histéresis, tales como la fatiga específica de acción, o si éstos son beneficiosos en los robots. Tampoco hemos estudiado en profundidad que ese umbral de activación pueda variar en el tiempo, ni el juego que ofrece esa dinámica, entroncandola con fenómenos como los *comportamientos de desplazamiento*.

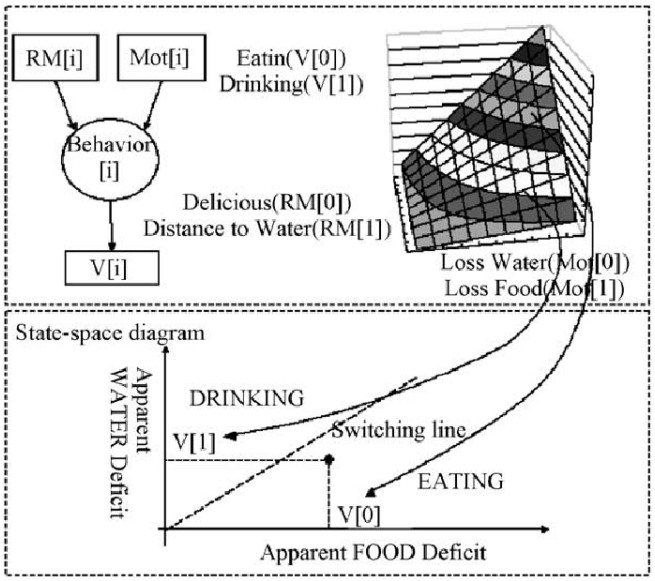


Figura 3.15: *Diagrama de estados que Arkin propone para resolver un arbitraje [Arkin* et al.*, 2003]*

La selección de acción de JDE también es similar al espacio de estados (*state-space diagram*) que utiliza Arkin en sus trabajos más recientes [Arkin *et al.*, 2001] para arbitrar entre comportamientos específicos. En el espacio de estados se establece una superficie de arbitraje que decide cuál de los esquemas contendientes realmente gana la competición por el control en cada instante. La figura 3.15 muestra un ejemplo concreto incluido en [Arkin *et al.*, 2003], donde se utiliza la superficie de control para decidir si conviene activar al esquema `comer` o al esquema `beber` en las distintas combinaciones de hambre y sed. El ganador no es necesariamente el de mayor activación numérica, porque la línea divisoria es una curva genérica, no tiene porqué ser la bisectriz del primer cuadrante. En JDE no tenemos números (*cuantificación* de la motivación de un esquema) y sí varios estados de activación (*cualificación* de la motivación). Quizá la cuantificación podría ser una línea futura a explorar.

Si bien ésta es la propuesta actual de Arkin, anteriormente propuso un planteamiento de selección de acción basado en un coordinador central [Arkin y MacKenzie, 1994]. Éste coordinador hace las veces de árbitro y selecciona qué conjunto de esquemas están activos en cada momento. La propuesta concreta proponía un árbitro codificado como un autómata finito de estados. La salida final del sistema se calculaba como una combinación lineal de salidas de los esquemas activos.

Una evolución natural de este planteamiento original de Arkin son los sistemas híbridos, donde el nivel secuenciador o el deliberativo hacen las veces de árbitro entre los componentes del nivel reactivo. De ésta forma la selección de acción sólo se presenta en un único nivel. Dos ejemplos significativos que ya hemos comentado son la arquitectura Saphira [Konolige y Myers, 1998] y la arquitectura de Saffiotti [Saffiotti, 1997], que son muy similares. En cuanto a selección de acción, ambas trasladan el problema a la capa deliberativa que activa a unos comportamientos reactivos y no a otros. Dentro de la capa reactiva utilizan la fusión borrosa como mecanismo principal de coordinación entre comportamientos que puedan estar activos simultáneamente: por cada actuador combinan de modo borroso la salida de todas las reglas ponderándola por su prioridad y su contexto de activación. Ese contexto de activación determina la relevancia de esas reglas de control a la situación actual, bajando su peso cuando no son aplicables. Por ello le imprime al sistema la orientación a la situación presente del entorno.

Una diferencia fundamental radica en que la selección de acción en JDE es excluyente, en ella no

hay fusión de comandos, puesto que sólo un esquema gana la competición por el control en cada nivel. Además, en JDE la orientación al objetivo la marca el padre, que activa selectivamente a sus hijos, y en ese contexto la situación percibida determina el ganador de la competición de control. El análogo en JDE a los contextos de activación de Saffiotti y Konolige sería el salto de ALERTA a PREPARADO en cada esquema. En cada uno de ellos, cuando se suma la motivación procedente de estímulos externos se está comprobando indirectamente si la activación de ese esquema es conveniente en la situación concreta del entorno. Si no lo es, la motivación no superará el umbral.

La selección de acción en JDE contrasta con la activación por prioridades de Brooks, que materializa un arbitraje por prioridades fijas, en el cual los niveles superiores siempre son más prioritarios que los inferiores. El mecanismo de selección de acción está esculpido en la propias conexiones entre autómatas y es por lo tanto fijo.

Motivación interna

Una de las hipótesis subyacentes en los sistemas reactivos y basados en comportamientos es que no es necesario estado interno alguno para generar comportamiento. ¿Basta pues una teoría de reflejos para explicar el comportamiento (teoría de reflejos encadenados [Vogel y Angermann, 1974])? En etología la evidencia más significativa en contra de esta hipótesis es que ante exactamente la misma situación del entorno el animal puede responder con diferentes pautas motrices. Por ejemplo, si se somete repetidamente, en cortos intervalos, al mismo individuo a los mismos estímulos externos, la respuesta varía con el tiempo. Las pautas motoras provocadas inicialmente son más intensas que las siguientes, llegando incluso a no dispararse tras un número elevado de exposiciones a los estímulos. Es lo que se conoce como *fatiga específica de acción*. No es un cansancio físico, pues el animal puede moverse perfectamente para exhibir otro comportamiento. Es una fatiga específica para esa pauta motora. Las respuestas diferentes ante los mismos estímulos sólo se explican si admitimos que el animal puede tener cierto estado interno, pues la situación externa es literalmente la misma.

Por ejemplo, cuando se enfrenta un gato hambriento a una población numerosa de ratones todas las fases de captura se disparan secuencialmente. Cuando el gato empieza a satisfacer su apetito las últimas fases dejan de dispararse, limitándose a atrapar y matar a los ratones, pero no a devorarlos. A medida que empieza a saciarse simplemente los atrapa, para después soltarlos y finalmente, a pesar de que hay ratones, el gato sólo acecha expectante, pero no realiza ningún movimiento. Una explicación simplista a este último caso es que entonces el gato no tiene hambre. En este sentido hay trabajos modernos que incluyen cierto estado interno en el juego por la selección de acción (Cañamero [Cañamero, 1997; Cañamero, 2000; Cañamero *et al.*, 2002], Tyrrell [Tyrrell, 1992], Arkin [Arkin *et al.*, 2001]). Por ejemplo, Tyrrell en su entorno ecológico simulado [Tyrrell, 1993a] y Cañamero en [Cañamero *et al.*, 2002] llegan a explicitar unas variables como hambre, sueño, necesidad-de-apareamiento, etc. que condicionan la selección de acción. Éstas variables internas tienen además su propia dinámica, en la que influyen a su vez las acciones del robot. Por ejemplo, en términos de Tyrrell, la necesidad de dormir disminuye cuando el robot está durmiendo, el hambre disminuye cuando se está comiendo algo, aumenta a medida que pasa el tiempo etc. Del mismo modo, Tyrrell enuncia que el objetivo del robot es mantener estas variables motivacionales dentro de cierto rango aceptable de valores. Este fenómeno de saciar necesidades y disminución de la motivación para cierta actuación también está presente en Ludlow [Ludlow, 1976].

Otro indicio etológico en contra de la suficiencia de la teoría de reflejos encadenados lo constituye la observación de que ciertas secuencias diferentes de actuación aparecen siempre juntas, previsiblemente afectados por la misma "variable motivacional". La predisposición a ejecutar cualquiera de las fases de la secuencia aumenta y disminuye paralelamente a ejecutar cualquiera de las demás.

En JDE el estado actual de activación del árbol de esquemas ya supone un estado interno, una predisposición ante ciertas respuestas para los mismos estímulos a los que en otro contexto se puede reaccionar de diferente manera. La activación de un padre marca un contexto de actuación y la acumulación de motivación interna hacia sus hijos. Sin embargo, no se debe trasladar eternamente hacia arriba en la jerarquía la pregunta sobre el origen de la predisposición interna. El padre marca un contexto, bien, pero ¿cómo se explica que el padre esté ACTIVO? En algún momento la variable interna hambre condiciona que cierto esquema tome el control, en cierto nivel de la jerarquía. En JDE el paso

de ALERTA a PREPARADO se produce cuando la acumulación de varios estímulos, externos *o internos*, sobrepasa cierto umbral. En esos estímulos internos tienen cabida las variables motivacionales que hemos comentado. De modo similar al modelo psicohidráulico de Lorenz [Lorenz, 1978] la motivación fluye de arriba a abajo, buscando por cual esquema hijo discrurrir en su descenso.

Además en JDE la dinámica propia de esas variables internas se puede implementar con nuevos esquemas que se ejecutan periódicamente. Por ejemplo, un esquema que incrementa a cierto ritmo controlado la variable interna *hambre*.

Tal y como vimos en el capítulo 2, otra materialización del estado interno, muy alejada de la etología, son los estado motivacionales en los agentes BDI. En ellos el comportamiento racional es fruto de las creencias (*Beliefs*), deseos (*Desires*) e intenciones (*Intentions*) que tiene el agente [Rao y Georgeff, 1995]. Por ejemplo, las intenciones almacenan compromisos del agente, y determinan que ante las mismas creencias sobre el estado del mundo, las acciones sean diferentes.

Monitorización continua

Como vimos anteriormente, la monitorización se ha incorporado al funcionamiento normal de los esquemas en JDE. Cuando un esquema está en ACTIVO, ha activado a otros esquemas hijos, y está continuamente vigilando (monitorizando) su desarrollo para responder a cualquier eventualidad. El resultado tangible de esa monitorización continua puede ser dejar las cosas como están, cambiar ligeramente la modulación de los hijos, o frecuentemente, activar a otros hijos. Así, la monitorización que ejerce el padre está muy imbricada con la selección de acción de los niveles inferiores.

Otra monitorización continua que se presenta en JDE es la que hacen los esquemas que en están ALERTA, al chequear permanentemente si se dan sus precondiciones y si reúnen suficiente motivación como para activarse. Tanto esta monitoriazación como la mencionada en el párrafo anterior se reflejan en posibles cambios en la selección de acción. Gracias a ellas el sistema es capaz reconfigurarse rápidamente porque en cada iteración se están renovando decisiones de selección de acción.

En los comienzos de la Inteligencia Artificial no había monitorización de planes, reflejando una confianza excesiva en los modelos utilizados y una falta de acercamiento al mundo real, donde los sensores, los actuadores y los planes pueden fallar. Una vez digerida la necesidad de monitorización, ésta se plantea como una vigilancia de las desviaciones respecto a un curso específico de actuación. Por ejemplo, en los RAP de Firby [Firby, 1987], cada unidad RAP es responsable de monitorizar si satisface sus objetivos o no. Así, esta monitorización se realiza dentro de cada unidad, que simplemente señaliza su éxito o fallo a quien lo invocó. El planificador que lanza un RAP debe contemplar la posibilidad de que falle, pero no tiene que preocuparse por detectarlo, de eso ya se encarga el propio RAP.

Otra arquitectura significativa que incluye monitorización explícita es TCA de Simmons [Simmons, 1994]. Por ejemplo, tras la ejecución de una acción pregunta al sensor para asegurarse de que la actuación fue ejecutada correctamente y tuvo los resultados previstos. Es pues una monitorización puntual, mientras que en JDE es continua. En este sentido esta monitorización de TCA es menos robusta porque sólo se hace la comprobación en un cierto instante. Sin embargo es mucho más eficiente computacionalmente. Para poder monitorizar en paralelo a la planificación y ejecución del plan, TCA también permite programar monitores que comprueban periódicamente cierta condición o que avisan si ocurre cierto evento [Simmons, 1994].

3.5.6. Limitaciones

Además de las ventajas que hemos argumentado de la arquitectura propuesta también hemos identificado problemas o cuestiones arquitecturales que aún no quedan bien resueltas con el planteamiento de JDE.

Un inconveniente de JDE es que hay que anticipar todos los casos, y la respuesta adecuada en ellos. El sistema no tiene iniciativa, sirve para lo que sirve y no para más. En este sentido contrasta con los sistemas deliberativos, que son capaces de encontrar la acción correcta si están provistos del modelo del mundo y las consecuencias de sus acciones. Son capaces de responder bien a una situación novedosa a la que no se han enfrentado nunca. El planteamiento de JDE pierde el ideal deliberativo: teniendo un motor de inferencias y las reglas del mundo, nos podemos enfrentar igualmente a los datos-I, que

a los datos-II, el sistema genera las actuaciones pertinentes para ambas situaciones. Este racionalismo exacerbado contrasta con la falta de generalización que hay en JDE, donde cada caso o situación se resuelve por separado. Esta ausencia de generalismo lógico no significa que no haya reutilización entre un caso y otro: la respuesta a un problema depende del corpus de los comportamientos ya disponibles, como han identificado los etólogos [Lorenz, 1978] en las pautas motoras y su evolución filogenética.

Dado que JDE tiene una clara inspiración etológica, se puede argumentar que en la naturaleza la mayor parte de los animales no son creativos, pero funcionan perfectamente en su entorno y desarrollan un repertorio de comportamientos que sin duda se pueden caracterizar como inteligentes. En este sentido, JDE se puede entender como la arquitectura de uno de estos animales, en los que es la especie, y no sus individuos, la que puede adaptarse con generaciones sucesivas a nuevas condiciones o cambios estructurales en el entorno.

JDE se puede considerar análoga a las arquitecturas que han utilizado los etólogos para describir los comportamientos instintivos de los animales. En este sentido genera un repertorio de comportamientos fijo que el diseñador puede ir enriqueciendo, pero no contempla ningún mecanismo automático de aprendizaje. Una posibilidad que abre JDE, al manejar como unidad básica los esquemas, es el establecimiento de conexiones entre diferentes estímulos (esquemas perceptivos) y diferentes "respuestas" (esquemas motores). Con ello facilita el manejo de unidades a diferentes niveles de abstracción.

Dado que en JDE tenemos que anticipar todos los casos, podemos proceder como Simmons en TCA [Simmons, 1994], materializando primero el caso regular, lo normal, y después ir incorporando la solución a casos especiales que puedan ir surgiendo. Para ello se puede retocar algún esquema ya existente o incorporar uno nuevo para que explícitamente trate la nueva situación en el contexto en el que aparece. Como vimos anteriormente, la escalabilidad de JDE facilita la inserción de nuevos esquemas o comportamientos para tratar esos nuevos casos.

Otra de las limitaciones que apreciamos en JDE es que dificulta la fusión de comandos. El principal obstáculo para ella es el mecanismo de selección de acción excluyente, que hace que dentro de cada nivel de abstracción el sistema sea monoobjetivo en cada instante. Sólo las preferencias del esquema ganador se tomarán en cuenta para decidir la actuación en ese nivel. Tal y como vimos anteriormente, para la satisfacción de varios objetivos se propone la alternancia en el tiempo y para ello la reconfiguración de la propia arquitectura. En cuanto a la fusión en el mismo instante, se puede pensar en activaciones parciales de varios hermanos, o que el ganador recoja las preferencias de los perdedores y trate de satisfacerlas en la medida de lo posible. En cualquier caso este problema no está aún resuelto.

Aunque en JDE no hay mecanismos para la fusión interesquema, sí hay fusión intraesquema. Un mismo esquema puede combinar varias tendencias y fundirlas en una única actuación final. Por ejemplo, un mismo esquema navegador puede tener en cuenta la información sensorial de sus alrededores (se la dan sus esquemas perceptivos hijos) y la dirección objetivo (por ejemplo un parámetro del propio esqeuema que modula su padre) para combinar ambas tendencias en un mismo comando de velocidad.

Capítulo 4

Implementación

> *AI has been brain-dead since the 1970s. [...] The worst fad has been these stupid little robots. Graduate students are wasting 3 years of their lives soldering and repairing robots, instead of making them smart. It's really shocking.*
> Marvin Minsky, 2003

En el anterior capítulo hemos planteado la arquitectura y los mecanismos que contempla para generar comportamiento. La prueba de fuego de una arquitectura es sin duda su implementación en robots reales, de modo que se demuestre su capacidad de resolver los problemas para los que se ha diseñado. En este sentido diferimos de Marvin Minsky y de la opinión suya que aparece en la cita. Nos alineamos más del lado de Rodney Brooks, que es a quien realmente va dedicada la crítica de Minsky. Pensamos que una buena teoría en robótica es mejor si va complementada con experimentos reales, sobre robots físicos. Razonar en el mundo teórico, con argumentos especulativos y lógicos es necesario, pero sólo la criba de los experimentos reales mantiene sanas y vigorosas las teorías. Uno de los riesgos de la teoría es el de alejarse de la realidad del problema que se quiere solucionar, y quedarse en meras especulaciones vacuas en las que todo cuadra y no se perciben las dificultades reales que hay que resolver. En el otro extremo dialéctico, pensamos que construir el cuerpo de un robot es un problema de ingeniería, resuelto en cierta medida, y por lo tanto no relevante para una tesis como ésta. Su programación para que genere comportamiento sí es un problema genérico aún no resuelto.

Este capítulo lo dedicaremos a describir la implementación en programas de la jerarquía JDE propuesta. Fundamentalmente detallaremos las herramientas de programación que hemos desarrollado para materializar los mecanismos previstos por la teoría, y la arquitectura sofware que hemos elegido para concretar la arquitectura conceptual. También mencionaremos algunos elementos en los que nos hemos apoyado, como alguna librería ya existente y varios de los robots en los que esta arquitectura se ha utilizado.

En concreto hemos articulado el capítulo en tres partes. En una primera describimos los dos principales robots reales sobre los que hemos hecho experimentos, tanto su parte física (sensores, actuadores, procesadores, etc.) como el entorno de programación con que los vende el fabricante. En una segunda parte explicamos la arquitectura de servidores y clientes que hemos desarrollado para permitir procesamiento distribuido, y el protocolo creado para que servidores y clientes se comuniquen. Finalmente detallamos la implementación de los esquemas de JDE que hemos realizado en software concreto, haciendo hincapié en cómo hemos materializado la selección de acción y enseñando el esqueleto típico para añadir nuevos esquemas a un sistema.

4.1. Robots reales: hardware y software

La arquitectura JDE se ha utilizado, en todo o en parte, sobre varios robots físicos diferentes. En esta sección vamos repasar brevemente no sólo el hardware de esos robots, sino también la plataforma software sobre la que es posible programarlos. Haremos especial hincapié en el robot Pioneer, que es el que consideramos de referencia para los experimentos de esta tesis.

En cuanto al hardware, mencionaremos brevemente los sensores instalados y los actuadores con los

que cuenta cada robot. También los elementos de interacción y los recursos típicos de procesamiento disponibles en cada plataforma.

En la parte de programación, veremos la arquitectura software básica con la que los vende el fabricante, los sistemas operativos instalados y de qué herramientas software se dispone para acceder a los sensores y los actuadores desde los programas. Además, la plataforma software del robot típicamente proporciona métodos para materializar multiprogramación, porque los programas del robot suelen tener que hacer varias cosas en paralelo. Por ejemplo, leer los sensores, atender a comunicaciones, a la interacción con el usuario, comandar órdenes a los actuadores, etc.. Veremos estos métodos y los mecanismos de comunicación entre esos flujos de ejecución concurrentes. También hablaremos de en qué lenguajes se escriben los programas para los robots y cómo se organiza el cómputo necesario.

Sistema operativo y lenguaje

Como veremos a continuación, los ordenadores personales se han insertado como elementos principales de cómputo en los robots de investigación. Al menos tres factores han impulsado esa irrupción: un precio relativamente asequible, la estandarización de herramientas de programación, y sobre todo el crecimiento exponencial de la capacidad de cálculo. Los procesadores específicos son más escasos y se han dejado para tareas de bajo nivel, muy concretas. Por ejemplo, el robot Pioneer o el B21 incorporan varios microcontroladores porque en su hardware incluyen control PWM para los motores.

El sistema operativo del PC sobre el cual corren todas los programas desarrollados en esta tesis es `GNU/Linux` (en adelante, Linux). Tanto el ordenador personal de trabajo como los ordenadores a bordo de los robots, portátiles o no, corren Linux, lo que facilita su interconexión. Linux es un sistema operativo similar a `Unix`, de libre distribución, multitarea y multiusuario, de 32 bits muy robusto y ágil. En concreto la distribución elegida ha sido Debian, porque es *software libre.* completamente y por ello ofrece una calidad contrastable. El código es escrutable. Otra ventaja es que está accesible gratuitamente a través de internet y la actualización del sistema es muy sencilla. Además ofrece un amplio abanico de aplicaciones, como el compilador de C/C++ `gcc`, el editor `emacs`, etc. Linux ofrece también las librerías necesarias para desarrollar interfaces gráficas en el sistema más extendido en el mundo Unix, X-Window. En cuanto a la comunicación entre varios programas ofrece los *sockets*, una abstracción que permite olvidarse de los detalles de red, protocolos de acceso al medio, etc.

Una de las desventajas del uso de Linux en robótica es que no es un sistema de tiempo real, pues no permite acotar plazos. En Linux no hay *garantía* de plazos, porque su algoritmo de planificación de procesos no implementa esas garantías. En este sentido contrasta con sistemas operativos similares como QNX, o la variante RT-Linux. Sin embargo, Linux ha resultado suficientemente ágil para los requisitos necesarios en nuestras aplicaciones. Por ejemplo, permite detener hebras durante milisegundos, de modo aproximado. Con la arquitectura JDE se desarrollan comportamientos vivaces, pero no necesita márgenes temporales estrictos, no es tiempo real duro.

En cuanto al lenguaje utilizado para programar al robot la incorporación creciente del ordenador personal ha abierto el paso a lenguajes de alto nivel. Lejos quedan los tiempos de programación en ensamblador para hacer un código eficiente en extremo. Un lenguaje que tuvo gran presencia en los primeros trabajos, y que tiene gran ascendiente de la inteligencia artificial es Lisp. Hoy día el lenguaje más extendido para programar robots es el lenguaje C. Esta también ha sido nuestra elección para desarrollar todos los programas de esta tesis. En primer lugar, este lenguaje supone un buen compromiso entre potencia expresiva y rapidez. Como lenguaje compilado su eficiencia temporal es superior a otros lenguajes interpretados. En segundo lugar, literalmente todos los robots y plataformas con los que hemos trabajado tenían soporte para C. Gran parte del software proporcionado por los fabricantes de robots está codificado en este lenguaje. Por ejemplo, C es el lenguaje elegido por el fabricante ActivMedia para codificar sus librerías en Saphira, o por el fabricante RWI para el entorno de desarrollo de sus robots. La integración de nuestros programas con sus entornos era pues más sencilla en C que en otros lenguajes. Además una comunidad muy amplia de grupos de investigación realiza sus desarrollos en C. Sirvan como ejemplo el paquete TCA y TCX de Reid Simmons [Simmons *et al.*,

1997a], o la suite de navegación CARMEN[1] del grupo de Sebastian Thrun en CMU. La reutilización o integración de software es más factible en este lenguaje común.

Recientemente se puede observar un crecimiento de los desarrollos en C++. Por ejemplo, el entorno ARIA de Kurt Konolige para programar robots de ActivMedia, el entorno de programación del perrito Aibo de Sony. El principal avance sobre C es que proporciona la abstracción de orientación objetos, con los mecanismos de herencia y polimorfismo asociados. Una ventaja es que la portabilidad desde C a C++ es relativamente sencilla.

Interfaces gráficas

Otro aspecto que normalmente incluyen las aplicaciones sobre robots es su propia interfaz gráfica. A través de ella el robot puede mostrar las estructuras perceptivas generadas autónomamente y el humano puede interactuar: especificando objetivos, modificando el comportamiento del robot mediante botones, diales, etc. La interfaz visual facilita el proceso de depuración y prueba de los distintos algoritmos perceptivos o de control, mejorando con mucho la realimentación que ofrece una interfaz meramente textual. Esta visualización no es una función vital para el comportamiento autónomo del robot, pero acelera el proceso de programación reduciendo el tiempo de desarrollo.

En robótica móvil uno de los requisitos a tener en cuenta en el sistema gráfico es la *visualización remota* porque el robot está continuamente moviéndose y no suele disponer de una pantalla a bordo. La visualización ha de ser necesariamente en la pantalla de otro ordenador. Como ya hemos mencionado el sistema gráfico más extendido en el mundo Linux es el sistema de ventanas `X-Window`, desarrollado en el MIT y DEC ya hace unos años. Precisamente este sistema resuelve la visualización remota de modo elegante, permitiendo que las salidas gráficas de cierto proceso que corre en una máquina (cliente-X) y se visualicen en otra diferente (servidor-X).

La librería que permite máxima flexibilidad y control de todos los detalles de la interacción gráfica en este sistema es `XLib`, pero suele considerarse compleja. En este sentido han surgido paquetes que se construyen encima de `XLib` ofreciendo objetos gráficos predefinidos y un repertorio acotado de patrones de interacción y visualización. La librería `XForms`[2] es uno de estos paquetes, y el que hemos elegido para crear la interfaz gráfica de nuestros programas, frente a otras alternativas como Qt o GTK+. Sus principales ventajas son que es software libre[3] y está escrita en C, por ello en perfecta sintonía con el entorno informático elegido. Además ofrece una herramienta visual, llamada `fdesign`, para crear de modo sencillo la interfaz gráfica de las aplicaciones.

`XForms` ofrece un amplio repertorio de objetos gráficos con los que confeccionar el frontal gráfico según se desee, cada uno de ellos con un determinado comportamiento y funcionalidad. Por un lado, estos objetos son sensibles a la interacción con el usuario a través del ratón y del teclado. Por otro lado, son accesibles y manipulables desde el código del programa aplicación, ya que tienen asociadas funciones que permiten muestrear y fijar su estado, consultar y cambiar sus atributos. Así permiten al usuario comunicar cosas al programa y al programa mostrar cosas al humano, esto es, la interacción gráfica entre ambos.

La librería `XForms` permite al programa de la aplicación *muestrear* de manera no bloqueante el estado de la interfaz y mantener el control del flujo de ejecución. Es este programa el que decide cuándo muestrear y cómo responder al estado en que se encuentre la interfaz. En este modo de operación la interfaz gráfica se inserta en los programas del robot como dos tareas: por un lado, muestrear si el usuario ha pulsado algún botón o dial, y por otro, periódicamente refrescar la imagen que se le está mostrando. Cuanto más corto sea el periodo de muestreo de los botones antes se entera el programa de la interacción demandada por el usuario y el sistema será más reactivo. Si la tarea de refresco se llama muy frecuentemente la visualización será muy vivaz, pero puede consumir mucho tiempo y ralentizar al resto de las tareas, lo cual no es deseable. En cualquier caso hay que establecer un compromiso.

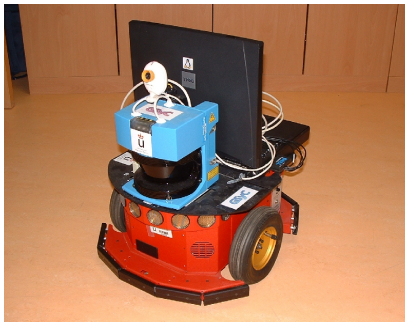
[1] http://www-2.cs.cmu.edu/∼carmen/

[2] http://bragg.phys.uwm.edu/xforms

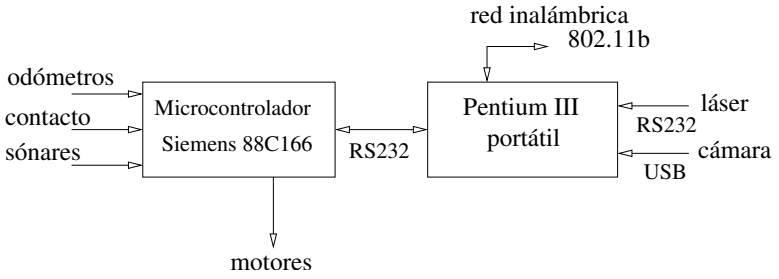
[3] Inicialmente se distribuía libremente el binario de la librería, pero no su fuente. Recientemente ha abierto su código fuente y se ha unido al software libre

4.1.1. El robot Pioneer

El robot que utilizamos como referencia para los experimentos con JDE que describiremos en los siguientes capítulos es el Pioneer 2DXE. Este robot lo comercializa la empresa norteamericana ActivMedia[4] y tiene una comunidad de usuarios bastante extensa, lo que facilita la resolución de dudas y problemas. El robot dispone de un equipo sensorial para medir el estado de su entorno, unos procesadores que le dan capacidad de cálculo y unos motores que le permiten moverse.



(a)



(b)

Figura 4.1: *Nuestro Pioneer (a) y su diagrama de bloques del hardware (b)*

En el apartado sensorial, el robot básico consta de una corona de sensores de contacto en su parte inferior, otra de sensores de ultrasonido y un par de odómetros asociados a sendas ruedas motrices. A este equipamiento le hemos añadido un sensor láser y una cámara en la configuración final, tal y como se muestra en la figura 4.1(a). Los actuadores principales de los que dispone esta plataforma son dos motores de continua, cada uno asociado a una rueda. Con ellos se dota al robot de un movimiento de tracción diferencial (tipo tanque). Con unas velocidades máximas de 1.8 m/s y de 360 $grados/s$. Es capaz de cargar 23 Kg de peso.

En el apartado de procesamiento, el robot dispone de un microprocesador Siemens 88C166 situado en la base motora, ejecutando instrucciones a 20 MHz. Sobre este microcontrolador funciona un sistema operativo especial llamado P2OS, que se encarga de recoger las medidas de los sónares, los odómetros y de materializar los comandos motores que le llegan. Todo ello a través de tarjetas especiales. Este micro se conecta a ordenadores externos a través de un puerto serie.

Tal y como ilustra la figura 4.1(b) la plataforma final se ha configurado con un ordenador portátil al que están conectados la base motora, el sensor láser y la cámara. En el ordenador portátil se tiene un procesador Pentium-III a 600 MHz. En él se ejecutarán la mayor parte de los programas para generar comportamiento. Una ventaja de este montaje es que se puede renovar fácilmente el ordenador a medida que se queda obsoleto. Al ser un elemento estándard, de gran mercado, se puede reemplazar por procesadores equivalentes más modernos y rápidos a un precio asequible.

Adicionalmente, el portátil está conectado a la red exterior a través de un enlace inalámbrico, con una tarjeta de red 802.11, que le proporciona un ancho de banda de 11 Mbps en sus comunicaciones hacia el exterior. De este modo el programa de control puede correr a bordo del robot o en cualquier otro computador. Además del funcionamiento autónomo, esta disposición permite la visualización e interacción en tiempo real con máquinas fijas.

Sónares y táctiles

Los sensores más sencillos que permiten detectar obstáculos son los de contacto. Nuestro robot tiene 5 sensores delante y 5 detrás, que le permiten discernir el punto de impacto. Éstos entregan lecturas binarias: choque, no-choque.

Situados unos 10 cm por encima de los *bumpers* hay una segunda corona, de sensores de ultrasonido, con 8 en la parte delantera y otros 8 en la parte trasera. Cada segundo se obtienen 3 lecturas completas de los 16. Los sensores de ultrasonido miden distancia a los obstáculos cercanos y por lo tanto ocupación

[4] http://www.activmedia.com

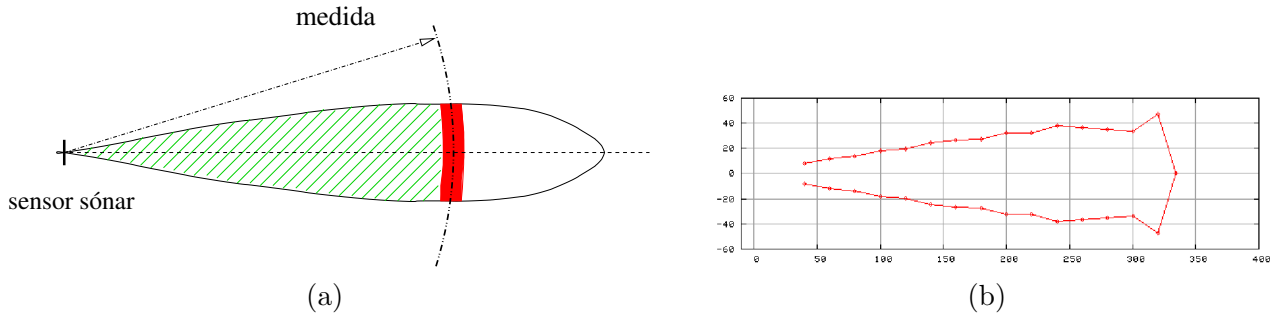


Figura 4.2: *Interpretación de la lectura sónar (a) y alcance experimental (b)*

del entorno. Emiten una onda ultrasónica que rebota en los obstáculos que pueda haber delante del sensor. Midiendo el tiempo que tarda en llegar el eco estiman la distancia hasta el obstáculo. Como veremos en el capítulo 5, estos sensores tienen ciertas incertidumbres asociadas. De momento baste decir que entregan valores de distancia. La energía ultrasónica se propaga con un frente de onda circular que se expande y conforma un patrón de energía de forma lobular como el de la figura 4.2(a). Tal y como ilustra esa figura, la intepretación geométrica más acertada de una lectura sónar es doble: en el interior del lóbulo hay espacio vacío, porque en caso contrario el eco hubiera retornado antes; en algún punto del arco frontal hay un obstáculo que ha provocado el retorno de cierta energía ultrasónica. En la figura 4.2(b) se muestra el modelo medido experimentalmente para los sensores sónar de nuestro Pioneer. Las distancias se expresan en milímetros y como se observa, tiene una forma cónica y un alcance máximo de 3 metros.

Odómetros

Otro sensor del robot Pioneer son los odómetros asociados a cada una de sus ruedas. Estos odómetros cuentan 500 pulsos por cada giro completo de la rueda, lo que se traduce en 66 $ticks/mm$. Su uso más importante es para estimar su posición, y con ellos se puede estimar además la velocidad real a la que se mueve el robot. Su principal problema es que acumulan error debido al deslizamiento de las ruedas y desalineamientos.

Estos sensores proporcionan lo que han girado las ruedas izquierda y derecha del robot en cierto intervalo. Desde esas lecturas, conocidas las coordenadas de su posición anterior $(x(t-1), y(t-1), \theta(t-1))$ es fácil calcular la nueva posición empleando un razonamiento incremental.

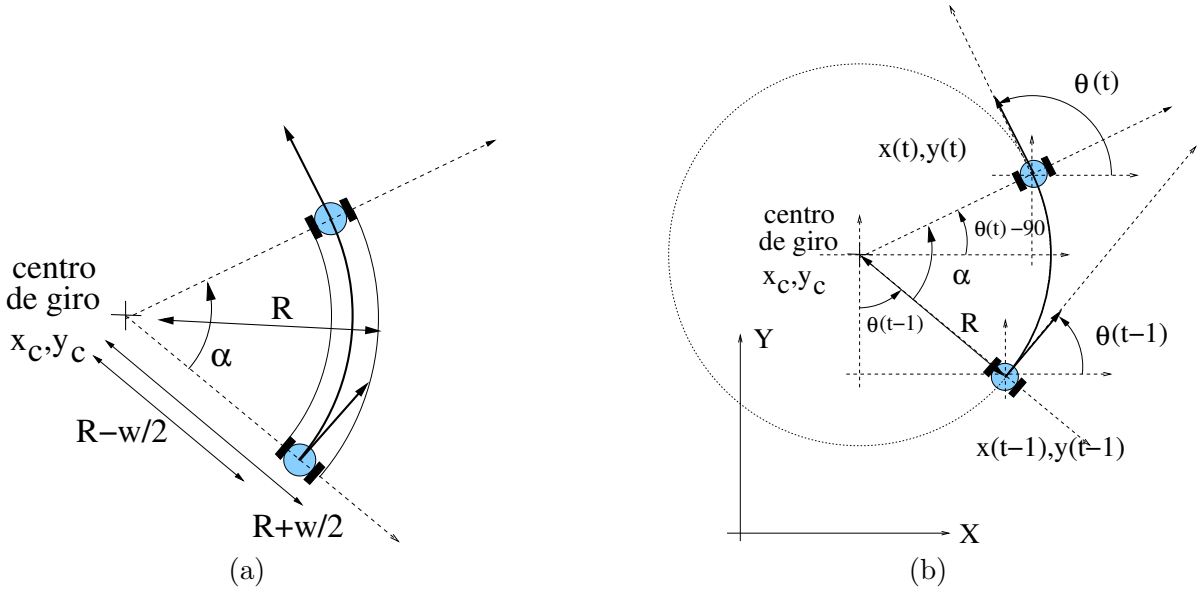


Figura 4.3: *Cálculo de la posición desde la lectura de los odómetros*

Se puede aproximar el movimiento en un diferencial de tiempo como un movimiento circular, de radio de giro R y ángulo α, tal y como indica la figura 4.3(a). Si en ese breve intervalo las distancias recorridas por la rueda izquierda y derecha han sido ΔS_i y ΔS_d respectivamente, entonces el ángulo recorrido respecto cierto centro de giro viene dado por las ecuaciones 4.3 y 4.4. El ángulo tiene un signo positivo si gira en sentido antihorario, es decir si $\Delta S_d > \Delta S_i$, y tiene un signo negativo en caso contrario.

$$\Delta S_i = \alpha(R - \frac{w}{2}) \tag{4.1}$$

$$\Delta S_d = \alpha(R + \frac{w}{2}) \tag{4.2}$$

$$\alpha = \frac{\Delta S_d - \Delta S_i}{w} \tag{4.3}$$

$$R = \frac{\Delta S_d + \Delta S_i}{2\alpha} \tag{4.4}$$

Si $\Delta S_d = \Delta S_i$ las dos ruedas han girado exactamente lo mismo y por lo tanto el robot ha avanzado en línea recta. En ese caso la nueva posición del robot en el instante t viene dada por las ecuaciones 4.5, 4.6 y 4.7.

$$\theta(t) = \theta(t-1) \tag{4.5}$$

$$x(t) = x(t-1) + R\cos\theta(t-1) \tag{4.6}$$

$$y(t) = y(t-1) + R\sin\theta(t-1) \tag{4.7}$$

Si los dos incrementos de odómetros no son idénticos significa que el robot se ha movido. Supondremos que lo ha hecho de un modo circular[5], girando un ángulo α alrededor de cierto *centro de giro* x_c, y_c con el radio de giro R. Siguiendo el desarrollo que ilustra la figura 4.3(b), se llega a que la nueva posición viene descrita por las ecuaciones 4.12, 4.13 y 4.14.

$$y_c = y(t-1) + R\cos(\theta(t-1)) \tag{4.8}$$

$$x_c = x(t-1) - R\sin(\theta(t-1)) \tag{4.9}$$

$$y_c = y(t) + R\cos(\theta(t)) \tag{4.10}$$

$$x_c = x(t) - R\sin(\theta(t)) \tag{4.11}$$

$$\theta(t) = \theta(t-1) + \alpha \tag{4.12}$$

$$x(t) = x(t-1) + R\cos\theta(t-1) - R\cos(\theta(t-1) + \alpha) \tag{4.13}$$

$$y(t) = y(t-1) - R\sin\theta(t-1) + R\sin(\theta(t-1) + \alpha) \tag{4.14}$$

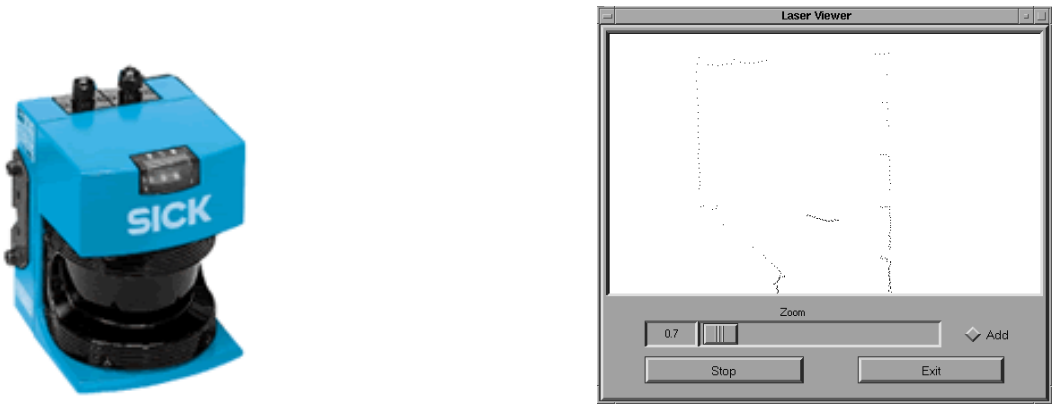
Láser

Posteriormente hemos añadido al Pioneer otro sensor de distancia a obstáculos: un sensor láser, de la marca Sick. En la configuración que hemos empleado, el modelo LMS-200 proporciona 10 barridos por segundo, con una precisión[6] de 1 grado y $2cm$. Tal y como muestra la figura 4.4(b), ofrece perfiles del entorno mucho más nítidos y fiables que los sónares. Su principal inconveniente es el precio, que lo hace prohibitivo para robots comerciales. Se alimenta con una tensión continua de 24 Voltios, que se extraen con una placa elevadora de tensión desde los 12 Voltios internos del robot. El sensor entrega sus lecturas a través de un puerto serie convencional.

Se ha reutilizado un *driver* de CARMEN, que configura adecuadamente al dispositivo láser en cuanto a su resolución angular, modo de operación, velocidad de emisión y utiliza el puerto serie a 38400. Este *driver* entiende el formato de mensajes en el puerto serie que establece el fabricante

[5] si el movimiento ha sido es diferencial, por ejemplo el de un intervalo temporal pequeño, la hipótesis circular es una buena aproximación

[6] se puede configurar para resoluciones de medio grado y otras frecuencias de barrido



(a) (b)

Figura 4.4: *Sensor láser (a) y ejemplo de lectura (b)*

del sensor, y refresca las variables en las que vuelca las medidas del láser. Además ofrece una interfaz de funciones en C para iniciar al dispositivo (`sick_start_laser`), lanzar o detener la captura (`sick_stop_continuous_mode`,`sick_start_continuous_mode`) y muestrear de modo no bloqueante si hay nuevos mensajes del sensor en el puerto serie.

Plataforma software: Saphira

Una vez descrita la plataforma física en la que hemos realizado los experimentos con nuestra arquitectura, detallaremos ahora la plataforma software que ofrece el fabricante para facilitar la creación de programas para el robot. La arquitectura Saphira nació en el grupo de investigación de Kurt Konolige y Alessandro Saffiotti, en el SRI International[7]. Posteriormente, ActivMedia la utilizó como plataforma software en sus robots: Pioneer, Amigo, etc. Saphira se ha extendido a muchos robots y funciona tanto en ordenadores con Linux y como en ordenadores con MS-Windows [ActivMedia, 1999].

En Saphira el estado sensorial del robot se materializa en dos estructuras de datos: `sfRobot` y `sbucket`. La rutina de inicialización de Saphira `sfInitBasicProcs` activa unas hebras básicas que refrescan continuamente esas estructuras. `sfRobot` es un registro con múltiples campos entre los que resaltamos `battery`, que indica el nivel de batería, `bumpers` que contiene el estado de los 10 sensores de contacto, las variables `ax,ay,ath` que almacenan la posición y orientación (x, y, θ) del robot respecto de cierto marco de referencia tal y como se deduce de sus odómetros. El array `sbucket` incluye las últimas 16 medidas de los sónares, cada una de ellas caracterizada por su rango `range`, la posición del sensor y un flag `snew` para señalar si es nueva o no.

En cuanto a las posibilidades de actuación, Saphira ofrece dos primitivas para controlar en velocidad los motores de las dos ruedas: `sfSetVelocity` y `sfSetRVelocity`. Además, si nuestro programa quiere controlar al robot en posición, Saphira ofrece otras sendas primitivas para la tracción y el giro: `sfSetHeading` y `sfSetPosition`.

Adicionalmente ofrece otras funciones para manejar una interfaz gráfica, construir mapas del entorno con forma de rejilla, detectar segmentos de paredes o puertas (`sfInitInterpretationProcs`), localizarse en el entorno (`sfInitRegistrationProcs`) y crear lo que Konolige llama *artefactos* en su arquitectura teórica [Konolige y Myers, 1998].

En cuanto al paralelismo Saphira proporciona dos mecanismos: microtareas (`microtasks`) y hebras (`threads`). En términos de sistemas operativos las microtareas son en realidad hebras de usuario, que ejecutan su código en iteraciones. Son por lo tanto cooperativas: si una se bloquea, se bloquean todas. Cada microtarea se crea con la función `sfInitProcess(*fn,name)`, donde `fn` es la función que se ejecutará periódicamente, en iteraciones cada 100 milisegundos, y `name` es el nombre de la microtarea. Cada microtarea tiene un estado asociado como suspendido (`sfSUSPEND`), interrumpido (`sfINTERRUPT`), éxito (`sfSUCCESS`), fracaso (`sfFAILURE`) o estados definidos por el

[7] antiguamente el Stanford Research Institute

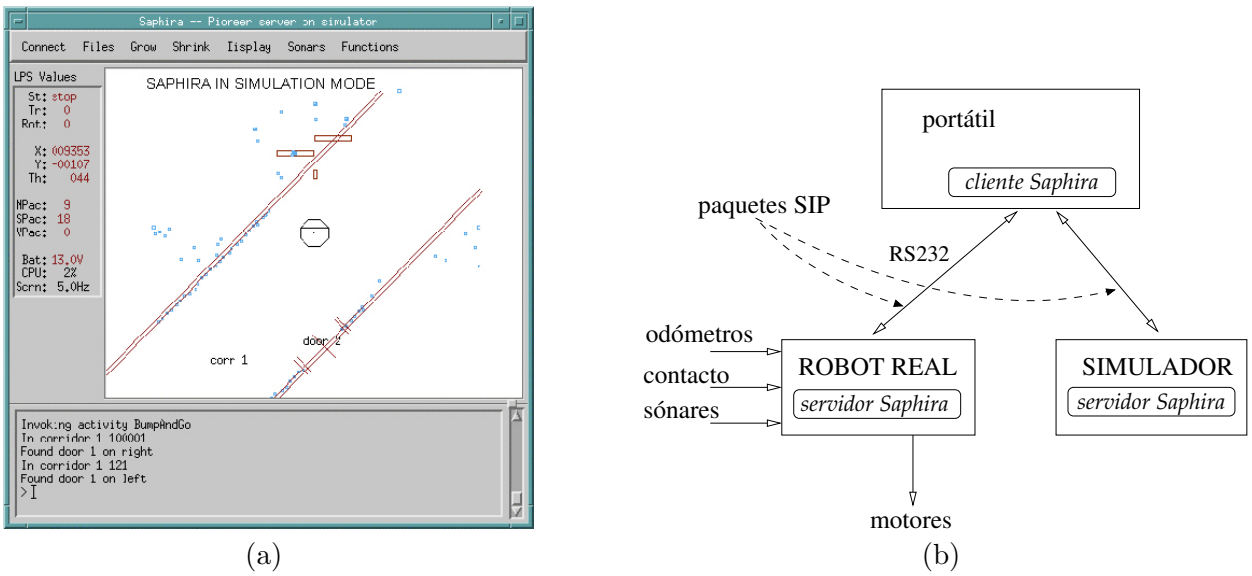


Figura 4.5: *Interfaz gráfico (a) y arquitectura software en Saphira (b)*

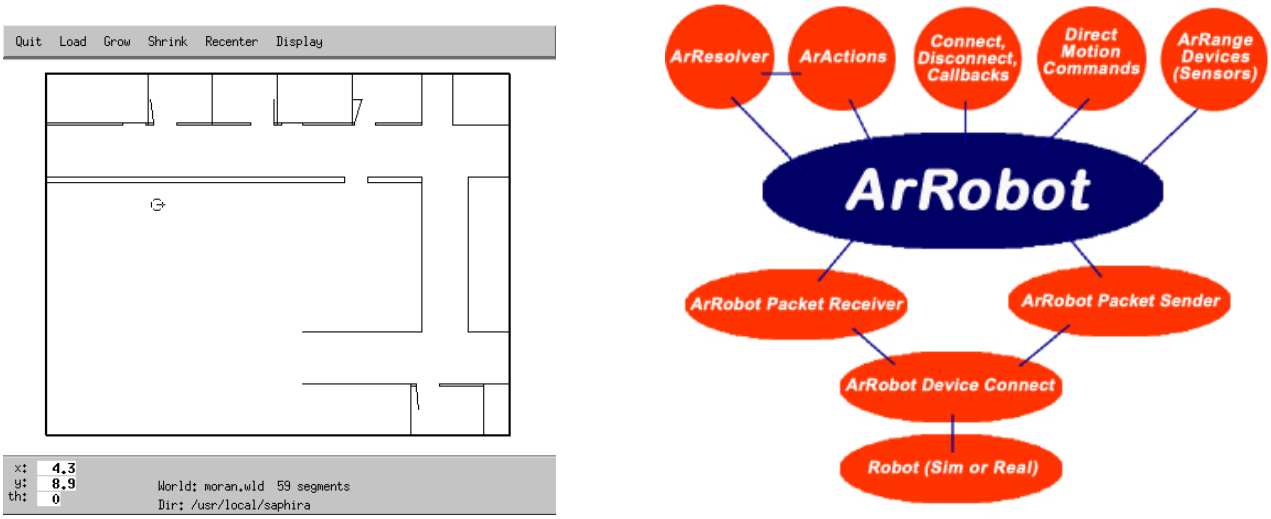
usuario. Típicamente lo que hace en cada iteración depende del estado en que se encuentre y una microtarea puede detener o activar a otras alterando su estado. Las microtareas programadas se arrancan con la función `sfStartup(0)`. Como todas las microtareas forman parte del mismo proceso no hay problemas de paralelismo, nunca dos de ellas accederán simultaneamente a la misma variable. Por ello no hay protecciones especiales para evitar condiciones de carrera.

Las hebras que ofrece Saphira son realmente hebras de kernel, que participan en la planificación de CPU que tiene el sistema operativo, en nuestro caso Linux. Su principal robustez es que son independientes y si una se bloquea las demás siguen su ejecución normalmente. Las hebras se crean con la función `sfstartThread` y se detienen con `sfDeleteThread`. Como las hebras son asíncronas pueden presentarse problemas de concurrencia. Saphira ofrece una especie de semáforos para evitar condiciones de carrera: `sfSuspendMT` y `sfResumeMT`. En realidad, para la plataforma Linux son un recubrimiento de la biblioteca `pthreads`. En cuanto a su inicialización las hebras se arrancan con la función `sfStartup(1)`. Típicamente en ese escenario puede haber varias hebras funcionando, una de las cuales contiene todas la microtareas básicas que refrescan las variables de estado del robot.

Como muestra la figura 4.5(b), esta librería se ejecuta en el ordenador que está conectado a la base motora del robot Pioneer. Tiene un diseño cliente-servidor, el servidor es el propio sistema operativo P20S que corre en la base motora del robot y se conecta a través de un cable serie. Cada 100 ms el sistema operativo del microcontrolador emite por el puerto serie paquetes de información del servidor (*Server Information Packet*), que contienen la información sensorial. La librería materializa un cliente Saphira en el ordenador, el cual incluye varias tareas básicas que actualizan las variables sensoriales procesando los mensajes provenientes del puerto serie, e incluye también funciones que materializan cierto comando motriz en un mensaje por el puerto. El cliente Saphira se ofrece en dos versiones, con interfaz gráfico y sin interfaz gráfico. El interfaz tiene el aspecto mostrado en la figura 4.5(a).

Una opción más que ofrece Saphira es el uso de un simulador bidimensional, SRIsim, cuya interfaz se aprecia en la figura 4.6(a). Como ilustra la figura 4.5(a), el cliente Saphira puede conectarse indistintamente a un servidor corriendo en el robot físico, o a un servidor corriendo en un simulador.

Recientemente se ha detenido el progreso de Saphira y se ha reservado ese nombre para el software de alto nivel, como la construcción de mapas, algoritmos de navegación, de localización, etc. Como sustitución del software básico ActivMedia ofrece la plataforma ARIA [ActivMedia, 2002], que es una reescritura en C++ de la anterior Saphira, pero con licencia GPL, es decir software libre. Las principales mejoras que incluye ARIA sobre Saphira anterior es que las microtareas tienen intervalos de 50 ms, se incluye más y mejor documentación, y se ofrece un recubrimiento de los sockets (ArSockets) para conexiones de red. El diagrama de clases genérico de ARIA se muestra en la figura 4.6(b).



(a) (b)

Figura 4.6: *Simulador SRIsim (a) y arquitectura software en ARIA (b)*

Cámara y Video4linux

Otro sensor que hemos añadido a la plataforma básica es una pequeña cámara de color. El modelo elegido fue una cámara de videoconferencia Philips, la PCVC-740K. Es una cámara CCD, digital, por lo que no requiere de tarjetas digitalizadoras. Se conecta a través de puerto USB y tiene soporte Linux[8]. Entrega imágenes de 640x480 pixels a un ritmo de 15 fps, o imágenes de 320x240 a 30 fps (utilizando un *driver* con compresión en ambos casos). Apenas tiene distorsión radial y su precio es muy reducido. Básicamente, la cámara proporciona información de color y de bordes, y si se calibra correctamente puede extraerse de las imágenes información espacial, geométrica.

`Video4linux` es una interfaz para captura de video en máquinas con Linux que se ha estandarizado recientemente[9]. Soporta gran variedad de tarjetas capturadoras e incluso sintonizadores de TV. Históricamente, para capturar imágenes en un robot se utilizaban cámaras analógicas y se necesitaba de tarjetas digitalizadoras. Para poner las imágenes a disposición de los programas se necesitaba el *driver* de la tarjeta, que solía parchearse con el código fuente del kernel de Linux y tenía su interfaz genuino. Con `video4linux` el soporte al dispositivo concreto se mantiene como un módulo de bajo nivel en el kernel, mientras que las aplicaciones interactúan con el dispositivo a través del módulo de alto nivel: `videodev`. Este interfaz es el mismo para todos los dispositivos, de manera que las aplicaciones pueden funcionar con diferentes dispositivos físicos sin necesidad de reescribirlas. Por ejemplo, el módulo `pwc` es el que soporta la cámara USB elegida, y el módulo `bttv` soporta las tarjetas digitalizadoras basadas en el popular chip BT878 y similares.

En `video4linux` el procedimiento típico de lectura de las imágenes en tiempo real es siempre el mismo. En primer lugar se abre el fichero `\dev\video`. En segundo lugar se configura al sistema de vídeo a través de varias llamadas al sistema `ioctl`. Con ella se puede preguntar al *driver* qué dispositivo concreto de video está presente, seleccionar el canal de entrada si procede, configurar sus opciones de captura como formato, tamaño de imagen, etc. El tercer paso consiste en preparar una zona de memoria compartida donde el *driver* va a volcar las imágenes capturadas y de donde el programa de usuario podrá leerlas. Esto se realiza con la llamada al sistema `mmap`.

Finalmente suele haber un bucle infinito de captura y análisis de imágenes. El *driver* responde a las interrupciones del dispositivo de vídeo que indican nueva imagen. La memoria común puede almacenar varios marcos para imágenes. El *driver* va rellenando esa memoria siguiendo una política FIFO entre marcos, y se detiene cuando llega a un marco que aún no ha sido leído por el usuario. Para coordinar este *driver* con el proceso de usuario se utilizan dos llamadas al sistema `ioctl`: una

[8] `http://www.smcc.demon.nl/webcam/`
[9] `http://bytesex.org/v4l/`

activarlo `ioctl(fdv4l,VIDIOCMCAPTURE,&gb)` y otra para liberar un hueco en la memoria común `ioctl(fdv4l,VIDIOCSYNC,&gb.frame)`. En las tarjetas capturadoras como las basadas en el chip BT848 este proceso de captura es muy eficiente porque la propia tarjeta transfiere la imagen a través de DMA a la zona de memoria señalada, con lo cual no se consume CPU. Con las cámaras USB la captura no es tan eficiente porque el procesador principal se encarga de hacer la transferencia.

4.1.2. El robot B21

Otro de los robots sobre los que se ha trabajado con JDE (en las versiones iniciales) es el B21, fabricado por la empresa Real World Interface[10]. Concretamente con el B21 del Instituto de Automática Industrial[11]. Este robot consta de dos módulos separables, la base y el cuerpo, en los cuales están insertos los sensores y los actuadores. Tanto la parte exterior de la base como del cuerpo se conforman con paneles de fácil apertura, 8 y 6 respectivamente. Estos paneles dan un sencillo acceso a las baterías y permiten manipular el hardware de los dos ordenadores internos que lleva. Encima del cuerpo hay montada una unidad cuello (*pantilt*) y sobre ésta una cámara convencional, tal y como detalla la figura 4.7(a). En la base están las baterías, que proporcionan energía al conjunto cuando se desenchufa de la red eléctrica.

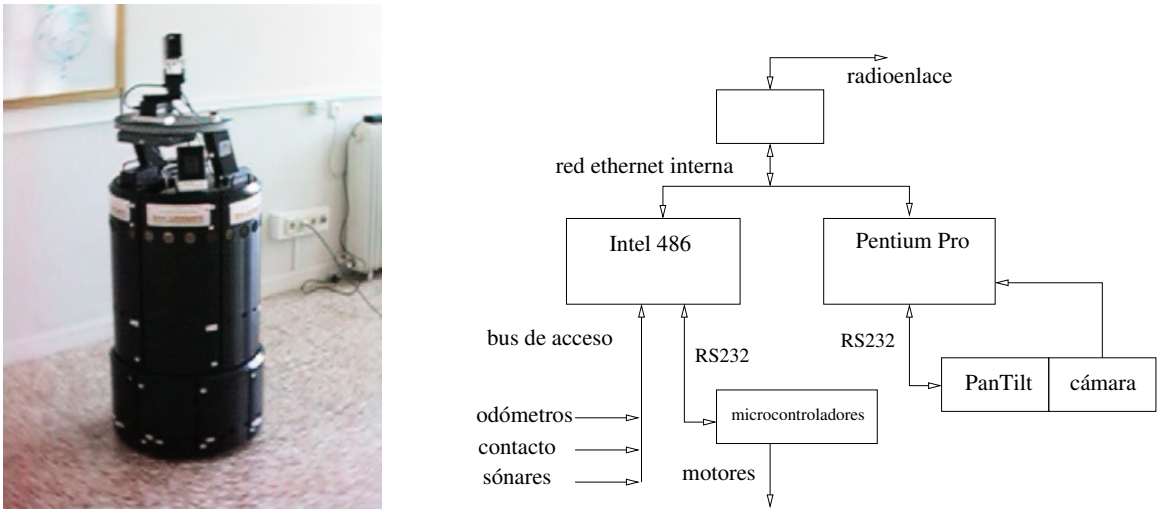


Figura 4.7: *Configuración del robot B21 (a) y diagrama de bloques de su hardware (b)*

En cuanto a los sensores, el robot cuenta con unos odómetros, sensores táctiles, tres cinturones de infrarrojos, un cinturón de ultrasonidos y una cámara. Hay odómetros que cuentan las vueltas de las ruedas motrices del motor y también los hay que miden la posición de la unidad cuello. Los táctiles son dos láminas metálicas que al ser presionadas entran en contacto. Cada panel tiene 4 sensores táctiles, uno en cada esquina, de manera que en total hay cuatro cinturones de táctiles, dos en los paneles de la base y dos en los paneles del cuerpo. Los sensores de ultrasonidos se distribuyen en una corona de 24 a media altura. Un sensor del voltaje permite medir si queda energía suficiente en la batería para funcionamiento autónomo o no. Los sensores sónares, los táctiles e infrarrojos se conectan por grupos a varios microcontroladores que recogen y procesan las medidas. Estos microcontroladores se conectan al ordenador a través de un bus especial, llamado `AccessBus`, como ilustra la figura 4.7(b). Una tarjeta capturadora especial recoge esas lecturas y pone esas medidas al alcance de los programas del usuario, gracias a un *driver* específico.

El único sensor novedoso respecto al Pioneer son los infrarrojos, que se distribuyen en tres coronas, una en el cuerpo y dos en la base. Una de ellas está apuntando hacia el suelo con idea de detectar los vanos de las escaleras. Los infrarrojos miden distancia a obstáculo cercano, pero tienen mucha más

[10] `www.rwii.com`. Recientemente esta empresa fue adquirida por iRobot: `www.irobot.com`
[11] `www.iai.csic.es`

resolución angular que los sónares. Emiten luz infrarroja y miden la cantidad de energía que rebota. Si reciben mucha es que el obstáculo está muy cerca. El inconveniente principal es que sus medidas dependen mucho del color del obstáculo y son muy sensibles a la luminosidad del ambiente.

También dispone de cuatro motores eléctricos de continua que proporcionan movimiento de tracción y de giro. Un puerto serie conecta la base con un ordenador interno, y a través de éste la base admite comandos de movimiento y envía las medidas de los odómetros. Los comandos de movimiento los procesan varios microcontroladores en la base que implementan rápidos bucles de control PID. Este control admite como parámetros la aceleración máxima, velocidad o posición deseadas, de modo que si un programa desea mover el robot a mayor velocidad sólo tiene que alterar este parámetro del bucle de control que corre continuamente en la base motora. Otro actuador más es la propia unidad *pantilt*, que se puede mover a voluntad apuntando la cámara en la dirección que más convenga. El modelo concreto es el PTU-46-17.5, de la casa Directed Perception[12].

A través del puerto serie se pueden enviar comandos de movimiento (control en posición) a la unidad, y preguntar la posición actual. Adicionalmente el robot dispone de una tarjeta reproductora de fonemas a través de la cual puede hablar. Esta tarjeta específica admite texto escrito y lo convierte en sonido, siguiendo las reglas de la fonética inglesa. Este actuador está orientado principalmente a la interacción con humanos.

El procesamiento a bordo del B21 se distribuye en dos ordenadores personales que hay montados en su interior. El PC derecho está destinado a ejecutar las aplicaciones de visión, consta de un Pentium-Pro a 180 Mhz y de una tarjeta digitalizadora de imágenes Matrox Meteor. El PC izquierdo consta de un procesador Intel 486 a 66 Mhz y tiene una tarjeta especial para recoger los datos del bus sensorial `AccesBus`. Ambos PCs llevan sendas tarjetas de red, de manera que forman una subred TCP-IP interior al robot. Esta subred se une a la red exterior a través de un puente radioenlace, de 2 Mbps de capacidad. El reparto entre recursos hardware y procesadores se ha hecho considerando que las aplicaciones de visión suelen demandar mucha capacidad de cálculo. De este modo hemos enganchado la unidad cuello y sobretodo la cámara al ordenador más potente, a través de un puerto serie y de la tarjeta digitalizadora, respectivamente. El ordenador izquierdo por su parte controla los movimientos de la base a través de un puerto serie y accede a los datos de sónares y táctiles, que son comparativamente más lentos y menos voluminosos que las imágenes.

Plataforma software: RAI y RAI-Scheduler

El fabricante del B21 proporciona varias librerías que permiten a los programas del usuario acceder a las medidas sensoriales de los sónares, los infrarrojos y los táctiles, así como comandar movimiento a los motores del robot. Todas ellas se agrupan bajo el nombre de `RAI` (*Robot Application Interface*). Una versión posterior de este mismo software es la plataforma `Beesoft`, y recientemente `Mobility`, que dan soporte a los robots de la empresa iRobot, heredera de RWI. Nos centraremos en describir la versión inicial, que corresponde a la utilizada en nuestros programas.

En cuanto al paralelismo, `RAI` incluye un planificador de tareas desarrollado por la universidad de Brown: `Rai-Scheduler`. Este planificador de tareas, escrito en C, permite al usuario programar en paralelo y pensar en las distintas tareas como distintas hebras que corren simultaneamente dentro del procesador. Su funcionalidad se ofrece en la librería `libRai.a`, con la cabecera `Rai.h`. Ofrece tareas de tres tipos: periódicas, de respuesta y de temporizador. Las *tareas periódicas* toman control periódicamente y se ejecutan de forma incremental, como iteraciones que son llamadas cíclicamente. Las *tareas respuesta* (*callback*) se ejecutan como respuesta a un evento cuando éste sucede. El evento es la escritura en cierto descriptor de fichero (fichero de disco, *socket*, *pipe*, etc.), lo cual indica que hay nuevas lecturas sensoriales, un nuevo dato en el canal de comunicaciones, etc.. Finalmente las *tareas temporizador* se ejecutan cuando cierto temporizador alcanza su cuenta final.

Normalmente el flujo de control de un programa sobre el robot móvil estará organizado con el planificador `Rai-Scheduler`. Esta librería obliga a pensar en la aplicación del usuario como un conjunto de tareas ejecutándose en paralelo. De este modo, las distintas tareas que debe realizar se materializan como tareas del planificador, dentro de un único proceso, que bien se disparan periódicamente o como

[12] `http://www.DPerception.com`

respuesta a algún evento. En realidad no es paralelismo estricto sino pseudo-paralelismo porque hay un único procesador, pero a efectos prácticos es equivalente. En términos de programación, son hebras de usuario. Para el sistema operativo aparecen formando parte de un único proceso que alterna su tiempo de procesador entre las distintas tareas. En este sentido, no hay problemas de concurrencia en accesos "simultáneos", porque en realidad hay pseudoparalelismo y no son estrictamente simultáneos.

```
typedef struct sonarType {
   int value; /* la distancia */
   int mostRecent; /* si este sensor es la más reciente */
   strict timeval time; /* tiempo de captura */
} sonarType;

sonarType sonarsNUM_SONARS];
irType irs[NUM_IR_ROWS][MAX_IRS_PER_ROW];
tactileType tactiles[NUM_TACTILE_ROWS][MAX_TACTILES_PER_ROW];
```

Tabla 4.1: *API de acceso a sónares del B21 con RAI*

Las variables que reflejan las medidas sensoriales de sónares, infrarrojos y táctiles se actualizan gracias a unas tareas respuesta asociadas al evento de que haya nuevas lecturas. Como muestra la tabla 4.1, los datos de ultrasonidos se almacenan en el *array* `sonars`. El software `RAI` incluye una tarea que sistemáticamente actualiza esa variable cada vez que hay nuevas lecturas. Después de inicializar con `sonarInit(void)` el programa del usuario puede activar y detener esa tarea con las funciones `sonarStart(void)` y `sonarStop` respectivamente. Adicionalmente, el programa puede asociar una rutina a modo de *callback* para hacer un procesamiento específico, es decir, una función que también será invocada cada vez que hay una nueva lectura sensorial (`registerSonarCallback(sonarCallbackType)`). Del mismo modo, los infrarrojos se almacenan en una matriz `irs`, y existen funciones similares para los infrarrojos como `irInit`, `irStart`, `irStop`, `registerIrCallback`. Los datos de los sensores táctiles se almacenan en una matriz `tactiles`, y existen las funciones `tactileInit`, `tactileStart`, `tactileStop` y `registerTactileCallback`.

```
typedef struct {
    unsigned long Request;
    unsigned long Clock;
    unsigned long GeneralStatus;
    unsigned long Xpos;
    unsigned long Ypos;
    unsigned long Heading;
    unsigned long BaseRelativeHeading;
    unsigned long TranslateError;
    unsigned long TranslateVelocity;
    unsigned long TranslateStatus;
    unsigned long RotateError;
    unsigned long RotateVelocity;
    unsigned long RotateStatus;
} statusReportType;

statusReportType activeStatusReport;
```

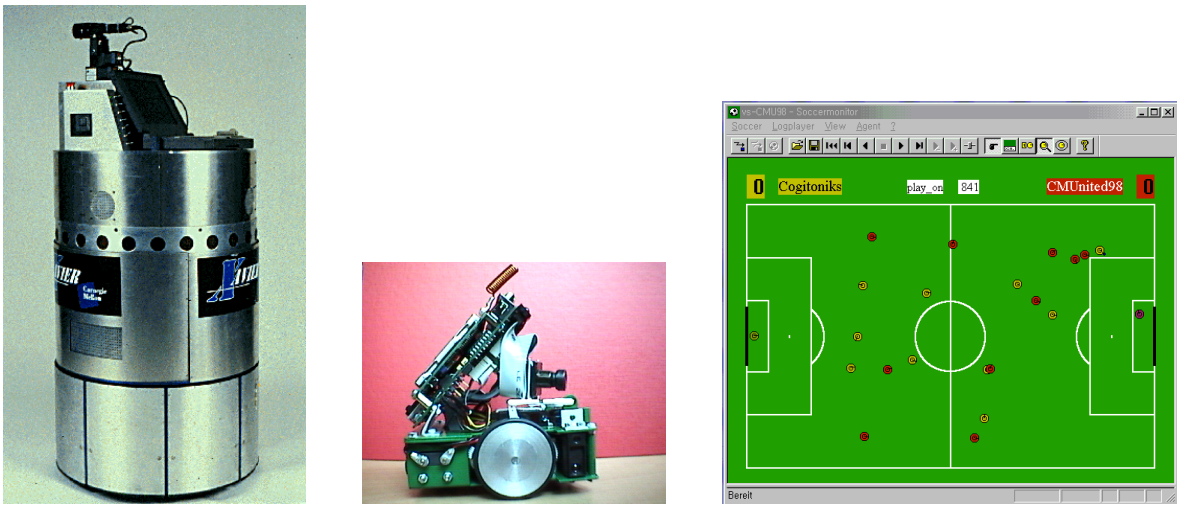
Tabla 4.2: *API de acceso a odómetros del B21 con RAI*

Como ilustra la tabla 4.2, la posición, la orientación y la velocidad del robot medidas desde los odómetros de la base motora, así como el nivel de baterías forman parte de la variable `activeStatusReport`. Esta variable se actualiza con los mensajes que periódicamente emite la base motora a través del puerto serie para refrescar sus valores. la función `statusReportPeriod` permite fijar esa periodicidad y `registerStatusCallback` se utiliza para enganchar una rutina que se ejecutará cuando se reciba un nuevo mensaje. La posición inicial de los odómetros se puede cargar

con la función `loadPosition` y la orientación inicial con `loadHeading`.

En el lado de actuación, `RAI` ofrece una colección de funciones para gobernar la tracción y el giro del robot, y permite controlar los movimientos en posición o en velocidad. Por ejemplo, para el control en velocidad `translateVelocityPos()` activa la tracción hacia delante, parametrizada con la velocidad y aceleración de referencia que se fijan con `setTranslateVelocity` y `setTranslateAcceleration`. Si se quiere activar la tracción hacia atrás se emplea `translateVelocityNeg()`. Funciones homólogas existen para controlar en velocidad el giro en ambos sentidos. En cuanto al control en posición, `RAI` ofrece `rotateRelativePos(units)` para que el robot gire en sentido antihorario cierto ángulo `units` y `rotateRelativeNeg(units)` para el sentido horario. Funciones homólogas existen para el avance o retroceso de cierta distancia. Adicionalmente `baseHalt` detiene el movimiento de todos los motores, mientras que `baseLimp` deja los motores desembragados.

Con el software `RAI` el programa del robot se estructura como un proceso que ejecuta varias tareas, las automáticas que refrescan las variables sensoriales y las genuinas del programa. Por ejemplo, una tarea periódica típica es la que toma decisiones de control en función de lo percibido en las variables sensoriales. Esta tarea puede implementar el bucle infinito de percepción, procesamiento y acción de control, ejecutando una vuelta del ciclo en cada iteración. Si se quiere elaborar representación del entorno se pueden utilizar los *callbacks* de las tareas sensoriales o bien utilizar otra tarea específica que muestrea (*polling*) las variables. Si el programa ofrece una interfaz gráfica se pueden incluir dos tareas más: una para refrescar la interfaz gráfica y otra para atender a cualquier evento del usuario humano sobre la interfaz. Si el programa del robot establece comunicaciones con otros procesos es común incluir otra tarea que se encarga del chequeo de las comunicaciones.



(a) (b) (c)

Figura 4.8: *Otras plataformas que se han programado con JDE*

Una característica de `RAI` es que proporciona un acceso local, es decir, el programa resultante ha de ejecutarse en la misma máquina que el sensor o que el actuador correspondiente. Por ejemplo, un programa puede leer periódicamente los datos de ultrasonido, pero si utiliza las librerías del fabricante ha de ejecutarse en el PC izquierdo, que es el que tiene la tarjeta hardware de acceso a los sensores. Además de este acceso local, `RAI` ofrece el software `RAI-servers-and-clients`, que permite el acceso remoto a los recursos y que comentaremos posteriormente.

4.1.3. Otras plataformas

Además del robot Pioneer y del robot B21, se ha utilizado JDE en otras plataformas. Por ejemplo, la parte perceptiva se ha programado también para el robot Xavier de la Universidad Carnegie Mellon durante las estancias realizadas allí. En ella los algoritmos perceptivos, escritos en C, se envolvieron en la arquitectura software TCA. Tres procesadores se repartían el cómputo y los servidores de la arquitectura: CTR, CTCamera, etc.. Xavier, que se muestra en la figura 4.8(a), es el diseño en que se

inspiraron los fabricantes del B21.

Un segundo ejemplo es el servidor de la RoboCup. En este entorno, que se ilustra en la figura 4.8(c), se establece una conexión por *sockets* entre el programa que controla el movimiento de los jugadores y el servidor. El servidor proporciona al controlador los datos sensoriales que percibe cada jugador, básicamente su visión local, e implementa los comandos de movimiento que éste genera.

Como ampliaremos en el capítulo 5, una tercera plataforma sobre la que se han programado varios comportamientos siguiendo la división que propugna JDE es el robot EyeBot, que se muestra en la figura 4.8(b).

4.2. Acceso remoto: servidores sockets

Hasta aquí hemos presentado los distintos recursos hardware de las plataformas utilizadas, básicamente sensores y motores, junto con el software de acceso local. Este software, tanto la capa `RAI`, como el *driver* de visión o `Saphira`, permite leer las medidas de los diferentes sensores, y mandar comandos de movimiento a los motores. Con ese software el programa que necesite un recurso debe ejecutarse forzosamente en el mismo ordenador al cual esté conectado el recurso hardware. Por ejemplo, en el caso del B21 cualquier programa que comande la base debería correr en PC izquierdo, y cualquiera que haga procesamiento de imágenes en PC derecho. En el caso del Pioneer, cualquier programa que mueva la base motora o utilice las imágenes de la cámara de videoconferencia ha de ejecutarse obligadamente en el portátil a bordo del robot.

Para superar esta limitación se ha desarrollado la actual arquitectura de servidores y clientes, tal y como ilustra la figura 4.9. De un lado, los servidores ofrecen acceso a los sensores y los comandos motrices a cualquier programa cliente, y del otro lado los distintos programas clientes solicitan esos servicios a través de la red. Cada servidor encapsula ciertos sensores y actuadores que hay en su máquina y ofrece su funcionalidad al resto de los programas a través de una *interfaz de mensajes*.

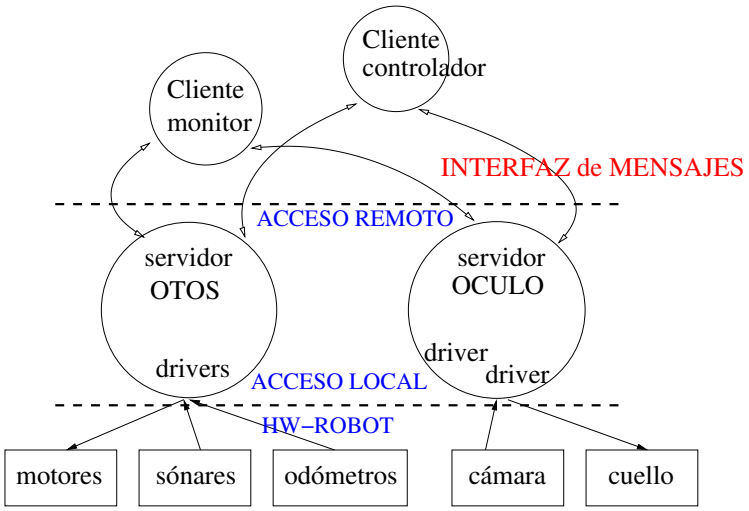


Figura 4.9: *Arquitectura software con servidores y clientes*

En concreto se han programado dos servidores: `otos` y `óculo`. El servidor `otos` incluye el acceso a los motores, los sensores de proximidad (como infrarrojos, sónar o láser), los sensores de contacto y el de voltaje. El servidor `óculo` se encarga de los sensores de imagen y caso de existir, del movimiento del cuello. Los clientes, con independencia de la máquina en que se ejecutan, solicitan medidas sensoriales y envían comandos intercambiando mensajes con los servidores a través de la red. Entre un cliente y un servidor se establece una conexión `tcp`, y a través de ella se establece entre ambos un diálogo regido por un protocolo que describiremos posteriormente en detalle.

La necesidad de dos servidores, `otos` y `óculo`, y no de uno único nació históricamente en el B21, en el cual los recursos de visión y de la base motora están ineludiblemente situados en dos máquinas diferentes. En el caso del Pioneer los recursos están asociados a la misma máquina, así que podrían

haberse unificado en un único servidor. Sin embargo, distribuirlo en dos no molesta en absoluto y es necesario para el B21. En el Pioneer los dos servidores son dos procesos ejecutándose en el mismo ordenador a bordo.

Esta arquitectura de servidores permite a un programa usuario acceder remotamente a cuantos recursos necesite a través de la red, vía *sockets*. Esto supone un valor añadido frente a entornos como Saphira o ARIA, que sólo ofrecen acceso local. Con el software que hemos desarrollado, un cliente se puede conectar a varios servidores y los servidores se han programado para que admitan varios clientes a la vez. Por ejemplo, si un programa está ejecutándose en el ordenador D y quiere acceder a las lecturas sónares actuales no tiene más que conectarse con el servidor `otos` y solicitarlos. Si ese mismo programa quiere acceder a las imágenes de la cámara solo necesita conectarse al servidor `óculo`. Puede estar conectado a ambos servidores a la vez si requiere ambos servicios.

Esta arquitectura también permite aprovechar mejor la capacidad de cómputo disponible. En la práctica facilita el procesamiento distribuido y paralelo de la información que proviene de los sensores. No es necesario que un cliente se ejecute en la máquina donde residen los sensores, puede estar en cualquier ordenador a bordo del robot, e incluso en cualquiera exterior enganchado a la red. De este modo se pueden poner más clientes en las máquinas más rápidas y balancear la carga de trabajo. Además, tampoco hay problema en que servidor y cliente se ejecuten en la misma máquina, la interfaz de mensajes es transparente a la ubicación de unos y otros.

Otra virtud de esta orientación a servidores y clientes es que hace a los programas más indepedientes del robot concreto. Primero, permite portar la arquitectura a varios plataformas con extrema facilidad. La interfaz de mensajes sigue valiendo aunque cambiemos el robot que tiene debajo. El único cambio necesario es incluir en los servidores el soporte para los nuevos *drivers*, es decir, el modo de conseguir los datos sensoriales en el nuevo robot, incorporando las llamadas a función pertinentes. Por ejemplo, la migración de las aplicaciones del B21 al Pioneer no ha sido traumática. Segundo, permite soportar nueva funcionalidad sin depender de una plataforma concreta de un fabricante. Por ejemplo, el soporte a la visión no está incluido en `RAI` ni en Saphira, pero se ha incorporado sin problemas en los servidores, añadiendo nuevos mensajes a la interfaz, que en el bajo nivel se sirven apoyándose en `video4linux`. La ampliación de la interfaz con nuevos mensajes no implica la recompilación de los clientes. Los antiguos clientes siguen funcionando sin retocarlos, aunque no utilicen los nuevos mensajes.

Estas ventajas no vienen gratuitamente y conllevan un coste en cuanto al aumento de los retardos entre que se lee el dato sensorial y éste llega, atravesando los servidores y la red, al programa donde realmente se procesa. Otro inconveniente es cierta desincronización de las medidas. En un único procesador al que se atan los sensores todas esas lecturas están datadas muy precisamente por el reloj del ordenador local. Al introducir los servidores, el reloj con el que se datan las lecturas en el programa que las recibe acumula los retardos variables de la red.

Una alternativa a `otos` y `óculo` es el software de `RAI`, que también proporciona un conjunto de programas servidores y librerías para el desarrollo de programas que accedan remotamente a los recursos. Este conjunto se llama `Rai-servers-and-clients`, y engloba un servidor de comunicaciones `tcxServer`, un servidor de los sensores básicos y la base motora `baseServer`, un servidor para la tarjeta reproductora de fonemas `speechServer` y un servidor para la unidad cuello `panServer`.

Los servidores de `Rai-servers-and-clients` se acompañan de unas librerías de acceso remoto, que se enlazan con el programa del usuario. Estas librerías mantienen literalmente los mismos nombres de funciones que se utilizan en el acceso local para que el programa usuario lea los sensores u ordene comandos de movimiento. Aunque la cabecera es la misma que el acceso local, su implementación implica ahora la comunicación por red con cierto servidor. De este modo la complicación de red queda oculta al programa de usuario, que llama a las mismas funciones que con el acceso local. La implementación de la librería de acceso remoto no es con llamadas a procedimiento remoto (RPC), sino que incorpora al programa usuario unas tareas que periódicamente hablan con los servidores para mantener actualizada su copia de las variables sensoriales. De esta manera el acceso no acumula todos los retardos que implicaría una llamada a procedimiento remoto bloqueante.

A pesar de estas ventajas, `Rai-servers-and-clients` cuenta con algunos inconvenientes que motivaron precisamente el nacimiento de los servidores desarrollados. Primero, sus servidores sólo permiten conectarse a un único cliente. De esta manera se proporciona acceso remoto pero se impide

el paralelismo que hay en la conexión de varios clientes al mismo servidor. Segundo, con objeto de mantener la misma interfaz de acceso local y remoto se obliga a que los clientes estén forzosamente programados con `Rai-scheduler`. Esto es así porque las rutinas de comunicación, incluídas en las librerías de acceso remoto de los clientes, están codificadas precisamente como tareas de `Rai-scheduler` y necesitan de ese planificador para darles vida.

Otras alternativas a la arquitectura software desarrollada son la arquitectura TCA [O'Sullivan *et al.*, 1997], con servidores como el CTR para los sónares o el CTCamera para las imágenes, y las arquitecturas de objetos distribuidos como CORBA. En contraste con ellas, con los servidores `otos` y `óculo` no tenemos transparencia de red ni resolución de nombres. No hay servidor de nombres, cada cliente debe saber en qué máquina está corriendo el servidor que le interesa y su funcionalidad, es decir, cuál es el subconjunto de mensajes a los que responde. La idea era no complicar en exceso la arquitectura software con servicios dinámicos o localización de servicios.

4.2.1. Protocolo JDE entre servidores y clientes

Entre los servidores y los clientes se establece un diálogo, una interacción que está regulada por el protocolo que hemos desarrollado. Este protocolo JDE entre servidores y clientes fija los mensajes posibles entre ambos y las reglas o semántica que los acompañan. Está pensado para el envío continuo (*streaming*) y en él hemos establecido tres patrones de interacción: la suscripción a sensores cortos, el envío bajo demanda de imágenes y las órdenes de movimiento. Iremos detallando estos patrones cuando comentemos los dos servidores desarrollados: `otos` y `óculo`.

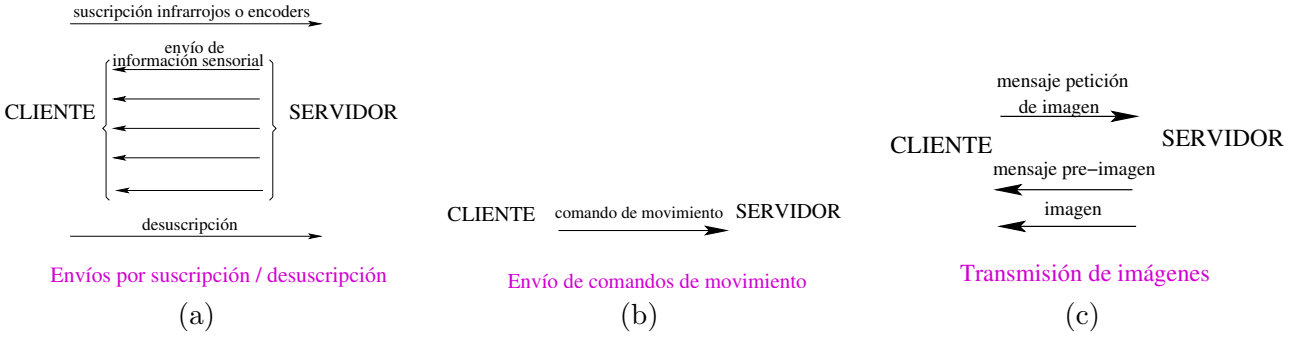


Figura 4.10: *Patrones de interacción asociados a sensores cortos (a), acciones (b) e imágenes (c)*

Como ya hemos anticipado, se define una *interfaz de mensajes* que ofrece el acceso remoto a los sensores y actuadores del robot. Así, la llamada a función o la consulta a una variable que se utilizan en el acceso local se sustituyen en el acceso remoto por el envío y recepción de ciertos mensajes. Si localmente las variables eran actualizadas por una tarea, ahora esas variables no residen en la máquina donde está el sensor y la tarea de actualización incluirá su petición al servidor correspondiente y la lectura desde la red de las respuestas oportunas. Del mismo modo, la llamada a función con que se comanda un movimiento se sustituye por el envío del mensaje correspondiente al servidor que realmente materializa esas órdenes a los motores. Gracias a los servidores no es necesario que el cliente enlace ninguna librería de acceso local a sensores o motores, con lo que el programa puede compilarse en cualquier máquina fuera del robot. No se invoca a ninguna función específica, simplemente se envían y reciben mensajes por la red. El único requisito es que los servidores deben estar funcionando antes que el programa cliente solicite sus servicios, para que puedan responderle.

2		3	8	1	8	9	6	3	5	2	1		1	4		1	3	8	2	.	0	\n

Figura 4.11: *Formato de línea con caracteres ASCII*

Para la implementación de la comunicación entre servidores y clientes hemos establecido un *formato de línea* en el cual se intercambian los mensajes del protocolo. Es similar al que existe entre el sistema operativo del Pioneer y el portátil por el puerto serie. En cuanto a su sintaxis, todos los mensajes

entre `otos`, `óculo` y sus clientes empiezan por un número entero que indica el tipo de mensaje, pueden incluir varios argumentos y acaban siempre en un retorno de carro. Por ejemplo, en la figura 4.11 se observa el mensaje utilizado para que el servidor envíe una lectura del sensor sónar: en primer lugar el número 2, para indicar que este mensaje contiene datos sónar y no otros, después un sello de tiempo 3818963521, después un entero que señala el número de sensor, y finalmente su lectura en milímetros. Todos los mensajes termina en un retorno de carro `\n` que señala el fin de mensaje y permite simplificar las rutinas de recepción. El protocolo permite enviar las lecturas de varios sensores sónares en el mismo mensaje, añadiendo tantos pares sensor-lectura como se necesiten.

Es de destacar que los códigos y argumentos se envían como texto ASCII, de manera que para enviar el número 19'5 se envían 4 caracteres, un 1, un 9, una coma, y un 5. Se ha elegido formato texto ASCII porque es fácilmente interpretable por clientes con diferentes representaciones internas de los números (*little-endian* o *big-endian*). Sólo se tiene que decodificar el texto que va en cada mensaje y hacer correctamente las conversiones a números. Además, este formato es independiente del sistema operativo, así un cliente puede estar programado en un sistema MS-Windows que entenderá los datos del servidor `otos` aunque éste corra en una máquina Linux. Describiremos a continuación la semántica asociada a los distintos mensajes del protocolo, adscritos al diálogo con alguno de los servidores disponibles: `otos` y `óculo`.

Servidor otos

El servidor `otos` reúne los servicios de los sensores de proximidad como los de ultrasonidos, el láser, los infrarrojos y los táctiles. Además, ofrece los datos que generan los odómetros de los motores de tracción del robot, tanto la posición como la velocidad estimada a partir de ellos. También entrega las medidas del sensor de voltaje de la batería y permite enviar comandos de movimiento a la base motora.

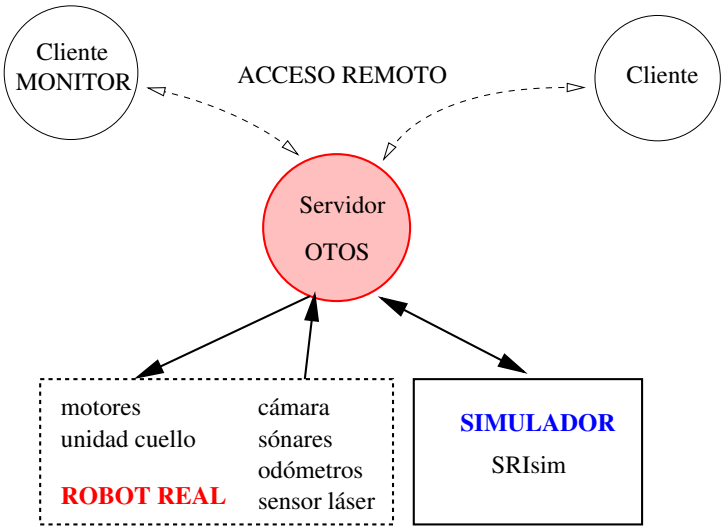


Figura 4.12: *Servidor `otos` con sensores y actuadores, reales o simulados*

Para todos esos sensores ofrece un *servicio de suscripción directa*, que consiste en que el cliente, una vez conectado, se suscribe a un determinado sensor y el servidor le entrega datos de ese sensor cada vez que haya medidas nuevas. Los clientes pueden suscribirse exclusivamente a los sensores de su interés, discrecionalmente, y desuscribirse a voluntad en cualquier momento. Por ejemplo, si un cliente no necesita las lecturas de odometría entonces no se suscribirá a los *encoders*, evitando sobrecargar el canal de comunicaciones con datos innecesarios.

En este planteamiento el servidor tiene la iniciativa y envía una lectura sólo cuando hay una nueva. Gracias a ello hay garantía de que al cliente le llegan todas las lecturas y cada una de ellas sólo una vez. Si el cliente pidiera periódicamente las medidas sensoriales podrían perderse lecturas o podría recibir medidas duplicadas, si los pidiera a un ritmo más lento o superior al de captura,

respectivamente. Además, se minimiza el tráfico de sobrecarga debido al protocolo (*overhead*) porque no hay que generar constantemente mensajes de petición de esos datos. Este esquema es viable dada la relativa *lentitud* y el poco volumen de los sensores asociados a este servidor y el flujo que es capaz de transportar la red. Por ejemplo, los sónares en el B21 entregan tres lecturas de la corona completa por segundo, es decir $24 * 3 = 72$ números, mientras que la cámara entrega 10 imágenes por segundo, que a 1 byte por pixel suponen $320 * 240 * 10 = 768000$ números por segundo.

Formato en línea del mensaje	*Descripción*	*Emisor*
OTOS_laser (tiempo, %d) (medida, %f) ... (medida, %f)	Datos láser expresados en milímetros desde el centro del robot. Hay 180 medidas en cada mensaje	servidor
OTOS_subscribe_laser	Suscripción a las medidas láser	cliente
OTOS_unsubscribe_laser	Desuscripción de las medidas láser	cliente
OTOS_sonars (tiempo, %d) (sensor, %d) (medida, %f)	Datos de los sónares, la medida es la distancia al obstáculo más cercano en la normal al sensor, en milímetros desde el centro del robot. Hay tantos pares sensor-medida como datos nuevos, normalmente dos	servidor
OTOS_subscribe_us	Suscripción a las medidas de ultrasonido	cliente
OTOS_unsubscribe_us	Desuscripción de las medidas de ultrasonido	cliente
OTOS_ir (tiempo, %d) (fila, %d) (columna, %d) (medida, %d)	Datos infrarrojos del sensor situado en esa fila y esa columna, la medida es la cantidad de luz reflejada, normalizada de 0 a 255	servidor
OTOS_subscribe_ir		cliente
OTOS_unsubscribe_ir		cliente
OTOS_tactiles (tiempo, %d) (fila, %d) (columna, %d) (medida, %d)	Datos táctiles del sensor situado en esa fila y esa columna, la medida es igual a 1 si en el sensor hay contacto, 0 si deja de haberlo	servidor
OTOS_subscribe_tactiles		cliente
OTOS_unsubscribe_tactiles		cliente
OTOS_encoders (tiempo, %d) (X, %f) (Y, %f) (Θ, %f)	Posición y orientación del robot respecto del sistema absoluto inicial, estimados por el sistema interno de odometría. X e Y en milímetros, Θ en grados	servidor
OTOS_subscribe_encoders		cliente
OTOS_unsubscribe_encoders		cliente
OTOS_battery (tiempo, %d) (voltaje, %f)	Voltaje de la batería, en voltios	servidor
OTOS_subscribe_battery		cliente
OTOS_unsubscribe_battery		cliente
OTOS_sensed_drive_speed (tiempo, %d) (velocidad, %f)	Velocidad sensada de tracción, en milímetros/seg.	servidor
OTOS_subscribe_sensed_drive_speed		cliente
OTOS_unsubscribe_sensed_drive_speed		cliente
OTOS_sensed_steer_speed (tiempo, %d) (velocidad, %f)	Velocidad sensada de giro, en grados/seg.	servidor
OTOS_subscribe_sensed_steer_speed		cliente
OTOS_unsubscribe_sensed_steer_speed		cliente

Tabla 4.3: *Mensajes sensoriales con el servidor OTOS*

La tabla 4.3 refleja los mensajes sensoriales posibles entre `otos` y sus clientes, es decir toda la funcionalidad sensorial que ofrece el servidor. En la primera columna se especifica el formato de línea de los mensajes. El primer componente indica el tipo de mensaje, que incluimos en forma de nombre para mayor claridad. Los argumentos van encerrados entre paréntesis con %d para indicar que es un número entero y %f un número real. La segunda columna da una breve descripción de la semántica del mensaje y la tercera columna indica dónde se genera ese mensaje, es decir, si lo emite el programa cliente o el propio servidor.

Por ejemplo, en el mensaje sónares se indica el sensor y la medida de distancia, en milímetros. El mensaje con las medidas láser incluye las 180 lecturas, que expresan la distancia en milímetros en los respectivos ángulos. En el mensaje de los odómetros se indica la posición del robot (en milímetros) y

su orientación (en grados). Estas coordenadas del sistema odométrico están referenciadas a un sistema absoluto externo, fijo, normalmente coincidente con la posición y orientación iniciales del robot. Las velocidades que van en los mensajes, tanto sensadas como comandadas siguen el criterio de signos que manda como positivo el sentido de avance para la tracción y el sentido antihorario para el giro. El sello temporal que aparece en todos los mensajes que incluyen lecturas sirve para datar esa medida. Tiene un valor entero y permite utilizar un tiempo local como referencia (por ejemplo en microsegundos desde cierto origen), con lo que se tiene no sólo el orden sino la distancia temporal entre dos lecturas.

Formato en línea del mensaje	*Descripción*	*Emisor*
OTOS_goodbye	Clausura de conexión	cliente
OTOS_hello	Mensaje de inicio, opcional	cliente
OTOS_drive_speed (velocidad, %f) (aceleracion, %f)	Control en velocidad de la tracción	cliente
OTOS_steer_speed (velocidad, %f) (aceleracion, %f)	Control en velocidad del giro	cliente
OTOS_limp	Deja los motores desembragados	cliente

Tabla 4.4: *Mensajes genéricos y de actuación con el servidor OTOS*

Respecto de los actuadores, el protocolo ofrece un *servicio de acciones*, que consiste en el envío de una órden, sin esperar confirmación de que se ha ejecutado. En la tabla 4.4 aparecen los dos mensajes principales de actuación en `otos`, para ordenar un control en velocidad de la tracción (`OTOS_drive_speed`) y del giro (`OTOS_steer_speed`). Tienen una semántica no acumulativa en el tiempo, materializando la ausencia de horizonte temporal de JDE, que hace hincapié en el presente. Estos mensajes se apoyan en los controladores PID de bajo nivel que incorporan tanto el Pioneer como el B21 en su base motora. Es de destacar que no se espera ningún mensaje de retorno que confirme el éxito o fracaso del comando. Esto cuadra con la monitorización distribuida que propugna JDE, el esquema que ordena el movimiento comprobará, si le interesa, leyendo otros sensores si efectivamente se ha conseguido la velocidad comandada.

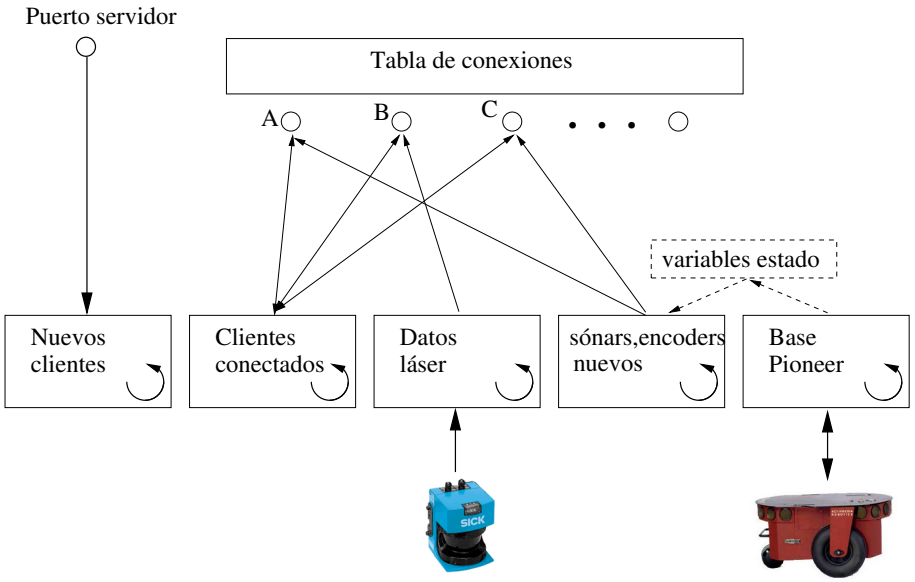


Figura 4.13: *Implementación del servidor otos con cinco hebras*

En cuanto a su implementación, el servidor `otos` se ha programado como 5 hebras concurrentes, tal y como muestra la figura 4.13. La primera se encarga de atender peticiones de conexión de nuevos clientes y de comprobar previamente si las hay. La segunda se encarga de servir a los clientes ya conectados, que pueden cambiar sus condiciones de suscripción (estado de su conexión), o enviar un comando motor. La tercera se encarga de muestrear en el puerto serie los datos provenientes del sensor láser, y de enviárselos a los clientes suscritos.

Se ha utilizado el entorno Saphira como *driver* de la base motora: los comandos motores, las

lecturas de los sónares y los odómetros se transmiten empleando las librerías de Saphira. La quinta hebra es una hebra automática de Saphira que se encarga de refrescar las variables que almacenan precisamente esos datos de sónares y odómetros. La cuarta hebra detecta esos refrescos y envía la información a los clientes suscritos a sónares u odometría.

Dado que no se espera un número masivo de clientes, no ha sido necesaria otra implementación más eficiente del servidor. Además, la multitarea necesaria para `otos` se ha conseguido con la que ofrece Saphira. Cada una de estas hebras se ha materializado como una microtarea de Saphira, que iteran cada 100 ms.

Servidor óculo

El servidor `óculo` aúna las funciones asociadas a la unidad cuello y a la cámara. Con ello permite a los clientes mover la unidad *pantilt* a voluntad, especificándole ángulos objetivo, y pone a disposición de los clientes el flujo de imágenes obtenidas con la cámara. Sus orígenes históricos están en el B21, donde cámara y cuello se conectaban al mismo ordenador.

Formato en línea del mensaje	*Descripción*	*Emisor*
OCULO_rgb24bpp_sifntsc_image_query	Petición de nueva imagen en color	cliente
OCULO_rgb24bpp_sifntsc_image (num_columnas, %d) (num_filas, %d) (bytes_per_pixel, %d) (tiempo, %d) (imagen)	Imagen actual de color, precedida de un mensaje cabecera con el número de columnas y filas de la imagen, el número de bytes por pixel y un sello de tiempo	servidor
OCULO_grey8bpp_image_query	Petición de nueva imagen	cliente
OCULO_grey8bpp_image (num_columnas, %d) (num_filas, %d) (bytes_per_pixel, %d) (tiempo, %d) (imagen)	Imagen actual, precedida de un mensaje cabecera con el número de columnas y filas de la imagen, el número de bytes por pixel y un sello de tiempo	servidor
OCULO_pantilt_encoders (pan_angle, %f) (tilt_angle, %f)	Indica posición actual de la unidad *pantilt* dando el ángulo horizontal y el vertical en grados	servidor
OCULO_subscribe_pantilt_encoders		cliente
OCULO_unsubscribe_pantilt_encoders		cliente

Tabla 4.5: *Mensajes sensoriales con el servidor ÓCULO*

La principal información que suministra `óculo` es la visual, las imágenes capturadas con la cámara. Éstas se ofrecen a los clientes a través de un servicio de *imágenes bajo demanda*: cada vez que el cliente necesita una imagen la solicita con un mensaje explícito y el servidor le responde con dos mensajes, uno de cabecera donde se especifica el tamaño de la imagen subsiguiente, y otro especial donde va la imagen en sí misma. Tal y como indica la tabla 4.5 el mensaje de cabecera acaba en retorno de carro y consta del código respectivo, dos enteros para indicar las filas y columnas de la imagen, un tercer número con los bytes que se dedican a cada pixel y un sello de tiempo. Inmediatamente después de él se envían los pixels de la imagen, como un chorro de bytes cuyo tamaño se conoce precisamente por el mensaje de cabecera previo.

Las imágenes suponen un volumen relativamente grande de bytes (comparado con otros sensores como los sónares) y el ritmo de captura de una cámara normal son unos 25 fps. Por ambas razones se requiere mucho ancho de banda para transmitir imágenes y esto hacía poco recomendable un servicio de suscripción directa con el sensor cámara (aunque se podría añadir sin problema si fuera conveniente). Una ventaja adicional del servicio bajo demanda es que permite desacoplar el ritmo de captura del ritmo al que es capaz de procesar imágenes el cliente. Por ejemplo, si un cliente ejecuta algoritmos de visión computacional y es capaz de digerir 5 imágenes por segundo, no tiene sentido saturarle con 25 fps que es capaz de capturar la cámara. En este sentido el servicio bajo demanda permite que al cliente le lleguen imágenes exactamente al ritmo que es capaz de procesar. Otra ventaja adicional es que al adelantar en el mensaje de cabecera la longitud en bytes de la imagen se consigue transparencia de datos. La imagen puede incluir retornos de carro en su interior, porque el receptor estará esperando cierta cantidad de bytes, sin atender a que llegue un retorno de carro como delimitador del fin de

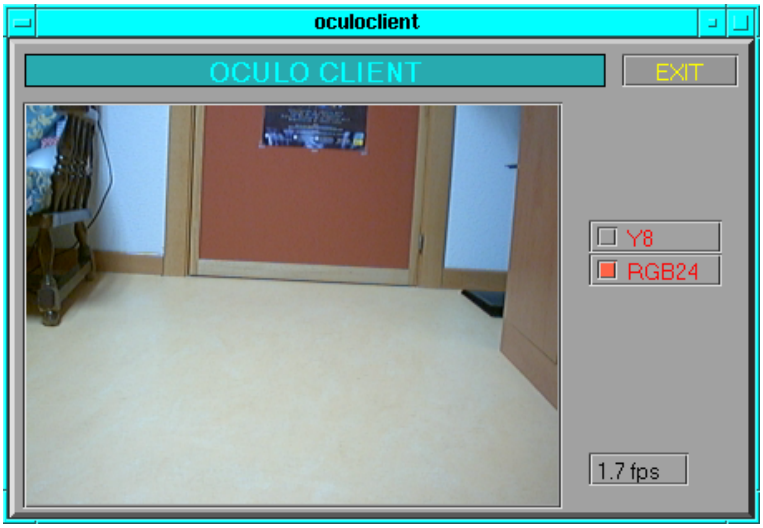


Figura 4.14: *`Oculoclient`, cliente de prueba del servidor `óculo`*

mensaje.

El servidor `óculo` puede transmitir varios formatos de imagen, cada uno de ellos necesita un mensaje preimagen característico. Se ha dado soporte a imágenes en niveles de gris, imágenes *logpolar* e imágenes RGB. De momento se envían imágenes planas, sin compresión. Una posibilidad a estudiar es el envío de imágenes comprimidas, viendo si compensa el tiempo de comprimir, transmitir comprimido y descomprimir frente al envío de la imagen plana.

En la figura 4.14 se aprecia la interfaz del programa `oculoclient`, que es un cliente de prueba que se conecta a `óculo` para solicitarle imágenes en blanco y negro, o en color.

Formato en línea del mensaje	*Descripción*	*Emisor*
OCULO_goodbye	Clausura de conexión	cliente
OCULO_hello	Mensaje de inicio, opcional	cliente
OCULO_pantilt_limits_query	Petición de los ángulos límite para el movimiento horizontal y el vertical	cliente
OCULO_pantilt_limits (min_pan, %f) (max_pan, %f) (min_tilt, %f) (max_tilt, %f)	Respuesta del servidor con los ángulos límite, en grados	servidor
OCULO_pantilt_position (pan_angle, %f) (tilt_angle, %f)	Orden de movimiento de la *pantilt* a esas coordenadas angulares absolutas, en grados	cliente
OCULO_pantilt_relative_position (pan_angle, %f) (tilt_angle, %f)	Orden de movimiento de la *pantilt* a esas coordenadas angulares relativas a las posición actual, en grados	cliente
OCULO_pantilt_origin	Orden de mover la unidad *pantilt* al origen de coordenadas angulares	cliente
OCULO_pantilt_halt	Orden de detener el movimiento de la unidad *pantilt*	cliente

Tabla 4.6: *Mensajes genéricos y de actuación con el servidor ÓCULO*

La posición actual de la unidad *pantilt* es una información sensorial, y se trata con un servicio de suscripción directa, tal y como se refleja en la tabla 4.5. Las órdenes de movimiento a la unidad cuello se especifican en la tabla 4.6. Básicamente son controles en posición, que se apoyan en los controladores internos de la propia unidad. El soporte para la *pantilt* se incluye en el servidor `óculo` que corre en el B21, pero no se utiliza en el Pioneer, que puede utilizar el propio robot para orientar la cámara. No se han incorporado a `óculo` actuaciones como los niveles de enfoque, o la activación y desactivación del autoiris, porque la cámara de videoconferencia utilizada no los permite. No obstante, el protocolo se puede ampliar con nuevos mensajes para ello si es necesario.

Como muestra la figura 4.15, el servidor `óculo` se ha implementado como cuatro hebras de kernel,

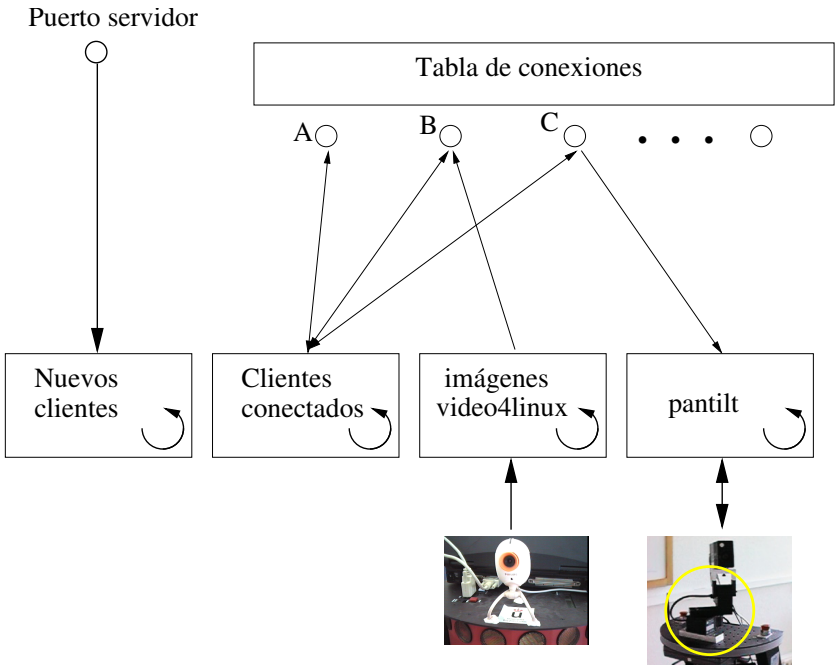


Figura 4.15: *Implementación del servidor óculo con cinco hebras*

hebras de `pthreads`. De modo similar a como hace `otos`, la primera se encarga de atender a nuevas peticiones de conexión y la segunda a los clientes ya suscritos. Para aumentar su eficiencia estas dos primeras hebras no muestrean periódicamente los *sockets*, sino que se quedan bloqueadas sin consumir CPU hasta que hay algún byte disponible en los *sockets* respectivos. Cuando se recibe una solicitud de imágen, esta hebra lee la última capturada y se la envía al cliente precedida de un mensaje preimagen. La tercera hebra se encarga de recoger las imágenes de la cámara local y actualizar su copia. Para ello utiliza la interfaz de `video4linux`. Una cuarta hebra se encarga de las comunicaciones con la unidad de cuello (*pantilt*), que se establecen a través del puerto serie. Las órdenes y los datos se envía y reciben por ese puerto en un formato fijado por el fabricante.

4.2.2. Comunicaciones

Los servidores implementados se apoyan en los mecanismos que proporciona Linux para la comunicación interprocesos. Ficheros, memoria compartida, *pipes* y *fifos* están orientados a comunicar procesos que ejecutan en la misma máquina. Para procesos en diferentes ordenadores, como es el caso genérico de clientes y servidores, Linux ofrece los *sockets* como abstracción de comunicaciones. En términos telemáticos el *socket* es la abstracción de comunicaciones en el nivel de transporte. Existen *sockets* orientados a datagrama, sobre `udp/ip`, que son muy eficientes pero en los que se contempla la posibilidad ocasional de perder paquetes o desordenarlos respecto del envío. En la implementación de las comunicaciones entre servidores y clientes se quieren conexiones fiables, por lo que hemos utilizado *sockets* orientados a conexión, sobre `tcp/ip`. La conexión `tcp` es segura y fiable, con lo que los problemas de pérdidas y reordenamiento de mensajes se dan por resueltos. Así, en el protocolo no hemos implementado ningún mecanismo de recuperación de mensajes.

Los clientes establecen una conexión `tcp` con el servidor, que les está esperando en cierto puerto, y sobre esa conexión dialogan empleando el protocolo descrito. En términos telemáticos, ese protocolo JDE entre clientes y servidores es un protocolo entre aplicaciones, al mismo nivel que `http` o `smtp`. A la hora de establecer conexión los *sockets* admiten nombres para designar máquinas, resolviendo el software de red la dirección `ip` asociada a cada nombre.

La funcionalidad de `tcp/ip` se ofrece en Linux a los programas a través de la librería de *sockets*. Un par de *sockets* representan los dos extremos de una conexión de red entre dos programas que están ejecutando en máquinas distintas conectadas en red. Los programas servidores preparan un puerto (`bind`) para que los programas clientes puedan conectarse (`connect`) a él, y se ponen a escuchar

(`listen`) nuevas conexiones. Por ejemplo, el servidor `otos` abre un *socket* en un puerto que el humano pasa como parámetro de ejecución al programa[13]. Una ejecución típica podría ser: `otos 3000`. El servidor escucha en ese puerto 3000 las peticiones de conexión que le llegan de los posibles clientes. Por cada cliente que se conecta se establece una comunicación particular, desocupando el puerto de enganche para otros potenciales clientes.

Una vez establecida la conexión, el cliente puede enviar mensajes al servidor y viceversa. Para las aplicaciones la conexión aparece como un descriptor de fichero en el cual a otro extremo está el programa remoto. Por ejemplo, la función `listen` entrega al servidor un descriptor de fichero detrás del cual hay un cliente que se acaba de conectar. De modo recíproco, la función `bind` entrega al cliente un descriptor detrás del cual está el servidor. En esos descriptores los programas pueden escribir datos para que lleguen al otro extremo (llamada al sistema `write`). También pueden leer de ese descriptor para recibir los datos que vienen de la otra parte (llamada al sistema `read`).

Hay que destacar que las conexiones de red se configuran por defecto de modo bloqueante, es decir, que una invocación a la llamada al sistema `read` sobre un descriptor asociado a un *socket* bloquea al llamante hasta que se reciban datos por la conexión. Normalmente tanto los servidores como los clientes tienen que hacer muchas tareas simultáneamente, y suele necesitarse que esas lecturas de red no sean bloqueantes. Un descriptor se configura no bloqueante con la llamada al kernel de Linux `ioctl(NON_BLOCK)`. En esta nueva situación la llamada `read` retorna -1 si en ese momento no hay datos disponibles en la conexión, devolviendo inmediatamente el flujo de control en lugar de bloquearse.

4.2.3. Clientes y servidores especiales

Además de los servidores `otos` y `óculo` y sus clientes de prueba, hemos desarrollado algunos servidores y clientes especiales, que comentaremos a continuación.

Cliente monitor

Un cliente que muestre continuamente los valores de los sensores resulta útil para monitorización. De este modo se puede verificar si hay algún sensor averiado. Por ejemplo, en el B21 era frecuente que los transductores se atascasen, con lo que desvirtuaban el efecto de los algoritmos. Con esta herramienta se puede detectar si están atascados o no. El monitor desarrollado también incorpora un *joystick* visual, que permite teleoperar al robot, moverlo a voluntad de un sitio a otro sin necesidad de hardware adicional. Por ello, este cliente también resulta ventajoso para desplazar el robot sin necesidad de empujarlo, o bien para controlar a mano su movimiento cuando se están probando algoritmos de percepción pasivos.

En la arquitectura descrita se pueden conectar varios clientes simultáneamente al mismo servidor. Esta característica se puede aprovechar para monitorizar los sensores *a la vez* que otros clientes utilizan esos mismos datos sensoriales para generar algún comportamiento. De este modo se puede monitorizar también en tiempo de ejecución, sin alterar ni reprogramar los programas de comportamiento autónomo. El cliente monitor se puede activar, conectar a voluntad y desactivarse cuando ya no sea necesario, para ahorrar recursos computacionales. Además, ese cliente monitor puede ejecutarse en una máquina fuera del robot, con lo que la sobrecarga en los ordenadores a bordo se limita a que los servidores den servicio a otro cliente adicional.

Servidor en diferido replay y cliente record

El servidor `replay` reemplaza a `otos` y a `óculo`, y permite reproducir en diferido los datos sensoriales almacenados en un fichero. Como se ilustra en la figura 4.16(b), el servidor ofrece dos puertos de conexión para emular la ejecución simultánea de ambos servidores originales. Para generar el contenido de ese fichero se utiliza el cliente `record`, que registra todos los datos sensoriales almacenándolos en un fichero histórico. Este cliente se conecta a los dos servidores `otos` y `óculo`, según muestra la figura 4.16(a), y asocia a cada lectura de sensor un sello temporal. La misma funcionalidad

[13] el número de puerto debe ser superior a 1023, dado que los 1024 primeros están reservados para otros servicios

se ha implementado también en un esquema de servicio dentro del programa que materializa los esquemas de JDE.

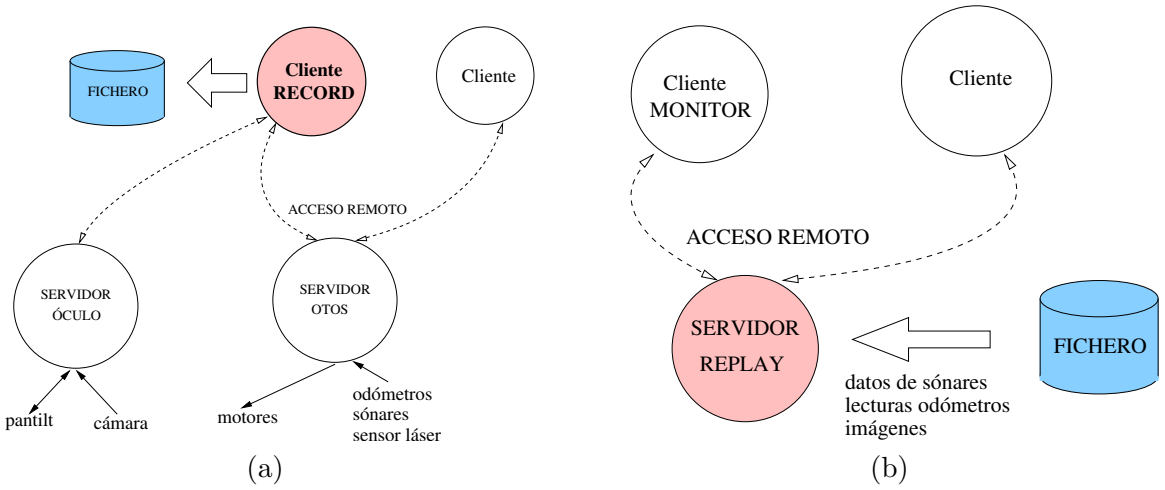


Figura 4.16: *Cliente* `record` *(a) y servidor* `replay` *(b)*

Estas herramientas resultan muy útiles cuando queremos probar diferentes algoritmos pasivos de construcción de representación del entorno exactamente sobre los mismos datos.

Una característica adicional es la reproducción a diferentes velocidades. El servidor en diferido reproduce por defecto al mismo ritmo que se grabaron las sensaciones, pero puede acelerar al ritmo que deseemos. Así, se puede reproducir a velocidades 1X, 2X, etc. teniendo como límite la velocidad de proceso del ordenador donde se ejecuta este servidor. Esta sincronización se apoya en las marcas de tiempo, en microsegundos, con las que se datan las lecturas sensoriales cuando llegan al cliente `record`. Para conseguir esta reproducción temporizada se detiene al proceso servidor una cierta cantidad de milisegundos entre línea y línea del fichero, de modo proporcional al tiempo entre los instantes de captura de ambas.

En el fichero se almacenan datos sensoriales como los sónares, los odómetros y las imágenes. El formato del fichero coincide con el formato de línea para facilitar su compatibilidad. En la tabla 4.7 se muestra un ejemplo. Las líneas que empiezan por 2 corresponden a datos sónar, con un sello de tiempo, el número de sensor y la medida. Las líneas que empiezan por 11 contienen datos odométricos. Finalmente, la línea que empieza por 1017 contiene un mensaje preimagen, en el que se especifica un sello temporal, el número de columnas y filas de la imagen, y cuántos bytes corresponden a cada pixel.

Otos sobre el simulador

Otra característica ventajosa que heredan los servidores y clientes de JDE es que el servidor `otos` se puede conectar indistintamente al simulador `SRIsim` o a la plataforma real. De este modo, los clientes que implementan alguna técnica de control con JDE se pueden probar en el simulador. Los datos sensoriales corresponden entonces al entorno simulado y a la situación en que se encuentre en él el robot simulado. Esto es muy útil para depuración. Además, el mismo cliente podrá ejecutarse sin recompilar nada en el robot real.

Gracias a la interfaz de mensajes, se abre la puerta a utilizar otros simuladores diferentes. Para poder utilizar un nuevo simulador basta programar un recubrimiento de su acceso a sensores y actuadores, y ofrecer esa misma funcionalidad en la forma de mensajes del protocolo desarrollado. Nuevos sensores se pueden incorporar con nuevos mensajes si es necesario.

4.3. Esquemas

Con el acceso remoto que hemos descrito en la sección anterior los datos sensoriales del robot pueden llegar a un proceso ejecutando en cualquier máquina y de él pueden partir las órdenes de movimiento

```
2 3818963505 11 1487.0
2 3818963521 14 1382.0
11 3818973846 0.0 0.0 6.28319
2 3819024225 1 3068.0
2 3819024282 2 1993.0
2 3819024299 5 2107.0
2 3819024314 9 6964.0
2 3819024330 10 1189.0
2 3819024345 13 2159.0
11 3819093349 0.0 0.0 6.28319
2 3819142829 0 283.0
2 3819142894 3 1689.0
2 3819142914 4 1019.0
2 3819142929 6 1681.0
2 3819142944 7 2331.0
2 3819142959 8 2332.0
2 3819142974 11 1488.0
2 3819142988 12 1118.0
2 3819143003 14 1380.0
2 3819143017 15 217.0
11 3819156096 0.0 0.0 6.28319
1017 3819156242 320 240 1
<E7><E7><E7><E8><E8><E9><EB><F2><F1><F4><F4><F4><F4><F4>
<EE><ED><E7><E0><DE><E0><CC><A0>y<83><A1><A6><AC><A7><A6>
<A4><A2><A1><9E><A1><9E><9D><A0><9D><9B><9A><9D><9A><9A><97>
<9C><9C><98><9C><98><96><98><94><95><97><98><93><80>
```

Tabla 4.7: *Fichero histórico con todas las sensaciones y actuaciones grabadas*

que se ejecutan en los actuadores del robot. Ahora bien, ¿cómo se materializan los esquemas que propugna la arquitectura JDE? Sea cual sea su implementación ha de cumplir algunos requisitos, como permitirlos ejecutar simultáneamente y que puedan activarse, desactivarse, modularse unos a otros e interactuar entre sí. Por ejemplo, los estímulos elaborados por los esquemas perceptivos deben ser visibles por los esquemas de actuación que los usan. Del mismo modo, los esquemas de actuación deben fijar los parámetros de modulación con los que sesgan el funcionamiento de sus esquemas hijos, los cuales deben leer el valor de esos parámetros. Además, los esquemas de actuación de cierto nivel deben competir unos con otros por tomar el control.

En esta sección describiremos cómo hemos materializado en programas las ideas principales de la arquitectura en cuanto a esquemas. Principalmente la implementación de cada esquema en una hebra independiente y el algoritmo de selección de acción programado. Concluimos la sección presentando algunas limitaciones de implementación que hemos identificado y una librería que hemos desarrollado para agilizar la codificación de esquemas de actuación con control borroso.

4.3.1. Una hebra por cada esquema

En la implementación realizada, cada esquema de los presentados en el capítulo 3 se materializa en una hebra de kernel independiente de las demás. Tanto si el esquema es perceptivo como si es de actuación, cada una de estas hebras ejecuta periódicamente una función de iteración, a un ritmo controlado. Esta hebra se ejecuta en paralelo con las hebras de los otros esquemas que no están en DORMIDO. Con esto se satisfacen los requisitos de la sección 3.1, en la que se definía un esquema-JDE como un flujo de ejecución independiente e iterativo.

Desde la óptica de sistemas operativos, una hebra es un flujo de control que ejecuta independientemente un conjunto de instrucciones y que comparte algunos recursos con el proceso que la alberga. Entre otros, todas las hebras de un proceso comparten el espacio de direcciones (memoria virtual) y la tabla de descriptores abiertos. Su principal ventaja es que el cambio de contexto entre dos hebras es más ágil que entre dos procesos normales. Reciben por ello el sobrenombre de procesos

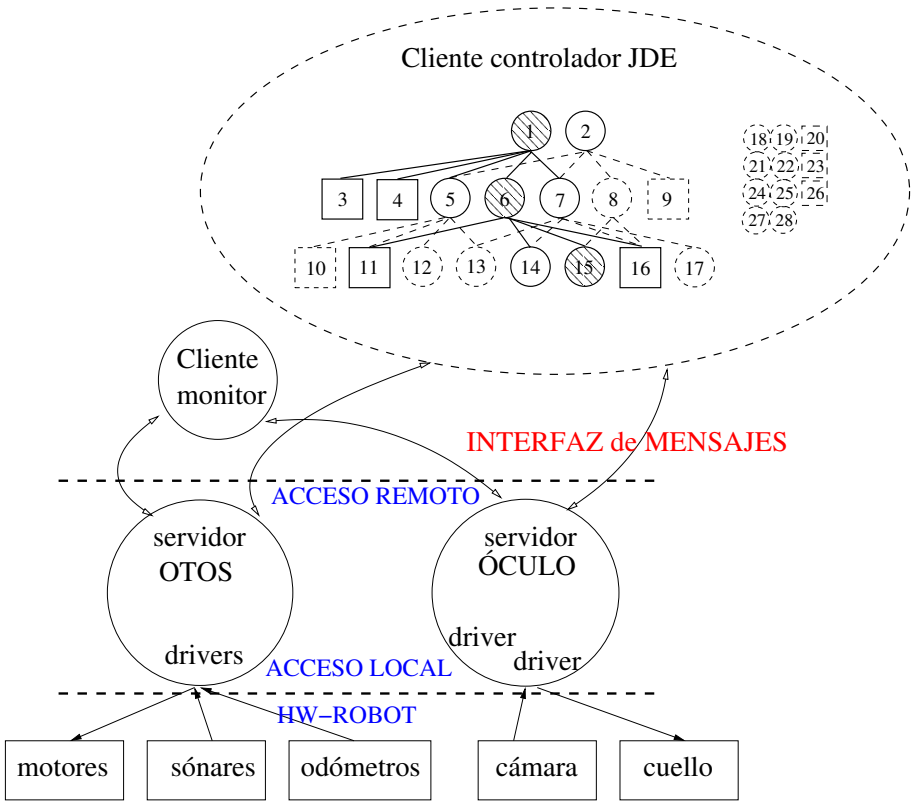


Figura 4.17: *Arquitectura completa con los esquemas de JDE como hebras de un proceso cliente*

ligeros. Como veremos a continuación, existen procesos ligeros de dos tipos: de kernel y de usuario.

En concreto, hemos utilizado las hebras de kernel de `PThreads`, que es la librería de referencia en Linux que materializa el estándard de hebras POSIX 1003.1. Al ser hebras de kernel, entran en el algoritmo de planificación de CPU del sistema nativo, y tienen más independencia unas de otras. La librería ofrece funciones para crear hebras (`pthread_create`) estableciendo la función que ejecutarán y sus parámetros, terminar su ejecución (`pthread_exit`), sincronizarla con la terminación de otras hebras (`pthread_join`) y otros mecanismos de sincronización como los *mutex* y las *variables condición*, que explicaremos posteriormente.

La otra opción de implementación disponible eran las hebras de usuario, pero éstas presentan algunos inconvenientes. Por ejemplo, si una se atasca se bloquean todas, porque son hebras cooperativas que se reparten el tiempo que el sistema operativo asigna a su proceso, y no tienen desalojo.

Así pues, todas las hebras que implementan esquemas pertenecen al mismo proceso, de manera que el sistema completo de control autónomo se articula en los dos procesos servidores y un único proceso que alberga a todos los esquemas, tal como ilustra la figura 4.17.

En nuestra implementación, cada esquema-JDE tiene asociado un identificador de esquema, una variable que almacena su estado de activación, un *mutex* y una *variable condición*. En realidad el estado, el *mutex* y la *variable condición* de todos los esquemas se almacenan en sendos *arrays* que se indexan precisamente por el identificador de esquema, que es un número entero distinto para cada uno de ellos. En cuanto a la programación, el sistema consta de un programa `jde.c` que centraliza las variables de estado, las *variables condición* y los *mutex* necesarios para todos los esquemas. Además, se encarga de establecer conexión con los servidores `otos` y `óculo`, activar las opciones de depuración de los distintos esquemas y arrancar la población potencial de esquemas.

Cada esquema se incorpora al software como dos ficheros, uno con la especificación (`miesquema.h`) y otro con la implementación (`miesquema.c`). En la especificación se declaran las funciones que otros esquemas pueden invocar relativas a este esquema. Por ejemplo, el esquema `motors.c` se encarga de materializar un control en velocidad sobre la tracción y el giro del robot. Las funciones que exporta son: `motors_startup` para arrancarlo, `motors_suspend` para detenerlo y `motors_resume(int *brothers,`

arbitration fn) para despertarlo, tal y como se muestra en la tabla 4.8. También se declaran las variables que el esquema acepta como parámetros de modulación, para que puedan ser accedidas desde el resto de esquemas. En este caso, la velocidad deseada de avance v, de giro w y el periodo de iteración del propio esquema motors_cycle, cuya memoria queda reservada y definida en motors.c.

```
extern void motors_shutdown();
extern void motors_startup();
extern void motors_suspend();
extern void motors_resume(int *brothers, arbitration fn);

extern float v; /* mm/s */
extern float w; /* deg/s*/
extern int motors_cycle;
```

Tabla 4.8: *Interfaz de un esquema: funciones arrancar, parar y despertar, y los parámetros de modulación.*

Cada nuevo esquema se compila generando el objeto miesquema.o. La colección de todos los esquemas desarrollados se enlaza con el jde.o para generar el ejecutable del sistema. Sólo los esquemas que explícitamente se han arrancado podrán activarse en algún instante. Existe un fichero de configuración jde.conf en el que se determina cuáles de ellos creará jde.c. Crear un esquema significa arrancar la hebra homóloga y situarlo en estado DORMIDO. A partir de entonces puede ser despertado para que forme parte de la arquitectura de control en cualquier momento.

Activación y desactivación

Para implementar el estado DORMIDO hemos usado las *variables condición* que ofrece la librería de Pthreads. Una *variable condición* es una abstracción que sirve para coordinar hebras que ejecutan en paralelo. Para usarlas, la librería Pthreads ofrece básicamente dos primitivas: pthread_cond_wait bloquea a la hebra que lo invoca hasta que otra hebra invoca a pthread_cond_signal sobre la misma *variable condición*. Utilizando estos mecanismos cada hebra puede quedarse dormida en su *variable condición*. Esta implementación es muy eficiente porque los esquemas en DORMIDO no consumen capacidad de cómputo del procesador.

Un esquema en DORMIDO puede activarse o reactivarse cuando otra hebra avisa con un pthread_cond_signal sobre la condición en la que está dormido. En general, cada esquema ofrece la posibilidad de su activación en una función miesquema_resume, que efectivamente emite un pthread_cond_signal sobre la condición y que el resto de los esquemas pueden invocar. Adicionalmente, esa función lleva como argumentos la lista de esquemas hermanos con la que se ha activado esta vez el esquema y una función de arbitraje, que describiremos posteriormente al hablar del algoritmo de selección de acción.

Típicamente cuando un esquema padre gana la competición de control en su nivel, fija los parámetros de sus hijos (escribiendo en las variables oportunas) e invoca a la funcion hijo_resume de cada uno de ellos (que efectivamente despierta al esquema hijo). Como vimos en el capítulo 3, cuando un esquema entra en estado ACTIVO entonces despierta a sus hijos, e inicia un nuevo nivel por debajo suyo, o emite directamente comandos a los motores si no tiene más niveles por debajo.

Cualquier esquema puede desactivar a otro escribiendo en su estado para ponerlo a DORMIDO. En realidad con esa escritura simplemente se solicita que se duerma. Los esquemas-JDE están programados de modo que la siguiente vez que el esquema receptor ejecuta su iteración se suspenda él solo en su condición asociada. Cada esquema ofrece la posibilidad de desactivación en una función miesquema_suspend que lo hace deternerse en su siguiente iteración, y que el resto de los esquemas puede invocar. Este mecanismo es el que emplea un esquema padre cuando sale del estado ACTIVO para detener a todos sus hijos, ejecutando hijo_suspend que duerme a cada uno de sus vástagos en su variable condición. Allí esperará sin consumir CPU a ser nuevamente despertado en otra situación en la que pueda aportar algo.

Con estas funciones se consigue que cualquier hebra, cualquier esquema, pueda activar y desactivar a otras. Se pone con ello de manifiesto que las hebras no pertenecen a ningún nivel a priori. Dependendiendo de quién las active pueden aparecer ahora en un nivel de la jerarquía y en otro

diferente un instante después. Se materializa así el principio enunciado en el capítulo 3 de que la jerarquía es dinámica, se reconfigura en el tiempo, y de que los esquemas no se adscriben a ningún nivel en concreto.

Variables compartidas para estímulos y parámetros

En el campo de la informática los mecanismos típicos de comunicaciones interprocesos se clasifican en dos: memoria compartida o paso de mensajes. En nuestra implementación, tanto los estímulos que elaboran los esquemas perceptivos como los parámetros de modulación de los esquemas se materializan en variables (memoria compartida). Esta opción simplifica la interacción interesquema y la agiliza enormemente. El coste computacional de leer una variable es mucho menor que el de elaborar mensajes y transmitirlos a través de la red continuamente. Por esto descartamos implementar los esquemas como procesos corriendo en diferentes máquinas y su interacción como diálogos entre ellos, a través de *sockets*, para pasarse mensajes.

En primer lugar, los estímulos perceptivos se materializan en variables, que los esquemas perceptivos definen y se encargan de refrescar periódicamente, y que los esquemas de actuación pueden leer cuando les interese. Así pues, la influencia que los esquemas perceptivos ejercen en los de actuación se da a través de variables comunes. La visibilidad está garantizada porque todas las hebras forman parte del mismo proceso y por ello comparten el espacio de memoria virtual donde se almacenan todas las variables. Del mismo modo, los esquemas perceptivos de un nivel inferior vuelcan sus resultados en ciertas variables que los esquemas perceptivos superiores leen para elaborar sus propios estímulos. En este sentido, la memoria virtual del proceso que alberga a todas las hebras se puede ver como un sistema de pizarra similar al de [Draper *et al.*, 1989].

En segundo lugar, los parámetros de modulación que acepta un esquema se materializan en variables que el propio esquema define. Los esquemas superiores que activen y quieran modular el funcionamiento de un esquema inferior escriben los valores adecuados en esas variables parámetro, refrescándolas a su ritmo. El esquema inferior se encarga de leer con su propia cadencia esa variable para comportarse de un modo o de otro dentro de la funcionalidad que implementa.

Una cuestión relevante es proteger el acceso en exclusión mutua a estas variables compartidas. Como los esquemas son hebras asíncronas y varios de ellos pueden intentar acceder simultáneamente a una misma variable, podrían aparecer condiciones de carrera. Para evitarlas hemos utilizado los *cerrojos* (*mutex*) que proporciona la librería `Pthreads`. La librería ofrece funciones para crearlos (`pthread_mutex_init`), destruirlos (`pthread_mutex_destroy`), echar el cerrojo (`pthread_mutex_lock`) y quitar el cerrojo (`pthread_mutex_unlock`). Típicamente, cada variable o conjunto de variables sensibles a las condiciones de carrera se protegen con un *mutex* y el convenio de los esquemas-JDE es solicitar el cerrojo antes de acceder a la variable, liberándolo después de usarla. Si en ese momento hay otra hebra accediendo a esa variable protegida la llamante se quedará bloqueada hasta que la primera suelte el cerrojo, con lo que se garantiza el acceso en exclusión mutua.

Esqueleto típico

El esqueleto típico de un esquema se ha programado con un bucle infinito que típicamente ejecuta una función de iteración en cada ciclo del bucle. En realidad, lo que haga en cada ciclo depende del estado de activación en que se encuentre y si es un esquema de actuación, del resultado de la competición por el control en ese momento y en ese nivel.

En la tabla 4.9 hemos puesto el bucle infinito de un esquema perceptivo a modo de ejemplo. En cada iteración se chequea primeramente si algún otro esquema decidió dormir al esquema. En ese caso la propia hebra se queda bloqueada sobre su variable condición, con la instrucción `pthread_cond_wait`. En caso contrario, toma el tiempo de sistema con `gettimeofday` y ejecuta una iteración de su función genuina (`perceptive_iteration` en la tabla 4.9). Al terminar esa función vuelve a tomar el tiempo del sistema. Finalmente la hebra se detiene un cierto intervalo temporal, invocando a la llamada al sistema `usleep`. Gracias a esta pausa se garantiza el ritmo periódico de ejecución de la hebra que enuncia JDE. Esa cadencia suele ser un parámetro que la propia hebra admite, en el caso de la tabla 4.9 sería la variable `perceptive_cycle`. La doble llamada al cronómetro local sirve para descontar el

```
for(;;)
   {
      if (state[SCH]==slept) pthread_cond_wait(condition[SCH]);
      else
         {
          gettimeofday(&a);
          perceptive_iteration();
          gettimeofday(&b);
          diff = (b-a);
          next = perceptive_cycle-diff;
          usleep(next);
         }
   }
```

Tabla 4.9: *Pseudocódigo del esqueleto típico de un esquema perceptivo*

tiempo de ejecución de la iteración genuina, que puede no ser despreciable, con lo que se ajusta más al ritmo periódico exacto.

En la tabla 4.10 se muestra también el esqueleto típico de un esquema de actuación. Es similar al de un esquema perceptivo, pero incluye la parte de selección de acción. Lo comentaremos posteriormente.

Figura 4.18: *Interfaz de depuración de la plataforma JDE*

Además de las hebras que corresponden a los esquemas de actuación y percepción del sistema, en nuestra implementación se han incluido varias hebras de servicio adicionales. Por ejemplo, se han incluido dos esquemas de servicio, `sensationsotos` y `sensationsoculo`, que se encargan de actualizar las variables internas que son la copia de los últimos datos sensoriales. Para mayor eficiencia computacional, estas hebras no ejecutan en iteraciones sino que se activan por el evento de nuevos datos en su comunicación con el servidor.

También se ha desarrollado una interfaz gráfica del sistema, que resulta muy útil para depurar

```
for(;;)
   {
   if (state[SCH]==slept) pthread_cond_wait(condition[SCH]);
   else {
      gettimeofday(&a);
      if (preconditions() <threshold)
         { /* my preconditions DONŤ match current situation */
         if (brothers_ready_or_active>1)
            { /* preconditions of another brother do match */
            state[SCH]=alert);
            }
         else
            { /* there is control absence */
            callforarbitration();
            if (state[SCH]==active) actuator_iteration();
             }
         }
      else
         {/* my preconditions match current situation */
         if (brothers_ready_or_active==0)
            {state[SCH]=winner)
            actuator_iteration(); }
         else
            { /* there is control overlap */
            callforarbitration();
            if (state[SCH]==winner) actuator_iteration(); }
         }
      gettimeofday(&b);
      diff = (b-a);
      next = actuator_cycle-diff;
      usleep(next);
      }
   }
```

Tabla 4.10: *Esqueleto típico de un esquema de actuación, incluyendo la selección de acción*

y observar estructuras internas. La interfaz se muestra en la figura 4.18, y se ha incluido como dos hebras de servicio más: una chequea si el usuario pulsó algún objeto gráfico, y la otra refresca la interfaz que el sistema le muestra al humano. Esta interfaz permite al diseñador ver los resultados de los esquemas perceptivos, cambiar parámetros de modulación en tiempo de vuelo o incluso activar o desactivar esquemas a voluntad. Contrasta con el monitor descrito en la sección anterior, que sólo permite visualizar los datos sensoriales crudos.

4.3.2. Algoritmo de selección de acción

El esqueleto típico de un esquema de actuación cuadra a grandes rasgos con el de un esquema perceptivo, pero además incluye la parte que implementa la selección de acción y que determina si la hebra ha de tomar el control ahora o no. En la tabla 4.10 ilustramos el esqueleto completo para una hebra de actuación, que incluye el algoritmo distribuido con el que hemos implementado el mecanismo teórico de selección de acción.

Tal y como muestra la cabecera de la función `miesquema_resume` en la tabla 4.8, cuando un padre despierta a un esquema hijo le pasa la lista de hermanos con la que tendrá que competir por el control. También le pasa a cada hijo una función propia de arbitraje explícito a la que invocar cuando éste detecte problemas en la competición por el control. Al despertarlos, el padre provoca que todos sus hijos pasen del estado DORMIDO a ALERTA. Esta selección refleja que el padre considera que sólo los esquemas despiertos pueden colaborar a conseguir su objetivo, y representa una vertiente finalista de la competición por el control en ese nivel. Hay cierta predisposición a que esos esquemas despiertos, y

no otros, tomen el control.

En cada iteración del hijo, si el padre ha decidido dormirlo (por ejemplo porque él mismo ha perdido la competición por el control en su nivel), entonces el hijo se duerme a si mismo en su variable de condición. Si por el contrario el padre no ha desactivado a sus hijos, cada hijo comprobará el estado de sus precondiciones. Como vimos en el capítulo 3 cada esquema de actuación tiene sus propias precondiciones, que miden el grado en que es aplicable a la situación actual. En esa función `preconditions()` de la tabla 4.10 es donde se implementa la consulta a los estímulos externos o internos y su adición siguiendo la suma heterogénea de estímulos presentada en el capítulo 3. Si la activación resultante supera cierto umbral entonces el esquema promociona de ALERTA a PREPARADO. De algún modo ese nivel de activación continua representa la motivación a disparar este esquema, que ahora lleva la impronta de la situación del entorno. Los estímulos del entorno hacen subir la motivación de los esquemas de actuación adecuados a la situación actual.

Si las precondiciones del hijo se satisfacen, entonces él pasa a estado PREPARADO y chequea el estado de activación de todos sus hermanos, buscando alguno que también esté PREPARADO o ACTIVO. Si no lo encuentra entonces es el único hijo PREPARADO en ese instante, así que promociona a ACTIVO y realmente ejecuta su función de iteración genuina. Si encuentra otros hermanos con intenciones de tomar el control entonces ha detectado un *solape de control* e invoca a la función de arbitraje explícito que le dio el padre. La función de arbitraje le mantendrá su estado en PREPARADO si no resulta elegido ganador. O bien modificará su propio estado a ACTIVO si él resulta elegido ganador, en cuyo caso habrá que ejecutar la función de iteración.

Si las precondiciones del hijo no se satisfacen, entonces chequea el estado de activación de todos sus hermanos, buscando alguno que sí esté PREPARADO o ACTIVO. Si lo encuentra entonces este hijo no hace nada en esta iteración, pero si no lo halla, entonces ha detectado un *vacío de control*, e invoca también a la función de arbitraje explícito que le dio el padre. La función de arbitraje modificará su propio estado a ACTIVO si él resulta forzado a tomar el control, o lo mantendrá en ALERTA si no tiene que hacer nada.

La función de arbitraje explícito devuelve a aquél que la invoca si es el elegido para tomar el control o no, bien sea por obligación en el caso de vacíos de control, bien por selección cuando haya solapes de control. Dentro de esta función se comprueba el estado de activación de todos los hermanos y con ello se detecta si hay que resolver un solape o un vacío de control. Adicionalmente, el padre puede consultar todos los estímulos que haya activado con idea de poder resolver el conflicto, quizá atendiendo a alguna medida sensorial o el valor concreto de algún estímulo.

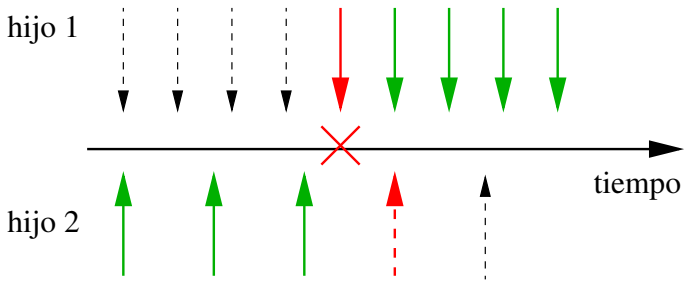


Figura 4.19: *Ejemplo de competición por el control con dos hijos*

En la figura 4.19 se muestra una posible evolución temporal de la competición por el control entre dos esquemas hermanos. Siguiendo el esqueleto de la tabla 4.10, cada uno de ellos comprueba periódicamente si se dan sus precondiciones llamando a su función `preconditions`, y el estado de su hermano. Inicialmente, el hijo 2 es el que gana el control (indicamos con flechas continuas el esquema que gana el control), y el hijo 1 permanece en estado ALERTA (flechas discontinuas). En cierto instante, señalado con una cruz roja, las condiciones del entorno cambian, y las precondiciones del hijo 1 se satisfacen. Al descubrir que su hermano está en ACTIVO invoca a la función de arbitraje, la cual le indica que él es el hijo conveniente. Por lo tanto, gana la competición y pasa a estado ACTIVO. En su siguiente iteración, el hermano 2 detecta la colisión por el control e invoca también a la función de arbitraje, que le indica que él no es el ganador. Por lo tanto, pierde la competición por el control y

pasa a estado PREPARADO. En la figura se señalan con flechas rojas los instantes en los que el estado de los esquemas varía.

El algoritmo de selección de acción implementado es distribuido, y se simplifica por el hecho de que cada hijo puede comprobar en exclusión mutua el estado de activación de sus hermanos. Cada vez que un hijo ejecuta una iteración suya se está evaluando si el arbitraje es el correcto en ese nivel. Esto responde a la pregunta de cúando se concreta el mecanismo de selección de acción. El algoritmo utiliza a los hijos como detectores de solapes y vacíos de control. El padre no ejecuta ningún código de arbitraje, realmente lo ejecutan los hijos, cualquiera de ellos, cuando detectan algún problema, aunque el código esté escrito en el fichero que contiene al esquema padre. De este modo, el código de arbitraje explícito queda centralizado, muy compacto y ordenado, porque es el padre el que tiene el contexto y la información relevante para resolver el conflicto. Pero su ejecución está distribuida entre todos sus hijos.

Las precondiciones configuran la región de activación del esquema y normalmente estarán moduladas por el padre para que siempre se active un hijo y sólo uno a la vez. En el caso normal ocurre literalmente eso y la competición se resuelve directamente, de modo muy eficiente, sin necesidad de invocar a la función de arbitraje explícito.

4.3.3. Control borroso en los esquemas de actuación

Además del mecanismo de selección de acción distribuida, la implementación en hebras de los esquemas y los esqueletos típicos para esquemas perceptivos y de actuación, se ha desarrollado una herramienta para facilitar la implementación de los esquemas de actuación como controladores borrosos. Se trata de una librería en C que permite definir completamente los controladores borrosos en un fichero escrito con una sintáxis sencilla, fácil de entender y modificar. Como vimos en la sección 2.3.3, con los controladores borrosos se pueden implementar muy rápidamente controles reactivos donde hay relaciones no lineales entre entradas y salidas. Utilizando la librería desarrollada, su ajuste además es muy sencillo, no hay más que retocar en el fichero los valores de las etiquetas borrosas y de las reglas.

```
label error(t) NLarge = -80 -80 -40 -10
label error(t) NSmall = -40 -20 -10 0
label error(t) Zero = -10 0 0 10
label error(t) PSmall = 0 10 20 40
label error(t) PLarge = 20 40 80 80
```

Tabla 4.11: *Conjuntos borrosos sobre la variable* `error(t)`*, tal y como se definen en el fichero de configuración*

Las variables usadas por el controlador, con sus nombres y los valores borrosos aceptados se especifican en el fichero de texto, tanto si son variables de entrada como si son de salida. Las etiquetas quedan especificadas con cuatro valores numéricos sobre el rango de la variable continua, que determinan la forma de la función de pertenencia. La librería soporta etiquetas con forma trapezoidal (a,b,c,d), rectangular (a,a,b,b), triangular (a,b,b,c; a,a,a,b), y *singleton* o impulso (a,a,a,a).

La tabla 4.11 muestra un ejemplo con cinco etiquetas borrosas `NLarge, NSmall, Zero, PSmall` y `PLarga` que se presentan como posibles valores de la variable `error(t)`. En esa tabla el identificador `label` es una palabra clave para la librería, `error(t)` es el nombre de una variable y `NLarge` es el nombre de una etiqueta borrosa. `error(t-1)` es otra variable diferente, que el diseñador se encarga de mantener actualizada correctamente.

El fichero también contiene la base de reglas del controlador, expresada como una lista no ordenada de reglas en la forma IF-THEN, con el operador AND en los antecedentes. La tabla 4.12 es un ejemplo. La implementación del operador AND es un simple producto de los grados de pertenencia de cada uno de los operandos. De este modo $x = A$ AND $y = B$ tiene el valor de verdad $A(x) * B(y)$. El mecanismo de inferencia elegido es truncar el trapezoide del consecuente a la altura proporcional al valor de verdad del antecedente completo, tal y como ilustra la figura 4.20.

Cuando varias reglas son de aplicación sus salidas forman un conjunto borroso de salida, cuya función de pertenencia toma en cada punto el máximo de las salidas de cada regla. La salida numérica

```
IF ( error(t) = Zero ) THEN ( pwm = VeryNarrow )
IF ( error(t) = NLarge ) AND ( error(t-1) = NLarge ) THEN ( pwm = VeryWide )
IF ( error(t) = NLarge ) AND ( error(t-1) = NSmall ) THEN ( pwm = Wide )
IF ( error(t) = NLarge ) AND ( error(t-1) = Zero ) THEN ( pwm = Wide )
IF ( error(t) = NLarge ) AND ( error(t-1) = PSmall ) THEN ( pwm = Wide )
IF ( error(t) = NLarge ) AND ( error(t-1) = PLarge ) THEN ( pwm = Wide )
```

Tabla 4.12: *Ejemplo de reglas de control tal y como aparecen en el fichero de configuración.*

final se calcula *desborrosificando* ese conjunto siguiendo la técnica del centro de masas. En el caso concreto de la figura 4.20 hay tres reglas activas, la primera sólo tiene un antecedente y su consecuente se aplica en escasa medida. La segunda regla tiene dos antecedentes y es relevante de modo notable. Su consecuente sesgará en gran medida la salida final. En la esquina inferior derecha de la figura 4.20 se puede apreciar el conjunto borroso final de salida y en rojo el valor concreto resultante.

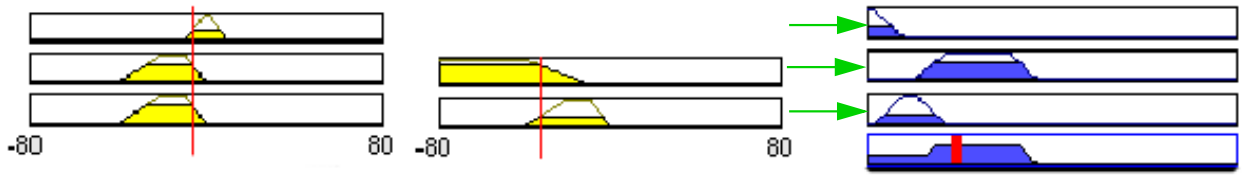


Figura 4.20: *Reglas borrosas*

La librería desarrollada, `fuzzylib`, ofrece la función `fc_open(char *filename)` para compilar en sus estructuras de programa el controlador descrito en el fichero `filename`. Las reglas y variables aparecen precedidas por una línea como `controlador seguimiento`, donde `controlador` es una palabra clave y `seguimiento` es el nombre concreto del controlador que se especifica a continuación en el fichero.

A la hora de utilizar los controladores definidos, el código que es usuario de la librería borrosa tiene que enlazar las variables simbólicas del fichero con sus variables correspondientes de programa. Tanto si son de entrada como si son de salida. Para ello la librería ofrece la función `fc_link(int c, char *varname, float *varpointer)`, donde el `varname` es el nombre de la variable en el fichero y `varpointer` es un puntero a la variable en el programa que va a enlazarse con la variable simbólica del fichero. Al estar enlazadas no hay ninguna primitiva para refrescar explícitamente las variables del controlador. Éste consultará la variable enlazada del programa, que otro esquema se encargará de mantener actualizada.

Cuando interesa saber el valor que el controlador borroso recomienda para cierta variable de control, se invoca a la función `fc_output(int c, char *varname, float *output)`, donde `varname` es el nombre de la variable en el fichero y `output` el puntero de la zona de memoria donde el controlador dejará su recomendación.

Con el código descrito en este capítulo proporcionamos la infraestructura necesaria (esquemas como hebras, funciones para la selección de acción, esqueletos típicos, etc.) para llevar a cabo los experimentos que se detallan en el siguiente capítulo.

Capítulo 5

Experimentos

> *One of the hard lessons we have learned over the years is that writing programs for robots is not an easy thing to do*
> Ronald C. Arkin, 2003

Una vez descrita la arquitectura conceptual y cómo hemos implementado sus mecanismos teóricos de selección de acción o el esqueleto de programación para los esquemas, en este capítulo describiremos el contenido concreto de los esquemas desarrollados. En el caso de los esquemas perceptivos, presentaremos las técnicas particulares empleadas para percibir y caracterizar ciertos estímulos, haciendo hincapié en los métodos de fusión sensorial. En el caso de los esquemas de actuación, describiremos los sistemas específicos de control empleados para conseguir determinado comportamiento. A través de varios ejemplos veremos cómo se combinan todos los esquemas en un sistema conjunto.

Describiremos técnicas concretas de estimación, de fusión sensorial, de control, etc, que nada tienen que ver con la arquitectura. Todas ellas se encuadran dentro de algún esquema y su coexistencia en el mismo robot se solventa precisamente empleando el marco de la arquitectura propuesta. Por ejemplo, la arquitectura resuelve cuándo entra en funcionamiento cierta técnica: si es perceptiva porque su esquema ha sido activado, o si es de actuación porque su esquema ha ganado la competición de control en su nivel. Como esos aspectos arquitectónicos ya han sido largamente comentados, aquí discutiremos la conveniencia de las técnicas perceptivas o de control particulares per se, juzgándolas por su eficiencia, su exactitud, etc.

El conjunto de esquemas programado ha permitido exhibir un repertorio variado de comportamientos en diferentes plataformas. Por supuesto, es un conjunto ampliable con nuevos esquemas y nuevos comportamientos, pero ya supone un corpus significativo. Ha sido necesario desarrollarlos para tener un robot real funcionando y exhibiendo comportamientos reales concretos. En primer lugar, suponen una validación experimental de la arquitectura propuesta. En segundo lugar, sirven de ejemplo pues todos ellos satisfacen la interfaz que especifica JDE y dan cuerpo a la filosofía que ésta propone para generar comportamiento y a los matices que contempla. Por ejemplo, en percepción los esquemas tratan de conseguir una descripción rica y dinámica del entorno local, que se articula como una colección ampliable de estímulos. En actuación, los esquemas de actuación reciben realimentación de los esquemas perceptivos a través de variables comunes. Las conductas complejas se implementan como la activación temporal y modulación de otras conductas más sencillas, también a través de variables comunes.

El capítulo se articula en dos grandes ramas: la primera describe los esquemas perceptivos más complejos que hemos elaborado; la segunda pormenoriza los esquemas de actuación programados y los comportamientos conseguidos utilizando los estímulos descritos anteriormente y otros más simples. En la primera parte hemos destacado los esquemas dedicados a percibir los obstáculos cercanos y los que sirven para identificar puertas en el entorno. En la segunda parte explicamos cómo hemos implementado con esquemas ciertos comportamientos sencillos como seguimiento o navegación local. Por ejemplo, las conductas `sigue-pelota`, `sigue-pared` y `avanza-rápido-sorteando-obstáculos`. También describimos otras conductas más complejas donde ha sido necesario utilizar la jerarquía de JDE, como la de `ir-a-punto`, el `saque-de-banda` y el sistema de comportamientos de un delantero

robótico para la Robocup.

5.1. Ocupación del entorno local

Una de las necesidades de los robots móviles es representar el entorno en que se mueven, en concreto representar la existencia de obstáculos con los cuales podría chocar en su movimiento por el mundo. Estos serán parte de los estímulos relevantes a la hora de tomar decisiones locales de movimiento. Por ello abordamos la construcción de una representación dinámica del estado de ocupación actual de los alrededores del robot. Al enmarcarse en JDE, un conjunto de esquemas perceptivos serán responsables de construir y actualizar esa representación. En concreto los dos mostrados en el diagrama de la figura 5.1. El primero mantiene una rejilla de ocupación local al robot, reflejando la posición dinámica de los obstáculos a su alrededor, y el segundo agrupa las celdillas vecinas ocupadas identificando segmentos en la rejilla, con el objetivo de tener una representación más compacta.

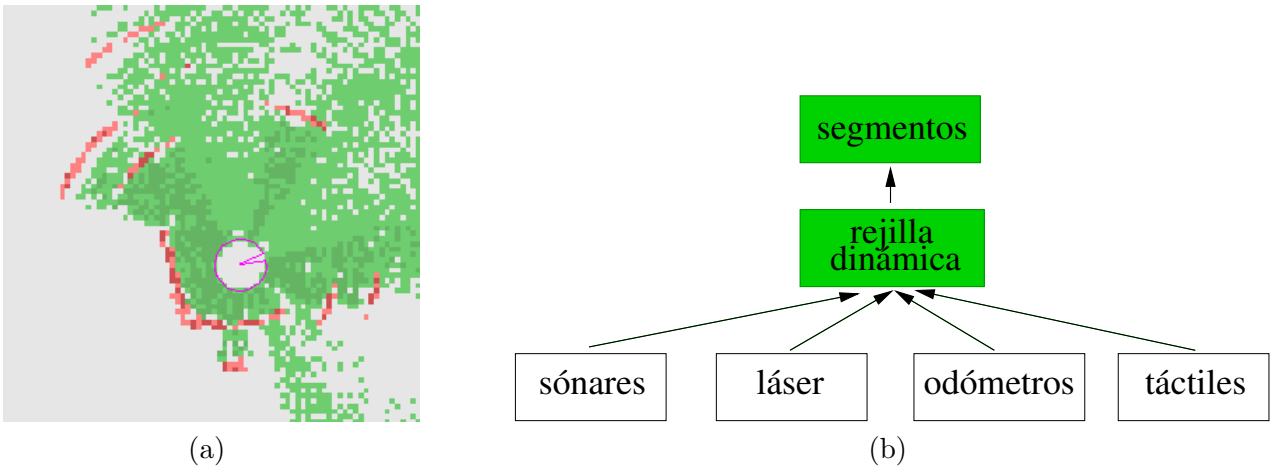


Figura 5.1: *Ejemplo de rejilla de ocupación (a) y esquemas perceptivos para estimar la ocupación del entorno local (b)*

La mayoría de los trabajos en representación del entorno para robots móviles dividen la información en dos partes que se tratan separadamente. Primero, un *mapa global* refleja los obstáculos estáticos como paredes, armarios, etc. y permite planificación de largo plazo. Segundo, una representación instantánea, básicamente la última lectura sensorial, que permite reaccionar a obstáculos imprevistos. En el caso de osbtáculos móviles, únicamente esta segunda representación instantánea contiene datos frescos. Como argumentaremos al final de esta sección, en JDE hemos preferido manejar una única representación, reducir su alcance al ámbito local del robot en cada momento, y conseguir que refleje los cambios temporales que puedan ocurrir. Este carácter local contrasta con la típica construcción de mapas que tiene alcance global. Dentro del planteamiento de JDE no abordamos la memoria a largo plazo, y por ello los mapas globales quedan fuera de nuestro estudio. Además, plantear de aquel modo dual la representación del espacio implica resolver el problema de localización, pues para utilizar un mapa global es necesario saber dónde se encuentra el robot dentro de él.

Otro matiz diferente y muy importante es la dinamicidad de la representación. Mientras que en los mapas tradicionales interesa identificar los obstáculos fijos del entorno, en la representación local que aquí perseguimos se pretende reflejar el dinamismo de los objetos móviles alrededor del robot. Si los obstáculos se mueven, la representación ha de reflejarlo a la mayor brevedad, para que las decisiones de movimiento puedan ser más acertadas. Es importante distinguir esta diferencia de requisitos, porque como veremos posteriormente, afecta al modo de construir esa representación.

En cuanto al formato de esa representación, hay dos muy difundidos en la literatura, los mapas topológicos y los mapas métricos. Los mapas topológicos guardan la conectividad entre lugares y en ellos no se puede estimar distancias o ángulos precisos. Por ejemplo, un grafo acíclico con arcos y nodos es un mapa topológico, los nodos son lugares relevantes y los arcos pasajes entre ellos. Los mapas métricos necesitan un sistema de coordenadas y en ellos sí se pueden inferir distancias y ángulos.

Dentro de los mapas métricos se puede distinguir dos paradigmas, el modelo de elementos geométricos [Leonard *et al.*, 1992] y el de rejillas de ocupación [Elfes, 1990][Moravec, 1989]. En el primero se tienen unas primitivas de representación (puntos, esquinas, paredes, objetos, etc.) cuya posición se estima constantemente desde la información sensorial. Hemos preferido las rejillas de ocupación, que no necesitan estructura en el entorno para conseguir una representación adecuada. La rejilla representa el espacio como un mallado regular de celdillas, cada una de las cuales contiene la creencia en que esa posición en el mundo esté ocupada o no. En la figura 5.1(a) se puede observar un ejemplo de una rejilla alrededor del robot, las casillas oscuras indican la presencia de un obstáculo en ellas, y las claras de espacio vacío.

Teniendo en cuenta el requisito de localidad, la rejilla de ocupación tiene un tamaño limitado, lo que la hace más manejable que si representara grandes superficies. Por ejemplo, la rejilla de la figura 5.1(a) ocupa 2x2 metros alrededor del robot y discretiza el espacio en celdillas de 5x5 cm. Además, es una estructura móvil: está siempre centrada en el robot y cambia su posición con el movimiento de éste, pues son sus alrededores los que interesa caracterizar. En cuanto al dinamismo, la regla de actualización de la rejilla debe capturar rápidamente el cambio en la realidad.

El problema de la construcción y el mantenimiento de rejillas de ocupación ha sido ampliamente abordado en la literatura, aunque prácticamente siempre con el objetivo de construir mapas estáticos. En general, este problema se ha dividido en dos etapas que se ejecutan cíclicamente. Primero, se captura toda la información que proporciona una nueva lectura del sensor sobre la ocupación del espacio, siguiendo determinado *modelo sensorial*. Segundo, esa información se utiliza para actualizar la creencia acumulada, siguiendo cierta *regla de actualización* y materializando con ella la fusión con otras medidas anteriores.

En nuestos experimentos se ha programado el esquema `rejilla-de-ocupación` que es responsable de construir y mantener actualizada la rejilla. Éste esquema chequea en cada una de sus iteraciones si hay datos nuevos de los sensores de proximidad como láser, sónares y táctiles. Si los hay, incorpora a la rejilla toda la información contenida en esas medidas, siguiendo el modelo sensorial correspondiente y aplicando la regla de actualización elegida. Además, comprueba las posiciones del robot y de la rejilla, y caso de haberse descentrado mucho, desplaza la rejilla para centrarla en la ubicación actual del robot. El tiempo de ciclo se ha ajustado a 200 ms en los experimentos. La rejilla se ha materializado como una matriz bidimensional de celdillas $C_{(x,y)}$, cada una de las cuales tiene asociado un *estado de ocupación*. Ese estado es una variable gradual, continua, que oscila entre un valor máximo E_{max} para señalar ocupación y un valor mínimo $E_{min} = -E_{max}$ para señalar certeza de vacío. Esta matriz es una variable común que otros esquemas consultarán para tomar decisiones de movimiento.

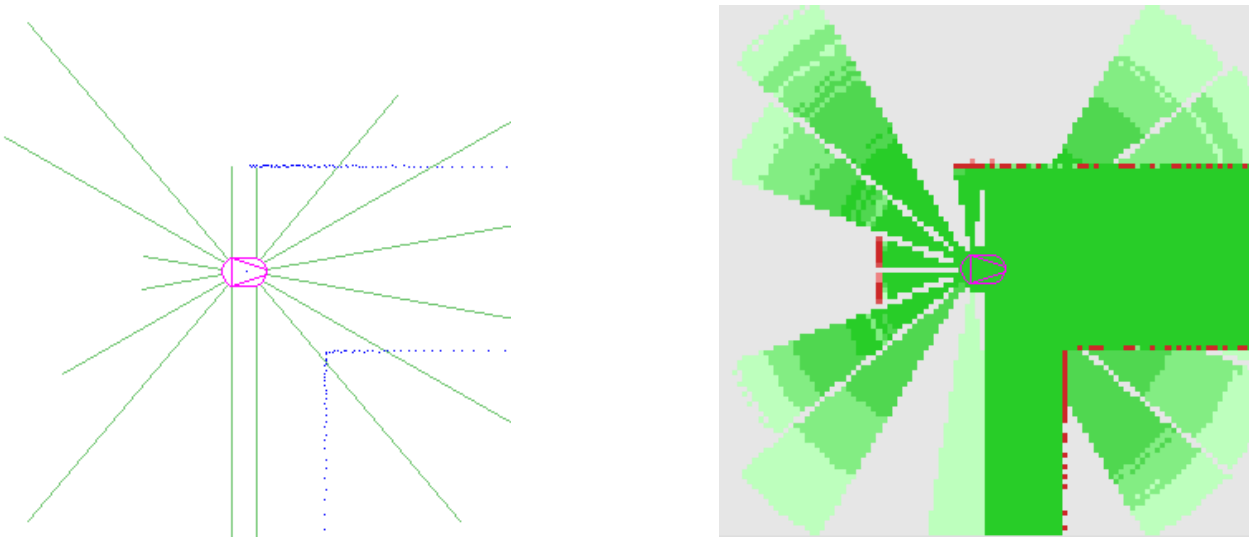
Este planteamiento hace que se muestree la información sensorial, pues no se incorpora al mismo ritmo al que se recibe, sino al ritmo del esquema `rejilla-de-ocupación`. Por ejemplo, se obtiene una lectura completa del láser cada 100 ms, pero sólo cada 200 ms se incorpora una a la rejilla, la última. Este desacople de ritmos respeta la independencia entre los esquemas, y no es perjudicial si el esquema de la rejilla tiene una frecuencia elevada. Para evitar incorporar dos veces la misma lectura sensorial, éstas tienen sellos temporales. Con objeto de tener siempre la rejilla a la última también se han implementado mecanismos de *callbacks* que permiten incorporar todas y cada una de las lecturas justo cuando se reciben. En la práctica se utilizan con los odómetros y los sónares, pero no con el láser, pues su incorporación es computacionalmente costosa y saturaría al sistema.

Para ajustar y optimizar la calidad de la rejilla hemos desarrollado la librería `libgrids`. Esta librería contiene distintos modelos sensoriales para diferentes sensores y varias reglas de actualización con las que hemos experimentado. Por ejemplo, contiene modelos sensoriales para sónares con distintas geometrías, modelos para sensor láser, etc.. En cuanto a reglas de actualización, contiene las más utilizadas en la literatura (probabilística, borrosa, basada en teoría de la evidencia e histográmica) y dos técnicas particulares que describimos a continuación.

5.1.1. Modelos sensoriales

Para construir la rejilla utilizamos información sensorial procedente de los sensores de proximidad del robot: sónares, láser y táctiles. Todos ellos proporcionan información sobre los obstáculos, la cual

se combina en la rejilla para estimar sus posiciones relativas. Las técnicas de actualización de la rejilla son literalmente las mismas para todos. Las peculiaridades de cada uno (por ejemplo su fiabilidad y alcance) quedan reflejadas en el modelo sensorial, tanto en su geometría como en la magnitud de los valores que emplea. Adicionalmente, para posicionar en el espacio las evidencias también se emplean los datos de los odómetros.



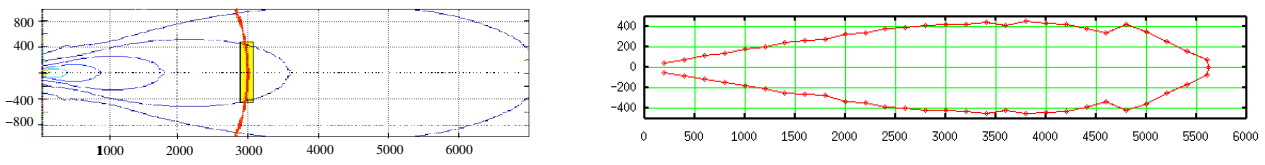
(a) (b)

Figura 5.2: *Instantáneas láser y sónar (a) y rejilla de ocupación obtenida con la fusión de ambos (b)*

Nos hemos centrado en los sónares porque son sensores muy difundidos en la comunidad robótica, y porque en ellos la necesidad de fusionar evidencias es más palpable debido a sus incertidumbres asociadas. La incorporación del láser a la plataforma real fue mucho más reciente.

Modelo sónar

Como ya presentamos en el capitulo 4, el sensor sónar mide la distancia al objeto más próximo utilizando el eco de una onda ultrasónica. El dispositivo puede tener varios transductores ultrasónicos, pero es muy común el uso del mismo transductor como emisor y como receptor, operando en modo pulso-eco. En este modo de funcionamiento el transductor emite una onda ultrasónica e inicia cierto temporizador. La propagación de la onda ultrasónica es anisótropa, la energía emitida se propaga en varios lóbulos desde el transductor, concentrándose en la zona central según muestra el patrón teórico en la figura 5.3(a). Pasado cierto intervalo (*intervalo de blanqueo*) se atiende a la energía que se recibe en el transductor. Cuando la energía ultrasónica que llega supera cierto *umbral de recepción* se detiene el temporizador y se traduce el tiempo transcurrido a distancia del obstáculo, considerando la velocidad de los ultrasonidos en el medio. Esa onda recibida corresponde al reflejo en el obstáculo más cercano de la onda emitida.



(a) (b)

Figura 5.3: *Difusión teórica energía ultrasónica (a) y modelo experimental (b)*

Comprender bien la física del sensor ayuda a plantear modelos sensoriales ajustados que extraigan el máximo de información de sus medidas. Compartimos con D.Pagac et al [Pagac *et al.*, 1998] la

creencia en que la evidencia de ocupación debe distribuirse uniformemente dentro del arco del cono ultrasónico, y no sólo en su punto central. Aunque en la zona central del lóbulo de emisión haya más energía que en las laterales, es imposible discernir qué partes del arco tienen más probabilidad de haber generado el eco, pues la reflexión en cualquier punto del arco del cono supera el umbral. Si el obstáculo está en la parte central la energía reflejada será mucho mayor que la reflejada si el obstáculo no está centrado, sin embargo esa es una información de la que no disponemos. Utilizamos exclusivamente el instante en el cual la energía reflejada supera el umbral, no la magnitud de la energía recibida. Por lo tanto, la evidencia de obstáculo debe repartirse uniformemente con el ángulo, y no sesgando hacia las partes centrales.

Las principales incertidumbres de los sensores reales de ultrasonidos son el error radial, la incertidumbre angular y las reflexiones especulares [Gasós y Martín, 1996]. El *error radial* suele ser pequeño (±2cm) y está relacionado con que la velocidad real de la onda no es exactamente la misma que la utilizada en la estimación de distancia. Como ya hemos comentado, la *incertidumbre angular* se debe a que la onda se abre (aproximadamente unos 15°) y es imposible discernir qué punto dentro de ese frente de onda provocó realmente el eco. Las *reflexiones especulares* se producen cuando la onda en vez de rebotar en el obstáculo hacia el emisor, se refleja en otro ángulo diferente, llegando tras múltiples rebotes al sensor. Esto provoca que se estimen distancias mayores de las reales.

El patrón experimental medido para los sensores del robot B21 aparece en la figura 5.3(b), y el obtenido para el robot Pioneer se mostró en la figura 4.2(b). En vez de emplear simplificaciones con geometría cónica o axial, hemos utilizado una tabla que captura la forma de cada lóbulo experimental, para ajustar perfectamente a la realidad el modelo sensorial. En concreto, el modelo elegido asigna un peso positivo ($\Delta_{obs}(t) = +1$) a las celdillas situadas en el arco de ocupación y un peso negativo ($\Delta_{obs}(t) = -1$) a las situadas en el interior del lóbulo para indicar evidencia de vacío. Cuanto mayor es el valor absoluto mayor es la influencia de la medida en esa celdilla. Además, descartamos las medidas más lejanas que cierta *distancia umbral*, pues esas lecturas lejanas muy probablemente proceden de reflexiones múltiples más que de observaciones directas. Éste valor está por debajo del alcance máximo medido en el patrón experimental.

Salvando las reflexiones especulares, la información de vacío de una lectura sensorial es una información concluyente: en el interior del lóbulo hay espacio vacío, porque si hubiera un objeto la medida habría sido otra. En contraste, la información de ocupación de un sónar es un mero indicio, debido precisamente a la incertidumbre angular.

Modelos del láser

El sensor láser de proximidad es un dispositivo que utiliza un haz de luz láser para medir distancia a obstáculos. El principio físico es el mismo que el del sensor sónar, medir el tiempo de vuelo entre la emisión y la recepción del eco, en este caso de la onda luminosa. Debido a la naturaleza del rayo de luz y a su menor dispersión, el sensor láser estima de modo mucho más preciso la distancia, sin incertidumbre angular. De hecho, en la figura 5.2(a) se puede apreciar cómo el láser refleja de modo mucho más preciso que el sónar el contorno de la esquina y el pasillo en los que se encuentra el robot.

Para lograr un barrido completo el haz láser se desvía con un espejo rotatorio de manera que barra todo un arco de unos 180 grados en horizontal. La lectura completa proporciona 180 medidas, una cada grado. El modelo utilizado tiene la geometría de un sector circular, pero de una apertura muy pequeña, apenas 1 grado [García Pérez, 2004]. Los valores numéricos del modelo dependen de la regla de actualización elegida, pero son más altos que los asociados a la lectura sónar, para reflejar la mayor fiabilidad del láser.

Sensores de contacto

Consiste en dos chapas que normalmente están separadas, pero que entran en contacto cuando un obstáculo empuja una de ellas al chocar con el robot. Al estar en contacto permiten el paso entre ellas de la corriente eléctrica. Este sensor es muy fiable, es muy dificil que haya falsos positivos o falsos negativos. El modelo sensorial utilizado para los táctiles sólo afecta a la celdilla en la que se sitúa el sensor y el valor asociado es extremo, precisamente por su fiabilidad. Esa certeza de ocupación

sólo se tiene en el instante del choque, pero no para siempre. Tal y como veremos con las reglas de actualización, con el paso del tiempo la creencia en ocupación debe disminuir si no hay nuevas evidencias.

Sensores de odometría

Otro de los sensores utilizados en la actualización de la rejilla son los odómetros. El robot se está moviendo continuamente y por ello es necesario ubicar espacialmente las información de ocupación de cada una de las medidas sensoriales. Para ello utilizamos la información de posición que ofrecen los odómetros del robot. Las coordenadas bidimensionales de los obstáculos detectados con los sónares, el láser o los táctiles, se expresan en el mismo marco de referencia que la posición del robot tomada desde los odómetros. Como vimos en el capitulo 4, la posición estimada por los cuenta vueltas adolece de errores que se acumulan en el tiempo, principalmente debido a desalineamientos y deslizamientos. Sin embargo, este error respecto de un sistema de referencia absoluto es irrelevante para la rejilla de ocupación. Como está siempre centrada en la posición actual del robot, la rejilla almacena la posición relativa de los obstáculos respecto del robot [Elfes, 1990], aunque cada celdilla tenga además su posición absoluta. Por ejemplo, todos los obstáculos detectados con la lectura actual del sónar y la posición presente del robot están afectados por el mismo error absoluto, de modo que no hay una deriva relativa relevante entre ellos (siempre que la memoria de la rejilla no tenga demasiada persistencia). En cierto sentido, es análogo a utilizar un reloj que se atrasa 5 minutos al día para cronometrar una carrera, con varios participantes, de 200 metros lisos. Aunque la marca no estaría homologada, sirve perfectamente para determinar el orden de llegada y estimar las diferencias entre todos los corredores.

Para mantener la rejilla más o menos centrada en la posición actual del robot, el esquema `rejilla--de-ocupación` comprueba en cada iteración si el autómata se ha salido de la zona central de la rejilla. Si lo ha hecho, entonces fuerza un desplazamiento de la rejilla a la nueva localización del robot, sin perder la información de las zonas que solapen entre la antigua y la nueva ubicación.

5.1.2. Regla de actualización

Cada vez que se recoge una nueva lectura sensorial, el estado de las celdillas afectadas en la rejilla se recalcula utilizando la regla de actualización. Hay muchas alternativas en la literatura a la hora de integrar la información ya existente con la aportada por la última lectura. Son muy conocidos los enfoques probabilístico [Thrun *et al.*, 1998], de teoría de la evidencia [Pagac *et al.*, 1998], de conjuntos borrosos [Oriolo *et al.*, 1998] o histográmico [Borenstein y Koren, 1991a]. Los tres primeros han ofrecido muy buenos resultados para la construcción de mapas globales estáticos, y por ello se estudió su posible aplicación para el mantenimiento de la rejilla dinámica de ocupación. Ese estudio comparativo lo enmarcamos dentro del anexo A donde evaluamos las características de esas técnicas de fusión sensorial y su comportamiento en rejillas dinámicas.

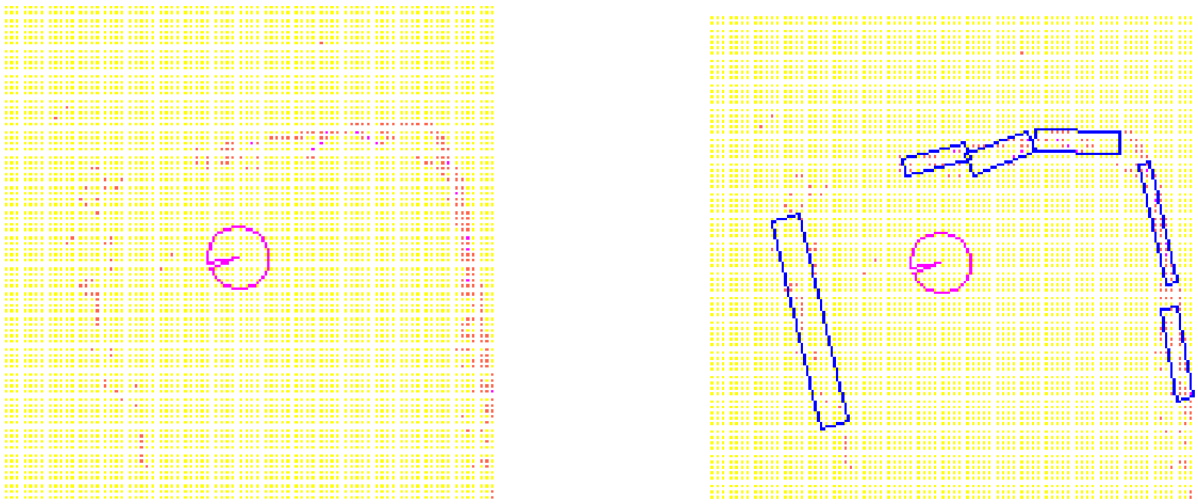
En esta sección simplemente presentamos las dos reglas de actualización propias, desarrolladas con la finalidad de preservar el dinamismo en la representación de la ocupación. Para ambos el modelo sensorial utiliza valores $\Delta_{obs}(t)$ acotados entre $[-1,+1]$, positivo para indicios de ocupación y negativo para evidencias de vacío. Valores próximos a los extremos señalan indicios contudentes, fiables, y valores cercanos a cero reflejan indicios endebles, suaves.

Actualización con ecuación diferencial

Con este enfoque se actualiza el estado de ocupación de cada celdilla siguiendo la ecuación diferencial (5.1), que tiene sus raíces en la propuesta de Hans Moravec [Moravec, 1989]. El cambio en la creencia de ocupación será un incremento o decremento dependiendo del signo de $\Delta_{obs}(t)$. La amplitud del cambio depende de varios factores.

$$creencia(C_{(x,y)},t) = creencia(C_{(x,y)},t-1) + \Delta_{obs}(t) * saturacion(t) * secuencia(t) * speed \quad (5.1)$$

$$saturacion = \begin{cases} \Delta_{obs}(t) > 0 & |E_{max} - creencia(x,y,t-1)| \\ \Delta_{obs}(t) < 0 & |E_{min} - creencia(x,y,t-1)| \end{cases} \quad (5.2)$$



(a) (b)

Figura 5.4: *Rejilla de ocupación (a) y segmentos conseguidos sobre ella (b)*

El *factor saturación* (5.2) acota el valor del incremento de tal modo que nunca se pasen los valores máximo y mínimo para la creencia. Este factor hace que afectando varias medidas con el mismo peso a la celdilla, las novedosas tengan más influencia, provoquen más cambio en la creencia. Esto permite en la práctica cambios de opinión muy rápidos, para reflejar el posible movimiento de obstáculos. Además, cuando se está muy seguro de la ocupación de la celdilla, nuevas observaciones en este sentido apenas aportan información.

El parámetro *speed*, entre 0 y 1, es el mismo para todas las celdillas y modula la velocidad de cambio de estado dentro de la rejilla. Con él se parametriza el número de medidas necesarias para cambiar totalmente de creencia, que es menor cuanto mayor es *speed*.

Finalmente, el *factor secuencia*, entre 0 y 1, refuerza el efecto de las medidas cuando éstas aparecen seguidas, de tal manera que las medidas aisladas quedan aminoradas. Su valor se encuentra entre 0 y 1, y se calcula sobre una pequeña memoria de evidencias asociada a la celdilla. Si la última evidencia aparece en una secuencia de evidencias del mismo signo su influencia será mayor que si las anteriores son de signo contrario. En cierto modo este factor retarda el efecto de las sorpresas hasta que se van confirmando con una secuencia de lecturas en el mismo sentido.

Por su propia naturaleza, la ecuación diferencial propuesta ofrece un alto dinamismo en la creencia y logra que las medidas recientes influyan en el estado de ocupación sistemáticamente más que las antiguas. Adicionalmente, hemos incluido un *mecanismo de olvido* que periódicamente (1 segundo) multiplica la creencia de todas las celdillas por un *factor de olvido* (en los experimentos 0.98). Este mecanismo acerca iterativamente la creencia de ocupación de todas las celdillas al estado de desconocimiento, es decir $creencia(C_{(x,y)}, t) \simeq 0$, y fuerza a que éstas se refresquen constantemente con nuevas observaciones.

Decisión por mayoría

Con este enfoque, en cada celdilla $C_{(x,y)}$ se almacena con orden temporal la información que aportan las últimas N medidas que afectan a la celdilla: $\Delta_{obs}(t-1), \Delta_{obs}(t-2), \ldots \Delta_{obs}(t-N)$. Sumando los valores en memoria tenemos el peso acumulado. El peso acumulado $\sum_{i=1}^{N} peso(i)$ oscila entre -N y +N. Con este peso acumulado se estima la ocupación de la celdilla siguiendo la función de la figura 5.5.

En el eje vertical se ve el estado de ocupación final y en las abscisas el peso acumulado en la memoria de la celdilla. El *umbral de ruido* de la figura 5.5 señala la cantidad mínima de evidencias necesarias para empezar a creer que la celdilla no está libre. Este umbral inferior filtra las medidas erróneas espúreas, pues se necesita más de una medida para confirmar que la celdilla puede estar ocupada. Las celdillas realmente ocupadas o vacías lo superan sin problemas acumulando enseguida evidencias que lo respaldan.

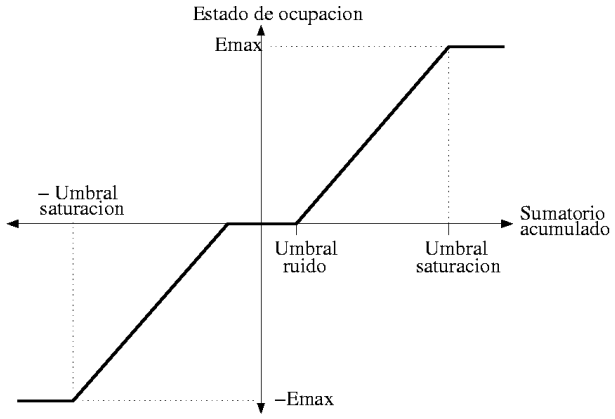


Figura 5.5: *Estado de ocupación en función de los pesos almacenados en la memoria de la celdilla*

A medida que se acumulan evidencias por encima de ese umbral, el estado de ocupación crece linealmente hasta alcanzar el valor máximo en el *umbral de saturación*. Este umbral superior introduce el fenomeno de saturación en la creencia. Esta saturación ecualiza las zonas del espacio por donde más y menos tiempo se ha movido el robot, con tal que las evidencias recogidas sean suficientes para concluir un estado u otro de la celdilla.

Sobre una misma celdilla pueden caer varias evidencias contradictorias, de ocupación y de vacío. Con este enfoque se observa claramente que unas lecturas compensan a otras. La idea aquí es que una lectura errónea no sesga la creencia frente a una mayoría de lecturas correctas. Para modular esa compensación, el enfoque permite asignar distintos pesos a evidencias de ocupación, lejanas, cercanas, de vacío, etc.

La inserción de observaciones nuevas refresca el contenido de la memoria local, con lo cual la creencia de ocupación está siempre actualizada. Adicionalmente, hemos añadido un mecanismo artificial de olvido que periódicamente inserta observaciones neutras para eliminar observaciones antiguas. Gracias a esto la creencia acaba envejeciendo en ausencia de lecturas recientes.

La rejilla de la figura 5.4(a) se obtuvo con el esquema de la rejilla activo y en paralelo un esquema de actuación reactivo que comandaba al robot para seguir paredes desde los datos sensoriales instantáneos. En ese experimento sólo se emplearon sónares y odómetros, utilizando una distancia umbral de 70 cm, que resultaba un buen balance entre filtrar medidas sónares erróneas y detectar los obstáculos con suficiente antelación para evitarlos a la velocidad del robot (0.1 m/s y 15 grados/s).

Un efecto nocivo observado con esta regla de actualización es que no filtra la inserción de nuevas evidencias. Por ello, una evidencia fiable puede salir de la memoria porque llega otra más reciente, aunque menos fiable. En las rejillas calculadas con sónares esto tiene un efecto muy negativo, porque las últimas medidas que quedan en la memoria de una celdilla acerca de una zona son las tomadas cuando el robot se alejaba de ella, por lo tanto las más distantes y menos fiables para dar información sobre esa zona. En la práctica, esto provoca que se borren los perfiles de ocupación que se habían conseguido formar con evidencias cercanas.

5.1.3. Segmentos dinámicos de ocupación

En la sección anterior hemos descrito el esquema que percibe la rejilla dinámica de ocupación. En ésta, explicaremos un segundo esquema, que se encarga de conseguir una representación de la ocupación más compacta que la propia rejilla. Para ello agrupa conjuntamente en segmentos las celdillas que están ocupadas y son próximas entre sí. Este segundo esquema se denomina `segmentador`, y construye una representación de la ocupación no sólo más compacta, sino además más abstracta que la rejilla. Sobre ella se pueden utilizar planificadores de caminos clásicos, e incluso estimar velocidades de objetos [García Pérez, 2003], sin salirse del planteamiento local y dinámico de JDE. El conjunto de segmentos hereda el alcance espacial de la rejilla y añade un cierto retardo en la representación del entorno (precisamente el tiempo necesario para adaptar los segmentos a la nueva situación). Además, los

segmentos son dinámicos porque se basan en el estado actual de la rejilla. Si un obstáculo se mueve, se reflejará en la rejilla y con cierta latencia, en los segmentos.

El algoritmo utilizado para segmentar es la técnica iterativa de estimación EM (*Expectation--Maximization*). Este algoritmo ha sido utilizado con éxito en visión computacional para segmentar imágenes [López de Teruel y Ruiz, 1998], y también en la generación de mapas como una optimización iterativa [Thrun *et al.*, 1998]. Su naturaleza iterativa cuadra muy bien con el carácter periódico de los esquemas perceptivos que propugna JDE. Los detalles de la aplicación de este algoritmo para segmentar rejillas de ocupación se pueden encontrar en [Cañas y García-Alegre, 1998].

Esta método interpreta el estado de ocupación de cada celdilla de la rejilla como una probabilidad de ocupación, en concreto como la probabilidad *observada* de ocupación. Independientemente del esquema `segmentador`, el esquema `rejilla-de-ocupación` actualiza este valor desde las observaciones sensoriales. Para tratarlo como probabilidad se desplaza y normaliza el estado de ocupación, definido entre $[E_{min}, E_{max}]$, a valores entre $[0, 1]$. A lo largo y ancho de la rejilla se construye una función bidimensional densidad de probabilidad observada de ocupación alrededor de la posición del robot.

El algoritmo EM parte de un modelo teórico consistente en una combinación lineal de k clases, donde cada clase es una gaussiana bidimensional G_k que aparece multiplicada por un factor de peso P_k, que se denomina *probabilidad de clase*. La probabilidad *teórica* de ocupación que refleja una celdilla $C_i = \{x, y\}$ según la gaussiana G_k corresponde al valor de esa gaussiana bidimensional en esa posición, es decir, $p(C_i/G_k)$ que sigue la ecuación 5.4. La probabilidad *teórica* de ocupación global de la celdilla C_i en los alrededores del robot viene dada por la combinación lineal de las diferentes probabilidades condicionadas, según se expresa en la ecuación 5.3.

$$P_{occupacy}(C_i) = \sum_{k,N} P_k \; p(C_i/G_k) \tag{5.3}$$

$$p(C_i/G_k) = \frac{1}{2\pi} \frac{1}{\sqrt{\sigma_{xx,k}^2 \sigma_{yy,k}^2 (1 - \sigma_{xy,k}^2)}} \frac{1}{e^{\frac{1}{2(1-\sigma_{xy,k})} \left[(\frac{x - \mu_{x,k}}{\sigma_{xx}})^2 + (\frac{y - \mu_{y,k}}{\sigma_{yy}})^2 - 2\sigma_{xy,k} (\frac{x - \mu_{x,k}}{\sigma_{xx}})(\frac{y - \mu_{y,k}}{\sigma_{yy}}) \right]}} \tag{5.4}$$

En sucesivas iteraciones EM, el algoritmo trata de ajustar la distribución teórica generada con las gaussianas a la observada, modificando en pequeños pasos los parámetros de las gaussianas y las probabilidades de clase. En la práctica estas gaussianas pueden asimilarse a objetos, de manera que su media $\{\mu_x, \mu_y\}$ determina su posición, y su matriz de covarianza $\{\sigma_{xx}^2, \sigma_{xy}^2, \sigma_{yy}^2\}$ define su tamaño y orientación. De esta manera, al mismo tiempo que se adaptan las gaussianas a la distribución de ocupación observada se están ajustando los segmentos al contenido de la rejilla de ocupación. Es decir, con esa adaptación continua se están calculando la posición, tamaño y orientación correctas de los obstáculos.

El algoritmo comienza con una población aleatoria de gaussianas que se refinan iterativamente en una secuencia de pasos de Expectativa y Maximización. Con una imagen estática los pasos se iterarían hasta que el algoritmo convergiera y los parámetros no cambiaran significativamente en nuevas iteraciones. En el uso que le hemos dado aquí se tiene una imagen dinámica del entorno en lugar de una fija (precisamente la rejilla de ocupación), cuyos valores pueden cambiar si la realidad lo hace. De este modo, el esquema `segmentador` ejecuta periódicamente iteraciones de EM para adaptar la población de segmentos a la población de la rejilla y para mantener el conjunto de gaussianas en consonancia con las medidas sensoriales recientes. Para un buen funcionamiento, la velocidad del algoritmo EM en perfilar las gaussianas debe ser superior a la velocidad de los objetos móviles del entorno.

En el paso de Expectativa[1] se calcula la probabilidad de las celdillas dado el conjunto actual de gaussianas $p(C_i/G_k^{t-1})$, y se usa el teorema de Bayes para estimar la probabilidad de las gaussianas dadas las celdillas actuales $p^t(G_k/C_i)$. Intuitivamente esta última cuantifica la proximidad de la gaussiana a la celdilla: cuanto mayor sea, más cerca se encuentran.

A lo largo de todo el algoritmo, y en particular para determinar la probabilidad de clase, las celdillas de la rejilla de ocupación se consideran muestras de la función densidad de probabilidad observada

de ocupación. Cada celdilla C_i pesa tanto como su estado de ocupación y el peso total de todas las celdillas se denomina I.

$$\forall k \in K, \forall i \in I$$

$$p^t(G_k/C_i) = \frac{P_k^{t-1}\, p(C_i/G_k^{t-1})}{\sum_{k=1}^{K} P_k^{t-1}\, p(C_i/G_k^{t-1})} \tag{5.5}$$

$$P_k^t = \frac{\sum_{i=1}^{I} p^t(G_k/C_i)}{I} \tag{5.6}$$

En el paso de Maximización[1] se reestiman los parámetros de cada gaussiana, usando los cómputos actualizados en el anterior paso de Expectativa y las ecuaciones (5.7)(5.8)(5.9)(5.10)(5.11). Intuitivamente la posición, el tamaño y la orientación de cada segmento se reestiman atendiendo a las celdillas que están más próximas a él, de manera que la gaussiana se ajusta mejor a los puntos que cubre.

$$\mu_{x,k}^t = \frac{\sum_{i=1}^{I} \frac{p^t(G_k/C_i)}{P_k^t} x_i}{I} \tag{5.7}$$

$$\mu_{y,k}^t = \frac{\sum_{i=1}^{I} \frac{p^t(G_k/C_i)}{P_k^t} y_i}{I} \tag{5.8}$$

$$\left[\sigma_{xx,k}^2\right]^t = \frac{\sum_{i=1}^{I} \frac{p^t(G_k/C_i)}{P_k^t} \left(x_i - \mu_{x,k}^{t-1}\right)^2}{I} \tag{5.9}$$

$$\left[\sigma_{yy,k}^2\right]^t = \frac{\sum_{i=1}^{I} \frac{p^t(G_k/C_i)}{P_k^t} \left(y_i - \mu_{y,k}^{t-1}\right)^2}{I} \tag{5.10}$$

$$\left[\sigma_{xy,k}^2\right]^t = \frac{\sum_{i=1}^{I} \frac{p^t(G_k/C_i)}{P_k^t} \left(x_i - \mu_{x,k}^{t-1}\right)\left(y_i - \mu_{y,k}^{t-1}\right)}{I} \tag{5.11}$$

Se han añadido dos mejoras al algoritmo estándard EM para aumentar su rendimiento sobre imágenes de ocupación. Primero, se ha definido una función *pertenece_a* para asignar cada celdilla a la gaussiana más próxima, si es que existe alguna aceptable (5.12). Las celdillas que están lejos de todos los segmentos actuales permanecen sin asignación. Con esta nueva relación la probabilidad de clase se calcula como la fracción de celdillas ocupadas que pertenecen a dicha clase (5.13). Este es un estimador para P_k^t mejor que el que ofrece el algoritmo estándard, y evita algún comportamiento degenerado.

$$C_i \in G_j \Leftrightarrow \begin{cases} p(C_i/G_j) = \text{máx}_{\forall k \in K} \quad p(C_i/G_k) \\ p(C_i/G_j) > umbral \end{cases} \tag{5.12}$$

$$P_k^t = \frac{\sum_{i=1}^{I} pertenece\_a(C_i, G_j)}{I} \tag{5.13}$$

Segundo, la probabilidad de clase se considera una medida de calidad. Las gaussianas que no cubren ninguna celdilla ocupada tienen una calidad nula. Los buenos segmentos deberían cubrir algunas celdillas, cuantas más mejor. Si este indicador de calidad cae por debajo de un umbral mínimo entonces ese segmento se descarta, y se introduce un nuevo segmento aleatorio en el proceso EM de ajuste. Para aceptar un segmento como representante válido de cierto objeto debe sobrepasar cierto umbral de calidad aceptable.

El número de segmentos en la población supone un compromiso entre la potencia expresiva del esquema segmentador y el coste computacional. Cuantos más segmentos haya en la población más rápidamente se incorporan nuevos obstáculos, porque hay segmentos disponibles para asimilarlos. Sin embargo, se ralentiza la actualización de todo el conjunto, porque ahora exige más cómputos. En los experimentos de las figuras 5.4(b) y 5.6 se emplearon 12 gaussianas, lo que permitía un ritmo de 2 iteraciones EM en el procesador Intel 486 a 66 Mhz del B21. Tal y como muestra el experimento de

[1] Los subíndices indican el número de segmento y los superíndices el número de iteración

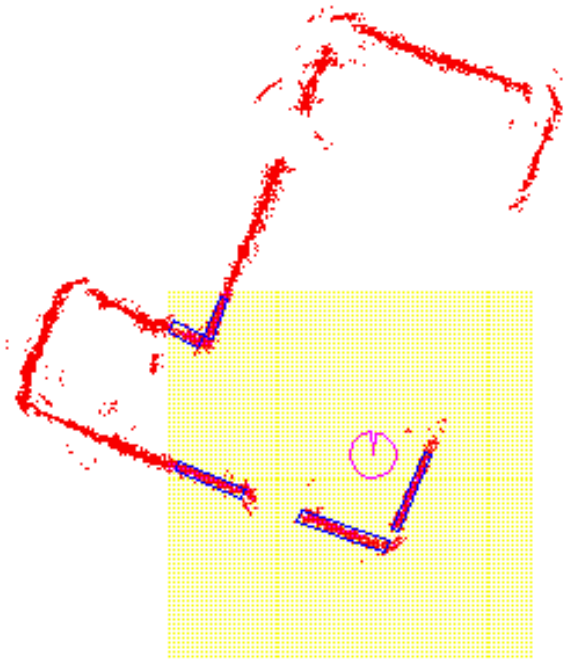


Figura 5.6: *Segmentos de ocupación que capturan el vano de la puerta y el rincón*

la figura 5.6 las paredes principales se representan como segmentos alineados, y también aparece un hueco de unos 60 cm que corresponde al vano de la puerta. En esa figura se representa la rejilla de ocupación con el cuadrado amarillo centrado en los alrededores del robot. También se ha incluido la nube de puntos almacenada durante el recorrido para recalcar el perfil correcto de la habitación. La anchura de los segmentos corresponde con el grado de incertidumbre asociado a la detección de las superficies, debido a los errores de los sónares o al de los odómetros. Es fácil confundir esto con la anchura de la pared, cosa que en ningún caso estima el algoritmo.

5.1.4. Discusión

La última lectura de todos los sensores proporciona al robot una *instantánea* sobre el estado de sus alrededores, como muestra la figura 5.2. Esta instantánea sensorial, continuamente refrescada, siempre está disponible en JDE para los esquemas de actuación. Debido a su simplicidad (en el caso de los sónares del Pioneer, 16 valores enteros) y a su vivacidad, esta representación ha sido utilizada en muchos casos para construir comportamientos reactivos sobre ella, por ejemplo el sorteo de obstáculos. La rejilla calculada también se puede utilizar para reacciones relativamente rápidas y navegación local. ¿Qué se gana con la rejilla respecto a los datos sónar instantáneos?

Primero, la rejilla facilita la fusión sensorial. En ella se pueden integrar observaciones tomadas desde posiciones dispares, en diferentes instantes, en incluso proveniente de sensores distintos como el láser, los táctiles u otros sónares. Esta fusión puede añadir información complementaria necesaria. Por ejemplo, si sólo utilizamos el láser, entonces el robot no podrá evitar el choque con obstáculos situados por debajo de la altura de barrido del sensor láser. Empleando la rejilla, nada impide la incorporación en ella de la información extraida de mapas de disparidad desde pares estereo de cámaras. También ellos aportan información de esos y otros obstáculos. Adicionalmente, la fusión de datos sensoriales hace que la rejilla sea más robusta a errores sensoriales que la última instantánea sensorial. La malla ofrece cierta acumulación temporal y espacial que ayuda a depurar errores sensoriales y perfilar mejor el contorno de los obstáculos, compensando las lecturas erróneas con las medidas correctas, previsiblemente más numerosas. Esta compensación es especialmente adecuada para los sensores sónares porque sus medidas son propensas a ruidos.

Segundo, la rejilla amplía el alcance espacial. Con una rejilla se puede recordar la información de zonas próximas recientemente sensadas, que quedan fuera del alcance sensorial actual. Con ella también se puede recordar información de zonas que de repente quedan ocluídas por un obstáculo

intermedio, y por lo tanto no están reflejadas en los valores sensoriales actuales. Gracias a la rejilla esas zonas pueden tenerse en cuenta en las decisiones de movimiento. Por ejemplo, en la plataforma experimental disponemos de un sensor láser, apuntando hacia el frente del robot, y tenemos una corona completa de sensores de ultrasonido. La rejilla almacena la información precisa del láser sobre la zona frontal, pero también cubre la parte trasera del robot con los datos de los sónares. Esa parte trasera puede ser relevante en ciertos momentos y se perdería si sólo utilizáramos la última instantánea del láser. La rejilla permite incorporar ambas de modo homogéneo.

Un tercer mérito de la rejilla es que representa explícitamente el espacio vacío, lo que resulta muy útil para la tarea de sortear obstáculos. En este sentido, captura toda la información de ocupación contenida en la medida, tanto la de ocupación como la de vacío. Ambas se tienen en cuenta a la hora de decidir el estado de ocupación de una celdilla. De este modo se refuerza la fiabilidad de la información almacenada. La información de vacío no se tiene explícita en los valores numéricos de las lecturas sónares. Gracias a esto se puede distinguir zonas no sensadas, de estado desconocido, de aquellas que se sabe están desocupadas. Esto puede resultar útil para la percepción activa, por ejemplo para navegar hacia zonas sobre las que se tienen pocas evidencias.

Cuarto, la fusión sensorial que se da en la rejilla permite la identificación de estímulos complejos, que no caben en una lectura instantánea. La malla regular permite acumular indicios, evidencias parciales. Como ya explicamos en la parte perceptiva del capítulo 3, esto resulta crucial cuando una instantánea sensorial por sí sola no es concluyente sobre la existencia de tal o cual estímulo. Por ejemplo, la última instantánea sónar se ve afectada por la existencia de una pared próxima, pero desde esa instantánea es imposible distinguir si se trata de una pared o de cualquier otro obstáculo. Es la acumulación de varias lecturas y el alineamiento de las celdillas ocupadas en la rejilla lo que permite concluir con más garantías que el obstáculo concreto es una pared. De este modo se puede explotar globalmente el conjunto de lecturas sónares, porque permiten la aparición de estímulos como segmentos, paredes, huecos, etc., no contenidos en una única medida.

Las ventajas de la fusión que permite la rejilla tienen un coste en tiempo. No ya en tiempo computacional, sino en latencia a la hora de reflejar cambios en la realidad. Quizá esta sea la razón por la que tradicionalmente han servido para reflejar mapas estáticos, donde la latencia no es importante, y se han empleado directamente los sensores para generar reacciones motrices, por ejemplo ante obstáculos imprevistos.

El enfoque que persigue la rejilla de ocupación aquí es totalmente diferente a los mapas estáticos, porque se busca que refleje a la mayor brevedad los posibles movimientos en los obstáculos alrededor del robot. Por ello las dos reglas desarrolladas para actualizar nuestra malla cuidan este aspecto dinámico. Como se puede ver en el anexo A, las reglas de actualización más tradicionales tienen demasiada inercia y por ello no son convenientes para reflejar obstáculos dinámicos en la rejilla. En la práctica, siempre hay que establecer un compromiso entre la rapidez de cambio en la creencia de la celdilla y la robustez frente a datos sensoriales espúreos.

En cuanto a la segmentación, la eficiencia computacional del algoritmo EM permite la construcción vivaz de una representación de segmentos continuamente actualizada. Estos mapas locales dinámicos pueden utilizarse por otros esquemas de actuación para navegación local, o incluso como paso para construir representación espacial a largo plazo.

El algoritmo EM resulta más indicado para segmentar que otros basados en conectividad cuando se tienen *imágenes* con mucho ruido y discontinuidades. Este es el caso de la rejilla de ocupación cuando se construye únicamente desde los sónares. En el caso de rejillas construidas desde el láser los perfiles de los obstáculos son mucho más nítidos y continuos. Entonces sí se pueden utilizar otros algoritmos de segmentación, como el empleado en [García Pérez, 2003], que recorren la vecindad de las celdillas ocupadas.

Otra ventaja EM es que es incremental: para calcular los segmentos en un instante se parte de los segmentos en la iteración anterior. Gracias a esta característica se puede apreciar desplazamientos del *mismo* segmento, reflejando la continuidad del movimiento de un objeto. Por ejemplo, esto resulta vital para estimar velocidad de los obstáculos alrededor del robot. Además, permite el seguimiento de objetos, de modo alternativo al filtro de Kalman [Cox y Leonard, 1994], porque los parámetros del mismo objeto se van modificando con el tiempo, adaptándose a las nuevas observaciones sensoriales.

5.2. Detección de puertas con visión

Otro de los estímulos relevantes para los robots de interiores son las puertas. Lejos del uso de visión para reconstruir en 3D, utilizaremos visión para enriquecer la representación del entorno con un estímulo específico, las puertas, siguiendo las especificaciones de JDE. En concreto, la detección y localización de puertas en el entorno del robot móvil se ha descompuesto en tres esquemas que funcionan en paralelo. Su organización y dependencias se muestra en la figura 5.7.

El esquema `jambas-3D` construye el estímulo `puerta` apoyándose en una rejilla de probabilidad de jamba. Esa rejilla integra la información de ocupación (láser, sónares, etc.) y la información de las `jambas visuales` identificadas en el flujo de imágenes. Para ello utiliza la regla de fusión probabilística. La dinámica de probabilidades que suben y bajan dependiendo de los datos sensoriales acaba confirmando la ubicación real de las jambas, desechando las ficticias, que son debidas a la ambigüedad de profundidad inherente a la visión monocular.

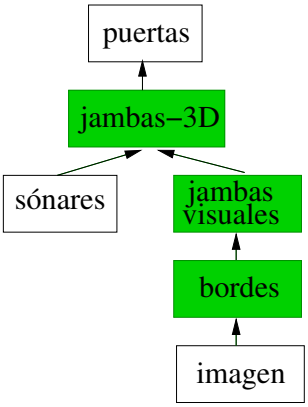


Figura 5.7: *Jerarquía de estímulos para las puertas*

El esquema `jambas-visuales` construye el estímulo `jamba visual` detectando bordes verticales largos en las imágenes. Depende de los bordes de intensidad, los cuales son calculados por el esquema `bordes` después de suavizar la imagen cruda en niveles de gris que procede de la cámara. Todos los esquemas están funcionando simultáneamente, de modo que el procesamiento y asimilación del flujo de imágenes es continuo. En esta sección describiremos cada uno de ellos con más detalle.

En cuanto a su utilidad, percibir las puertas puede facilitar las estrategias de navegación para atravesarlas sin problemas, puede resultar conveniente como delimitadores naturales entre cuartos, o como ayuda a la localización. Además, las puertas son un estímulo muy frecuente en entornos de oficina, laboratorios, etc., por lo que la técnica descrita en esta sección puede ser de utilidad en robótica de interiores.

Por ejemplo, suele ser muy complicado hacer que un robot atraviese puertas relativamente estrechas si utiliza exclusivamente sensores sónares [Budenske y Gini, 1994]. Las estrategias de navegación basadas en sónares funcionan bien si las puertas son anchas, pero resultan insuficientes para atravesar puertas estrechas, debido a la alta incertidumbre angular asociada a sus medidas. Esa falta de resolución lleva a percibir los huecos de una anchura menor a la real, lo cual resulta crítico con puertas que ofrecen poco margen aparte de la anchura del robot. En esta situación, la detección de las puertas y la caracterización de su posición y anchura reales utilizando visión puede hacer posible que el robot atraviese la puerta fiablemente.

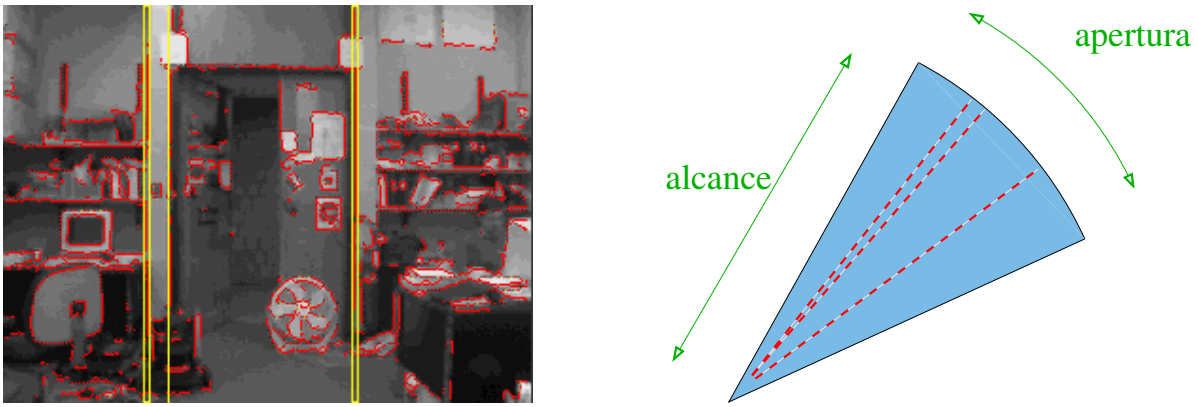
En otros planteamientos, la percepción de las puertas también puede ser útil como ayuda a la autolocalización inicial del robot. Las puertas detectadas en el entorno tras arrancar el robot pueden ayudar a discriminar mejor cuál es su posición en cierto mapa global que ya pudiera tener. Además, puede ser útil para un robot repartidor de correo en interiores, como ajuste fino para llegar a una habitación concreta después de que la navegación global haya llegado aproximadamente al destino y deje al robot en los alrededores de la puerta, dando por concluída su tarea (véase [Simmons y Koening, 1995]). Estos dos últimos ejemplos son de utilidad directa en Xavier (figura 4.8), el robot de CMU para el cual se programó inicialmente este sistema perceptivo.

5.2.1. Jambas en la imagen

Para percibir la puerta en profundidad primeramente hay que detectar sus jambas en cada imagen monocular tomada por la cámara. El invariante que hemos encontrado en la mayoría de las puertas son sus jambas. Independiente de que sea más o menos alta o estrecha, de un color u otro, todas las puertas tienen jambas. El esquema `jambas-visuales` se encarga precisamente de extraer de la imagen monocular las posibles jambas que pueda haber.

La información de partida son los bordes de intensidad en la imagen, que actualiza por separado el esquema `bordes`. Este último esquema, primero toma las imágenes de entrada, que vienen en niveles de gris, y les pasa un filtro de suavizado. Con ello se eliminan los pixels espúreos, que sólo introducen ruido en los cálculos posteriores. Segundo, aplica un filtro de bordes estándard. En concreto, hemos aplicado la técnica y el código de Stephen Smith descrito en [Smith y Brady, 1997]. Este filtro clasifica cada pixel de la imagen como punto borde o no.

En cada iteración, el esquema `jambas-visuales` busca en la imagen de bordes líneas verticales largas. La cámara del robot está montada paralela al suelo y sólo se permiten movimientos horizontales. Por ello las jambas aparecen siempre como líneas verticales en las imágenes. En concreto, se seleccionan las columnas en las cuales más del 80 % (*porcentaje umbral*) de sus puntos sean de borde. Como la verticalidad estricta es difícil de conseguir, en la práctica se tienen en cuenta las $n = 4$ columnas más próximas. Todas las columnas que superen ese porcentaje umbral representan una línea vertical pronunciada, posiblemente la jamba de una puerta.



(a) (b)

Figura 5.8: *Detección de la puerta en la imagen (a) y modelo sensorial de visión de esa imagen a la rejilla con probabilidad de jamba (b)*

Por ejemplo, la figura 5.8(a) muestra una imagen típica, de 320*240 pixels, obtenida con una cámara de 55° de apertura. En ella se pueden observar los puntos borde superpuestos a la imagen suavizada, y la salida final del detector de bordes verticales, con tres columnas resaltadas, las dos jambas reales y un tercer borde vertical muy pronunciado. Obsérvense los bordes resaltados y el falso positivo al lado de la jamba izquierda.

En los experimentos, este detector funciona de modo vivaz sin suponer demasiada sobrecarga al procesador. Como todos los sensores, este detector de jambas visuales tiene asociado un alcance máximo: sólo detecta jambas próximas, puesto que las puertas lejanas ocupan pocos pixels y nunca superarán al *porcentaje umbral*. No obstante, en entornos de oficina las distancias no suelen ser muy grandes.

Con él, los bordes de las puertas aparecen resaltados, también los límites de armarios o los bordes verticales de pizarras que contrasten con el color de la pared. Sin embargo, no hay problema con esos falsos positivos, está previsto desecharlos posteriormente. Lo importante en este esquema es no quedarse ninguna jamba real sin detectar, puesto que lo que no se vea en esta etapa no existe de ningún modo para el esquema `jambas-3D`, y por ello se pierde.

Para que esta técnica sea válida se necesita que haya contraste de luminosidad entre el marco y la pared, o entre el marco y la hoja de la puerta, que suele ser habitual. Es interesante resaltar que en

esta implementación no se ha necesitado utilizar el color como discriminante.

A la hora de detectar las jambas en la imagen se decidió hacerlo como un esquema independiente, en vez de como parte del esquema `jambas-3D`. De este modo el esquema `jambas-3D` se usa principalmente para detectar la existencia de la puerta, y se puede tener otro diferente para el mantenimiento fresco y exacto de su posición. Una vez que se sabe segura su existencia se puede desactivar `jambas-3D` y corregir la posición relativa estimada de la puerta únicamente atendiendo a las `jambas visuales` directamente. Esto permite una actualización mucho más vivaz que esperar a que las evidencias de la rejilla cambien su estado convenientemente (por ejemplo para corregir desviaciones odométricas).

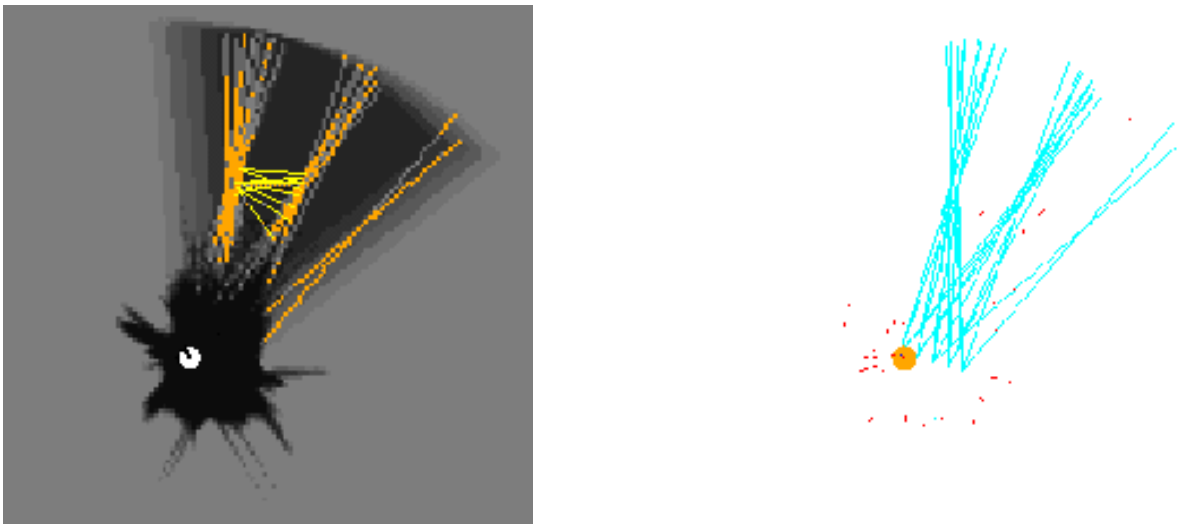
5.2.2. Esquema jambas-3D

Para extraer profundidad usando visión se necesita al menos un par de imágenes desde distintas posiciones para resolver la ambigüedad inherente a la visión monocular. Cada una de las imágenes sólo proporciona información parcial acerca de la puerta, es decir, información no concluyente. Este esquema perceptivo utiliza el marco probabilístico para integrar la información de varias imágenes tomadas por el robot desde posiciones distintas, así como de los sensores ultrasónicos, con el objetivo de inferir la existencia y localización de las puertas (figura 5.9).

En concreto, para representar la creencia que tiene el robot sobre la existencia o no de puertas en su entorno, éste mantiene un *rejilla de probabilidad de jamba*. El esquema tiene dos partes bien diferenciadas que se ejecutan en cada iteración. Primero, la actualización de esa rejilla con las evidencias de jamba recogidas desde la última iteración. Y segundo, utilizando la información de la rejilla, buscar si algún par de puntos con alta probabilidad de ser jamba cumple las condiciones de puerta. El resultado final son unas variables con las posiciones relativas de las puertas detectadas, si es que existe alguna.

Rejilla de probabilidad de jamba

La rejilla es una matriz bidimensional, situada en un plano paralelo al suelo a la altura de los sónares ($\approx 1m$ en Xavier). Cada celda almacena la probabilidad de que en esa posición haya una jamba de puerta, y en los experimentos concretos se les eligió un tamaño de $10cm * 10cm$. Esta representación horizontal resulta suficiente para representar las puertas, su orientación, posición y anchura. Las jambas reales son verticales y en este plano aparecen como puntos. Con un par de ellos se representa una puerta. Puesto que esta representación está dirigida exclusivamente a la identificación del estímulo `puerta` no aparecen en ella objetos por debajo de la altura de los sónares (como las papeleras).



(a) (b)

Figura 5.9: *Rejilla con la probabilidad de jamba tras cinco observaciones desde varias posiciones (a) y rayos visuales procedentes de imágenes obtenidas desde ellas (b)*

Esta malla se va renovando a medida que el robot recibe nueva información sensorial, bien desde la cámara, bien desde los sónares y constituye la base sobre la que se decide interpretar lo visto como puerta o no. La idea subyacente es que después de unas cuantas observaciones sólo la posición real de la jamba recibirá la realimentación positiva de la mayoría de las imágenes. El resto de localizaciones se verán desmentidas con alguna observación y su probabilidad de jamba no podrá ser tan alta como en la posición auténtica.

De modo análogo a la rejilla de ocupación, su actualización viene gobernada por los modelos sensoriales y la regla de actualización. Los sensores que aportan información respecto a que una jamba pueda estar situada en una celdilla o no son el de `jambas visuales` y los de proximidad. Los describimos a continuación.

Modelo sensorial de la cámara Cada vez que se recibe una imagen *independiente* se actualiza la probabilidad asociada a las celdillas de la malla que caen dentro del campo visual. Intuitivamente, si en la imagen se observa una jamba visual, es indicio de que hay realmente una jamba *en algún lugar* del plano perpendicular al suelo que proyecta en los pixels de ese borde. El plano proyectante intersecta con la rejilla horizontal en una línea, y en cualquier punto de esa línea puede estar físicamente la jamba. La probabilidad de jamba asociada a esas celdillas debe aumentar. Por el contrario, si en la imagen no se aprecia ningún borde vertical pronunciado, eso es síntoma de que en el campo de visión abarcado por la cámara no hay ninguna puerta y la estimación de probabilidad de jamba debe bajar en él. En resumen, se aumenta la probabilidad de jamba en todas aquellas posiciones que podrían ser la causa de que se haya apreciado una jamba en la imagen y se disminuye en el resto.

El campo visual utilizado en la implementación de este esquema tiene forma de sector circular, y viene determinado por un *alcance* y una *apertura* angular, como muestra la figura 5.8(b). Siguiendo este modelo se aumenta la probabilidad de jamba en las celdillas sobre las líneas discontinuas porque todas ellas proyectan en pixels donde se aprecia una jamba en la imagen de la figura 5.8(a). También se baja la probabilidad en las celdillas oscuras que caen dentro del campo visual, porque en su proyección no se ha apreciado un borde vertical pronunciado. Empíricamente ajustamos los valores a $p(jamba_{x,y}/img(t)) = 0{,}72$ si proyecta en una jamba de imagen o $p(jamba_{x,y}/img(t)) = 0{,}42$ en caso contrario. Estos valores concretos del modelo suponen un compromiso entre el peso de cada observación individual y la tolerancia a algún error en la percepción.

Modelo sensorial de ultrasonidos Los sensores de ultrasonido, con sus medidas de distancia al obstáculo más próximo, también aportan algo de información de interés en la detección de puertas. Básicamente permiten descartar posiciones tentativas de las jambas. En las zonas donde se percibe espacio vacío, difícilmente habrá una jamba.

El modelo de sónar utilizado consiste en un lóbulo que se abre desde la posición del sensor hasta el radio medido, con el eje en la perpendicular del sensor y siguiendo una apertura que refleja precisamente la de la onda ultrasónica. De hecho, hemos utilizado literalmente el mismo modelo que el empleado para la rejilla de ocupación, con la salvedad de no incluir ningún valor positivo en el arco de ocupación. Todas las celdas que caigan dentro de ese lóbulo disminuyen su probabilidad de ser jamba, puesto que el sónar ha percibido en ellas espacio vacío.

Regla de actualización Bayesiana El estímulo que se pretende capturar con esta rejilla son las `jambas-3D`, que tienen naturaleza estática. Por ello, la regla de actualización aquí no trata de reflejar movimientos rápidos, sino que debe estar orientada a acumular evidencias tolerando ciertos datos ruidosos (por ejemplo las falsas `jambas visuales`). Por este motivo se decidió utilizar la regla probabilística, que ya ha demostrado su buen funcionamiento en una aplicación similar, los mapas estáticos.

En cada momento t, la estimación que se tiene de la probabilidad de jamba en el punto (x, y), $p_{jamba}(x, y, t)$, se define como la probabilidad de jamba en ese punto condicionada a las observaciones que se han obtenido hasta ese momento. Así lo expresa la ecuación 5.14, donde $data(t-1)$ supone el conjunto de observaciones acumuladas hasta el instante $t-1$ e $imagen(t)$ la observación actual.

$$p_{jamba}(x, y, t) = p(jamba/img(t), data(t-1)...) \tag{5.14}$$

Inicialmente todas las casillas de la rejilla tienen valor 0.5, reflejando el desconocimiento total. La probabilidad en cada una de las casillas se va actualizando siguiendo la *regla de Bayes* a medida que se incorpora información. Utilizando un desarrollo análogo al que se detalla en el anexo A, se llega a la expresión incremental de la ecuación (5.15). Si una imagen proporciona información sobre el estado de determinada celdilla (x, y), es el valor del modelo de sensor $p(jamba/img(t))$ en esa posición el que determina si allí la probabilidad de jamba sube o baja después de la nueva observación. La información proveniente de las lecturas sónares se trata del mismo modo, pero con su propio modelo sensorial.

$$\rho_{jamba}(x, y, t) = \frac{p(jamba/img(t))}{1 - p(jamba/img(t))} * \frac{1 - p_{jamba}}{p_{jamba}} * \rho_{jamba}(x, y, t-1) \tag{5.15}$$

Estímulo puerta

La segunda parte en cada iteración del esquema `jambas-3D` consiste en buscar posibles puertas, una vez que la rejilla está actualizada. Básicamente, consiste en buscar pares de jambas con alta probabilidad y ver si están a la distancia adecuada para formar parte de una puerta.

Una puerta viene definida por sus extremos, *punto*1 y *punto*2, que son precisamente sus jambas. A cada posible puerta le asociamos una *probabilidad de puerta* que definimos con la fórmula (5.16), es decir, el producto de la probabilidad de cada una de sus jambas por un factor que indica en qué medida esa posible puerta satisface las condiciones de `puerta`. La condición principal evalúa si su anchura casa con la esperada en las puertas del entorno. En la implementación realizada ese factor es binario: 1 si la distancia entre jambas caen entre un mínimo y un máximo parametrizables, y 0 fuera de ese intervalo. También se anula si las dos jambas están contenidas en el mismo rayo óptico, ya que en ese caso no puede tratarse de una `puerta`.

$$p_{puerta}(punto1, punto2, t) = p_{jamba}(x1, y1, t) * p_{jamba}(x2, y2, t) * condiciones\_de\_puerta \tag{5.16}$$

Para examinar la existencia o no de `puertas`, primero se explora la rejilla de probabilidades de jamba, y sólamente teniendo en cuenta aquellas celdillas más probables, se elabora un *conjunto de puertas candidatas*, formado por todos los posibles pares. Seguidamente, se evalúa la probabilidad de cada una de estas puertas candidatas y el robot asume que ha detectado realmente una puerta cuando su probabilidad está por encima de cierto *umbral de aceptación*. Es decir, cuando sus jambas están confirmadas por las observaciones sensoriales y la distancia entre ellas es la adecuada.

En la figura 5.9(a) se puede observar el estado de la rejilla de probabilidad de jamba después de la observación del entorno desde cinco posiciones distintas, separadas entre sí unos 30cm. Alrededor del robot la probabilidad de jamba es muy baja (color oscuro), gracias a los sónares, que han ayudado a descartar las hipótesis en ese área. En la parte central superior se aprecian tres conjuntos de alta probabilidad correspondientes a las proyecciones de las distintas jambas visuales detectadas en las sucesivas imágenes. Las zonas oscuras intermedias corresponden a la proyección de la zonas de imagen con ausencia de jambas visuales.

Las partes intermedias de los dos haces centrales son las que reciben más probabilidad porque es donde intersectan más proyecciones. En color amarillo, más o menos horizontales entre esos dos haces aparece superpuesto el conjunto de puertas candidatas obtenido hasta ese momento. Obsérvese que no hay ninguna puerta candidata con un extremo en el haz de rayos de la derecha, porque no han acumulado suficiente probabilidad. La longitud de todas las puertas candidatas generadas está dentro de los márgenes aceptables.

5.2.3. Percepción activa: movimiento para percibir la puerta

En los experimentos descritos hasta ahora, el movimiento del robot y la orientación de la cámara venían teleoperados por el humano. Los esquemas mencionados funcionan en modo pasivo, procesando los sónares y todo lo que caiga delante de la cámara. Con esos experimentos se avala la capacidad de

esos esquemas para concluir la existencia y ubicación de la puerta desde unas observaciones razonables obtenidas desde posiciones distintas.

Adicionalmente hemos desarrollado un esquema de actuación, el esquema `busca-puerta` para acelerar la búsqueda de `puertas`. Su objetivo es dirigir los movimientos del robot y de su cámara para que encuentre y discrimine con mayor rapidez las `puertas`. En la terminología etológica sería el comportamiento apetitivo del estímulo `puerta`, ya que favorece su aparición.

Puesto que no todas las observaciones aportan la misma información, si se eligen bien los puntos desde los cuales se toman nuevas imágenes se podrá discriminar con mayor celeridad la existencia o no de las jambas. Por ejemplo, debido al campo visual limitado, cobra importancia el lugar desde dónde se observa el entorno. Dependiendo de ese lugar se percibirán unos estímulos y se ignorarán otros. Además, los lugares de observación pueden discriminar más o menos entre el conjunto de puertas candidatas que se está manejando en cada momento.

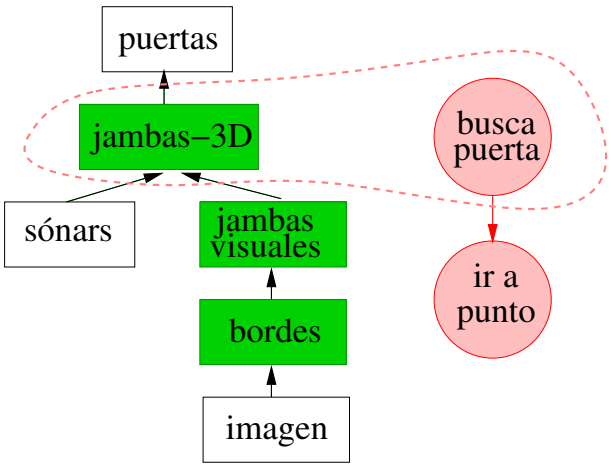


Figura 5.10: *Jerarquía de esquemas en la detección activa de puertas*

Como muestra la figura 5.10, el esquema `busca-puerta` se activa en paralelo al esquema `jambas-3D`. Juntos forman un sistema de percepción activa, capaz de detectar autónomamente las puertas, que en este caso implica moverse para percibir. Un esquema superior puede activarlos conjuntamente si por alguna razón interesa localizar rápidamente las `puertas` y en función de que se encuentre o no, comportarse de un modo u otro. En la figura 5.10 también se puede observar que se apoya en el esquema de actuación `ir-a-punto`, el cual comentaremos posteriormente.

En una primera fase, el esquema `busca-puerta` hace que el robot dé una vuelta completa sobre sí mismo, mirando en todas direcciones, buscando todas las posibles jambas que pueda haber en sus cercanías. Después de ese primer vistazo en derredor se tiene ya una cierta información sobre dónde pueden estar las `puertas`, caso de existir alguna en el entorno. Por ejemplo, en la figura 5.9(a) muestra el aspecto tras una vuelta inicial. Las `jambas visuales` que se hayan apreciado en las imágenes se proyectan en la rejilla, elevando la probabilidad de todas las posiciones que explican esas observaciones. Ya hay algunas jambas candidatas, y debido a la ambigüedad de profundidad, tanto la posición real como otras hipotéticas aparecen resaltadas. Entre ellas hay que discernir la puerta real con nuevas observaciones.

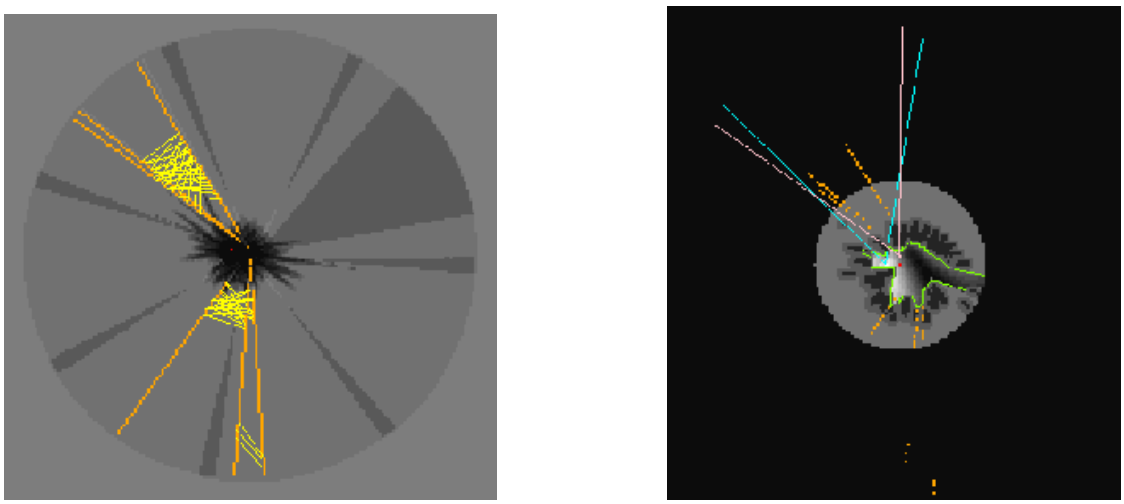
A partir de ese momento la estrategia de movimiento autónomo es iterativa: se calcula el punto óptimo para tomar la siguiente imagen y se desplaza el robot hasta él mediante navegación local. Una vez allí se vuelve a analizar la imagen de la cámara, se actualiza al rejilla de probabilidad de jamba y se concluye si se ha detectado alguna puerta, si hace falta seguir explorando o si es mejor abandonar la búsqueda. En caso de necesitar la acumulación de más evidencia sensorial se calcula el nuevo punto óptimo y así sucesivamente.

El punto óptimo es el que maximiza la discriminación entre las puertas candidatas que se tienen en cada momento. Una vez calculado el punto óptimo para situar la cámara se hace uso de la arquitectura activando el esquema `ir-a-punto`, que se encarga de la navegación local hasta aquella posición. No es necesario acudir a técnicas de navegación más complejas porque, como ahora veremos, uno de los

requisitos del punto óptimo es que esté próximo y sea alcanzable con un giro y una traslación. Aunque la arquitectura lo permite, se ha preferido no procesar imágenes mientras el robot se está moviendo.

Cálculo del punto desde el cual observar

Según se describió anteriormente, desde la rejilla de probabilidad de jamba se extrae el *conjunto de puertas candidatas* formando pares con las celdillas de mayor probabilidad de jamba. Para cada uno de esos pares se evalúa su probabilidad de puerta y el conjunto global queda ordenado de mayor a menor probabilidad. Tras la vuelta inicial, ningún par habrá conseguido acumular evidencia suficiente como para sobrepasar el *umbral de aceptación* y dar la búsqueda por concluída. Sin embargo, ese conjunto representa las mejores hipótesis que el robot tiene para explicar las imágenes que ha visto. Previsiblemente, la ubicación real estará contenida en ellas y será conveniente dirigir las nuevas observaciones para que confirmen o desmientan estas candidaturas.



(a) (b)

Figura 5.11: *Aspecto de la rejilla después de la vuelta inicial (a) y cálculo del punto desde el cual tomar la siguiente imagen (b)*

El lugar (x, y) y la orientación θ de la cámara desde los cuales tomar la siguiente imagen del entorno se calculan maximizando una *función de utilidad*, definida en la ecuación (5.17). Por un lado, cuantas más jambas candidatas se observen desde una posición su utilidad asociada es mayor, porque se pueden constrastar más candidaturas. En este sentido, el sumatorio de (5.17) se extiende a todas las jambas candidatas que caigan dentro del campo visual desde (x, y, θ). Por otro lado, la utilidad es mayor cuanto menor es el solapamiento en la imagen entre las candidatas que se observen. Por ejemplo, si desde una posición tres jambas candidatas aparecen en los mismos pixels entonces desde esa posición no se puede discriminar cuál de ellas es la verdadera. Ese punto de observación es menos recomendable que otro desde el cual esas mismas candidatas aparezcan en pixels distintos y por lo tanto se pueda comprobar mejor la validez de cada una de ellas. En este sentido, el sustraendo de (5.18) resta el solapamiento medio a la información que aporta esa jamba candidata observada desde ese punto. El término $solapamiento(jamba_i, jamba_j)$ es una función triangular que se hace 1 si ambas jambas candidatas proyectan en el mismo pixel y disminuye hasta 0 a medida que crece la distancia entre esos pixels de proyección.

$$utilidad(x, y, \theta) = \sum_{jamba_i \in campo\_visual} informacion(jamba_i) \tag{5.17}$$

$$informacion(jamba_i) = 1 - \frac{\sum_{i \neq j} solapamiento(jamba_i, jamba_j)}{\#jambas \in campo\_visual} \tag{5.18}$$

Para agilizar los cálculos la función de utilidad sólo se calcula en posiciones aceptables: que no estén muy alejadas de la posición actual del robot (a menos de 3 m), que no estén cerca de ningún

obstáculo y que sean directamente accesibles con navegación local (un giro y una traslación). Estas condiciones reducen considerablemente el espacio de búsqueda.

En la figura 5.11(b) se puede apreciar la función de utilidad evaluada en las cercanías del robot después de la vuelta inicial reflejada en la figura 5.11(a). Para cada celdilla dentro de la zona aceptable se representa la utilidad de la mejor orientación de la cámara, en oscuro valores bajos y en claro valores más altos. Los puntos sueltos naranjas son las jambas candidatas que necesitan confirmación. El punto central es el robot, y en él tiene el vértice el sector que representa la orientación actual de la cámara. Desde el punto central se abre también un contorno de celdillas directamente accesibles con navegación local sin chocar con los obstáculos, los cuales aparecen como las manchas oscuras. Las zonas accesibles se calculan bien desde la rejilla de ocupación, bien desde los sónares instantáneos. Dentro de ese contorno se evalúa la función de utilidad, generando en el ejemplo un patrón que favorece a los puntos situados a la izquierda. En este caso, la utilidad se hace máxima en el vértice del segundo sector, que es el punto y orientación óptimos recomendados.

5.2.4. Discusión

El esquema `jambas-3D` es otro ejemplo más de las ventajas de las rejillas como medio para acumular evidencias parciales. En este caso, sirve para obtener la profundidad de las jambas, que es una información no contenida de modo completo en ninguna de las observaciones sensoriales por separado: ni en las imágenes monoculares ni en los sónares. Este uso de la visión y la rejilla permite estimar profundidad incluso con una sóla cámara. Además, muestra una ventaja concreta de la función sensorial al incorporar los sónares, que descartan jambas candidatas y aceleran la detección.

En contraste con las técnicas de visión estéreo clásicas, se extrae la posición de la puerta sin necesidad de buscar homólogos. Para extraer mapas de profundidad, los pares estéreo emparejan puntos homólogos entre la imágenes derecha e izquierda y aplican triangulación. El problema de asignación de homólogos puede ser intratable si las imágenes han sido tomadas desde posiciones completamente distintas. Esa es precisamente nuestra situación con el robot, que puede observar la puerta desde lugares distantes más de un metro. En lugar de resolver explícitamente esta asignación de homólogos la rejilla de probabilidad permite representar la ambigüedad de cada imagen y solucionar el problema de modo implícito.

Cualquier metodo de identificación basado en el procesamiento en una sola imagen requiere contemplar en la misma imagen la puerta completa. Por ejemplo, buscar dos bordes verticales y un tercero horizontal (o inclinado) para enlazarlos. En contraste con ellos, la técnica presentada no requiere observar los dos bordes de la puerta en la misma imagen. Por eso es valida incluso si el robot está debajo del quicio.

En cuanto a la regla de actualización, un mérito de la probabilidad para el estímulo `puerta` es que tolera relativamente bien el ruido, los falsos positivos y los fallos de observación. Por ejemplo, si temporalmente deja de observarse una jamba por la presencia espúrea de un obstáculo interpuesto, el sistema funcionará bien si obtiene nuevas imágenes buenas con las que compensar ese fallo, la probabilidad acabará subiendo. Para materializar esa robustez deben diseñarse cuidadosamente los modelos de sensor.

Uno de los puntos débiles de este enfoque es el ruido en los odómetros. Para poder mezclar distintas imágenes su información se ancla espacialmente a la posición del robot cuando tomó cada una de ellas. Si la estimación de posición es errónea la mezcla de probabilidades no es válida. Hemos utilizado odómetros para estimar la posición, que adolecen de error acumulativo. Al igual que comentamos con la rejilla de ocupación, las posiciones de las jambas sólo necesitan ser válidas como posiciones relativas, es decir, precisas respecto de la posición actual del robot, y no en términos absolutos. Ese tipo de precisión se tiene con los odómetros si sólo se tienen en cuenta medidas recientes, dentro de una ventana temporal en la que no da tiempo a acumular demasiado error relativo de posición. Además, la propia rejilla asume bien errores de localización pequeños.

Otra desventaja es el actual tamaño de las celdillas, 10 cm, que pone en duda la precisión obtenible con este sistema. Sin embargo, el enfoque se muestra eficiente para comprobar la propia existencia de puertas, y su posición dentro de la resolución de la rejilla. Si se quiere más precisión se puede complementar con otros algoritmos, que resultan mucho más sencillos una vez que la existencia del

estímulo `puerta` ha sido verificada. Por ejemplo, una fase de detección como la descrita y otra fase, mucho más eficiente, que permite interpretar directamente las `jambas visuales` como jambas de puerta si son compatibles con la zona en la que aproximadamente se ha detectado la `puerta`.

Además del trabajo realizado con pares estéreo la idea de combinar varias imágenes para inferir características 3D ha sido explorada profusamente. Por ejemplo, Margaritis [Margaritis y Thrun, 1998] utiliza también la probabilidad como marco para localizar objetos en 3D a partir de visión. También Collins [Collins, 1996][Collins, 1997] extrae estructura 3D del entorno, en su caso los edificios de un campus universitario, desde varias imágenes aéreas. Como primitiva visual también utiliza bordes y los retroproyecta en el espacio 3D desde todas las imágenes disponibles. Las celdillas con los bordes reales quedan resaltadas porque recibirán más retroproyecciones que el resto.

5.3. Comportamientos sin jerarquía

Una vez explicado cómo se capturan con esquemas dos estímulos significativos como son la ocupación y las puertas del entorno local, en esta sección describiremos algunos comportamientos concretos conseguidos con JDE sin necesidad de utilizar jerarquía. En concreto, presentaremos cómo hemos materializado con esquemas las conductas `sigue-pelota`, `sigue-pared`, y el `avance-rápido--sorteando-obstáculos`.

Como son comportamientos relativamente sencillos no se ha requerido el uso de la jerarquía que ofrece JDE. La activación simultánea en un único nivel de los esquemas de actuación y de percepción relevantes ha sido suficiente. En este sentido, los esquemas de actuación no activan a otros, sino que emiten directamente comandos a los actuadores. Estos experimentos respaldan la descomposición del comportamiento en dos partes, la perceptiva y la de actuación, que están activas simultáneamente por separado. Esta separación ya ha sido validada por otros autores [Arkin, 1989b], los experimentos aquí descritos lo ratifican sobre los robots elegidos y utilizando la plataforma software desarrollada, cuya operatividad que queda de este modo acreditada. Además, estos experimentos enfatizan distintos matices como la subjetividad, la localidad y el dinamismo asociados a la percepción dentro de JDE.

5.3.1. Seguimiento con visión local de una pelota

Este comportamiento consiste en que el robot siga a una pelota, manteniéndose a una distancia de referencia, y empleando como sensor únicamente la cámara local. Este comportamiento resulta de utilidad en el entorno de la RoboCup[1], y se puede generalizar al seguimiento de cualquier objeto utilizando visión. La RoboCup es un campeonato anual en el que los robots compiten entre sí jugando al futbol en equipos. Este concurso plantea el problema del fútbol entre robots como reto concreto y como marco común para impulsar avances en robótica e inteligencia artificial. De hecho, supone un escenario muy exigente, dinámico, y competitivo en el que se pueden investigar técnicas de control, comportamiento, cooperación, inteligencia artificial distribuida, etc. Tiene distintas categorías, como la liga simulada, la de robots pequeños, la de robots medianos y la de perritos de Sony entre otras.

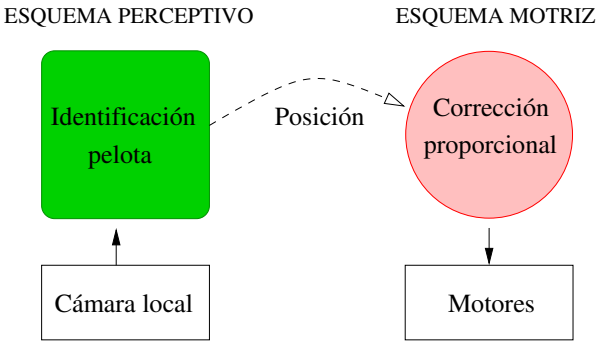


Figura 5.12: *Diseño en esquemas del comportamiento sigue-pelota utilizando visión local*

[1] http://www.robocup.org

Hemos materializado esta conducta en el robot EyeBot, que ya presentamos en el capítulo 4. En cuanto a sensores, el EyeBot incorpora una pequeña cámara de color, tres sensores de infrarrojos y dos odómetros, uno en cada una de sus ruedas. En cuanto a actuadores, este robot incorpora dos motores, un servomotor para mover la cámara en horizontal y otro para chutar la pelota. Para la conducta `sigue-pelota` hemos utilizando como sensor su cámara frontal, y como actuadores los motores de sus dos ruedas motrices.

El comportamiento programado materializa la coordinación visuomotora necesaria para generar la conducta de seguimiento. El objetivo concreto es situarse frente a la bola, si ésta se desplaza a la derecha, entonces el robot debe girar en ese sentido para no perderla, y de modo recíproco si la bola se mueve hacia la izquierda. Si la pelota se aleja, entonces el robot debe avanzar, y si está demasiado cerca, el robot debe retroceder.

Tal y como señala la figura 5.12 este programa se organiza siguiendo la arquitectura JDE, y se divide en dos partes. Primero, un esquema perceptivo que se encarga de identificar la pelota a seguir y su posición en la imagen de la cámara. Segundo, un esquema de actuación que decide qué velocidades ordenar a cada motor del robot. Estos dos esquemas los hemos implementado en sendas hebras ejecutando en el EyeBot, a las que hemos añadido una tercera hebra de servicio para finalizar la ejecución. Como el programa se ejecuta íntegramente sobre el procesador del EyeBot y accediendo a recursos locales, no hemos utilizado los servidores de acceso remoto, ni `otos` ni `óculo`.

Percepción de la pelota

El estímulo relevante para este comportamiento es sin duda la `pelota`, y el esquema perceptivo se encarga de identificarla y localizarla en la imagen de la cámara. El balón de juego reglamentario en la liga pequeña de la RoboCup es una pelota de golf, de color naranja. Esta bola es fácilmente identificable por su color chillón. Su posición dentro de la imagen es la información significativa para que el robot se mueva de un modo u otro.

Hemos dividido el esquema perceptivo en tres fases que se ejecutan secuencialmente en cada iteración del esquema. Primero, un filtro de color que clasifica los píxels de la imagen en pelota o no-pelota. Segundo, una segmentación de la imagen filtrada, que identifica las diferentes ventanas con alta densidad de pixels naranjas. Muy probablemente, estas zonas corresponden a pelotas del entorno. Finalmente, explorando la ventana mayor se calcula el punto central de la pelota (no tiene porqué ser el centro de la ventana) que se tomará como referencia para el seguimiento. El resultado conjunto global se puede apreciar en la figura 5.15(a), donde el robot tiene perfectamente capturada internamente la posición en la imagen de la pelota que tiene situada enfrente.

El sistema operativo del EyeBot, llamado ROBIOS, ofrece funciones para acceder a la imagen de color que captura la cámara local, en formato RGB. Un ejemplo de imagen se muestra en la figura 5.13(a). Para identificar los pixels de la pelota ajustamos un filtro de color en ese espacio. Un pixel se considera naranja si $99 < R < 255$, $65 < G < 206$, $48 < B < 184$ y $1{,}25 < R/G < 1{,}8$. Estos valores, ajustados a mano, resultan suficientemente discriminativos y robustos para la mayor parte de las condiciones de luminosidad en las que realizamos los experimentos.

(a) (b)

Figura 5.13: *Percepción de la pelota en la imagen*

Una vez clasificados los pixels, aún no se tiene información de si realmente hay alguna pelota en la

imagen, varias o ninguna. Para averiguarlo agrupamos en segmentos rectangulares los pixels naranjas que estén próximos. En la figura 5.13(b) se puede apreciar en amarillo el segmento que aglutina la mayoría de los pixels naranjas de la imagen filtrada.

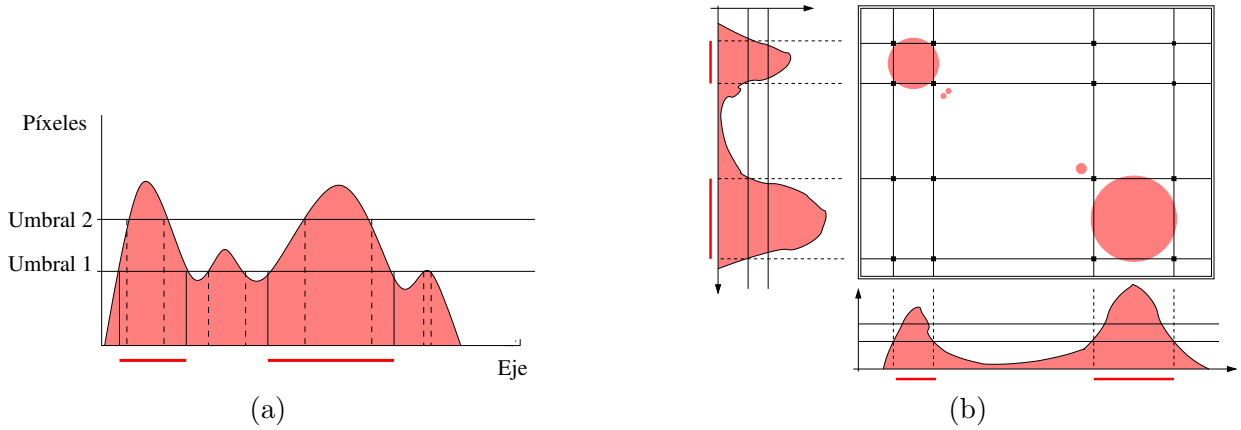


Figura 5.14: *Segmentación basada en doble umbral*

Esta segmentación la hemos hecho analizando el histograma de pixels naranjas, tanto en las filas como en las columnas. Como se muestra en la figura 5.14(a), para cada columna se calcula cuantos píxels de color naranja hay en ella. El histograma de columnas se recorre de la primera a la última. Si en una columna la densidad de pixels naranja pasa a superar el *umbral-1* se marca como posible comienzo de segmento. Si en las columnas siguientes la densidad se mantiene por encima de ese umbral y llega superar el *umbral-2*, entonces aquella columna se ratifica como inicio de segmento. Cuando la densidad vuelva a caer por debajo del primer umbral se anota la finalización de ese segmento. El análisis prosigue hasta finalizar todas las columnas, y del mismo modo se estudia el histograma de filas. Combinando los segmentos calculados de filas y columnas se obtienen las ventanas de segmentación. En la figura 5.15(c) se puede ver un ejemplo completo. Para discriminar las ventanas fantasma de las reales se exige que los pixels naranjas dentro de cada ventana superen cierto número. La ventana mayor se considera representativa de la pelota más cercana, y se escoge ésa como pelota a seguir.

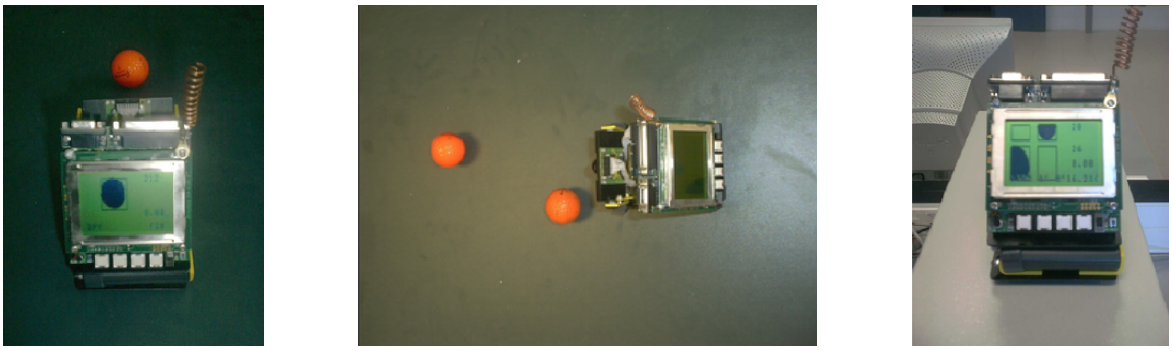


Figura 5.15: *Segmentación con una y dos pelotas en frente*

Con un umbral único, si el umbral es muy bajo entonces las zonas no significativas con cierto ruido llegan a superarlo, creando demasiados segmentos. Por el contrario, si el umbral es muy alto, los segmentos obtenidos se comen buena parte de los objetos, como se puede apreciar en la figura 5.14(b). La técnica de doble umbral supone un compromiso entre ambos extremos.

Adicionalmente, la principal ventaja de esta segmentación es su rapidez. La relativa lentitud del procesador a bordo del EyeBot condiciona la elección de algoritmos de segmentación muy eficientes en tiempo, aunque no ajusten perfectamente las regiones. El tratamiento independiente de filas y columnas agiliza los cómputos, con la limitación de manejar regiones exclusivamente rectangulares. Para el objetivo de este comportamiento estas ventanas han resultado más que suficientes.

Una vez localizada en la imagen la franja de la pelota a seguir, se calcula cuál es el centro del balón. Ese centro será la referencia que se tome para decidir si la pelota está lejos, cerca, alineada o no con el robot. Debido a las sombras el centro de la pelota en la imagen no suele coincidir con el centro de la ventana de segmentación, aunque es una buena aproximación. Si se quiere estimar de modo más preciso, una opción es calcular el centro de masas de los pixels naranjas dentro de la ventana. Otra alternativa que hemos desarrollado estima de modo aún más exacto el centro de la pelota, aprovechando su forma esférica. Debido a su geometría la bola se proyecta siempre como una circunferencia en la imagen[2]. Con toda probabilidad, los pixels con los valores extremos de abcisas y ordenadas dentro de la ventana pertenecen al contorno de la pelota. Cogiendo tres de ellos se puede estimar esa circunferencia de modo preciso, su centro y su radio, como muestra el círculo rojo de la figura 5.13(b).

El ritmo máximo de captura de imágenes que ofrece el sistema operativo es de unas 4 *fps*. Al incluir el procesamiento, el ritmo de este esquema perceptivo baja a unas 3.5 *fps*. Ese escaso retardo adicional se debe a la eficiencia temporal de los algoritmos escogidos. A pesar de la lentitud del procesador (Motorola 68330, 20MHz), esta cadencia todavía deja margen para materializar un comportamiento reactivo. Si hubieramos utilizado algoritmos computacionalmente más pesados, el ritmo habría bajado aún más, impidiendo el desarrollo vivaz del comportamiento.

Es interesante recalcar que no ha sido necesaria ninguna representación espacial o absoluta. Toda la información necesaria para el seguimiento se extrae de la imagen, y así la descripción de la pelota es totalmente subjetiva. No hay ningún *objeto pelota* con tal o cual posición en una representación objetiva del mundo. De hecho, se trabaja sin referencias espaciales o coordenadas absolutas de posición. Por ejemplo, los datos de los odómetros se han ignorado completamente.

Control del movimiento

El esquema perceptivo descrito calcula y refresca continuamente la posición actual de la pelota en la imagen. Simultáneamente, el esquema de actuación toma en cuenta esa información (su última actualización) para determinar cuál es la velocidad de movimiento adecuada con objeto de generar la conducta de seguimiento.

Este esquema de actuación se ha concebido como un doble control en velocidad que trata de centrar la pelota en la imagen, tanto en el eje vertical como horizontalmente. Por un lado, las desviaciones laterales ($X_{pelota} - X_{centro\_imagen}$) se intentan corregir modificando la velocidad de giro del robot. Si la pelota aparece en la zona central el robot no girará, si aparece desviada hacia la izquierda el robot girará en ese sentido tratando de perseguirla. De modo recíproco si se desvía a la derecha. Por otro lado, las desviaciones verticales ($Y_{pelota} - Y_{centro\_imagen}$) se intentan corregir cambiando la velocidad de tracción. Si aparece en la zona central el robot no deberá avanzar ni retroceder. Si la pelota aparece en la zona alta significa que está demasiado lejos, y el robot intentará acelerar para perseguirla. Si por el contrario aparece en la parte inferior, el robot deberá retroceder porque está demasiado cerca del balón[3].

La figura 5.16 muestra el perfil de velocidades que realimentan al giro dependiendo de las desviaciones horizontales de la pelota. En las abscisas está la diferencia entre la X_{pelota} y X central de la imagen, X_{centro_imagen}. En las ordenadas se tiene la velocidad de giro que se comanda a los motores. A grandes rasgos, presenta tres zonas: una zona inicial en la cual no se ordena movimiento porque la pelota se considera centrada, una segunda zona que materializa una especie control proporcional y una zona de saturación en la que la velocidad ordenada es la misma independientemente de la desviación. Una vez que el esquema determina la V y la W comandadas, una librería interna proporcionada por fabricante traduce esas consignas a valores numéricos para los motores de cada rueda.

Si el robot se encuentra en movimiento, entonces la velocidad comandada es la del perfil continuo. Por el contrario, si el robot está parado la velocidad que realmente se ordena viene marcada por la línea discontinua de la figura 5.16, y se denomina *pico de arranque*. Esta diferencia fue determinante

[2] salvo en distancias excesivamente próximas al foco de la cámara

[3] El control de velocidad de tracción se podía haber hecho atendiendo al tamaño de la pelota en la imagen: avanzar si es menor que cierta referencia y retroceder si es mayor. Sin embargo, ofrecía el inconveniente de calibrar el sistema dependiendo de la bola que se quiere seguir

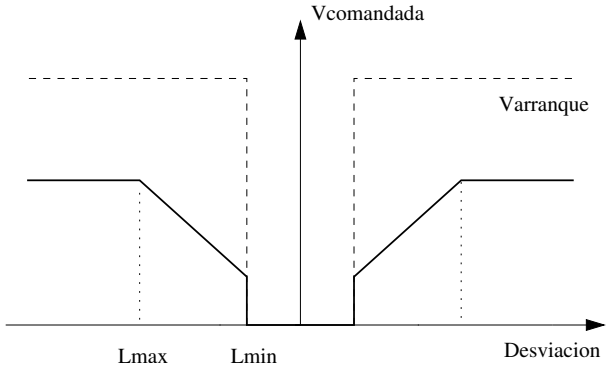


Figura 5.16: *Perfil de realimentación*

para que este comportamiento se generara correctamente. Los perfiles como el continuo funcionan bien cuando el robot ya se está moviendo, pero con esos valores no es capaz de poner en marcha la plataforma cuando parte del reposo. Si para evitar esto utilizamos un perfil similar con valores mayores, entonces sí arranca, pero el control en movimiento es demasiado brusco e inestable, el robot oscila y pierde enseguida la pelota. La causa de esta disfunción es que el rozamiento estático que ofrece la superficie al movimiento del robot es mucho mayor que el rozamiento dinámico. Para resolverlo adecuadamente utilizamos el pico de arranque, que es capaz de poner en marcha la plataforma desde el reposo y una vez que se está moviendo se utilizan valores más suaves para realimentar. Se pueden encontrar más detalles en [Gómez *et al.*, 2003].

El ritmo del esquema de actuación se ha fijado a 5 iteraciones por segundo, cerca del ritmo al cual el esquema perceptivo entrega nueva información. El resultado conseguido con estos dos esquemas es un seguimiento completo de la pelota en movimiento, siempre que no sea muy veloz. El robot describe trayectorias suaves y consigue no perderla si ésta no se mueve demasiado deprisa. El principal cuello de botella identificado es la relativa lentitud del procesador, que ralentiza el esquema perceptivo a un ritmo de 3.5 iteraciones por segundo. Como quedó demostrado en otro experimento posterior [Martínez, 2003], si el esquema perceptivo fuera capaz de analizar más imágenes por segundo, entonces actualizaría su información con más frecuencia, y la calidad del seguimiento sería mayor.

5.3.2. Conducta sigue-pared

Otro comportamiento que hemos desarrollado en el EyeBot utilizando la descomposición en esquemas de JDE es el seguimiento de paredes. El objetivo concreto es que el robot, autónomamente, sea capaz de avanzar paralelo a una pared situada a su derecha, a una cierta distancia. Si llega a una rincón el robot debe girar a su izquierda para seguir a la pared frontal. Si llega a una esquina (esquina convexa) el robot debe girar a su derecha para continuar paralelo a la nueva pared. Para ello, el único sensor permitido es la cámara local a bordo del EyeBot, y como actuadores se emplean los motores de sus dos ruedas motrices.

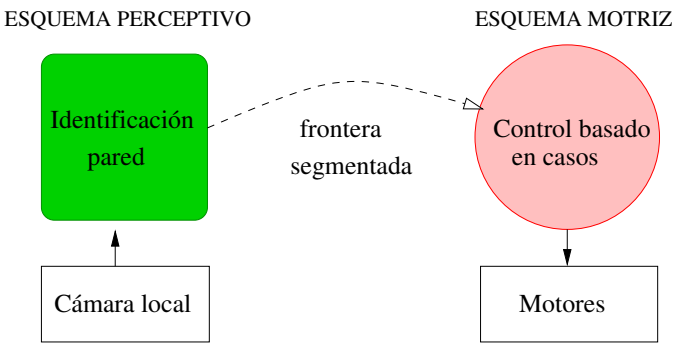


Figura 5.17: *Diseño en esquemas del comportamiento sigue-pared*

El diseño en esquemas de este comportamiento se muestra en la figura 5.17, y es enteramente similar al `sigue-pelota` descrito anteriormente.

Percepción de la pared

El estímulo relevante para este comportamiento es obviamente la pared, pero representada en un formato operativo para el robot (para que se puedan tomar decisiones sobre él) y extraíble desde la imagen cruda (para que esté anclado continuamente en la realidad sensorial). La construcción de ese estímulo puede apoyarse en unos invariantes que sean lo más genéricos posible, pero no es necesario que funcionen en *todas* las condiciones imaginables. Es decir, tienen que funcionar en los contextos en que se puede activar el esquema, no en más. Por ejemplo, no se requiere que funcionen cuando el robot sea volteado, o cuando la pared y el suelo sean indistinguibles. Un invariante adecuado que hemos detectado de la pared es precisamente su frontera con el suelo, que se presenta en practicamente cualquier situación. Su identificación y caracterización desde el flujo de imágenes crudo es la tarea de este esquema perceptivo. Ese procesamiento se ha dividido en tres fases, cuyos resultados se aprecian en la figura 5.18.

(a) (b) (c)

Figura 5.18: *Percepción de la pared en la imagen*

Primero, un filtro de color se encarga de seleccionar los pixels del suelo, que se identifica por un color característico. El suelo de laboratorio donde se hicieron las pruebas tiene un marcado tono verdoso, que se discrimina con $60 < R < 140$, $70 < G < 150$, $50 < B < 100$, $G < R$ y $B > G$. Segundo, una vez identificados los pixels del suelo, se buscan los puntos frontera con la pared. Para ello se recorre cada columna de la imagen de modo ascendente, buscando la ordenada en la cual los pixels dejan de tener el color del suelo. Ese corte es el inicio de la pared. La búsqueda se realiza con cierta tolerancia al ruido, lo cual permite saltarse los pixels aislados.

Después de los dos primeros pasos se tienen los puntos frontera en la imagen, pero inconexos y sin estructura. Para facilitar las decisiones de control se necesita un pequeño análisis, como la identificación de segmentos dentro de los puntos frontera. El número de ellos, así como su posición e inclinación permitirán decidir el movimiento conveniente. Para calcular los segmentos hemos desarrollado un algoritmo propio muy eficiente en tiempo, condicionados por el procesador relativamente lento que tiene el EyeBot.

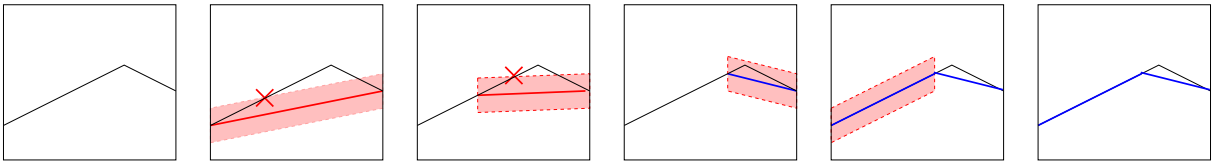


Figura 5.19: *Algoritmo de segmentación de la frontera*

El algoritmo es abductivo, es decir, hipotetiza segmentos que se confirman o descartan con las evidencias de la imagen. Su funcionamiento se muestra en la figura 5.19. Primero, hipotetiza un segmento desde el primer al último pixel de la frontera, de izquierda a derecha. Alrededor de ese segmento se construye una franja de confirmación. Si todos los pixels frontera intermedios están contenidos en la franja, entonces el segmento se acepta por válido ya que explica, con cierta tolerancia, todos los pixels frontera observados. Para confirmarlo se recorre la frontera desde la izquierda a la derecha, y si cierto pixel del borde suelo-pared queda fuera de la franja entonces el segmento se desecha.

Precisamente desde ese pixel disonante se hipotetiza un nuevo segmento hasta el último pixel, y el proceso de confirmación se repite para este nuevo segmento hipotetizado. De este modo, al final de la primera pasada se encuentra un segmento que explica la parte final, más a la derecha en la imagen, de la frontera. En la segunda pasada se trata de segmentar el resto de la frontera, desde la columna más a la izquierda hasta el comienzo del segmento ya admitido. Por ejemplo, en la figura 5.19 basta un segundo segmento para completar todo el borde pared-suelo. En el caso general, después de un número finito y pequeño de pasadas se tiene una segmentación completa de la frontera.

Este algoritmo tiene la ventaja de ser sorprendentemente rápido, sobre todo para casos simples cuando la línea frontera puede explicarse con pocos segmentos. Por ejemplo, en el caso de *pared infinita* (figura 5.20(e)), una única pasada a la frontera es suficiente para tenerla segmentada. Además, debido a la franja es muy robusto al ruido, mientras que otros algoritmos basados en derivada son muy sensibles.

De nuevo la descripción del estímulo es subjetiva y operativa, al estilo de los sistemas basados en comportamientos y contrastando con las descripciones objetivas de los sistemas deliberativos clásicos. Con esto se demuestra que no es necesario procesar la imagen para obtener una distancia y orientación relativas entre la pared y el robot. Se puede enganchar actuaciones a descripciones menos simbólicas de la realidad. Adicionalmente, y en contraste con los reactivos puros, el estímulo no se encuentra directamente en el sensor, sino que es necesario cierto procesamiento, en este caso el filtrado y la segmentación.

Control del movimiento

Paralelamente al esquema perceptivo, el esquema de actuación va tomando decisiones de movimiento sobre la representación del borde suelo-pared como una colección de segmentos. Para determinar los valores convenientes de velocidad de giro y tracción de la plataforma, hemos utilizado un control basado en casos. Hemos identificado 6 casos relevantes, dependiendo de los segmentos: *todo-pared* (figura 5.20(g)), *todo-suelo*, *pared infinita* (figura 5.20(e)), *esquina concava* (figura 5.20(f)), *esquina convexa* y *desestructurado* (figura 5.20(h)).

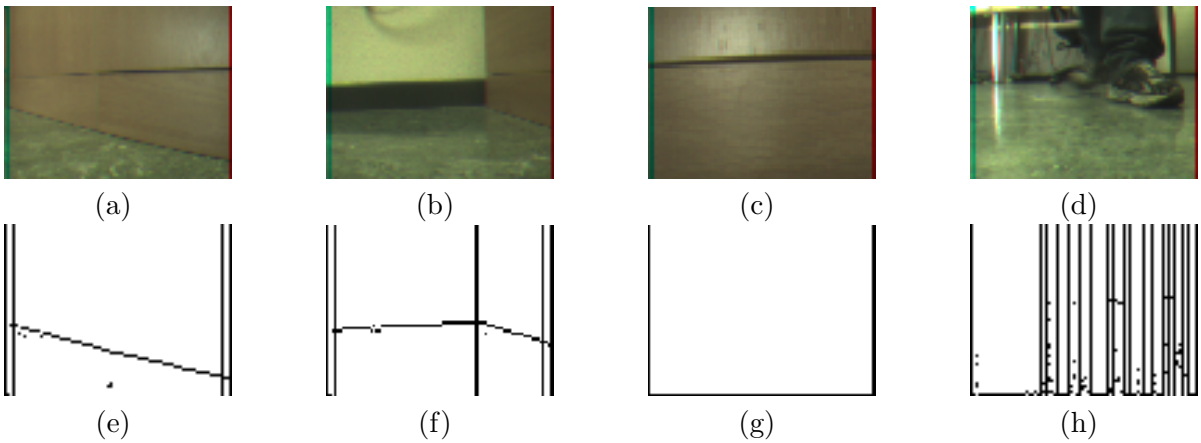


Figura 5.20: *Percepción de la pared en la imagen*

El caso más sencillo es el de *pared infinita*, que se identifica cuando la frontera tiene un único segmento (segmento guía), con cierta inclinación, o dos siendo el más a la derecha horizontal y situado en la parte baja de la imagen. En esta última situación el segmento guía es el no horizontal. El control adecuado en ambas depende de la inclinación del segmento guía, y de la posición de su extremo derecho. Si es similar a cierta inclinación de referencia el robot debe seguir recto (velocidad de giro nula, cierta velocidad de avance) porque se desplaza paralelo a la pared. Si la inclinación es más pronunciada, entonces debe girar suavemente hacia la derecha, y si es menor debe girar ligeramente hacia la izquierda. Se pueden encontrar más detalles en [Gómez *et al.*, 2003].

El caso de *esquina cóncava* (es decir, rincón) se identifica por dos segmentos, el primero de ellos horizontal y el segundo con cierta pendiente. En este caso, dependiendo de la altura del segmento

horizontal el robot se encuentra más o menos lejos de la pared frontal y debe empezar a girar hacia la izquierda para ir alineándose con ella.

El caso de *todo-pared*, la frontera aparece como un único segmento situado en la parte inferior de la imagen, pues no hay pixels con el color del suelo en la imagen. En esta situación, el robot debe girar firmemente hacia la izquierda pues está demasiado cerca de la pared y el choque es inminente si no corrige su trayectoria. El caso recíproco es el caso de *todo-suelo*, con la frontera en la parte superior de la imagen, e incluso el caso *desectructurado*. Cualquiera de ellos puede aparecer si el robot ha girado demasiado y tiende a alejarse de la pared. En ambos es recomendable que el robot gire hacia la derecha, tratando de buscar la pared que ha perdido de la imagen.

Para seguir paredes por la izquierda habría que desarrollar un esquema de control análogo. El esquema perceptivo sería perfectamente utilizable, pero la interpretación que haría el esquema de actuación sería totalmente diferente. Las decisiones de control serían simétricas, analizando los segmentos frontera en sentido opuesto al actual, es decir, de derecha a izquierda.

5.3.3. Avance rápido sorteándo obstáculos

Otro comportamiento que hemos materializado con JDE es el `avance-rápido-sorteando--obstáculos`, que sirve de navegación local segura. Esta conducta acepta como parámetro de modulación una orientación objetivo, en la que el robot debe tratar de avanzar. Debe trasladarse en esa orientación absoluta a velocidad alta sin chocar con ningún obstáculo. Obviamente, se permiten pequeñas desviaciones para sortear los obstáculos que pudieran aparecer, siempre y cuando a medio plazo se avance de modo efectivo en la dirección deseada.

Hemos adaptado el enfoque de ventana dinámica [Fox *et al.*, 1997] y el método de curvatura--velocidad [Simmons, 1996], que es su inspiración original. La implementación de este algoritmo con esquemas ha resultado directa, con un esquema perceptivo que elabora la representación de ocupación del entorno local y el esquema de actuación, que toma la decisión de movimiento siguiendo el algoritmo de la ventana dinámica (figura 5.21).

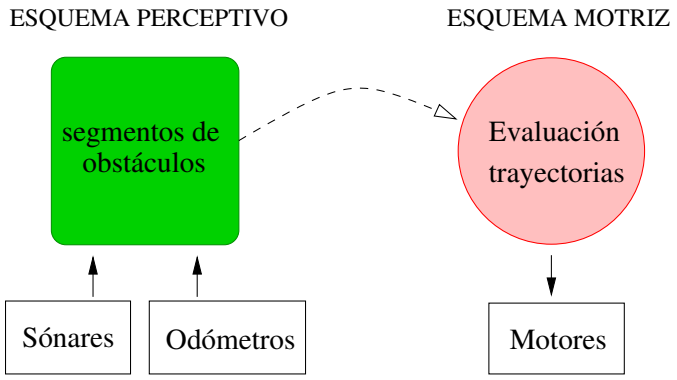
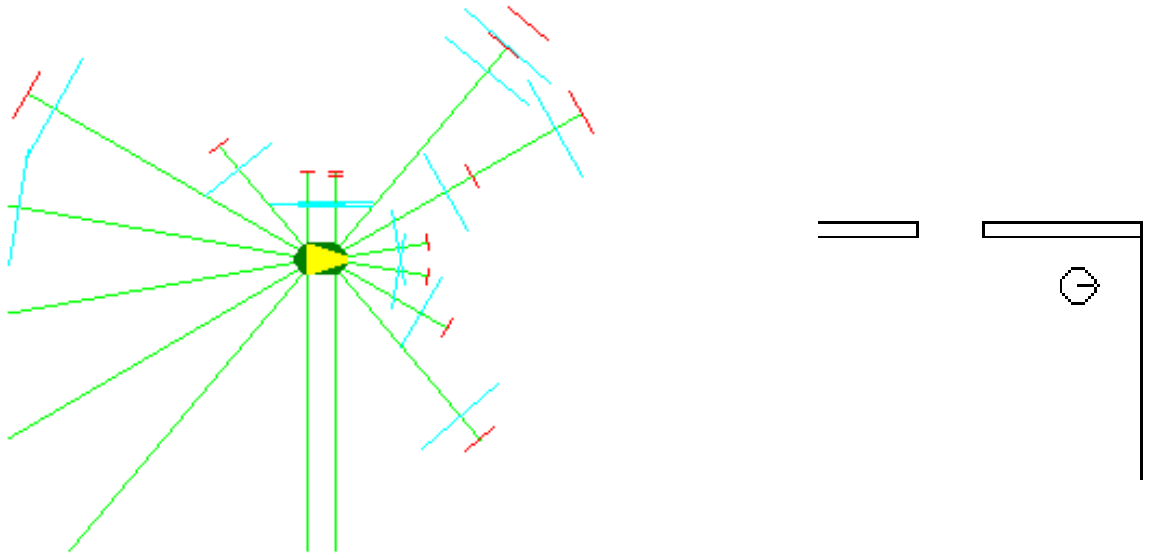


Figura 5.21: *Esquemas para la navegación local con ventana dinámica*

En este caso lo hemos implementado en el robot Pioneer, utilizando el servidor `otos` y la plataforma software JDE. Por ello, puede ejecutarse tanto en el portátil a bordo del robot, como en cualquier otro ordenador conectado a la red. Además, el desarrollo de este comportamiento es una muestra de la utilidad del simulador, porque fue probado y depurado primeramente en él. Una vez que estuvo maduro, se hicieron las pruebas y ajustes experimentales en el robot real.

Percepción de los obstáculos

El esquema perceptivo elabora una representación que consiste en una colección de segmentos que se extraen de las tres últimas observaciones de la corona completa de sónares. Cada lectura de un sónar se asimila como un segmento, situado a la distancia medida por el sensor, y cuya anchura crece linealmente con la distancia. Esta anchura refleja la incertidumbre angular del sónar, cuya onda se propaga siguiendo una apertura angular de unos 20^o, como ya vimos en el capítulo 4.



(a) (b)

Figura 5.22: *Representación de los obstáculos (a) cuando el robot está en una esquina (b)*

Para poder considerar al robot como un punto (lo cual simplifica enormemente los cálculos), los segmentos se alargan por cada extremo y se acercan a los sensores una cierta distancia, precisamente el radio del robot más un margen de seguridad. La figura 5.22(b) muestra el rincón en el que se encuentra el robot (en el simulador), y la vecina figura 5.22(a) las medidas sónares instantáneas, así como los segmentos construidos y alargados por este esquema perceptivo.

La velocidad de iteración de este esquema es de unas tres iteraciones por segundo. Está limitada por la máxima frecuencia de captura de los sensores sónares, que es de una lectura de la corona completa cada 330 ms aproximadamente.

Evaluación de trayectorias

El esquema de actuación de este comportamiento determina cuál es el mejor movimiento en el instante actual, es decir, cuáles son las velocidades de tracción y giro óptimas teniendo en cuenta las velocidades actuales (el robot tiene inercia), la situación de los obstáculos locales y la orientación en la que se quiere avanzar.

Para ello se trabaja con el espacio de velocidades, que aparece en la figura 5.23(b). Las ordenadas de este espacio representan velocidades de tracción V, y las abscisas velocidades angulares W, en sentido horario las positivas y antihorario las negativas. En él se define la *ventana dinámica* como el subconjunto de velocidades que el robot podría alcanzar hipotéticamente en la siguiente iteración del esquema, contando con las velocidades actuales y las aceleraciones máximas de la plataforma física. Cada punto de esa ventana supone cierta combinación de V/W, y suponiendo valores constantes, trayectorias circulares de diferente curvatura en el espacio real. Se puede encontrar la justificación de esta aproximación en [Lobato, 2003].

El algoritmo evalúa la trayectoria generada por cada combinación, y descarta las que comprometen la seguridad del robot, bien porque provocan el choque, bien porque impiden frenar a tiempo para evitar la colisión. Además, evalúa las trayectorias válidas según una función objetivo, la cual refleja una combinación de criterios. Los criterios incorporados son: que la velocidad de tracción sea alta, que deje al robot en una orientación similar a la deseada, y que pase lo más lejos posible de los obstáculos del entorno.

Por ejemplo, en la situación de la figura 5.23(a), el robot se encuentra avanzando por el pasillo y girando suavemente hacia la derecha, y tiene que decidir qué velocidades comandar para el siguiente instante. El espacio de velocidades y su ventana dinámica asociada se muestra en la figura 5.23(b). Las

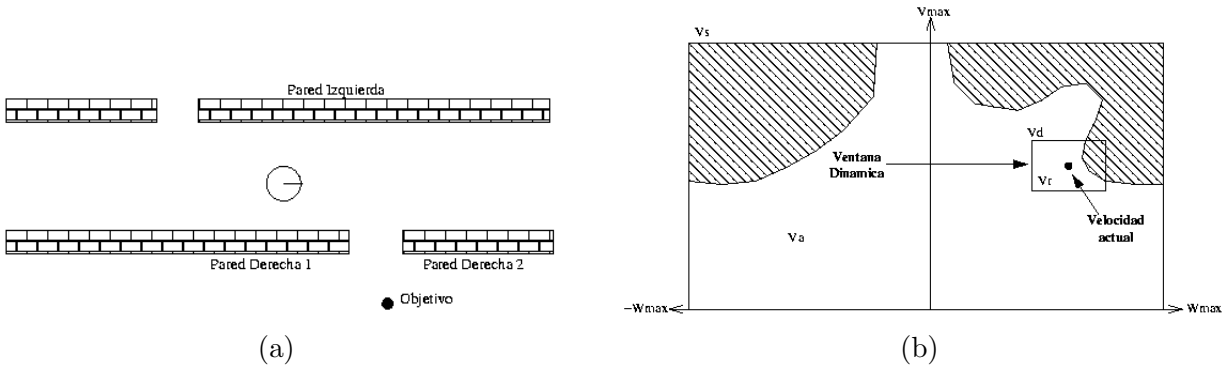


Figura 5.23: *Evaluación de trayectorias de la ventana dinámica*

partes sombreadas son combinaciones descartadas, que se corresponden a giros bruscos tanto hacia la derecha como hacia la izquierda, porque provocan la colisión con las paredes del pasillo. El golfo que hay en las abscisas positivas corresponde al hueco que hay en la pared derecha. La ventana dinámica es el recuadro situado alrededor del punto. Los valores V/W para la siguiente iteración salen de ese rectángulo, calculando el máximo de la función objetivo para los puntos interiores. Se pueden encontrar más detalles en [Lobato, 2003].

5.4. Comportamientos con jerarquías simples

En la anterior sección hemos descrito varios comportamientos sencillos, implementados con esquemas pero sin necesidad de jerarquía. En esta sección vamos a explicar varios comportamientos ligeramente más complejos, que ya ejemplifican el uso de la jerarquía dentro de JDE.

5.4.1. Comportamiento ir-a-punto

Como ya se comentó en la sección 3.3, esta conducta de navegación local se encarga de gobernar los movimientos del robot para llegar a un cierto punto cercano, que se especifica en coordenadas odométricas. Si el camino está libre el robot debe orientarse hacia el destino a la vez que avanza hacia él. Si durante su camino se encuentra con algún obstáculo, dinámico o no, debe sortearlo y seguir avanzando hacia el destino, evitando en todo momento la colisión con él.

El diseño con esquemas que hemos implementado para generar este comportamiento es el de la figura 5.24(a). En este caso la jerarquía consta de dos niveles. El esquema `ir-a-punto` es el principal, y el responsable último de la navegación. Admite como parámetro de modulación las coordenadas del punto destino. Al ser un parámetro de JDE puede variar dinámicamente, es decir, el usuario de este esquema puede cambiar sobre la marcha el destino al que quiere que el robot llegue. En los experimentos realizados el objetivo lo especificaba el usuario humano pinchando en una representación, a vista de pájaro, de las proximidades del robot. Este destino se expresa en coordenadas odométricas, con lo que el esquema puede calcular en todo momento la distancia y el ángulo que lo separan de él, utilizando sus odómetros.

Para materializar su objetivo, este esquema despierta a tres esquemas de actuación: `para`, `sigue--pared` y `avanza`. En cada iteración el esquema `ir-a-punto` compara la lectura de los odómetros con las coordenadas de su destino. Si comprueba que ha llegado al punto final, con cierta tolerancia, entonces desactiva a sus hijos y él mismo pasa a estado PREPARADO, pendiente de que le dé un nuevo objetivo. Sus precondiciones pasan por tener un objetivo insatisfecho. Si aún no ha llegado, entonces simplemente refresca la modulación de sus tres esquemas hijos. Por ejemplo, actualiza la orientación local de avance poniendo justo la que apunta en dirección al destino, que puede haber cambiado respecto de la iteración anterior.

El esquema `para` detiene al robot si hay obstáculos demasiado cerca. Su principal parámetro de modulación es la *distancia de seguridad* por debajo de la cual se activa. Si los obstáculos no invaden esa *distancia de seguridad* entonces sus precondiciones no se satisfacen y el esquema permanece ALERTA.

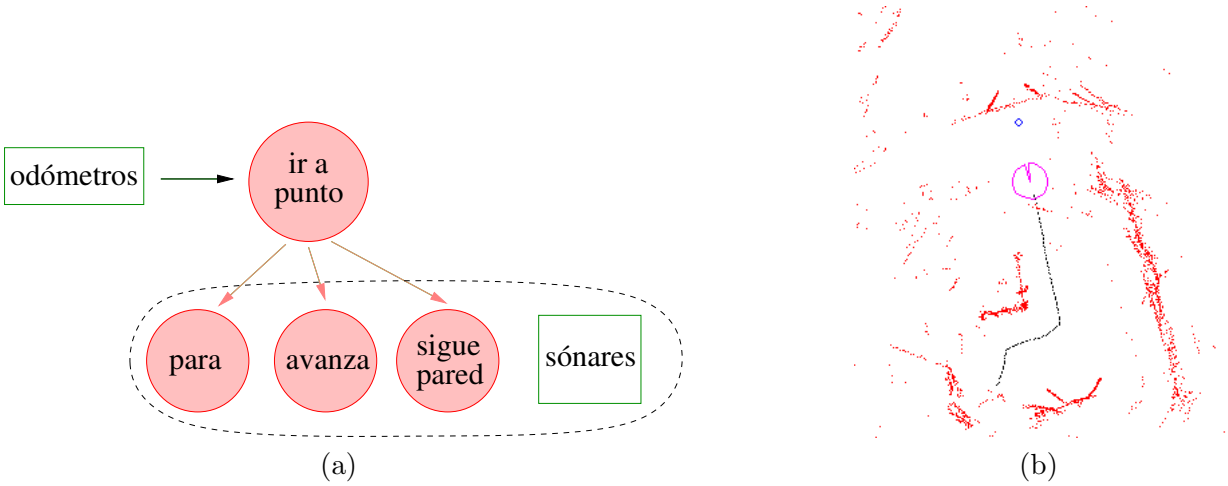


Figura 5.24: *Esquemas del comportamiento ir-a-punto (a) y trayectoria descrita (b)*

La información de los obstáculos la extrae de las últimas medidas ultrasónicas, por lo que necesita que el esquema `sónares` esté activado.

El esquema `avanza` mueve el robot en cierta orientación, de modo rápido si no percibe obstáculo en esa dirección. Esa orientación es un parámetro de modulación suyo. Si el robot está muy desalineado respecto de ella, entonces girará sin avanzar hasta orientarse en esa dirección. Cuando la desviación sea moderada emprenderá el avance. Si el camino está despejado en la orientación deseada, entonces el movimiento será más veloz que si percibe algún obstáculo a cierta distancia. De hecho, las precondiciones de este esquema chequean que haya camino libre en esa orientación hasta cierta distancia, y si no lo hay, entonces el esquema no tratará de activarse.

Está programado como un controlador borroso, utilizando la librería `fuzzylib` comentada en el capítulo 4. La figura 5.1 muestra las reglas utilizadas cuando el esquema efectivamente gana la competición por el control en su nivel. El control de la velocidad angular no depende de la distancia al obstáculo en la orientación objetivo (`d_obstaculo`), sólo del error angular (`alfa`) respecto de la orientación objetivo. Este control se implementa como un control proporcional, con la saturacion propia del sistema borroso. El control de la velocidad de tracción depende tanto de la distancia al obstáculo como del error angular. Si éste es muy alto entonces no se avanza, sólo gira.

```
IF ( alfa = alto_pos ) THEN ( w = alta_pos )
IF ( alfa = medio_pos ) THEN ( w = baja_pos )
IF ( alfa = nulo ) THEN ( w = nula )
IF ( alfa = medio_neg ) THEN ( w = baja_neg )
IF ( alfa = alto_neg ) THEN ( w = alta_neg )

IF ( d_obstaculo = baja ) THEN ( v = nula )
IF ( alfa = alto_neg ) THEN ( v = nula )
IF ( alfa = alto_pos ) THEN ( v = nula )

IF ( d_obstaculo = media ) AND ( alfa = nulo ) THEN ( v = baja )
IF ( d_obstaculo = media ) AND ( alfa = medio_neg ) THEN ( v = baja )
IF ( d_obstaculo = media ) AND ( alfa = medio_pos ) THEN ( v = baja )
IF ( d_obstaculo = alta ) AND ( alfa = nulo ) THEN ( v = alta )
IF ( d_obstaculo = alta ) AND ( alfa = medio_neg ) THEN ( v = alta )
IF ( d_obstaculo = alta ) AND ( alfa = medio_pos ) THEN ( v = alta )
```

Tabla 5.1: *Reglas borrosas del esquema* `avance`

El esquema `sigue-pared` necesita los datos sensoriales de ultrasonido y mueve el robot paralelo al obstáculo que hay en cierta orientación. Por sí solo materializa la conducta homónima. Su reutilización para el comportamiento `ir-a-punto` es un ejemplo de reutilización de esquemas: empleado

con otra región de activación y en conjunción con otros esquemas, puede ayudar a conseguir otro comportamiento completamente diferente. Como precondiciones, este esquema necesita que efectivamente haya una pared a la que seguir, lo que en este contexto equivale a que haya un obstáculo al que contornear.

El esquema `ir-a-punto` modula en sus hijos unas regiones de activación que son aproximadamente disjuntas. Como distancia de seguridad para `para` elige 20 cm. Como distancia mínima libre para `avance` escoge 100 cm en la dirección del punto destino. La precondición que exigirá para su activación el esquema `sigue-pared` es precismente la presencia de obstáculos en esa zona. El arbitraje para casos de solape y vacío de control elige la opción más conservativa, con algunas excepciones. Por ejemplo, si la colisión por el control se produce entre `para` y `avanza`, entonces da prioridad a este último, pues muy probablemente se debe a un obstáculo muy próximo pero en la zona trasera o lateral del robot, que no impide la progresión.

En la figura 5.24(b) se puede apreciar la trayectoria descrita por el robot cuando el esquema `ir-a-punto` está activado y tiene como objetivo, especificado en un parámetro suyo, el punto azul en la parte superior de la figura. Inicialmente el robot realiza un giro sobre si mismo para alinearse en la dirección del objetivo. Posteriormente, gracias al esquema `avanza`, se desplaza hacia el destino hasta que se encuentra un obstáculo. Entonces se activa el esquema `sigue-pared`, que provoca el comportamiento de contorneo del obstáculo. Una vez que el camino hacia el objetivo se despeja, vuelve a tomar el control el esquema `avanza` que consigue llevar al robot hasta el destino. Allí, el propio esquema `ir-a-punto` se desactiva porque su precondición, que es tener un objetivo pendiente, ha dejado de satisfacerse.

Es interesante resaltar que el funcionamiento es enteramente reactivo, no hay ninguna ruta especificada de antemano. Se mantiene la capacidad de respuesta ante obstáculos que irrumpen entre el robot y su punto destino. También es significativo que no hay esquemas perceptivos específicos, porque con los datos sensoriales crudos, sónares y odómetros, ha bastado para un funcionamiento aceptable. Hay muchos modos de conseguir este comportamiento descritos en la literatura. Desde los planificadores locales, hasta los más reactivos como los basados en potencial. Esta implementación con esquemas quizá no es la más eficiente en tiempo, o en cómputo, pero respeta la incrementalidad del sistema y fomenta la reutilización de esquemas.

5.4.2. Saque de banda

Un comportamiento que hemos desarrollado, y que es útil en el entorno de la RoboCup, es el de saque de banda. En concreto, lo hemos generado en el robot EyeBot. Esta conducta consiste en que el robot se acerque a la banda, donde el árbitro ha situado la pelota, y la introduzca en el terreno de juego para que el partido continúe. Para generar esta conducta dentro de JDE se ha diseñado un sistema con dos esquemas perceptivos y cuatro esquemas motores, tal y como muestra la figura 5.25.

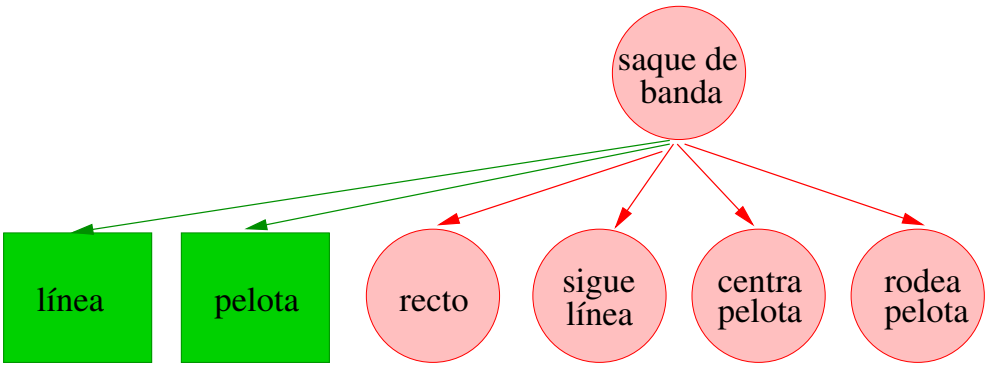


Figura 5.25: *Esquemas para el comportamiento saque de banda*

No tiene una jerarquía muy compleja, pues todos los esquemas excepto el padre se utilizan en el mismo nivel. No obstante, este comportamiento supone un ejemplo más elaborado que los casos sencillos descritos en la sección 5.3. El padre es necesario para unificar toda la funcionalidad en un

único esquema y además mantiene ciertas variables de estado como si se está dentro o fuera del campo, que condicionan el desarrollo de la conducta. Se pueden encontrar más detalles en [Peño del Barrio, 2003].

En cuanto a percepción, los estímulos relevantes identificados para este comportamiento son las líneas del campo y la pelota. Para percibir dichos estímulos hemos desarrollado sendos esquemas: `línea` y `pelota`. Para implementar el esquema `línea` hemos reutilizado el esquema perceptivo del comportamiento `sigue-pared`. Puesto que se apoya en la diferencia de color entre suelo y pared para distinguir la frontera entre ambos, el esquema se puede utilizar de modo análogo para discriminar entre el terreno de juego y la línea que lo delimita. Para identificar la pelota hemos reutilizado directamente el esquema homónimo empleado en el comportamiento `sigue-pelota`.

En cuanto a la actuación, la idea es ir hacia la banda y seguir la línea del campo hasta detectar pelota. Una vez detectada, hacer un giro para que el robot se sitúe por detrás de la pelota, y avanzar recto para introducir de nuevo la pelota en el terreno de juego. JDE, con los esquemas que están latentes en ALERTA permite materializar esta secuencia de modo flexible, saltándose pasos si se presentan oportunidades para ello. Para simplificar el desarrollo de este comportamiento asumimos como hipótesis de partida que no hay obstáculos, es decir, otros jugadores, entre el robot que va a efectuar el saque y la banda.

El esquema `recto` mantiene al robot marchando hacia delante, con una velocidad de rotación nula y una . Se activa cuando los esquemas perceptivos no detectan ni línea ni pelota en las cercanías del robot. La idea es que tarde o temprano el robot llegará a la línea de banda y ésta será percibida. También se activa cuando se ha completado la maniobra de rodear a la pelota y el robot está fuera del campo, para empujar el balón hacia dentro del terreno de juego.

El esquema `sigue-línea` materializa el seguimiento de la divisoria utilizando la información visual del esquema perceptivo. Se activa cuando se ha detectado la línea pero no la pelota. Su objetivo es llevar al robot paralelo a la línea que se ha detectado, por dentro del campo. Su implementación es similar al esquema de actuación en el comportamiento `sigue-pared`.

El esquema `centra-pelota` se activa cuando se ha detectado pelota, para alinearse con ella. Si durante su acercamiento a la banda el robot desemboca en la pelota, la conducta `sigue-línea` no se activará nunca, ya que en ese caso no es necesaria.

El esquema `rodea-pelota` describe un círculo moviéndose durante cierto tiempo a velocidad de rotación y de tracción constantes. Se activa cuando se ha detectado la pelota, ésta se encuentra más o menos centrada en la imagen y el robot está dentro del campo.

Este comportamiento supone un ejemplo de reutilización de esquemas. Los esquemas perceptivos de detección de línea y de pelota son los que se emplearon en el comportamiento `sigue-pared` y `sigue-pelota` respectivamente. En este sentido, hay que destacar nuevamente que no se utilizan los odómetros para nada, ni estimación alguna de posición absoluta.

También se puede apreciar cómo la información que elaboran los esquemas perceptivos se utiliza no sólo en la ejecución de los esquemas de actuación, sino también en el arbitraje entre ellos.

5.4.3. Jugador de la RoboCup

El tercer experimento con la jerarquía JDE consiste en su aplicación para generar los comportamientos de un jugador de fútbol. En concreto, de un jugador dentro de la liga simulada de la RoboCup, participando como atacante. El comportamiento deseado es el que se espera de un delantero jugando al fútbol: meter gol, sortear a los defensas, combinar con otro compañero mejor situado, etc.. Este ejemplo es el más complicado de los que hemos realizado. Como muestra la figura 5.26, la arquitectura implementada en este caso tiene varios niveles, para poder generar la heterogeneidad de movimientos que debe exhibir el delantero dependiendo de la situación concreta.

Como ya se adelantó en el capítulo 4, el entorno software en que se ha desarrollado el jugador es el que se ofrece a los participantes del campeonato simulado. El jugador es un proceso, dentro del cual se ejecutan todas las hebras que materializan sus esquemas. Ese proceso se conecta vía *sockets* al servidor `SoccerServer`, al igual que todos sus compañeros de equipo y los jugadores contrarios. El servidor simula los sensores y materializa los movimientos ordenados por cada jugador. Además, simula la dinámica de la pelota y realiza el arbitraje. Por ejemplo, él determina el modo de juego en

cada momento: prepartido, normal, saque de banda, saque de corner, fuera de juego, lanzamiento de falta, de penalti, etc..

Periódicamente el servidor comunica a cada jugador lo que éste percibe en el campo, bien sea información visual, auditiva o propioceptiva. En cuanto a los datos visuales, el servidor simula un cono de visión para cada jugador, e informa de todos los objetos que caen dentro de ese cono. Dos ejemplos de información propioceptiva son la velocidad actual de movimiento y el nivel de energía (*stamina*) del jugador. Éste último refleja su cansancio y la capacidad de correr velozmente o golpear fuerte el balón.

Estos mismos comportamientos de futbolista han sido materializados con otras arquitecturas. Saffiotti [Saffiotti y Wasik, 2003] en la categoría de perritos y Aguirre [Aguirre *et al.*, 2001] en la simulada, emplean arquitecturas borrosas, con fusión de comandos similar a la de Saphira. Dos ejemplos híbridos son [Han y Veloso, 1998] y [Oller *et al.*, 2000]. En contraste, [Chang *et al.*, 2002] y [Dorer, 1999] abordan el problema desde sendas arquitecturas basadas en comportamientos.

Descomposición en esquemas

La conducta del jugador que hemos implementado se organiza siguiendo los esquemas de la figura 5.26. La percepción se ha dividido en dos esquemas: `sensor` y `avsensor`. Mientras que la perpeción necesaria es relativamente sencilla, la actuación se ha descompuesto en cuatro niveles que desglosaremos a continuación. El ejemplo de esta sección hace hincapié en la descomposición de la actuación a la JDE, y da varias muestras de cómo unos esquemas pueden usar jerárquicamente a otros. Se pueden encontrar más detalles en [Martínez Gil, 2003].

Dentro del jugador se ha desarrollado siguiendo JDE, el esquema `sensor` recoge la información del servidor a través de la conexión de red. Este esquema se limita a poner en el proceso del jugador los datos sensoriales crudos que proporciona el simulador `SoccerServer`. Cada 50 ms recibe datos como la información subjetiva de la pelota, de las dos porterías, el modo de juego actual, etc. Adicionalmente, estima la posición absoluta del delantero en el terreno de juego, utilizando la lista de objetos que se observan en el cono visual subjetivo. Para ello emplea una librería proporcionada por la organización RoboCup junto con el servidor.

Un segundo esquema básico, denominado `motor`, se encarga de enviar al servidor, vía *sockets*, los comandos de movimiento que decide la jerarquía de esquemas. Las principales actuaciones que permite el simulador son `turn` (gira el cuerpo del jugador ciertos grados), `dash` (acelera el movimiento con cierto empuje), `kick` (chuta la pelota con cierta fuerza en cierta dirección), y `catch` (atrapa la pelota, para el rol de portero). Tanto `motor` como `sensor` son esquemas de servicio que se encuentran permanentemente activos. Su función es básicamente la de establecer comunicación con el servidor, en ambos sentidos.

El segundo esquema perceptivo, `avsensor`, elabora información más avanzada, como evaluar si se observa algún compañero en el cono de visión, o algún contrario, y estimar su distancia y orientación relativas. También estima si el jugador está en su zona de referencia o no. Estas zonas se idearon para organizar el comportamiento del equipo, adscribiendo a cada rol una zona de referencia en el campo. Estas posiciones permiten una cierta coordinación implícita entre los jugadores, que se reparten con cierto sentido por el terreno de juego. Por ejemplo, la zona de referencia para el delantero centro incluye el área contraria y la zona central próxima, para los extremos incluye la parte más adelantada de la banda. El comportamiento del jugador dependerá de que se encuentre en su zona o no.

Con respecto a la actuación, en primera instancia, el jugador estará atacando, volviendo a su posición de referencia o marcando al contrario. Para materializar cada una de estas conductas, hemos desarrollado los esquemas `atacar`, `volver` y `marcar` respectivamente. La competición por el control entre ellos se resuelve con la información aportada por `sensor` y `avsensor`.

El esquema `volver` consigue que el jugador regrese a su posición de referencia. Se activa cuando el jugador se ha alejado de su localización de referencia y no ve la pelota.

El esquema `marcar` logra que el robot simulado se acerque al contrario más próximo para marcarlo, entorpecer sus movimientos y que pueda recibir el balón. Se activa cuando el modo de juego indica que va a sacar el equipo adversario, bien una falta, un saque de banda, etc. Consiste en mantener al jugador cerca del contrario, y si no lo ve, girar en redondo para localizarlo y hostigarlo.

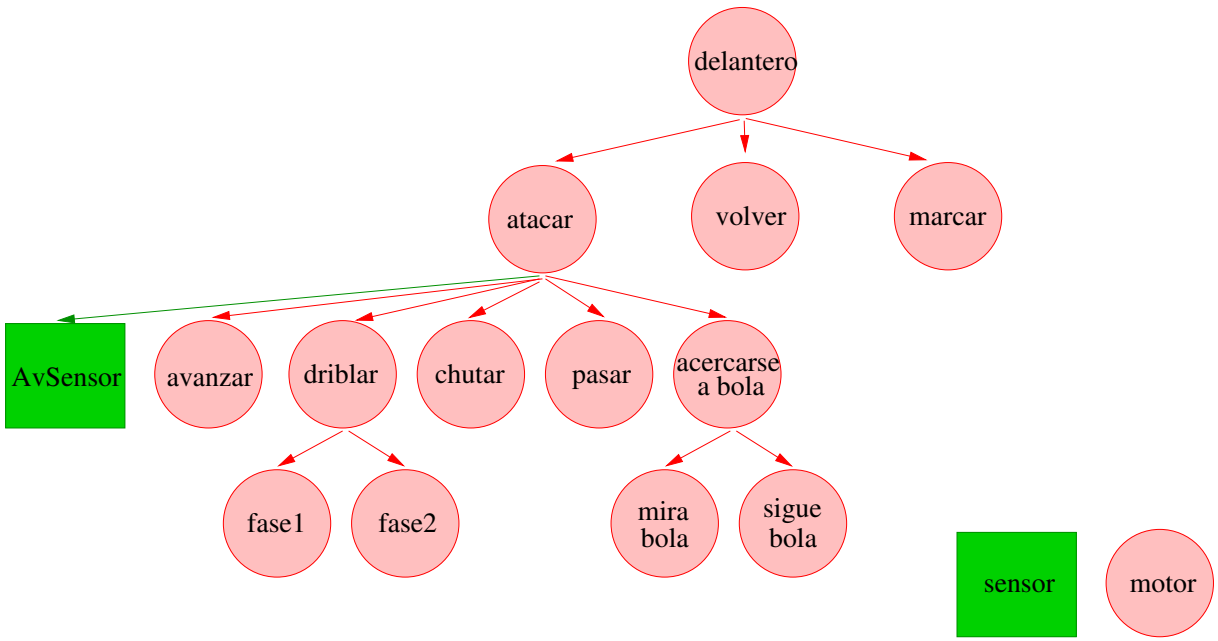


Figura 5.26: *Esquemas JDE del sistema de comportamientos para el jugador de RoboCup*

El esquema `atacar` tiene por objetivo conseguir la pelota y meter gol con ella. Se activa cuando el jugador está dentro de su zona de referencia de delantero, o bien la pelota está cerca. Este esquema no emite órdenes directas a los actuadores, sino que emplea a otros esquemas para materializar su objetivo: `pasar`, `driblar`, `avanzar`, `chutar` y `acercarse-a-bola`, que veremos a continuación.

Comportamientos de ataque

Cuando en el primer nivel gana la competición el esquema `atacar`, entonces entran en estado ALERTA sus cinco hijos, entre los que se establece una nueva competición.

El esquema `pasar` envía la pelota a un compañero que se encuentre más cerca de la portería contraria. Su activación se produce cuando, teniendo la pelota cerca de sus pies, el jugador detecta un compañero por delante suya. Entonces chuta la pelota en esa dirección con una fuerza proporcional a la distancia relativa estimada. Para ello utiliza el comando `kick`.

El esquema `chutar` trata de meter gol disparando la pelota hacia la portería contraria. Se activa cuando la pelota está cerca de sus pies y la portería del adversario esté más próxima que cierta distancia umbral. Consiste en un comando `kick` en dirección al centro de la meta contraria.

El esquema `acercarse-a-bola` localiza la pelota y se acerca hasta ella. Se activa cuando la pelota está lejos o no se percibe. Engloba a dos esquemas hijo: `mira-bola` y `sigue-bola`, que toman el control alternativamente. El primero se activará cuando no se vea la pelota en el cono de visión local, o cuando ésta aparezca descentrada. Consiste en girar el jugador hasta centrar la pelota en el cono. El segundo se activará cuando la pelota esté centrada ($\pm 10^o$), y consiste en avanzar hacia la pelota, corrigiendo ligeramente el rumbo si es necesario.

El esquema `driblar` trata de esquivar a un jugador contrario mientras se avanza con la pelota, realizando un autopase. Se activa cuando la pelota está cerca de los pies del jugador y se detecta un oponente en la dirección de avance hacia la meta contraria. Es un esquema compuesto que tiene dos hijos `driblar-fase1` y `driblar-fase2`. Inicialmente, el padre activa a `driblar-fase1` para chutar el balón desviándolo 45^o de la dirección del contrario. El padre comprueba permanentemente que la pelota deja de estar cerca de los pies del jugador. Cuando esto ocurre, duerme a su primer hijo y activa a `driblar-fase2`. Éste hijo comanda a los motores una velocidad muy alta y va corrigiendo la trayectoria hasta reencontrarse con la pelota.

El esquema `avanzar` consigue que el jugador se aproxime con la pelota a la portería contraria. Se activa cuando la pelota está cerca de los pies, la distancia a la meta del oponente supera cierto umbral

(en caso contrario podría `chutar`), no hay ningún compañero delante (en caso contrario podría `pasar`), ni ningún adversario cerca (en caso contrario podría `driblar`). Su actuación de salida es una patada al balón en dirección a la meta del adversario, pero no tan fuerte como en `chutar`. La progresión del jugador con la pelota se produce como efecto de la alternancia entre este esquema y el esquema `acercarse-a-bola`.

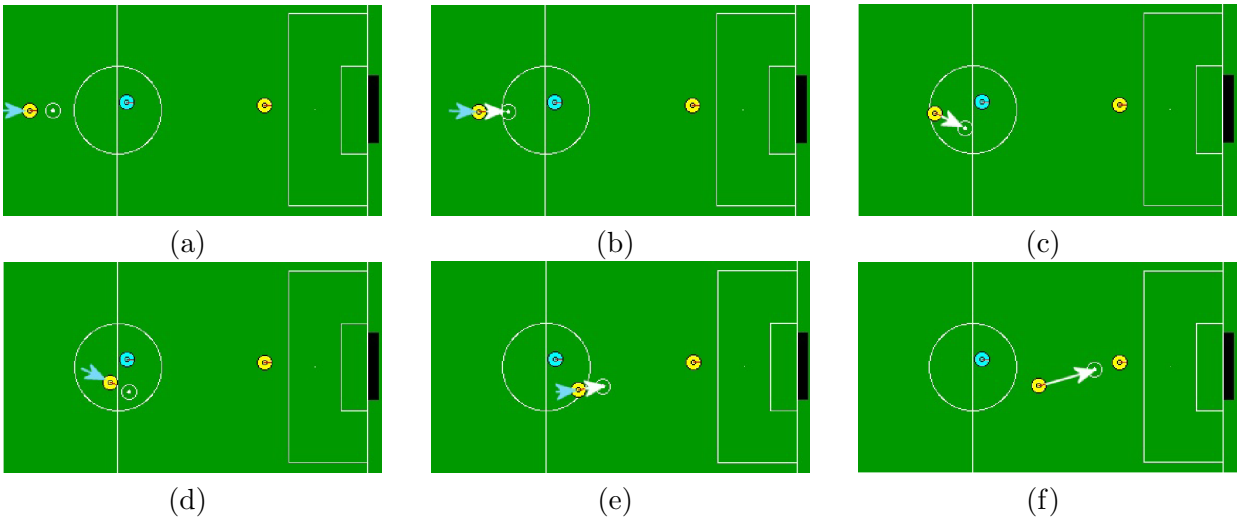


Figura 5.27: *Escenario con comportamiento driblar y pasar*

En la figura 5.27 se tiene una demostración de la jerarquía en acción. El jugador inicialmente se aproxima a la pelota gracias al esquema `acercarse-a-bola` (figura 5.27(a)). Cuando llega hasta ella, se activa el esquema `avanzar`, con el que empieza su progresión hacia la otra portería (figura 5.27(b). Cuando detecta la presencia de un oponente bloqueando el camino hacia la meta contraria entra en funcionamiento `driblar`. En un primer movimiento (figura 5.27(c)) realiza el autopase. En una segunda maniobra (figura 5.27(d)) acelera hasta recuperar de nuevo la pelota. Superado el adversario sigue avanzando hasta que percibe un compañero más adelantado. En ese momento toma el contro el esquema `pasar` (figura 5.27(f)). Este ejemplo muestra como la jerarquía es capaz de responder correctamente a distintos estímulos que aparecen.

A la hora de implementar el movimiento del jugador fue necesario realizar ciertas adaptaciones. Por ejemplo, el simulador `SoccerServer` permite básicamente un control en posición (avanza 1 metro, gira 30 grados) mientras que JDE está más orientada a un control en velocidad, sin horizonte temporal. Por ejemplo, la emisión de varios comandos `turn` o `dash` provoca su acumulación, y un avance real que depende del ritmo al que se emitan esas actuaciones. Esta adaptación se ha realizado en el esquema `motor`. Por un lado, acepta comandos de velocidad, que ahora puede actualizarse a cualquier ritmo sin que haya efecto nocivo de la acumulación temporal. Por el otro, emite los comandos de posición permitidos por `SoccerServer`, controlando su amplitud y su *ritmo* para que materialicen efectivamente la velocidad deseada por el resto de los esquemas.

El ejemplo descrito hasta aquí explica la implementación de un delantero, que se comporta bien en los escenarios de prueba ensayados. Para lograr más eficiencia en partidos competitivos es necesario dotarle de más comportamientos, incorporándoselos a su jerarquía. Para desarrollar un equipo completo hay que generar los sistemas de comportamientos de otros roles del conjunto, como defensa, portero, etc.

Capítulo 6

Conclusiones

La verdadera investigación consiste en buscar a oscuras el interruptor de la luz. Cuando la luz se enciende, todo el mundo lo ve muy claro.
Desconocido

Una vez descrita la arquitectura propuesta, su implementación software y algunos experimentos realizados con ella, terminamos esta memoria resumiendo los principales aportes de la tesis y las líneas más prometedoras que se abren, a la luz de este trabajo, para continuar la investigación.

Primeramente recapitularemos los puntos centrales de la arquitectura presentada y haremos una evaluación somera, una vez que la lectura de los capítulos anteriores permite tener una visión de conjunto de la propuesta. Segundo, repasaremos los objetivos que se plantearon en el primer capítulo y el grado en que se han satisfecho. También expondremos los aportaciones originales hechas en esta investigación, así como las limitaciones que hemos observado. Finalizamos esbozando las líneas por las que apreciamos que se puede continuar el trabajo realizado.

6.1. Jerarquía de esquemas para generar comportamientos

El problema fundamental que hemos abordado en esta tesis es la generación de comportamiento autónomo en un robot móvil. En concreto, hemos enfocado la investigación en la arquitectura cognitiva (si es que se puede llamar enfocar a abarcar un tema tan extenso). Como vimos en el capítulo 1, la arquitectura organiza las capacidades perceptivas, de procesamiento y de actuación del robot para que sea capaz de desarrollar un repertorio amplio de conductas.

Históricamente, la escuela reactiva y la deliberativa han sido dos aproximaciones fundacionales en la generación de comportamiento autónomo. Cada una tiene sus ventajas y también serias limitaciones, como explicamos en el capítulo 2, por lo que en los últimos años se han explorado otras vías para generar conductas. Una alternativa a esta dialéctica son los sistemas basados en comportamientos, que combinan varias unidades reactivas. Pese a los buenos resultados prácticos en conductas relativamente sencillas, estas nuevas propuestas no terminan de escalar en complejidad para sistemas amplios. Otra alternativa son los sistemas híbridos, que tratan combinar elementos reactivos y deliberativos en varias capas. Suelen limitar el uso de componentes reactivos a una sola capa, y son los sistemas de mayor éxito en la actualidad. En este escenario, JDE se puede encuadrar como una evolución de los sistemas basados en comportamientos, que revisita algunos conceptos claves para mejorar la escalabilidad.

El punto de partida de JDE es la definición de esquema utilizada en neurociencia y en los trabajos de Arbib [Arbib, 1989] y Arkin [Arkin, 1989b]. El comportamiento se considera la combinación de percepción y actuación en la misma plataforma. Ambas partes se dividen en pequeñas unidades que se llaman esquemas. La hipótesis fundamental de JDE es que se puede generar comportamientos complejos mediante una jerarquía dinámica de esos esquemas. Los principios esenciales de esta arquitectura se han explicado en el capítulo 3 y quedaron resumidos en la tabla 3.5.1. También en ese capítulo se explicaron las propiedades ventajosas que este ordenamiento ofrece para la generación del comportamiento, tales como la atención, la acotación de complejidad para la selección de acción, la coordinación entre percepción y actuación, la modulación continua, la percepción estructurada, etc..

Los criterios definidos en el capítulo 1 para evaluar una arquitectura de control de robots son los de reactividad, orientación a objetivos, robustez, escalabilidad y versatilidad. A continuación vamos a evaluar el cumplimiento de estos criterios en la arquitectura propuesta.

En cuanto a la reactividad, JDE proporciona mecanismos que dotan al sistema de agilidad frente a cambios en el entorno, tanto para detectar los nuevos estímulos que aparecen, como para la rápida toma de decisiones en respuesta. Los esquemas perceptivos capturan los cambios en la situación y los esquemas activos responden inmediatamente. En cuanto la percepción, el sistema tiene activos a los esquemas perceptivos que buscan a los estímulos relevantes, para detectarlos y caracterizarlos en cuanto aparecen. En cuanto a la actuación, la reactividad de la arquitectura reside en el uso de esquemas reactivos y en la rapidez del mecanismo de selección de acción. Los continuos bucles reactivos de información mantienen al sistema anclado, situado en la realidad que le rodea. Ante ligeros cambios, los esquemas de actuación pueden cambiar la modulación de sus hijos y tomar decisiones ligeramente diferentes. Ante novedades más significativas el sistema puede reaccionar reconfigurándose, cambiando los esquemas que están activos de modo que sean los convenientes para generar la respuesta en el nuevo escenario. Esta reconfiguración es rápida porque esos esquemas estaban latentes, chequeando constantemente si podían tomar el control.

Los comportamientos `sigue-pelota` descritos en la sección 5.3 suponen un ejemplo validador de la reactividad conseguida con JDE. Variaciones de posición de la pelota se detectan y corrigen con rapidez, consiguiendo un comportamiento vivaz. Si por cualquier circunstancia la pelota desaparece del campo visual, la arquitectura detiene instantáneamente el movimiento.

La orientación a objetivos de JDE se asienta en el mecanismo de selección de acción. Un padre sólo despertará a aquellos esquemas hijos que colaboran de algún modo en la consecución de sus propios objetivos. En este sesgo se apoya la finalidad del sistema, todos los esquemas activos están despiertos porque ayudan a conseguir cierto objetivo en determinada situación. En este sentido, el objetivo general se estructura en una jerarquía de posibles subobjetivos, que materializan desde la finalidad más abstracta hasta las más concretas a medida que se desciende en la jerarquía. Es importante recalcar que no hay un orden temporal preestablecido entre ellos, sólo concomitancias y predisposiciones. El desarrollo temporal queda pendiente de las condiciones del entorno (y de ciertas variables endógenas) que se vaya encontrando en cada momento.

Un ejemplo que muestra la validez de JDE en lo referente a la orientación a objetivos es el sistema de comportamientos del jugador de futbol descrito en la sección 5.4. En este sistema se dispone de un conjunto de comportamientos básicos reactivos. A lo largo del tiempo y dependiendo de la situación, se va alternando cuáles de ellos están activos. Con esta activación selectiva y finalista se consigue que cada uno se active sólo cuando contribuye al objetivo global del delantero.

En cuanto a robustez frente a fallos de los esquemas, en JDE no se necesitan mecanismos *especiales* para ello. La propia ejecución de la arquitectura implica que el esquema padre monitorice el funcionamiento de sus hijos y si efectivamente están colaborando a su propio objetivo o no. Él detectará el fallo y puede tomar acciones correctoras. Por ejemplo, el padre puede tener varios modos diferentes (varias colecciones de hijos distintas) de conseguir su labor. Puede monitorizar el progreso de una alternativa y si fracasa entonces desactivarla y lanzar los hijos de la segunda alternativa. También se pueden despertar a la vez a varios hijos, entendidos como maneras distintas de realizar la misma labor, y que sean las condiciones sensoriales quienes activen uno u otro según su adecuación a la situación concreta que se presenta. La recuperación de esos fallos se contempla de modo natural en la arquitectura, no se ha articulado ningún mecanismo adicional, particular para esa labor.

En cuanto a la robustez frente a errores sensoriales, se han implementado técnicas dentro de JDE, pero no son estructurales, sino el contenido concreto de algún esquema. Por ejemplo, en los experimentos de la sección 5.1 se han descrito las técnicas de fusión sensorial empleadas en la estimación correcta de la ocupación del espacio. Ellas hacen robusta la percepción de los obstáculos, móviles o estáticos, a pesar de la incertidumbre sensorial. También en la sección 5.2 se ha explicado la combinación de información de varios sensores y obtenida en distintos instantes para percibir de modo fiable el estímulo `puerta`.

La versatilidad de JDE ha sido validada en los experimentos, al utilizar la arquitectura con éxito en varios entornos diferentes: la RoboCup y varios robots de interiores como el Pioneer y el EyeBot. El

haber funcionado con distintos robots avala que la arquitectura no es dependiente de una plataforma específica, y realmente puede ser portada a diferentes robots sin pérdida de generalidad.

Con respecto a la escalabilidad, se han justificado los argumentos teóricos que la facilitan y se han evitado en el diseño las dificultades de las que adolecían otras arquitecturas. La jerarquía más compleja que hemos implementado es la del sistema de comportamientos del jugador de la RoboCup, descrita en la sección 5.4. No hemos probado ningún ejemplo en el que se utilicen más allá de cuatro niveles, por lo cual los argumentos que tenemos para defender la escalabilidad de JDE en este sentido se fundamentan en esa experimentación y en la discusión teórica. Serían necesarios más experimentos con repertorios cada vez más extensos de comportamientos, que ratificaran las ventajas argumentadas. Seguro que al implementar tareas suficientemente complejas como para necesitar más niveles aparecen problemas nuevos no contemplados.

6.2. Aportes principales

El objetivo principal de esta tesis era el diseño de una arquitectura cognitiva que organizara el sistema de control para generar un repertorio de comportamientos. La aportación en este sentido es la arquitectura JDE. En el capítulo 3 se ha detallado su descripción teórica. En el 4 se ha explicado la implementación software con la que la hemos materializado. Finalmente, en el capítulo 5 se analizan diversos experimentos realizados con ella sobre robots reales, que la validan. Relacionados con ese objetivo principal también enunciamos cuatro subobjetivos concretos que sustentan al principal. Los describimos a continuación, viendo el grado en que se han satisfecho.

El primero era un análisis del estado del arte. Este análisis se pormenoriza en el capítulo 2. En él se describen las principales aproximaciones existentes en la literatura al problema de la generación del comportamiento. El barrido ha sido deliberadamente amplio, multidisciplinar si se quiere, para tener una mejor perspectiva del problema del comportamiento. Por ejemplo, se incluyen como mecanismos generadores la teoría de control y los descritos por algunos etólogos, que no suelen aparecer en los trabajos de inteligencia artificial, típicamente más computacionales.

Además de dar los ejemplos más significativos de arquitecturas particulares, en el análisis se han tratado de identificar las líneas comunes y de clasificar esas arquitecturas concretas en alguna de las familias registradas, sistematizando el estado del arte. Este esfuerzo taxonómico no se ha limitado a una enumeración, sino que se ha incluido una revisión crítica de los argumentos centrales de cada familia, enfocando la discusión en sus virtudes y defectos.

El segundo subobjetivo era realizar una evaluación conceptual de la arquitectura. Esta evaluación se ha realizado por análisis comparativo, en el capítulo 3. Examinando cómo aborda JDE diversas cuestiones arquitectónicas como la selección de acción, la incrementalidad de comportamientos, etc. y cotejándolo con las respuestas que ofrecen otras arquitecturas a estos mismos problemas.

El tercero era la implementación en software real de los mecanismos descritos en la teoría, que sirve de validación experimental. La satisfacción de este subobjetivo ha supuesto un enorme esfuerzo de programación y de concreción. Como ya indicamos anteriormente, la arquitectura regula la organización del sistema, cómo se conectan entre sí las distintas unidades del comportamiento, pero no regula el contenido concreto de cada una de ellas. En este punto se han diseñado y escrito unos programas que materializan las directrices de JDE. Ellos constituyen la infraestructura software sobre la que desarrollar e integrar diferentes esquemas particulares para generar comportamientos sobre robots reales. Atendiendo a este subobjetivo hay que destacar varias aportaciones, que se describen con detalle en el capítulo 4:

1. Se han programado dos servidores, `otos` y `oculo`, como acceso básico, tanto local como remoto, a los sensores y actuadores del robot. Como vimos en la sección 4.2, estos servidores ofrecen una interfaz de mensajes de texto para leer las imágenes de la cámara del robot, o enviarle una orden de movimiento a sus motores. Ambos servidores se han codificado siguiendo técnicas de multiprogramación y utilizando *drivers* del estado del arte actual (por ejemplo `ARIA` y `video4linux`). Se han usado siempre bibliotecas de software libre, distribuidas con licencia GPL, y el propio código de los servidores es abierto. Con ello se favorece la validación y reutilización por terceros grupos de los experimentos descritos aquí.

2. Los esquemas se han implementado como hebras de kernel. Los flujos de control independientes que define JDE se han materializado como diferentes hebras dentro del mismo proceso. Así, la comunicación interesquema se realiza a través de variables comunes. Para reforzar la autonomía entre esquemas se han empleado hebras de kernel, y no de usuario. De este modo cada uno de ellos participa independientemente en la planificación de procesador que hace el sistema operativo local y si uno se bloquea por alguna razón, no detiene al resto. Se han utilizado `pthreads`, que es la biblioteca estándard dentro de GNU/Linux para la multitarea.

3. Para los esquemas de actuación, el mecanismo de selección de acción distribuido se ha programado como dos funciones, el chequeo de las precondiciones y el arbitraje explícito, que sitúan a cada esquema en un estado de activación de los posibles.

4. Se ha creado un paquete software con varias herramientas para facilitar el uso de JDE. Según vimos en el capítulo 4, el paquete incluye varios esquemas de servicio, un esqueleto para desarrollar nuevos esquemas y varios ejemplos sencillos de uso[1].

   Para simplificar la inserción de nuevos esquemas en el sistema se ha creado un esqueleto de código fuente que incluye las características de los esquemas de JDE. Por ejemplo, el uso de variables de modulación, el ritmo controlable, las funciones de selección de acción, las funciones de dormir y despertar a un esquema, etc. También se han realizado varios esquemas de servicio necesarios para montar un programa de control sobre los servidores descritos, `otos` y `oculo`. Finalmente se han programado varios ejemplos sencillos: `teleoperador`, `sigue-pared`, etc, que se incluyen en el paquete como muestra.

El cuarto subobjetivo era generar un repertorio de comportamientos y percibir un cierto conjunto de estímulos reales utilizando la arquitectura JDE. Este punto, junto con el anterior, expresan el deseo de que esta tesis no se limitara a una guía teórica, de principios, sino que incluyera soporte experimental. Los experimentos proporcionan una realimentación real, que es imprescindible para decidir qué asuntos necesitan revisión, cuáles funcionan bien, o por dónde continuar la ampliación de la arquitectura. Las aportaciones originales a este respecto se han descrito con detalle en el capítulo 5 y se pueden resumir en:

1. Programación de varios esquemas para percibir el entorno de ocupación del robot, incluso los obstáculos dinámicos. La estimación de ocupación de los alrededores del robot se refleja en una rejilla. Cada celdilla fusiona la información de ocupación procedente del láser, los sónares y los sensores de contacto. Además de esta integración de sensores heterogéneos, la rejilla supone una integración temporal, porque almacena evidencias obtenidas en distintos instantes y desde lugares diferentes.

   La regla de actualización de la rejilla se ha diseñado para que la estimación sea dinámica y se adapte a la realidad cambiante cuando los obstáculos son dinámicos. Adicionalmente, las celdillas de ocupación se agrupan en segmentos para construir una representación más abstracta e igualmente dinámica.

2. Diseño y codificación de los esquemas que permiten al robot percibir las puertas de su entorno [Cañas *et al.*, 2001]. Se parte de un estímulo `jamba`, que es el invariante que hemos encontrado para todas las puertas. Las `jambas` se buscan en las imágenes utilizando el heurístico de bordes verticales largos. Esos indicios visuales se fusionan en una rejilla de probabilidad de jamba con la información de ocupación. Acumulando las evidencias en la rejilla la posición real de las jambas destaca después de incorporar varias imágenes obtenidas desde diferentes posiciones. De este modo se infiere la profundidad de la puerta indirectamente, sin necesidad de un par estéreo de cámaras, ni de encontrar puntos homólogos.

   Adicionalmente, se ha programado un esquema motriz que elige en cada momento el punto de observación más discriminativo para tomar la siguiente imagen. Dirigiendo los movimientos del

[1] El código de este paquete software, así como de los servidores, se encuentra disponible en http://gsyc.escet.urjc.es/jmplaza

robot hacia ese punto se obtiene un sistema de percepción activa basado en esquemas, que acelera la detección de las puertas.

3. Realización de varios comportamientos sencillos con la arquitectura JDE. Se han programado varias conductas simples respetando la división entre esquemas perceptivos y de actuación. Por ejemplo, se ha conseguido que un robot sea capaz de perseguir una pelota en movimiento y de seguir en paralelo una pared. Ambos casos se han resuelto con dos esquemas, uno perceptivo y otro motriz. En el primero se identifica la posición de la pelota en la imagen y se utiliza un control proporcional con ciertos ajustes para mover al robot y que la pelota aparezca centrada. En el segundo se extrae la frontera entre la pared y el suelo analizando la imagen, y se usa un control basado en casos para situarse paralelo a ella.

   También se ha programado un comportamiento de navegación veloz, que consigue que el robot progrese en cierta orientación sin chocar con los obstáculos. Se apoya en la información de ocupación que hemos descrito anteriormente. Sobre la rejilla de ocupación se utiliza una técnica de ventana dinámica para calcular el mejor comando de movimiento [Lobato, 2003]. Por el mejor se entiende aquel que resulta un compromiso satisfactorio entre los criterios de velocidad alta, distancia a los obstáculos y orientación próxima a la deseada.

4. Programación de varias conductas más elaboradas, que implican el uso de algunos mecanismos jerárquicos de JDE. En esta línea se han programado varios casos concretos, como el comportamiento `ir-a-punto` y un sistema completo de comportamientos que exhibe un jugador de fútbol simulado en la liga de la RoboCup.

   El comportamiento `ir-a-punto` se ha programado como la combinación en jerarquía de cuatro esquemas de actuación. Uno de ellos hace de padre y reutiliza a otros tres más simples: el esquema `para`, el esquema `avanza` y el esquema `sigue-pared`. Modulando las precondiciones de unos y otros, y resolviendo los casos de colisión en la función de arbitraje, se consigue un comportamiento de navegación local capaz de sortear a los obstáculos del entorno. Esta conducta supone una muestra de reutilización de esquemas, pues el esquema `sigue-pared` utilizado en otro contexto desarrolla el comportamiento de avance en paralelo a la pared.

   El ejemplo más completo que hemos realizado para explotar los mecanismos de jerarquía es el sistema de comportamientos del delantero simulado. A grandes rasgos, la conducta del delantero se divide en comportamientos de ataque, defensa y retorno a su posición de referencia. Se ha desarrollado un abanico completo de conductas de ataque como: `chutar` a gol, `driblar`, `pasar` a un compañero, `acercarse-a-bola` y `avanzar` con el balón. Para todos ellos se establece una competición por el control que determina, siguiendo ciertas precondiciones y arbitrajes en JDE, cuál de ellos gana el control de los actuadores en cada momento. Adicionalmente, el comportamiento `driblar` se ha subdividido en dos esquemas hijos que materializan dos fases separadas de la conducta: realizar el autopase y después correr en busca de la pelota. El comportamiento `acercarse-a-bola` se ha implementado como la combinación de otros dos más sencillos: `mirar-bola` y `seguir-bola`.

5. Para demostrar que dentro de JDE se pueden utilizar diferentes técnicas concretas de control se han programado varios esquemas siguiendo distintos paradigmas de control. En el capítulo 5 se ha descrito varios experimentos donde se utilizan controladores borrosos (esquema `avanza` en la conducta de `ir-a-punto`), la realimentación proporcional (comportamiento `sigue-pelota`), el control basado en casos (comportamiento `sigue-pared`) y técnicas con un mayor enfásis deliberativo que anticipan el efecto de seguir posibles trayectorias (conducta de avance rápido sorteando obstáculos).

Aportes conceptuales

Una vez repasados los objetivos conseguidos, queremos resaltar los aportes originales de este trabajo desde una perspectiva más conceptual:

1. Un aporte de esta tesis es la propia definición de *arquitectura*, y los problemas que debe resolver. Aunque la atención y la selección de acción se habían identificado como problemas relevantes en robótica, aparecían cada uno por separado. Ambos se han detectado como escollos fundamentales que dificultan el camino hacia sistemas de comportamientos más amplios y flexibles. En esta tesis los dos se han identificado como partes del problema genérico de la organización del sistema. De hecho, JDE propone un ordenamiento que da respuesta a ambos problemas.

2. Se ha propuesto la jerarquía como principio esencial para escalar en complejidad. No es mérito de esta tesis la división del comportamiento entre percepción y actuación, ni siquiera la cuantización de ambas en pequeñas unidades que ejecutan en paralelo y que se llaman esquemas. Sí lo es su apuesta por la organización en jerarquías de esquemas. Como hemos visto a lo largo del capítulo 3, esta jerarquía tiene unas características propias que la diferencian de otras aproximaciones jerárquicas como RCS, TCA, etc..

   En cuanto a actuación, la jerarquía se entiende como una coactivación de los hijos, que materializa una predisposición hacia ciertos modos de actuar. Las condiciones del entorno elegirán entre los modos predispuestos al más adecuado para la situación actual. Esta concepción de la jerarquía es novedosa en robótica y combina de manera flexible la orientación hacia objetivos y hacia la situación que son deseables en cualquier algoritmo de selección de acción.

   La complejidad de la selección de acción se acota al plantearla como una competición entre un reducido número de esquemas de actuación hermanos. En vez de una competición centralizada, de todos contra todos, la selección de acción se divide jerárquicamente en varias competiciones más simples, una por nivel de la jerarquía. Cada una de ellas se resuelve de manera distribuida entre los hermanos y el padre.

   Esta interpretación de jerarquía en actuación permite que los padres monitoricen permanentemente a sus hijos, puedan modularlos de manera continua e incluso puedan cambiar de hijos con rapidez. Gracias a ello y a la agilidad del algoritmo de selección de acción, la jerarquía puede reconfigurarse rápidamente para atender a cambios de objetivos o en la situación.

3. Otro aporte significativo de esta tesis es la inclusión explícita de la parte perceptiva dentro de la arquitectura. Aunque la descomposición de la complejidad en la actuación ha recibido numerosas propuestas, no es tan frecuente el planteamiento jerárquico en la percepción. En JDE la percepción se presenta como una colección jerarquizada de estímulos, cada uno de los cuales es detectado y actualizado por un esquema perceptivo homónimo.

   La coordinación entre percepción y actuación se resuelve en JDE asociando a cada esquema de actuación los esquemas perceptivos que elaboran los estímulos que él puede necesitar. De esta manera la percepción tiene un carácter subsidiario y se imbrica con la actuación siguiendo la estructura jerárquica de los esquemas de actuación. Los esquemas perceptivos se activan en el momento en que lo hacen los esquemas de actuación que los necesitan.

   La atención aparece de manera natural en la jerarquía de JDE cuando el esquema padre decide activar ciertos esquemas perceptivos y otros no. Esto permite no desperdiciar capacidad de cálculo y facilita la ampliación la arquitectura cuando hay estímulos computacionalmente costosos de percibir.

   Como vimos en el capítulo 3, otros fenómenos como la interpretación, la percepción estructurada y la percepción activa surgen dentro de la jerarquía de esquemas perceptivos. Por ejemplo, la percepción estructurada se presenta cuando un esquema perceptivo depende de otros esquemas perceptivos, los cuales elaboran estímulos de menor abstracción, para producir el suyo.

4. Se han identificado las raíces de varios problemas de arquitectura en otras aproximaciones como la subsunción o los sistemas deliberativos clásicos, que dificultan su uso en comportamientos complejos. Estos problemas quizá no se manifiestan cuando sólo hay que generar un repertorio pequeño de conductas, porque entonces casi cualquier arquitectura puede resolverlo. Sin embargo se tornan importantes cuando se amplía el repertorio de conductas a generar y se crece en complejidad.

En los sistemas deliberativos es frecuente la descomposición funcional para dividir la complejidad en la toma de decisiones. Típicamente la llamada a funciones admite parámetros únicamente en el momento de la invocación y el llamante suele quedarse bloqueado hasta que la función se ejecuta y devuelve algún resultado. Estas consecuencias se han mostrado muy nocivas para los sistemas generadores de comportamientos. Por ejemplo, se traslada al módulo que materializa la función la responsabilidad de detectar las sorpresas que atañen al llamante, que no puede detectarlas mientras tanto porque está bloqueado. Frente a esto, JDE propone una modulación continua, que es mucho más flexible, y un funcionamiento en paralelo entre esquemas padres y esquemas hijos, que permite a cada uno seguir pendiente de detectar los estímulos o sorpresas que le incumben.

Un problema de la arquitectura de subsunción que dificulta su escalabilidad es lo que hemos bautizado como *incrementalidad substractiva*. Cada nuevo comportamiento o nivel de competencia que se añade debe subsumir explícitamente las salidas de los niveles inferiores. De este modo cuando se añade algun módulo inferior, todos los superiores a él han de recodificarse para que lo anulen. Frente a esto, JDE propone una *incrementalidad aditiva*, más flexible para añadir nuevos esquemas. La incorporación de un nuevo esquema en los niveles inferiores no afecta en absoluto a los superiores ya existentes, porque el nuevo estará inactivo por defecto.

5. Se ha estudiado el dinamismo de los mecanismos de fusión de creencias más habituales. Tal y como se describe en el anexo A, las técnicas probabilística, de teoría de la evidencia y la borrosa clásica presentan demasiada inercia en la creencia acumulada. Por ello resultan desaconsejables para reflejar fenómenos dinámicos como la ocupación en entornos con obstáculos móviles. Se ha identificado la propiedad asociativa de la regla de fusión (Bayes, Dempster-Shafer y unión borrosa respectivamente) como la causa principal de su inconveniencia, pues hace irrelevante el orden temporal en el que se incorporan las medidas. Además, se han desarrollado dos técnicas propias orientadas a que la estimación interna preserve el dinamismo del estado real de ocupación. La primera actualiza la creencia con una ecuación diferencial y la segunda emplea una memoria local de evidencias para decidir a partir de ella la estimación por mayoría.

6. Quizá el mérito más importante de esta tesis robótica sea el enlace con los modelos etológicos, haber identificado la afinidad entre etología y robótica, y plantear en este último escenario mecanismos similares a los propuestos en la etología para explicar el comportamiento.

   Muchas son las miradas que se dirigen actualmente al comportamiento de los animales como fuente de inspiración para diseñar el comportamiento en los robots. Sobre todo en una época en la que parece hay un cierto estancamiento en cuanto a nuevas arquitecturas en robótica. Más reducido es el número de trabajos que se sumergen en los modelos de comportamiento directamente propuestos por los etólogos. Esta tesis pretende establecer un enlace más entre ambas disciplinas, reconociendo el potencial que ofrece la etología.

   En este sentido se han explorado arquitecturas propuestas por Lorenz, Tinbergen, Ludlow, etc. a la luz de sus observaciones con los animales. Ellas han servido de inspiración directa en el diseño de JDE, y muchos de los mecanismos que ésta ofrece tienen sus raíces en ellas. Por ejemplo, el planteamiento de la jerarquía como predisposición. Esta idea resulta central en JDE y supone un cambio innovador que facilita la escalabilidad del sistema.

6.3. Limitaciones

Lamentablemente el problema de la generación de un repertorio de comportamientos no está solucionado. Atendiendo a la cita que abre este capítulo, la luz sigue apagada. La arquitectura presentada no es la panacea, la solución definitiva, y deja inevitablemente algunos problemas sin resolver. Simplemente pretende ser un avance más, un paso, en una dirección que nos parece correcta. Ya apuntamos algunas limitaciones de la arquitectura en la sección 3.5.6, y en algunos puntos del capítulo 5. En esta sección haremos un recuento de los problemas identificados, tanto los que son

achacables a la implementación realizada, como otros más conceptuales que no están resueltos desde el punto de vista teórico en la arquitectura JDE.

En cuanto a las limitaciones de implementación, señalar en primer lugar que en la implementación realizada de JDE todos los esquemas deben ejecutarse en la misma máquina. Al materializarse en hebras de kernel que comparten un espacio de direcciones, todas las hebras corren dentro del mismo ordenador. Esto puede resultar en un cuello de botella en cuanto a capacidad de cómputo, como en el caso de utilizar muchos esquemas que hagan procesamiento de imágenes. Esta restricción queda ligeramente paliada con la flexibilidad que aportan los los servidores `otos` y `oculo`, porque se puede elegir dónde ejecutar el programa.

Otra limitación de la implementación realizada se manifestó al programar los experimentos descritos en la sección 5.4. En concreto al implementar con JDE el comportamiento de un jugador en la liga simulada de la RoboCup [Martínez Gil, 2003]. El simulador oficial proporciona como actuaciones básicas giros de un pequeño ángulo y avances de una cierta distancia (control en posición). Ocurre entonces que se da un efecto acumulativo: si se emite dos veces la actuación `avanza-10cm` entonces el resultado neto es que se avanzan 20 cm. Para materializar un control en velocidad y paliar este efecto acumulativo hubo que controlar la frecuencia a la que se enviaban comandos de posición al simulador.

Las raíces de este problema concreto residen en que JDE no acepta actuaciones que se acumulen en el tiempo, porque todo es presente, no hay un horizonte temporal en las actuaciones. Por ejemplo, es perfecto para el control en velocidad, con órdenes del tipo: avanza a 20 cm/s. Sin embargo, no funciona bien con unidades de bajo nivel cerradas que tienen un alcance temporal más allá de lo instantáneo, con órdenes como: avanza 10cm. En JDE no hay unidades de comportamiento que encapsulen un comportamiento que dura cierto intervalo de tiempo. Se pueden tener varios niveles de abstracción si son necesarios, pero operando todos en el presente, sobre valores instantáneos de las variables.

Una limitación adicional detectada en la implementación realizada de JDE es que no admite varias instancias del mismo esquemas activas a la vez. Sin embargo esta posibilidad sí está contemplada conceptualmente, pues se pueden tener instancias del mismo esquema activadas simultáneamente en distintos niveles de la jerarquía, previsiblemente con modulaciones diferentes.

En cuanto a las limitaciones conceptuales, quizá la principal crítica que se le puede hacer a JDE es que el sistema no tiene iniciativa: si se encuentra en una situación novedosa, donde no aparecen los estímulos que es capaz de reconocer, entonces no sabe cómo comportarse. Es decir, el robot diseñado con JDE sabe hacer lo que sabe hacer y no más, literalmente. El sistema no tiene capacidad de innovación o aprendizaje. En éste sentido conviene mencionar que explícitamente dejamos fuera de esta tesis el aprendizaje, y recordar que la funcionalidad que aspira a generar esta arquitectura es la de animales inferiores como pájaros o peces. Como ya avanzamos en el capítulo 1 nos enfocamos en el comportamiento instintivo (por expresarlo en el vocabulario etológico), sin adentrarnos en los comportamientos aprendidos. No hemos estudiado la modificación de comportamiento con las experiencias. De hecho, bastante tenemos con conseguir que se comporte bien sin aprendizaje.

Una argumentación en esa línea es que los monos no saben escribir a máquina, aunque tengan el hardware para hacerlo. Simplemente porque su arquitectura *software* no se lo permite, no les ha dotado con los mecanismos de control para ello. Los animales son sistemas generadores de comportamientos que resultan flexibles y exitosos en su nicho, si se cambia de habitat se mueren. Por ejemplo, si aparece una glaciación, los comportamientos programados filogenéticamente en su sistema nervioso dejan de ser útiles y la especie se extingue. Con los comportamientos innatos se lleva el adjetivo situado hasta sus últimas consecuencias. A ese éxito y flexibilidad se aspira con los robots programados con JDE.

El robot que utiliza JDE no modifica su comportamiento con el tiempo y además no recuerda nada. Aprendizaje y memoria suelen estar relacionados, y no se han planteado porque son asuntos complicados, que por sí sólos requieren más de una tesis para abordarlos. En el estado actual de la cuestión nos parecía prematuro tratar siquiera de incluirlos en una propuesta seria de arquitectura.

Una de las críticas habituales a los sistemas sin memoria es que pueden quedarse estancados en bucles, porque son incapaces de detectar que están repitiendo la actuación y ya pasaron por la misma situación recientemente. No obstante, dentro de JDE sí hay cierta memoria, dado que algunos esquemas guardan estado, y no hay motivos para que no lo hagan. No hay problema en que un esquema utilice información de largo plazo (por ejemplo, un mapa), o que incluso la genere. Sin embargo, nos

parece que un tratamiento en profundidad de estas componentes del comportamiento requiere un replanteamiento integral, para no ser un cúmulo de arreglos ad-hoc.

Otra limitación conceptual de JDE es que no se pueden tomar acciones de compromiso, que satisfagan a varios necesidades en cierta medida simultáneamente. Esta limitación radica en que sólo puede haber un esquema ganador en cada nivel, de manera que sólo sus órdenes llegan a los actuadores en ese instante. No se combinan las salidas de varios esquemas.

Como mencionamos en la sección 3.5.6, la arquitectura hereda cierto carácter secuencial en cuanto a la consecución de objetivos. Para satisfacerlos todos ellos establece implícitamente un reparto de los actuadores en el tiempo. Esto contrasta con otras arquitecturas que utilizan técnicas de fusión borrosa o de campos de potencial para combinar las salidas de varios esquemas.

6.4. Líneas futuras y perspectivas

Una vez localizadas las limitaciones más relevantes, en esta sección concluimos presentando las líneas de investigación por las cuales estimamos que merece la pena continuar el trabajo desarrollado en esta tesis. Más que el principio del final, esta sección cierra el final del principio.

En primer lugar señalar que es necesaria mucha más experimentación sobre la jerarquía. En esta tesis se han realizado algunos comportamientos empleando los mecanismos jerárquicos de JDE, pero resultan más una prueba de validación que un estudio experimental sistemático. Sería conveniente realizar nuevos experimentos sobre los mecanismos jerárquicos, para probar más profusamente la escalabilidad y versatilidad de la arquitectura. Es de esperar que la nueva realimentación permitirá revisar algunos temas simplemente esbozados en JDE y orientar mejor por dónde extender la arquitectura. Por ejemplo, un aspecto con escasas pruebas experimentales es la reconfiguración dinámica de la jerarquía en cambios radicales de la situación.

Parte de los nuevos experimentos podrían dirigirse a aumentar el abanico de comportamientos. Con un sistema completo y diverso, que utilice información visual desbordante, se podría defender sólidamente la conveniencia o no de esta propuesta, pues supone a día de hoy un reto aún no resuelto y el escenario natural para que demuestre su potencial. En este sentido está previsto enriquecer el repertorio de estímulos y conductas de la plataforma Pioneer, hasta completar un sistema autónomo de comportamientos variados.

Un segundo conjunto de líneas de continuación se centra en mejorar la implementación realizada, resolviendo las carencias detectadas. En este sentido, sería conveniente realizar una implementación de la arquitectura que permitiera a los esquemas residir en máquinas diferentes. La principal ventaja de esta nueva implementación sería la multiplicación de la potencia de cómputo, pues los esquemas se podrían repartir por un conjunto de ordenadores. En este nuevo escenario habría que recodificar el algoritmo de selección de acción y la comunicación interesquema, quizá pensando en algún tipo de paso de mensajes, pues la memoria compartida dejaría de ser viable. Habría que explorar si esta distribución física compensa realmente el retardo en la comunicación interesquema.

Otra línea por la que proseguir este trabajo es añadir a la implementación realizada la capacidad de tener activas simultáneamente varias instancias del mismo esquema. Con ello se materializaría una de las posibilidades que permite la arquitectura teórica que no está recogida en la implementación descrita. Por ejemplo, creando las hebras dinámicamente cada vez que el esquema homónimo es activado. El padre podría señalar en el momento de la creación las variables concretas de las que esa instancia del esquema que ejerce como hijo tiene que leer sus entradas y en las que escribir sus salidas. De esta manera instancias diferentes operarían sobre variables de entrada y salida distintas.

Un tercer conjunto de líneas de trabajo futuro tiene que ver con ampliaciones estructurales de la jerarquía, extendiendo el ámbito de comportamientos que pueden acomodarse en ella. Es enorme la cantidad de preguntas que quedan abiertas.

Siguiendo las limitaciones identificadas, una extensión lógica es explorar la inclusión en la arquitectura de la memoria a largo plazo. Por ejemplo la construcción, la actualización y el uso de mapas de ocupación de largo plazo en los cuales se refleje la estructura de un entorno de tamaño medio (i.e. una planta de un edificio) que supere el ámbito local con el que hemos trabajado hasta ahora. En este aspecto sería conveniente aclarar cuestiones como qué cosas almacenar en esa memoria, cómo

construirlas desde la información sensorial, qué cosas olvidar y cómo se accede a la información que almacena desde la situación presente. Se podría utilizar, por ejemplo, acceso por contenidos.

En cuanto al aprendizaje, una perspectiva abierta por JDE es la asociación de alto nivel entre esquemas. Es decir plantear el aprendizaje no ya como exploración del enlace correcto entre entradas sensoriales y salidas a actuadores, sino como enlace entre estímulos, es decir esquemas perceptivos, y esquemas actuadores que generan el comportamiento adecuado de respuesta. En este sentido JDE abre la posibilidad de aprendizaje a un nivel de abstracción superior. Adicionalmente, habría que contemplar la posibilidad de aprender nuevos comportamientos combinando los esquemas ya existentes, o incluso la crear nuevos esquemas de modo autónomo.

Un punto digno de estudio sería el de comportamientos emergentes en agrupaciones de individuos. En este sentido se puede continuar el trabajo programando dentro de JDE comportamientos que dependen de las actuaciones de otros congéneres. Por ejemplo programando los esquemas perceptivos que detectan la presencia de los otros e identifican lo que está haciendo. Percibir esos estímulos sociales puede facilitar la programación de comportamientos de grupo o de simple interacción. Este tipo de conductas se presenta con mucha frecuencia en animales sociales, por ejemplo la migración de aves en bandadas, el cortejo o los combates prenupciales en los peces, que siguen unas pautas de estímulos muy marcadas. Un escenario para probarlo en robótica podría ser la RoboCup simulada.

Otra línea interesante a explorar es el control homeostático, introducir ciertas variables internas como hambre, sed, etc. para regular y coordinar los comportamientos. Estas variables se podrían incorporar al mecanismo de selección de acción y tendrían su propia dinámica. De este modo se establece un juego de objetivos internos que entran en competición con los exógenos. En esta dirección se puede probar qué prestaciones ofrece el mecanismo implementado para coordinar estas diversas tendencias o motivaciones.

Un asunto pendiente de mayor estudio es la incorporación de la percepción activa a JDE. El ejemplo descrito en la sección 5.2 acompaña a los esquemas perceptivos con un único esquema de actuación, lo que resulta válido si lo único que tiene que hacer el sistema en ese momento es percibir puertas. Sin embargo ha quedado abierta la integración de esos esquemas de *actuación para percibir* en los mecanismos de selección de acción generales y su coordinación con los otros esquemas de actuación finalista. Una alternativa posible sería que el esquema perceptivo con hijos de actuación compita por el control con sus propios hermanos.

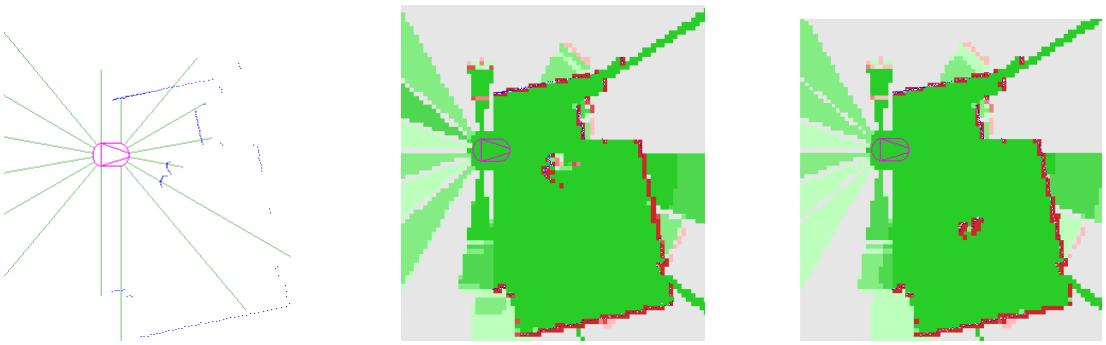
Una línea de investigación que se antoja fructífera es profundizar en la etología, en concreto estudiar modelos etológicos más recientes y su potencial aplicación a la robótica. Creemos que esta es una vía prometedora, que puede arrojar mucha luz al diseño de robots versátiles. Quizá se puede argumentar que esta vía queda lejos del salto en inteligencia que aparece entre los homínidos superiores a los hombres. También se puede aducir que es una línea poco ambiciosa, que no incluye las características esenciales que distinguen la inteligencia humana, como el razonamiento o el aprendizaje. Sin embargo a nosotros nos parece una línea realista para avanzar en la generación de comportamientos, y creemos que es un camino que hay que recorrer. Nó sólo porque la evolución natural haya tenido que pasar por estas etapas antes de dar lugar a la inteligencia humana, sino porque aún no hemos sido capaces de hacerlo mejor que ella. La flexibilidad de comportamientos de los animales y su éxito en entornos diversos como pájaros, peces, pequeños mamíferos, etc. está todavía muy por delante de los mejores resultados actuales con robots.

Apéndice A

Técnicas de fusión de evidencias

Como expusimos en la sección 5.1, la rejilla de ocupación implementada es una técnica robusta y vivaz de fusión de evidencias. A la hora de decidir su regla de actualización experimentamos con muchas otras alternativas que están muy asentadas en la literatura, sobre todo procedentes de la construcción autónoma de mapas. De hecho, empezamos los experimentos probando la fusión probabilística, pero descubrimos con ellos que no funcionaba bien para reflejar obstáculos dinámicos.

Por ejemplo, si el robot está un cierto tiempo dentro de una misma zona, pongamos tres minutos, y de repente aparece un obstáculo móvil que pasa por delante del robot durante 10 segundos, en la rejilla probabilística ese objeto no aparece. Aunque esos 10 segundos sea tiempo más que suficiente para confirmar que no son datos sensoriales espúreos. Del mismo modo, si una persona ha estado esos tres minutos en cierto lugar y después se mueve, la rejilla no refleja su desaparición de donde estaba, ni su aparición donde está ahora. Este ejemplo es el que se muestra en la figura A.1. En la figura A.1(a) la instantánea sensorial típica con los datos sónares y láser que alimentan la rejilla. En la figura A.1(b) se aprecia la rejilla probabilística conseguida después de tres minutos. Diez segundos después del movimiento del obstáculo que el robot tiene en frente, la rejilla actualizada con la regla de Bayes aún no ha incorporado ese desplazamiento, sigue siendo la de la figura A.1(b). Por contra, la rejilla actualizada con ecuación diferencial, sí ha incorporado ese cambio en el mundo real, como muestra la figura A.1(c): la ubicación antigua está ahora vacía, y la posición actual del obstáculo está ocupada[1].



(a) (b) (c)

Figura A.1: *Imagen sensorial típica (a) y las rejillas dinámicas conseguidas con actualización bayesiana (b) y utilizando la ecuación diferencial (c)*

Motivados por ese contraejemplo decidimos estudiar la adecuación de las técnicas más conocidas de fusión sensorial para construir la rejilla *dinámica* de ocupación. En concreto, hemos analizado el dinamismo del enfoque probabilístico bayesiano, de la teoría de la evidencia, de los conjuntos borrosos, del enfoque histográmico de Borenstein y de las dos reglas de actualización descritas en la sección 5.1. Todas ellas han sido implementadas en la mencionada librería `libgrids` y probadas.

[1]Se puede argumentar que un filtro de medidas independientes hubiera eliminado las lecturas repetidas, pero eso no afecta a la naturaleza del hecho. Si el robot se estuviera moviendo e incorporara muchas evidencias independientes de la misma zona, entonces se produciría el mismo fenómeno.

En este anexo haremos una descripción de las técnicas más conocidas de fusión de evidencias, y realizaremos un estudio comparado de su dinamismo, tomando como caso particular la ocupación del entorno. El estudio es aplicable a otros estímulos y sensores diferentes, porque no consideramos la geometría de los modelos sensoriales, ni el fenómeno concreto que se representa, sino el comportamiento de la regla de fusión de evidencias. De hecho, ha servido como fundamento para elegir la regla pertinente en la percepción de los obstáculos móviles y en la detección de puertas, que como vimos en la sección 5.2 también se apoya en una rejilla para fusionar evidencias.

A.1. Enfoque probabilístico bayesiano

El enfoque probabilístico es el más utilizado en la bibliografía y fue iniciado por Alberto Elfes [Elfes, 1990] y Hans Moravec [Moravec, 1989]. En él se asume que cada celdilla de la rejilla puede estar únicamente uno de los dos estados: ocupada o vacía, el cual se tratará de estimar desde las observaciones sensoriales acumuladas. El conocimiento que el robot tiene en el instante t sobre la ocupación de la celdilla situada en (x, y) se refleja en la *probabilidad* de que la celdilla esté en alguno de los dos estados posibles, condicionada a las observaciones que se han obtenido hasta ese momento. Así lo expresa la ecuación (A.1), donde $data(t-1)$ supone el conjunto de observaciones acumuladas hasta el instante $t-1$ y $obs(t)$ la observación actual. Cuando la probabilidad de ocupación es cercana a 0 entonces se está muy seguro que tal celdilla está vacía. Por el contrario, cuando es próxima a 1 entonces se tiene mucha confianza en que esa celdilla está ocupada. Inicialmente todas las casillas de la rejilla tienen valor 0.5, reflejando el desconocimiento total.

$$p_{ocupada}(x, y, t) = p(ocupada/obs(t), data(t-1)) \tag{A.1}$$

A.1.1. Actualización con regla de Bayes

La ecuación (A.1) no es operativa, de modo que se buscaron equivalentes suyos más fáciles de ejecutar, como el desarrollo teórico siguiente, paralelo al de Margaritis [Margaritis y Thrun, 1998] y Martin [Martin y Moravec, 1996]. Aplicando a (A.1) la regla de Bayes obtenemos (A.2). Si asumimos que las observaciones son *independientes en sentido markoviano* (A.3) y utilizamos nuevamente la regla de Bayes (A.4) llegamos a la expresión (A.5). La independencia markoviana supone que dado el hecho de la ocupación real, la $obs(t)$ no depende de las observaciones anteriores $data(t-1)$, según expresa (A.3). Esta suposición facilita enormemente el tratamiento probabilístico de la información. En general, el mundo no es markoviano, pero en la práctica asumimos esa independencia entre observaciones incorporando a la rejilla exclusivamente observaciones tomadas desde posiciones relativamente separadas (o siendo desde el mismo sitio, con valores diferentes).

$$p_{ocupada}(x, y, t) = \frac{p(ocupada/data(t-1)) * p(obs(t)/ocupada, data(t-1))}{p(obs(t)/data(t-1))} \tag{A.2}$$

$$p(obs(t)/ocupada, data(t-1)) = p(obs(t)/ocupada) \tag{A.3}$$

$$p(obs(t)/ocupada) = \frac{p(ocupada/obs(t)) * p(obs(t))}{p(ocupada)} \tag{A.4}$$

$$p_{ocupada}(x, y, t) = \frac{p(ocupada/data(t-1)) * p(ocupada/obs(t)) * p(obs(t))}{p(ocupada) * p(obs(t)/data(t-1))} \tag{A.5}$$

De modo análogo a (A.5) se obtiene la ecuación (A.6). A partir de ahí, se maneja el *ratio de probabilidad* en lugar de la probabilidad de ocupación $p_{ocupada}$. El *ratio de probabilidad* se definido como $\rho_{ocupada} = p_{ocupada}/p_{\overline{ocupada}}$. Con esta simplificación se elimina de (A.5) la dependencia de $p(obs(t))$ y de $p(obs(t)/data(t-1))$ sin perder información, puesto que afectan por igual a $p_{ocupada}$ y a $p_{\overline{ocupada}}$.

$$p_{\overline{ocupada}}(x, y, t) = \frac{p(\overline{ocupada}/data(t-1)) * p(\overline{ocupada}/obs(t)) * p(obs(t))}{p(\overline{ocupada}) * p(obs(t)/data(t-1))} \tag{A.6}$$

$$\rho_{ocupada} = p_{ocupada}/(1 - p_{ocupada}) \quad (A.7)$$
$$p_{ocupada} = \rho_{ocupada}/(1 + \rho_{ocupada}) \quad (A.8)$$

Sustituyendo (A.5) y (A.6) en (A.7) llegamos a (A.9), que posibilita un tratamiento incremental de la información. Con cada nueva observación la probabilidad acumulada se multiplica por un factor $\frac{p(ocupada/obs(t))}{1-p(ocupada/obs(t))}$ que es precisamente el valor dado por el modelo de sensor, y una constante $\frac{1-p_{ocupada}}{p_{ocupada}}$ determinada por la probabilidad de ocupación a priori, sin ningún otro conocimiento. Trabajando con logaritmos, los productos se convierten en rápidas sumas. Si en algún momento se quiere obtener la probabilidad exacta basta con deshacer los logaritmos y aplicar (A.8).

$$\rho_{ocupada}(x,y,t) = \frac{p(ocupada/obs(t))}{1 - p(ocupada/obs(t))} * \frac{1 - p_{ocupada}}{p_{ocupada}} * \rho_{ocupada}(x,y,t-1) \quad (A.9)$$
$$\rho_{obs} = \frac{p(ocupada/obs(t))}{1 - p(ocupada/obs(t))} \quad (A.10)$$
$$\rho_{apriori} = \frac{1 - p_{ocupada}}{p_{ocupada}} \quad (A.11)$$
$$\rho_{ocupada}(x,y,t) = \frac{\rho_{obs}}{\rho_{apriori}} * \rho_{ocupada}(x,y,t-1) \quad (A.12)$$

A medida que el robot recibe nuevas observaciones sensoriales, su información se va incorporando a la rejilla, actualizando las probabilidades almacenadas y haciéndolas evolucionar. Si una lectura sónar proporciona información sobre el estado de determinada celdilla $C_{(x,y)}$, el valor del modelo de sensor $p(ocupada/obs(t))$ en esa posición determina, a través de ρ_{obs}, si allí la probabilidad de ocupación sube o baja después de la nueva observación. El denominador $\rho_{apriori}$ simplemente normaliza la influencia de ρ_{obs}. Si $p(ocupada/obs(t)) = p(ocupada)$ entonces la observación no aporta ninguna información adicional sobre el conocimiento a priori y la probabilidad acumulada no cambia. Si $p(ocupada/obs(t)) > p(ocupada)$ entonces aumenta la probabilidad global en la ocupación de esa celdilla. Recíprocamente, cuando $p(ocupada/obs(t)) < p(ocupada)$ la probabilidad acumulada disminuye.

Este modelo sensorial $p(ocupada/obs(t))$ se denomina *modelo de sensor a posteriori*, y marca la probabilidad de que la celdilla está ocupada o no dada una lectura del sónar $obs(t)$. Por ejemplo, en [Matía y Jiménez, 1998] se utiliza un modelo sónar que vale $p(ocupada/obs(t)) = 0{,}4$ en las celdillas más cercanas al sensor que el radio observado y $p(ocupada/obs(t)) = 0{,}6$ en las celdillas más o menos coincidentes con ese radio. Para celdillas más distantes, el modelo ofrece $p(ocupada/obs(t)) = 0{,}5$, que no aporta ninguna información en el enfoque probabilístico. Cuanto más se acerque a los extremos de probabilidad, 0 ó 1, más certidumbre aporta esa medida, en un sentido u otro.

Utilizar probabilidades permite tener un marco teórico fiable a la hora de realizar ciertas operaciones, cálculos e hipótesis con la información disponible. Otra ventaja sustancial es que (A.12) permite una formulación incremental, muy eficiente desde el punto de vista de tiempo y memoria requeridas en la actualización.

Uno de los inconvenientes de la actualización con regla de Bayes es que requiere que las distintas observaciones que se incorporan al grid sean *independientes*, al menos en sentido markoviano. Esto no siempre se puede asegurar cuando se tiene un flujo continuo de sensaciones. Otra desventaja es que no da medida alguna de *confianza*.

A.2. Teoría de la evidencia

La teoría de la evidencia se basa en la definición de un *campo de discernimiento* Θ, que es un conjunto de etiquetas que representan eventos mutuamente excluyentes. Tal y como se describe en [Pagac *et al.*, 1998], para nuestra aplicación de rejillas de ocupación las etiquetas interesantes son $\Theta = \{E, F\}$ porque las celdillas de la malla pueden estar vacías, E, u ocupadas, F. Se define también

una *asignación básica de probabilidad* como una función $m : \Psi \rightarrow [0,1]$, donde Ψ es el conjunto de todos los subconjuntos posibles de Θ, en nuestro caso $\Psi = \{\emptyset, E, F, \{E,F\}\}$.

El estado de cada celdilla se define asignando números de probabilidad a cada etiqueta en Ψ, en nuestro caso cuatro números. Sin embargo, asumiendo $m_{x,y}(\emptyset) = 0$ y aplicando (A.13) basta almacenar dos de ellos, $m_{x,y}(E)$ y $m_{x,y}(F)$, para caracterizar el conocimiento sobre la ocupación de la celdilla en este enfoque. El desconocimiento absoluto se refleja en $m_{x,y}(E) = 0$, $m_{x,y}(F) = 0$ y por lo tanto $m_{x,y}(E,F) = 1$. Cuando se está seguro de que una celdilla está vacía entonces $m_{x,y}(E) = 1$ y el resto se anula. Recíprocamente cuando se está seguro de que está ocupada $m_{x,y}(E) = 0$.

En cuanto a los valores del modelo sensorial, un ejemplo es el modelo que emplea [Pagac *et al.*, 1998], con geometría de cono de propagación. Para las celdillas dentro del arco el modelo viene dado por $(m_{obs}(F) = \frac{1}{n}, m_{obs}(E) = 0)$, donde n corresponde al número de celdillas situadas en el arco. Para las celdillas en el interior del sector, el modelo utilizado es $(m_{obs}(F) = 0, m_{obs}(E) = \rho)$, donde ρ es un factor constante de ajuste que iguala la masa total de evidencia asignada a las celdas vacías y a las ocupadas en cada lectura.

$$m_{x,y}(E) + m_{x,y}(F) + m_{x,y}(E,F) = 1 \tag{A.13}$$

A.2.1. Actualización con regla de Dempster-Shafer

La regla de Dempster-Shafer permite combinar evidencias sobre el evento A, por ejemplo $m_1(A)$ y $m_2(A)$. En nuestro caso, serán las asignaciones básicas de probabilidad acumuladas en cada celdilla de la rejilla para el evento ocupado F y las proporcionadas por la última lectura sónar (análogamente para el evento vacío E). Siguiendo el desarrollo de [Pagac *et al.*, 1998] se llega por ejemplo a (A.15).

$$m^t_{x,y}(E) = (m^{t-1}_{x,y} \oplus m_{obs(t)})(E) \tag{A.14}$$

$$m^t_{x,y}(E) = \frac{m^{t-1}_{x,y}(E)m_{obs(t)}(E) + m^{t-1}_{x,y}(E)m_{obs(t)}(\{E,F\}) + m^{t-1}_{x,y}(\{E,F\})m_{obs(t)}(E)}{1 - m^{t-1}_{x,y}(E)m_{obs(t)}(F) - m^{t-1}_{x,y}(F)m_{obs(t)}(E)} \tag{A.15}$$

Una ventaja de este enfoque es que contempla explícitamente la ambigüedad, tanto en las observaciones como en la creencia acumulada. El factor $m^t_{x,y}(\{E,F\})$ representa la incertidumbre almacenada. También se representa la contradicción: una misma celdilla puede recoger a lo largo del tiempo tanto lecturas que indican que está ocupada $(m^t_{x,y}(E))$ como lecturas contradictorias que apuntan lo contrario $(m^t_{x,y}(F))$. Si queremos resumir la creencia en un único valor, entonces necesitamos destilar esa creencia final convenientemente. En esa combinación irá implícitamente una compensación entre las evidencias de ocupación y vacío almacenadas en cada celdilla.

A.3. Enfoque borroso

En la aproximación borrosa, descrita en [Poloni *et al.*, 1995], [Gambino *et al.*, 1996],[Ribo y Pinz, 2001] y [Oriolo *et al.*, 1998], la rejilla de ocupación se almacena como *dos conjuntos borrosos* no complementarios: el de zonas vacías ε y el de zonas ocupadas o. Cada $celdilla(x,y)$ del espacio pertenece en cierta medida a cada uno de los conjuntos y esa pertenencia es una función de pertenencia borrosa $\mu_\varepsilon(x,y)$, $\mu_o(x,y)$.

La información de una lectura sónar k se captura igualmente con dos conjuntos borrosos ε^k y o^k, que reflejan precisamente la evidencia de vacío y ocupación que aporta esa lectura k a las diferentes celdillas del espacio. Por ejemplo, en [Poloni *et al.*, 1995] se utilizan los modelos de la figura A.2

A.3.1. Actualización con el operador borroso unión

Los conjuntos borrosos con las creencias globales se definen como la unión borrosa de las evidencias recogidas en cada lectura (A.16)(A.17). La operación de unión borrosa es asociativa, por ello estas ecuaciones (A.16) y (A.17) permiten una implementación incremental, eficiente desde el punto de vista

(a) (b)

Figura A.2: *Modelo borroso para la información de ocupación (a) y de vacío (b) de una lectura sónar [Poloni* et al.*, 1995]*

práctico. En la formulación clásica [Poloni *et al.*, 1995] se han propuesto varios operadores de unión borrosa: producto algebraico (A.18), producto acotado (A.19), operador Dombi, operador Yager.

$$o = \bigcup_{i=1}^{i=k} o^i = (\bigcup_{i=1}^{i=k-1} o^i) \cup o^k \tag{A.16}$$

$$\varepsilon = \bigcup_{i=1}^{i=k} \varepsilon^i = (\bigcup_{i=1}^{i=k-1} \varepsilon^i) \cup \varepsilon^k \tag{A.17}$$

$$(A \cup B)(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) * \mu_B(x) \tag{A.18}$$

$$(A \cup B)(x) = min(1, \mu_A(x) + \mu_B(x)) \tag{A.19}$$

Una de las ventajas de la aproximación borrosa señaladas en [Gambino *et al.*, 1996] es que no necesita tantas asunciones teóricas como el desarrollo probabilístico y se tiene más libertad a la hora de diseñar el modelo sensorial y los operadores borrosos.

Las evidencias de ocupación y de vacío no son contradictorias en este enfoque. Precisamente por ello, la aproximación borrosa exhibe una mayor robustez frente a medidas erróneas esporádicas que el enfoque probabilístico, como señalan [Gambino *et al.*, 1996] y [Oriolo *et al.*, 1998]. Combinando los dos conjuntos borrosos globales se puede distinguir entre zonas ambiguas y zonas desconocidas, es decir, entre la información contradictoria y la ausencia de información.

A.4. Enfoque histográmico

El enfoque histográmico fue presentado por Johann Borenstein y Y. Koren [Borenstein y Koren, 1991a]. En él cada celdilla mantiene un valor de certidumbre CV indicando la confianza en la existencia de un obstáculo en esa posición, que se mueve entre $CV_{min} = 0$ y $CV_{max} = 15$. Para utilizar la rejilla se suele binarizar la creencia de ocupación comparando el valor almacenado en cada celdilla con cierto umbral, por ejemplo 12. Sólo las casillas con evidencia superior se consideran realmente ocupadas.

El modelo histográmico utilizado en [Borenstein y Koren, 1991a] tiene geometría axial, sólo modifica las celdillas del eje central perpendicular al sensor que ha realizado la medida. Para la celdilla en el radio medido $\Delta_{obs}(t) = +3$ y en casillas más próximas $\Delta_{obs}(t) = -1$. La mezcla de información se hace empleando una regla aditiva heurística (A.20) que suma el valor del modelo sensorial al acumulado en la celdilla.

$$CV_{x,y}(t+1) = CV_{x,y}(t) + \Delta_{obs}(t) \tag{A.20}$$

En el trabajo de Borenstein [Borenstein y Koren, 1991a] sí hay un estudio explícito del carácter dinámico de la representación. La regla de actualización contempla la posibilidad de que la creencia pueda cambiar completamente de sentido con un número finito de observaciones sensoriales. La creencia puede cambiar tantas veces como se necesite e independientemente de lo confiado que se estuviera en

la creencia anterior. Se considera el *número crítico de medidas* necesarias para dar una creencia por firme. Ese valor marca la velocidad máxima de los obstáculos que puede reflejar la rejilla tal y como está construida.

Otra ventaja es que no necesita que las observaciones sensoriales sean independientes, se incorporan todas. Tampoco se hipotetiza cómo se distribuyen las medidas del sensor dada una configuración del mundo. Es la compensación entre unas y otras la que va conformando la *distribución de probabilidad* en el espacio.

A.5. Análisis comparado del dinamismo

En la literatura existen muchas comparativas [Gambino *et al.*, 1996],[Ribo y Pinz, 2001],[Poloni *et al.*, 1995],[Matía y Jiménez, 1998] pero pocas veces se ha evaluado explícitamente el comportamiento dinámico de las posibles reglas de actualización de la rejilla. Como a continuación justificaremos, los enfoques más utilizados tienen un carácter estático subyacente, en el cual no importa demasiado la velocidad en adquirir determinada creencia de ocupación, más bien su corrección. La compensación entre medidas persigue principalmente corregir algunas incertidumbres relativas al sensor.

En el caso de la rejilla dinámica de ocupación, el estado real de las celdillas puede cambiar con el tiempo. Por lo tanto, la regla de actualización debe también tratar de compensar las lecturas antiguas con las recientes. Es deseable que la creencia cambie rápidamente si las lecturas nuevas apuntan un cambio en el estado de ocupación actual, para reflejar con vivacidad los movimientos de los obstáculos. Este comportamiento dinámico afecta finalmente a la calidad de las rejillas de ocupación construidas, como se puede apreciar en el ejemplo de la figura A.1.

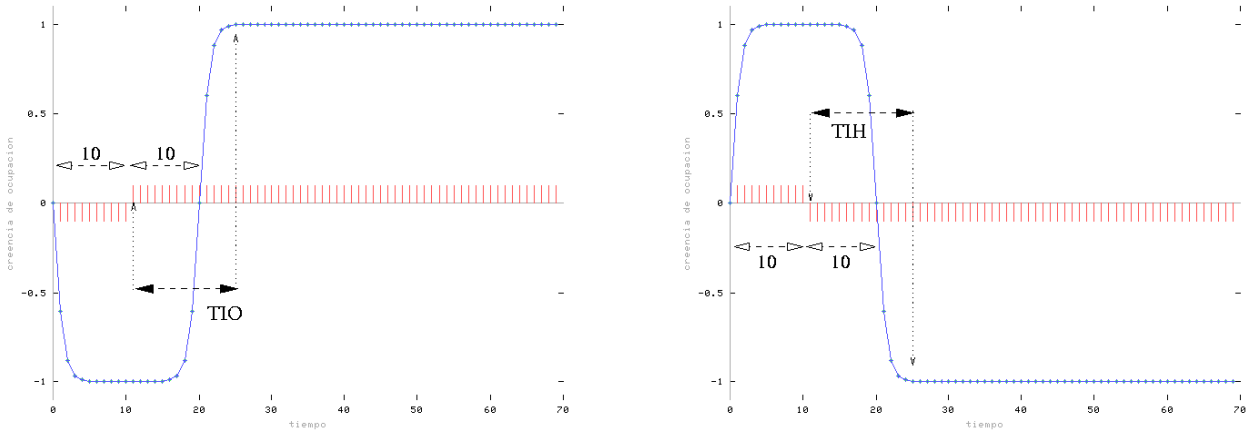
En contraposición a este criterio de vivacidad tenemos el de robustez frente a las lecturas inciertas. La compensación entre medidas inciertas y medidas válidas necesita cierta latencia para implementarse y una sola observación no modifica significativamente la creencia hasta que se confirma con nuevas observaciones. Entre estos dos criterios contrapuestos el algoritmo elegido debe establecer un compromiso.

Para poder comparar el dinamismo de las diferentes reglas de actualización hemos definido dos ratios que caracterizan su comportamiento dinámico: *tiempo en incorporar obstáculo (TIO)* y *tiempo en incorporar hueco (TIH)*. Estas dos propiedades miden precisamente el número de medidas necesarias para que la creencia de ocupación confirme la ocupación o el vacío, respectivamente. Para evaluar el TIO y TIH representativos de cada enfoque utilizaremos unas *secuencias de prueba*. Estas secuencias corresponden a observaciones que atañen a una misma celdilla y permiten ver la evolución temporal de la creencia en la ocupación de esa celdilla, actualizada con la regla correspondiente.

A.5.1. Dinamismo del enfoque probabilístico

El modelo probabilístico es el más difundido en la comunidad. Su carácter implícitamente estático se pone de manifiesto, por ejemplo, cuando tenemos una lectura sensorial absolutamente fiable. Por ejemplo, si la celdilla se observa vacía de modo muy fiable, $p(ocupada/obs(t)) = 0$, con lo cual $\rho_{obs} = 0$ y utilizando (A.9) lleva a 0 el ratio $\rho_{ocupada}(x, y, t)$ y ya no cambiará sean cuales sean las siguientes lecturas. Así, la estimación probabilística se queda bloqueada en $p(ocupada/obs(t)) = 0$. Del mismo modo, cuando $p(ocupada/obs) = 1$, el ratio $\rho_{ocupada}(x, y, t)$ pasa a valer ∞ y ya no se modificará. La estimación probabilística se queda enganchada entonces en $p(ocupada/obs(t)) = 1$. Estos bloqueos obedecen al carácter estático de la estimación probabilística: cuando una medida es absolutamente cierta entonces ella da el estado real, sea cual sea la creencia acumulada, y la estimación ya no debe cambiar. La regla de Bayes así lo refleja. Sin embargo, este funcionamiento obvia que el estado real puede cambiar con el tiempo.

Si no se tienen lecturas absolutamente ciertas, es decir, si los valores del modelo probabilístico de sensor se alejan de los extremos [0,1], entonces unas medidas compensan a otras. En estos casos, la regla de Bayes ofrece una determinada evolución temporal cuando las medidas que afectan a la celdilla cambian con el tiempo. En la figura A.3 se caracteriza esa evolución temporal a través de su TIO y su TIH. El eje horizontal representa el paso del tiempo, la incorporación de una nueva lectura en cada tic. El sentido de la lectura concreta que se añade en cada tic (ocupación o vacío) se aprecia



(a) (b)

Figura A.3: *TIO (a) y TIH (b) en el enfoque probabilístico. Para compensar 10 lecturas de vacío se necesitan 10 de ocupación y viceversa.*

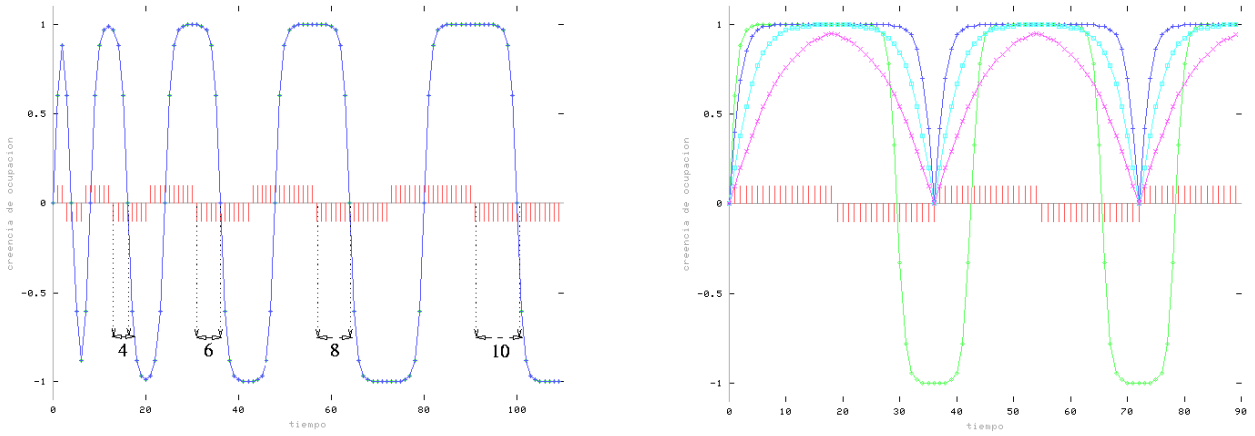
en las pequeñas barritas (+0.1 ó -0.1) alrededor de $y = 0$: si es positiva entonces esa lectura aporta una evidencia de ocupación sobre la celdilla en cuestión, si es negativa, evidencia de vacío. Estas lecturas concretas, vistas a lo largo del eje del tiempo, constituyen la secuencia de prueba. En el caso de la figura A.3(a) consta de 10 medidas iniciales de vacío y a continuación 60 lecturas que apuntan ocupación en la celdilla.

En el eje vertical tenemos el valor de la probabilidad de ocupación acumulada en cada momento, normalizada entre [-1,1]. El modelo sensorial empleado utiliza $p(ocupada/obs(t)) = 0{,}7$ para indicar ocupación y 0,3 para vacío. En la figura A.3(a) se observa que tarda unas 15 medidas en cambiar de creencia desde el instante 10 al 25, es decir tiene un *Tiempo en Incorporar Obstáculo* (TIO) de 15 lecturas. También se puede apreciar que para compensar la certeza de vacío que se ha conseguido con 10 lecturas de vacío se necesitan otras 10 lecturas en el sentido contrario. Con ese modelo, el TIH medido experimentalmente con la secuencia recíproca es el mismo que TIO, según se observa en la figura A.3(b).

También hemos constatado que se tarda más en cambiar de creencia (es decir, TIO y TIH mayores) cuantas más medidas soportan la creencia anterior. Esto se aprecia en la figura A.4(a), donde la secuencia de prueba consta de intervalos cada vez mayores de lecturas de ocupación y de vacío. La relación concreta entre TIH y TIO depende de los valores concretos del modelo sensorial, pero para valores de modelo simétricos (alrededor de 0.5) ambos parámetros toman igual valor. Esta inercia ralentiza en exceso el cambio de creencia cuando la presencia actual tiene muchas medidas pasadas que la avalan.

Se podría pensar que valores más pronunciados del modelo sensorial, valores más cercanos a los extremos de probabilidad, agilizan esta dinámica. Sin embargo, los experimentos realizados lo desmienten. La figura A.4(b) muestra cómo evoluciona la creencia para distintos valores del modelo probabilístico, $p(ocupada/obs(t))$ igual a 0.2, 0.3, 0.4 y 0.45 respectivamente. En este caso, la secuencia de prueba corresponde a varios periodos de longitud fija, alternando tantas lecturas de ocupación como lecturas de vacío. Para valores más extremos del modelo, los incrementos de probabilidad acumulada son efectivamente mayores que con modelos suaves. Sin embargo, la latencia en el cambio de opinión es exactamente la misma para los modelos con $p(ocupada/obs(t))$ 0.3, 0.4 y 0.45. Este experimento constata que el valor del modelo probabilístico no afecta a la latencia en cambiar de opinión.

Este comportamiento se puede explicar desde la ecuación (A.9), donde la compensación entre lecturas la marca el factor ρ_{obs}. Con modelos simétricos alrededor de 0.5, ese factor tiene valores inversos. Por ejemplo, para $p(ocupada/obs(t)) = 0{,}4$ y 0,6 se cumple que $\rho_{obs}(0{,}4) = \frac{1}{\rho_{obs}(0{,}6)}$. Esta simetría hace que multiplicar por $\rho_{obs}(0{,}4)$ y después por $\rho_{obs}(0{,}6)$ deje la probabilidad acumulada igual que estaba. Una observación de vacío compensa a otra de ocupación, y viceversa. Si la creencia de ocupación está avalada por n lecturas, entonces se necesitan otras tantas en sentido contrario para



(a) (b)

Figura A.4: *La inercia probabilística depende de la certeza acumulada (a). Evolución de la creencia con modelos probabilísticos simétricos $p(ocupada/obs(t))$ =0.45, 0.4, 0.3 y 0.2 respectivamente (b)*

compensar esa certeza acumulada. Si el modelo es más extremo, entonces la probabilidad de ocupación sube más rápidamente para el mismo número de lecturas que con modelos suaves, sin embargo el valor de probabilidad alcanza valores mayores y por eso tarda más en bajar, aunque esa bajada también se haga más deprisa que con modelos suaves. Valores asimétricos del modelo sólo consiguen acelerar las subidas de probabilidad frente a las bajadas, o viceversa.

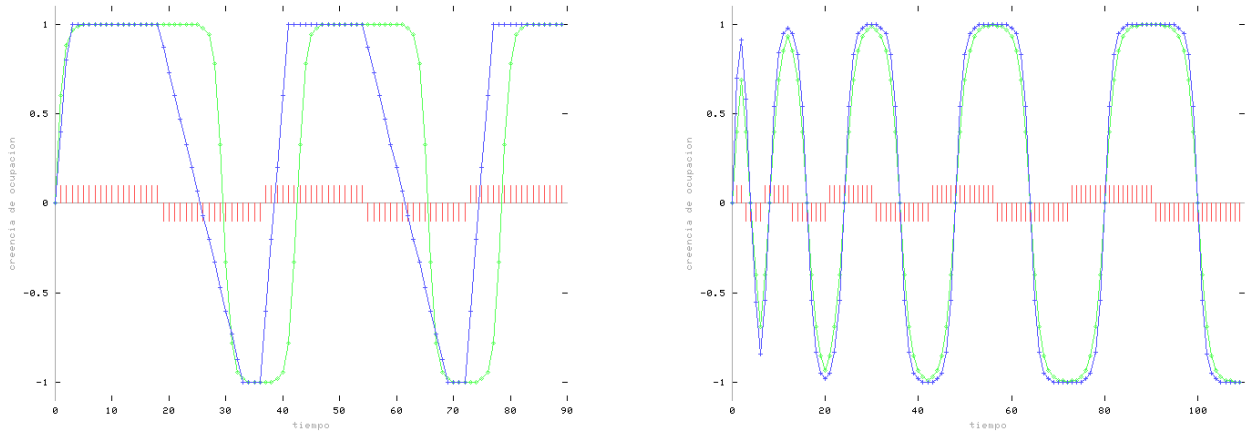
En la práctica, la regla de Bayes de la ecuación (A.9) deja de aplicarse para valores extremos de probabilidad. Por limitaciones de representación el ordenador no puede distinguir valores muy próximos a 1 de 1, ni valores muy próximos a 0 de 0. Una vez que la probabilidad acumulada alcanza 1 ó 0 la regla de Bayes ya no modifica ese valor, sea cual sea la observación. Para evitar este bloqueo debido exclusivamente cuestiones prácticas, normalmente se acota el valor de la probabilidad entre $[\delta, 1-\delta]$ (por ejemplo con $\delta = 0{,}0000001$) y se trunca el valor devuelto por la actualización (A.9) para mantenerlo dentro de ese intervalo. Gracias a esta *saturación*, dejan de acumularse evidencias en los extremos, que aproximarían aún más la creencia a 1 ó a 0. Este efecto limita la inercia implícita en la aproximación probabilística y hace que valores de modelo más pronunciados tengan efectivamente dinámicas más veloces de cambio de opinión cuando se entra en saturación. Se alcanza antes la saturación que con modelos suaves. Esto explica que la latencia en cambiar de opinión sea menor en la figura A.4(b) para el modelo con $p(ocupada/obs(t)) = 0{,}2$. Es el único que consigue cambiar realmente de opinión porque efectivamente alcanza la saturación. El valor exacto de δ y el modelo marcan el tiempo preciso en cambiar de opinión, que entonces es independiente de la cantidad de medidas que avalan la creencia anterior.

A.5.2. Dinamismo de la teoría de evidencia

Siguiendo la teoría de la evidencia la información de una lectura sónar en una celdilla se representa por dos valores: $(m_{obs}(E), m_{obs}(F))$. Estos valores actualizan la creencia acumulada siguiendo la regla de Dempster-Shafer (A.15). Esta regla hace que las evidencias acumuladas, $m_{x,y}(E)$ y $m_{x,y}(F)$, puedan subir o bajar dependiendo de la lectura concreta incorporada.

Al igual que ocurre en el caso probabilístico cuando tenemos una lectura cierta, bien de ocupación $(0, 1)$ bien de vacío $(1, 0)$, esta información lleva la creencia acumulada a la certeza y ahí se queda bloqueada irremediablemente, con independencia de las nuevas medidas que se vayan obteniendo. Este bloqueo refleja el carácter estático inherente a esta aproximación.

En los experimentos realizados, después de la incorporación de unas lecturas iniciales, la evolución de la creencia es paralela a la evolución probabilística. Por ejemplo, la figura A.5(b) muestra la evolución de ambos enfoques frente a la misma secuencia de prueba. Salvo pequeñas diferencias son indistinguibles. La acumulación inicial de evidencias reduce la incertidumbre $m_{x,y}(E, F)$ hasta valores próximos a cero y entonces la regla de Dempster-Shafer ofrece el mismo dinamismo que la regla de



(a) (b)

Figura A.5: *Latencia en el enfoque histográmico (a) y con teoría de la evidencia (b) frente al probabilístico (en verde).*

Bayes. Los mismos comentarios sobre la inercia dependiente de la evidencia acumulada son aplicables en este caso también.

Por la misma limitación práctica que comentamos para el caso probabilístico, la evidencia acumulada con este enfoque puede quedarse bloqueada si alcanza esos valores indistiguibles de 1 ó 0. Para evitar este bloqueo también hemos acotado el valor de la evidencia $m_{x,y}(E)$ y $m_{x,y}(F)$ entre $[\delta, 1-\delta]$. Esto se traduce en que forzamos una incertidumbre mínima de $m_{x,y}(E,F) = 2\delta$.

A.5.3. Dinamismo del enfoque borroso

Con el enfoque borroso, a medida que se recogen lecturas sónares se van actualizando los conjuntos borrosos vacíos ε y ocupados o con el operador borroso unión. Con los operadores borrosos del producto algebraico (A.18), producto acotado (A.19), operador Dombi y operador Yager utilizados en [Poloni *et al.*, 1995], [Gambino *et al.*, 1996] y [Ribo y Pinz, 2001] se produce un bloqueo irreversible que inhabilita este enfoque cuando se tienen *muchas* medidas sonar. Con esos operadores, el grado de pertenencia es una función creciente, que nunca disminuye y acaba llegando a 1. Cuando tanto μ_ε como μ_o alcanzan su valor máximo entonces, sea cual sea la compensación que se establezca entre unos y otros, la creencia final de ocupación no cambia, se bloquea indefinidamente.

El problema principal radica en que la evidencia acumulada no disminuye, lo cual no permite rectificar. Supongamos que inicialmente una celdilla está vacía, entonces acumula cierta evidencia de vacío. Si su estado real cambia entonces comienza a acumularse evidencia de ocupación de tal manera que la creencia final $F = \varepsilon \cap \overline{o}$ se acerca más a la ocupación. Sin embargo, la evidencia anterior no se olvida. Llega un momento en que ε u o alcanzan su valor máximo y ya no cambian su valor. Fenómenos como éste ponen de manifiesto la necesidad de un mecanismo de olvido que permita neutralizar la influencia de unas medidas que ya no se corresponden con el estado actual de la celdilla.

Para solventar este bloqueo, en trabajos más recientes se redefine el operador borroso de unión como una media aritmética ponderada [Oriolo *et al.*, 1998], que da mayor peso a la creencia acumulada. La función de pertenencia al conjunto de celdillas libres ε y al de celdillas ocupadas o entonces sí pueden disminuir cuando se incorpora la información de una nueva lectura.

A.5.4. Dinamismo en enfoque histográmico

Este enfoque tiene una aritmética sencilla de compensación entre unas medidas y otras dada por el modelo sensorial. Esta aritmética hace pesar +3 a las evidencias de ocupación y -1 a las de vacío. Por esta razón se tarda tres veces menos en incorporar un obstáculo TIO que en incorporar un nuevo hueco TIH , según ilustra la figura A.5(a). La linealidad del cambio de creencia contrasta con el efecto más o menos exponencial del enfoque probabilístico en la misma figura (modelo: $p(ocupada/obs(t)) = 0{,}2$

y 0,8).

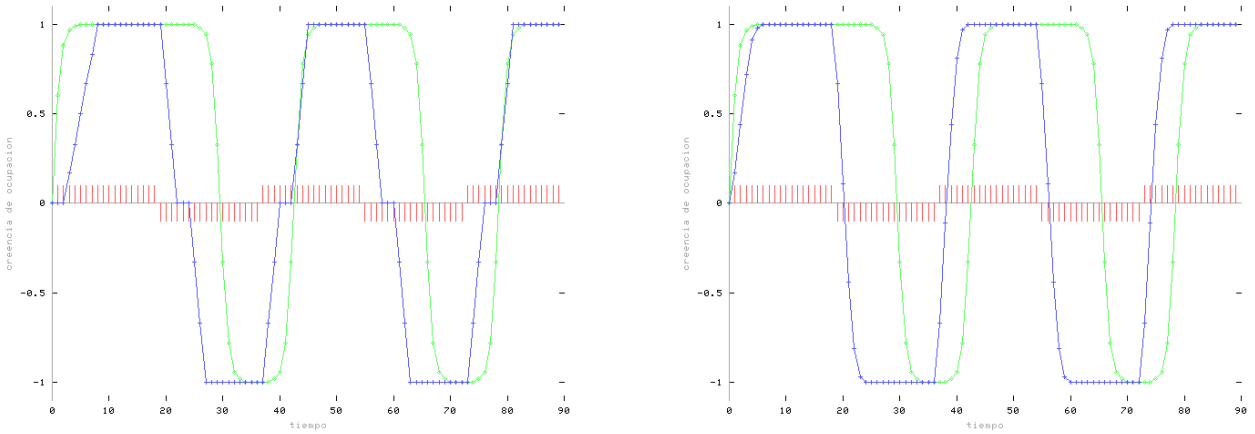
Por otro lado la creencia de ocupación se mueve aquí dentro del intervalo [0,15], saturando en ambos extremos. Si la creencia ya se encuentra en un extremo nuevas lecturas en el mismo sentido no cambian literalmente en nada la creencia acumulada, que ya es máxima.

A.5.5. Dinamismo los enfoques con ecuación diferencial y por mayoría

Una vez estudiado el dinamismo de las técnicas de fusión más conocidas, describiremos ahora el de las dos técnicas utilizables en el esquema perceptivo que mantiene la rejilla de ocupación. Ambas fueron descritas con detalle en la sección 5.1.

En la figura A.6(b) se aprecia que el enfoque histográmico con ecuación diferencial puede cambiar más rápidamente de creencia que el enfoque probabilístico (modelo típico: $p(ocupada/obs(t)) = 0{,}2$ y 0,8). Además, este dinamismo contrasta con el cambio de creencia *a ritmo constante* (en el producto) que se da en el enfoque bayesiano.

También se puede observar cierta saturación cuando se acumulan evidencias en el mismo sentido, de modo que observaciones recurrentes cada vez provocan menor incremento en la creencia. Sin embargo, cuando la nueva observación contradice la creencia acumulada entonces el cambio es significativo, aunque el *factor secuencia* ayuda a diferir ese efecto hasta que se confirma con nuevas observaciones. Por ejemplo, se puede ver como la primera observación contraria no tiene tanta influencia como una segunda y una tercera también contrarias.



(a) (b)

Figura A.6: *Latencia en el enfoque por mayoría (a) y con ecuación diferencial (b) frente a probabilístico (en verde).*

En la figura A.6(a) se aprecia que la creencia actualizada con el enfoque por mayoría es capaz de evolucionar en ambos sentidos si suficientes medidas avalan el cambio. Además de la saturación, también se aprecia el efecto del umbral de ruido, que mantiene la creencia nula (indecisa) hasta que se acumulan suficientes evidencias. La latencia depende del tamaño N de la memoria y los umbrales. Para los experimentos realizados ($N = 10$, $umbral\ ruido = 2$ y $umbral\ saturacion = 8$) se ofrece menor latencia que el modelo típico con la técnica probabilística ($p(ocupada/obs(t)) = 0{,}2$ y 0,8).

A.6. Discusión

En esta sección se han repasado las técnicas más populares de construcción y mantenimiento de mapas métricos en forma de rejilla. Aunque la comparación exacta depende de los valores numéricos concretos de los modelos sensoriales, la comparativa pone de manifiesto fenómenos sistemáticos que sí obedecen a la naturaleza de la regla de actualización. Estos fenómenos caracterizan el comportamiento dinámico de la creencia de ocupación en cada enfoque, y por ello, la calidad de las rejillas dinámicas de ocupación construidas con ellos.

El enfoque probabilísitico, la teoría de evidencia y el enfoque borroso son técnicas perfectamente válidas para construir mapas estáticos, como avalan los abundantes trabajos de la literatura. En nuestro caso, queremos aplicar estas exitosas técnicas a un problema completamente distinto, como es la construcción de mapas dinámicos. En concreto, como reglas de actualización de la rejilla dinámica de ocupación, que está orientada a capturar los obstáculos, móviles o no, alrededor del robot. Los experimentos realizados, tanto con las secuencias de prueba como las rejillas construidas (figura A.1) constatan que no ofrecen un buen rendimiento en este ámbito dinámico. Este mal comportamiento nos hizo desarrollar dos nuevas técnicas, más orientadas a preservar en la rejilla el dinamismo de los obstáculos: la actualización con ecuación diferencial y por mayoría.

El enfoque probabilístico bayesiano muestra una inercia proporcional a las evidencias acumuladas, lo que ralentiza en exceso su cambio de creencia. En general, necesita tantas evidencias de ocupación como de desocupación para cambiar el sentido de su estimación. En la práctica, ofrece un mayor dinamismo debido a una limitación práctica que obliga a manejar valores de probalibidad en el intervalo $[\delta, 1-\delta]$. Sin embargo, esta restricción desvirtúa todas las asunciones de probabilidad hechas y no forma parte explícita del formalismo bayesiano.

La teoría de la evidencia deriva en los mismos resultados que la probabilística una vez que se incorpora un reducido número de lecturas iniciales. Después de esas medidas, la ambigüedad en la estimación $m_{x,y}(E,F)$ se anula y la evolución de ambos enfoques es similar. Las mismas restricciones se aplican.

El enfoque borroso clásico presenta un bloqueo inaceptable tras incorporar un pequeño número de lecturas sensoriales. No funciona bien si se tiene un flujo continuo de medidas. La razón de este bloqueo radica que el operador borroso de unión sea una función monótona creciente. Nuevos trabajos dentro de este enfoque [Oriolo *et al.*, 1998] proponen nuevos operadores borrosos que superan este bloqueo y lo acercan a una sencilla media aritmética.

Así pues, estos enfoques resultan inválidos para representar características que puedan cambiar con el tiempo, por ejemplo la ocupación del espacio cuando hay obstáculos móviles. La principal razón que hemos identificado es que tanto la regla de Bayes, la regla de Dempster-Shafer y el operador borroso de unión exhiben la *propiedad asociativa*: dada una secuencia de lecturas sensoriales, el estado final de las celdillas de la rejilla es el mismo con independencia del orden en que se incorporen esas lecturas: manejando probabilidad, $p(ocupada/obs_1, obs_2 ... obs_n) = p(ocupada/obs_n, obs_{n-1} ... obs_1)$; el operador borroso de unión $\varepsilon = \bigcup_{i=1}^{i=k} \varepsilon^i$ es asociativo; y lo mismo ocurre con la regla de Dempster-Shafer, $((m_1 \oplus m_2) \oplus m_3)(E) = ((m_2 \oplus m_3) \oplus m_1)(E)$.

Esta propiedad les confiere un carácter inherentemente estático que les invalida para representar características dinámicas. En el caso de las rejillas dinámicas de ocupación interesa que los valores sensoriales recientes pesen más que los antiguos, posiblemente obsoletos, a la hora de estimar la situación real. Sin embargo, en estos enfoques la influencia de una medida en la creencia no depende de la antigüedad de la observación, todas pesan por igual, y su efecto no se olvida con el paso del tiempo. Esto hace que la influencia de una nueva observación tenga a *todas* las lecturas pasadas como inercia.

Por ejemplo, supongamos una $secuencia_A$ de 200 observaciones con las 100 primeras apuntando ocupación y las 100 últimas apuntando vacío, correspondiendo a un obstáculo que se sale de una celdilla. Asumamos también una $secuencia_B$, inversa de la primera, correspondiente a un obstáculo que irrumpe en una celdilla, anteriormente vacía. Sin mencionar los bloqueos que hemos comentado anteriormente, la creencia acumulada sería la misma con ambas secuencias, debido a la propiedad asociativa. Sin embargo, parece claro que la creencia final debería ser distinta para una secuencia que para su inversa.

Por el contrario el enfoque histográmico, de decisión por mayoría y el basado en ecuación diferencial sí reflejan el dinamismo de la realidad. Todos ellos distinguen entre $secuencia_A$ y $secuencia_B$, el estado final es distinto en ambos casos. En estos enfoques, por muy seguros que estemos de que tal celdilla está ocupada, basta un número relevante de lecturas en sentido contrario para cambiar radicalmente de creencia. Este dinamismo de representación es imprescindible para representar obstáculos móviles y resulta útil incluso con obstáculos estáticos si se arrastran errores de localización[2]. No es casualidad

[2]El uso de técnicas estáticas obliga a mantener una localización absoluta precisa, para no mezclar evidencias de

que en otros trabajos que también utilizan dinamismo, como [Murphy *et al.*, 1999], hayan tratado sólo rejillas histográmicas.

Un modo relativamente sencillo de hacer más dinámicas las técnicas que hemos identificado como estáticas consiste en incorporar una ventana de tiempo. De este modo, la creencia de ocupación en una celdilla se calcula sólo desde las observaciones obtenidas en un cierto intervalo a descontar desde el instante actual. Con ello se rompen algunas asunciones de base, pero se garantiza que toda la información que se incorpora en la estimación es relativamente reciente. Este mecanismo tiene además otras consecuencias, como la tendencia al olvido si no hay evidencias nuevas. Un trabajo relevante en esta línea es el de Arbuckle [Arbuckle *et al.*, 2002].

distintas celdillas.

Bibliografía

[ActivMedia, 1999] ActivMedia. Saphira operations and programming manual. Technical Report version 6.2, ActivMedia Robotics, August 1999.

[ActivMedia, 2002] ActivMedia. ARIA reference manual. Technical Report version 1.1.10, ActivMedia Robotics, November 2002.

[Agre y Chapman, 1987] Philip E. Agre y David Chapman. Pengi: an implementation of a theory of activity. In *Proceedings of 6th AAAI National Conference on Artificial Intelligence*, pages 268–272, Seattle, WA, 1987. Morgan Kaufmann.

[Agre y Chapman, 1990] Philip E. Agre y David Chapman. What are plans for? In Pattie Maes, editor, *Designing Autonomous Agents: theory and practice from Biology to Engineering and Back*, pages 17–34. MIT Press, 1990.

[Aguirre *et al.*, 1998] Eugenio Aguirre, María García-Alegre, y Antonio González. A fuzzy safe follow wall behavior fusing simpler fuzzy behaviors. In *Proceedings of the 3rd IFAC Symposium on Intelligent Autonomous Vehicles IAV'98*, pages 607–612, Madrid, March 1998.

[Aguirre *et al.*, 2001] Eugenio Aguirre, Juan C. Gámez, y Antonio González. Un sistema multiagente basado en lógica difusa aplicado al ámbito de la robobup. In *Actas del 2do Workshop de Agentes Físicos, WAF'2001*, pages 131–145, Móstoles, Madrid, March 2001. ISBN 84/699-4585-8.

[Albus, 1993] James S. Albus. A reference model architecture for intelligent systems design. In Panos J. Antsakis y Kevin M. Passino, editors, *An introduction to intelligent and autonomous control*, pages 27–56. Kluwer Academic Press, 1993. ISBN: 0-7923-9267-1.

[Albus, 1996] James S. Albus. The engineering of mind. In Pattie Maes, Maja J. Mataric, JeanArcady Meyer, Jordan B. Pollack, y Stewart W. Wilson, editors, *Proceedings of the 4th International Conference on Simulation of Adaptive Behavior SAB'96*, pages 23–32. MIT Press, 1996.

[Albus, 1999] James S. Albus. The engineering of mind. *Information Sciences*, 117(1-2):1–18, 1999.

[Ali y Arkin, 1998] Khaled S. Ali y Ronald C. Arkin. Implementing schema-theoretic models of animal behavior in robotics systems. In *Proceedings of the 5th International Workshop on Advanced Motion Control, AMC'98*, pages 246–253, Coimbra (Portugal), 1998.

[Arbib y H., 1987] M. A. Arbib y Hanson D. H. Depth and detours: an essay on visually guided behavior. In M.A. Arbib y A.R. Hanson, editors, *Vision, Brain and Cooperative Computation*, pages 129–163. A Bradford Book/ MIT Press, 1987.

[Arbib y Liaw, 1995] Michael A. Arbib y Jim-Shih Liaw. Sensorimotor transformations in the worlds of frogs and robots. *Artificial Intelligence*, 72:53–79, 1995.

[Arbib, 1981] M. A. Arbib. Perceptual structures and distributed motor control. In Brooks V.B., editor, *Handbook of Physiology. The nervous system II: motor control*, pages 1449–1480. American Physiological Society, 1981.

[Arbib, 1989] Michael A. Arbib. Schemas and neural networks for sixth generation computing. *Journal of parallel and distributed computing*, 6:185–216, 1989.

[Arbuckle *et al.*, 2002] Daniel Arbuckle, Andrew Howard, y Maja Mataric. Temporal occupancy grids: a method for classifying the spatio-temporal properties of the environment. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-02)*, Switzerland, September 2002.

[Arkin *et al.*, 2000] Ronald C. Arkin, Khaled S. Ali, Alfredo Weitzenfeld, y Francisco Cervantes-Perez. Behavioral models of the praying mantis as a basis for robotic behavior. *Robotics and Autonomous Systems*, pages 39–60, 2000.

[Arkin *et al.*, 2001] Ronald C. Arkin, Masahiro Fujita, Tsuyoshi Takagi, y Rika Hasegawa. Ethological modeling and architecture for an entertainment robot. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, volume 1, pages 453–458, Seoul, Korea, May 2001.

[Arkin *et al.*, 2003] Ronald C. Arkin, Masahiro Fujita, Tsuyoshi Takagi, y Rika Hasegawa. An ethological and emotional basis human-robot interaction. *Robotics and Autonomous Systems*, 42:191–201, 2003.

[Arkin y Balch, 1997] Ronald C. Arkin y Tucker Balch. Aura: principles and practice in review. *Journal of Experimental and Theoretical AI*, 9(2-3):175–188, April-September 1997.

[Arkin y MacKenzie, 1994] Ronald C. Arkin y Douglas MacKenzie. Temporal coordination of perceptual algorithms for mobile robot navigation. *IEEE Transactions on Robotics and Automation*, 10(3):276–286, June 1994.

[Arkin, 1989a] Ronald C. Arkin. Dynamic replanning for a mobile robot based on internal sensing. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, volume 3, pages 1416–1421, 1989.

[Arkin, 1989b] Ronald C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, August 1989.

[Arkin, 1995] Ronald C. Arkin. Reactive robotic systems. In M. Arbib, editor, *Handbook of the brain theory and neural networks*, pages 793–796. MIT Press, 1995.

[Arkin, 1998] Ronald C. Arkin. *Behavior based robotics*. MIT Press, 1998.

[Baerends, 1976] Gerard P. Baerends. The functional organization of behavior. *Animal Behavior*, 24:726–738, 1976.

[Bajcsy, 1988] R. Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):996–1005, August 1988.

[Benayas *et al.*, 2002] J.A. Benayas, J.L. Fernández, R. Sanz, y A.R. Diéguez. The beam curvature method: a new approach for improving local real time obstacle avoidance. In *Proceedings of the 15th IFAC Triennial World Congress*, Barcelona (Spain), July 2002.

[Betgé-Brezetz *et al.*, 1996] Stéphane Betgé-Brezetz, Hébert Patrick, Raja Chatila, y Michel Devy. Uncertain map making in natural environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1048–1053, Minneapolis (USA), April 1996.

[Blumberg, 1994] Bruce Blumberg. Action-selection in hamsterdam: lessons from ethology. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior SAB'94*, pages 108–117, Brighton, 1994.

[Blumberg, 1997] Bruce M. Blumberg. *Old tricks, new dogs: ethology and interactive creatures*. PhD thesis, Massachusetts Institute of Technology, 1997.

[Bonasso *et al.*, 1997] R. Peter Bonasso, R. James Firby, Erann Gatt, David Kortenkamp, David P. Miller, y Marc G. Slack. Experiences with an architecture for intelligent reactive agents. *Journal of Experimental and Theoretical AI*, 9(2):237–256, 1997.

[Borenstein y Koren, 1989] J. Borenstein y Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):1179–1187, September 1989.

[Borenstein y Koren, 1991a] J. Borenstein y Y. Koren. Histogramic in-motion mapping for mobile robot obstacle avoidance. *IEEE Journal of Robotics and Automation*, 7(4):535–539, 1991.

[Borenstein y Koren, 1991b] J. Borenstein y Y. Koren. The vector field histogram fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, June 1991.

[Boumaza y Louchet, 2001] A.M. Boumaza y J. Louchet. Dynamic flies: using real-time parisian evolution in robotics. In Egbert J. W. Boers, Jens Gottlieb, Pier Luca Lanzi, Robert E. Smith, Stefano Cagno ni, Emma Hart, Günther R. Raidl, y H. Tijink, editors, *Applications of Evolutionary Computing, EvoWorkshops 2001*, volume 2037 of *Lecture Notes in Computer Science*, pages 288–297. Springer, 2001.

[Braunl y Graf, 1999] Thomas Braunl y Birgit Graf. Autonomous mobile robots with on-board vision and local intelligence. In *Proceedings of 2nd IEEE Workshop on Perception for Mobile Agents*, pages 51–57, Fort Collins (CO, USA), 1999.

[Braunl, 1999] Thomas Braunl. Eyebot: a family of autonomous mobile robots. In *Proceedings of 6th IEEE International Conference on Neural Information Processing, ICONIP'99*, volume 2, pages 645–649a, Perth (WA, Australia), 1999.

[Brooks *et al.*, 1988] Rodney A. Brooks, Jonathan H. Connell, y Peter Ning. Herbert: a second generation mobile robot. Technical Report AI-Memo 1016, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, January 1988.

[Brooks *et al.*, 1998] Rodney A. Brooks, Cynthia Breazeal, Robert Irie, Charles C. Kemp, Matthew Marjanovic, Brian Scassellati, y Matthew M. Williamson. Alternative essences of intelligence. In *Proceedings of the 15th AAAI National Conference on Artificial Intelligence*, pages 961–968, Madison (Wisconsin), July 1998.

[Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[Brooks, 1987] Rodney A. Brooks. A hardware retargetable distributed layered architecture for mobile robot control. In *Proceedings of the 1987 International Conference on Robotics and Automation*, pages 106–110, Raleigh-NC, March 1987. Computer Society Press.

[Brooks, 1991a] Rodney A. Brooks. Challenges for complete creature architectures. In J-A. Meyer y S.W. Wilson, editors, *From Animals to Animats*, pages 434–443. MIT Press, 1991.

[Brooks, 1991b] Rodney A. Brooks. Intelligence without reason. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 569–595, 1991.

[Brooks, 1991c] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence Journal*, 47:139–160, 1991.

[Budenske y Gini, 1994] John Budenske y Maria Gini. Why is it so difficult for a robot to pass through a doorway using ultrasonic sensors? In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 3124–3129, San Diego, CA (USA), 1994.

[Budenske y Gini, 1997] John Budenske y Maria Gini. Sensor explication: knowledge-based robotic plan execution through logical objects. *IEEE Transactions on Systems Man and Cybernetics*, 27(4):611–625, 1997.

[Bustos *et al.*, 2001] P. Bustos, P. Bachiller, J. Vicente, M. Broncano, y C. Fernández. Murphy: hacia un robot con visión estereoscópica. In *Actas del 2do Workshop de Agentes Físicos, WAF'2001*, pages 91–104, Móstoles, Madrid, March 2001. ISBN 84/699-4585-8.

[Bustos, 1997] Pablo Bustos. *Generación de comportamiento complejo en robots autónomos.* PhD thesis, Universidad Politécnica de Madrid, 1997. Facultad de Informática.

[Capek, 1923] Karel Capek. *RUR: Rossum's universal robots.* 1923.

[Cañamero *et al.*, 2002] Lola Cañamero, Orlando Avila-García, y Elena Hafner. First experiments relating behavior selection architectures to environmental complexity. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3024–3029, Lausanne (Switzerland), October 2002.

[Cañamero, 1997] Dolores Cañamero. Modeling motivations and emotions as a basis for intelligent behavior. In *Proceedings of the ACM International Conference on Autonomous Agents*, pages 148–155, Marina del Rey (USA), feb 1997.

[Cañamero, 2000] Dolores Cañamero. Designing emotions for activity selection. Technical Report PB 545, Dept. of Computer Science, University of Aarhus (Denmark), February 2000.

[Cañas *et al.*, 2001] José M. Cañas, Reid Simmons, y María García-Alegre. Detección probabilística de puertas con visión monocular activa. In *Actas del II Workshop de Agentes Físicos, WAF'2001*, pages 113–128, Universidad Rey Juan Carlos, March 2001.

[Cañas y García-Alegre, 1998] José María Cañas y María García-Alegre. Segmentación estadística em en tiempo real del entorno local de un robot móvil. Technical Report TR-09/98, Departamento de Sistemas, Instituto de Automática Industrial (CSIC), 1998.

[Cañas y García-Alegre, 1999a] José M. Cañas y María García-Alegre. Real time EM segmentation of occupancy grid for robots navigation. In *Proceedings of IJCAI-99 Workshop Adaptive Spatial Representations of Dynamic Environments*, pages 75–79, Estocolmo, Suecia, August 1999.

[Cañas y García-Alegre, 1999b] José M. Cañas y María C. García-Alegre. Modulated agents for autonomous robot piloting. In *Actas de la VIII Conferencia de la Asociación Española para la Inteligencia Artificial*, Universidad de Murcia, 1999.

[Cañas y Matellán, 2002] José M. Cañas y Vicente Matellán. Dynamic schema hierarchies for an autonomous robot. In José C. Riquelme y Miguel Toro Francisco J. Garijo, editor, *Advances in Artificial Intelligence - IBERAMIA 2002*, volume 2527 of *Lecture notes in artificial intelligence*, pages 903–912. Springer, 2002.

[Chang *et al.*, 2001] Mark Chang, Brett Browning, y Gordon Wyeth. ViperRoos 2000. In T.Balch P.Stone y G.Kraetzschmar, editors, *Robocup-2000: Robot soccer world cup V, Lecture Notes in Artificial Intelligence 2019*, pages 527–530. Springer-Verlag, 2001.

[Chang *et al.*, 2002] Mark Chang, Brett Browning, y Gordon Wyeth. ViperRoos: developing a low cost local vision team for the small size league. In S.Coradeschi A.Birk y S.Takodoro, editors, *Robocup-2001: Robot soccer world cup V, Lecture Notes in Artificial Intelligence 2377*, pages 305–311. Springer-Verlag, 2002.

[Chatila, 1995] Raja Chatila. Deliberation and reactivity in autonomous mobile robots. *International Journal on Robotics and Autonomous Systems*, 16:197–211, 1995.

[Cires *et al.*, 1998] Juan Cires, Felipe Bertrand, y Pedro J. Zufiria. Two modeling approaches for navigation control of a nomad 200 mobile robot. In *Proceedings of the 3rd IFAC symposium on Intelligent Autonomous Vehicles*, Madrid, March 1998.

[Collins, 1996] Robert T. Collins. A space-sweep approach to true multi-image matching. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 358–363, San Francisco, June 1996.

[Collins, 1997] Robert T. Collins. Multi-image focus of attention for rapid site model construction. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 575–581, San Juan, Puerto Rico, June 1997.

[Connell, 1992] Jonathan H. Connell. Sss: a hybrid architecture applied to robot navigation. In *Proceedings of the 1992 IEEE Conference on Robotics and Automation ICRA'92*, pages 2719–2724, Nice (France), May 1992.

[Coradeschi y Saffiotti, 2001] Silvia Coradeschi y Alessandro Saffiotti. Perceptual anchoring of symbols for action. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 407–412, Seattle, WA (USA), 2001.

[Corbacho y Arbib, 1995] Fernando J. Corbacho y Michael A. Arbib. Learning to detour. *Adaptive Behavior*, 5(4):419–468, 1995.

[Coste-Maniére y Simmons, 2000] Ève Coste-Maniére y Reid G. Simmons. Architecture, the backbone of robotic systems. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 67–72, San Francisco, CA (USA), April 2000.

[Cox y Leonard, 1994] Ingemar J. Cox y John J. Leonard. Modelling a dynamic environment using a bayesian multiple hypothesis approach. *Artificial Intelligence*, 66:311–344, 1994.

[Crowley, 1985] James L. Crowley. Navigation for an intelligent mobile robot. *IEEE Journal of Robotics and Automation*, 1(1):31–41, March 1985.

[Crowley, 1996] J. Crowley. Mathematical foundations of navigation an perception for an autonomous mobile robot. In L. Dorst, editor, *Reasoning with Uncertainty*, pages 9–51. Springer Verlag, 1996.

[de la Rosa *et al.*, 1997] J. Ll. de la Rosa, A. Oller, J. Vehí, y J. Puyol. Soccer team based on agent-oriented programming. *International Journal on Robotics and Autonomous Systems*, 21(2):167–176, 1997.

[Delahoche *et al.*, 1998] Laurent Delahoche, Claude Pégard, El Mustapha Mouaddib, y Pascal Vasseur. Incremental map building for mobile robot navigation in an indoor environment. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2560–2565, Leuven (Belgium), May 1998.

[Dorer, 1999] Klaus Dorer. Behavior networks for continuous domains using situation-dependent motivations. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Stockolm, 1999.

[Draper *et al.*, 1989] Bruce A. Draper, Robert T. Collins, John Brolio, Allen R. Hanson, y Riseman Edward M. The schema system. *International Journal of Computer Vision*, 2:209–250, 1989.

[Duchon *et al.*, 1998] Andrew P. Duchon, William H. Harren, y Leslie Pack Kaelbling. Ecological robotics. *Adaptive Behavior*, 6(3/4):473–507, 1998. Special Issue on Biologically Inspired Models of Spatial Navigation.

[Elfes, 1990] Alberto Elfes. Occupancy grids: a stochastic spatial representation for active robot perception. In *Proceedings of the 6th AAAI Conference on Uncertainty in AI*, pages 60–70, July 1990.

[Española, 1992] Real Academia Española. *Diccionario de la lengua española.* 21 edition, 1992.

[Fikes y Nilsson, 1971] R.E. Fikes y N.J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 1971.

[Firby, 1987] R. James Firby. An investigation into reactive planning in complex domains. In *Proceedings of the 6th AAAI National Conference on Artificial Intelligence*, pages 202–206, Seattle, WA, 1987.

[Firby, 1992] R. James Firby. Buiding symbolic primitives with continuous control routines. In *Proceedings of the 1st International Conference on AI Planning Systems AIPS'92*, pages 62–69, College Park, MD (USA), June 1992.

[Firby, 1994] R. James Firby. Task networks for controlling continuous processes. In *Proceedings of the 2nd International Conference on AI Planning Systems AIPS'94*, pages 49–54, Chicago, IL (USA), June 1994.

[Fischer *et al.*, 1994] Klaus Fischer, Jörg P. Müller, y Markus Pischel. Unifying control in a layered agent architecture. Technical Report TM-94-05, German Research Center for Artificial Intelligence (DFKI GmbH), 1994.

[Flynn, 1985] Anita M. Flynn. Redundant sensors for mobile robot navigation. Master's thesis, Massachusetts Institute of Technology, October 1985.

[Flynn, 1988] Anita M. Flynn. Combining sonar and infrared sensors for mobile robot navigation. *The International Journal of Robotics Research*, 7(6):5–14, December 1988.

[Fox *et al.*, 1997] Dieter Fox, Wolfram Burgard, y Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, pages 23–33, March 1997.

[Gallardo *et al.*, 1998] Domingo Gallardo, Francisco Escolano, Ramón Rizo, Otto Colomina, y Miguel Angel Cazorla. Estimación bayesiana de características en robots móviles mediante muestreo de la densidad a posteriori. In *Actas del I Congrés Catalá d'Intelligéncia Artificial*, Tarragona, October 1998.

[Gambino *et al.*, 1996] Fabio Gambino, Giuseppe Oriolo, y Giovanni Ulivi. A comparison of three uncertainty calculus techniques for ultrasonic map building. In *Proceedings of the 1996 SPIE Symposium on Aerospace/Defense Sensing and Control, Applications of fuzzy logic technologies-III*, pages 249–260, Orlando (Florida,USA), April 1996. International Society for Optical Engineering.

[Gambino *et al.*, 1997] Fabio Gambino, Giovanni Ulivi, y Marilena Vendittelli. The transferable belief model in ultrasonic map building. In *Proceedings of the 6th IEEE Conference on Fuzzy Systems*, pages 601–608, Barcelona, Spain, July 1997.

[García-Alegre *et al.*, 1993] M.C. García-Alegre, A. Ribeiro, J. Gasós, y J. Salido. Optimization of fuzzy behavior-based robots navigation in partially known industrial environments. In *Proceedings of the 3rd IEEE International Conference on Industrial Fuzzy Control and Intelligent Systems*, pages 50–54, Houston, December 1993.

[García-Alegre y Recio, 1998] Maria C. García-Alegre y Felicidad Recio. Basic visual and motor agents for increasingly complex behavior generation on a mobile robot. *Autonomous Robots*, 5:19–28, 1998.

[García Pérez, 2003] Lía García Pérez. Laser rangefinder perception of static and dynamic objects in outdoor environments. Technical report, Instituto de Automática Industrial, March 2003.

[García Pérez, 2004] Lía García Pérez. *Navegación autónoma de robots en agricultura: un modelo de agentes*. PhD thesis, Universidad Complutense de Madrid, 2004. en preparación.

[Gasós y Martín, 1996] Jorge Gasós y Alejandro Martín. A fuzzy approach to build sonar maps for mobile robots. *Computers in Industry*, 32:151–167, 1996.

[Gasós *et al.*, 1990] J. Gasós, P.D. Fernández Zuliani, M.C. García-Alegre, y R. García Rosa. Environment for the development of fuzzy controllers. In *Proceedings of the IASTED International Symposium on Artificial Intelligence Application & Neural Networks*, pages 121–124, Zurich, Switzerland, June 1990.

[Gat, 1992] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the 10th AAAI National Conference on Artificial Intelligence*, pages 809–815, San Jose, CA, 1992.

[Gibson, 1977] James J. Gibson. The theory of affordances. In J. Bransford R. Shaw, editor, *Perceiving, acting and knowing*, pages 67–82. Wiley, New York, 1977.

[Giralt *et al.*, 1983] Georges Giralt, Raja Chatila, y Marc Vaisset. An integrated navigation and motion control system for autonomous multisensory mobile robots. In M. Brady y R. Paul, editors, *First International Symposium on Robotics Research*, pages 191–214. MIT Press, 1983.

[González y Pérez, 1998] Antonio González y Raúl Pérez. Refinement of a fuzzy control rule set. *Mathware & Soft Computing*, 5:175–187, 1998.

[Gómez *et al.*, 2003] Victor M. Gómez, José M. Cañas, Félix San Martín, y Vicente Matellán. Vision based schemas for an autonomous robotic soccer player. In *Actas del IV Workshop de Agentes Físicos, WAF'2003*, pages 109–120, Universidad Alicante, March 2003. ISBN 84/607-7171-7.

[Gómez, 2002] Víctor M. Gómez. Comportamiento sigue pared en un robot con visión local. Proyecto fin de carrera, Universidad Rey Juan Carlos, September 2002.

[Haigh y Veloso, 1996] Karen Zita Haigh y Manuela Veloso. Interleaving planning and robot execution for asynchronous user requests. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS-96)*, pages 148–155, Osaka, Japan, November 1996.

[Hallam y Hayes, 1992] Bridget Hallam y Gillian Hayes. Comparing robot and animal behavior. Technical Report RP-598, University of Edinburgh, 1992.

[Han y Veloso, 1998] Kwun Han y Manuela Veloso. Reactive visual control of multiple non-holonomic robotic agents. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven (Belgium), May 1998.

[Horswill, 1997] Ian Horswill. Real-time control of attention and behavior in a logical framework. In *Proceedings of the ACM International Conference on Autonomous Agents*, pages 130–137, Marina del Rey (USA), February 1997.

[Hough, 1959] P.V.C. Hough. Machine analysis of bubble chamber pictures. In *Proceedings of the International Conference on High Energy Accelerators and Instrumentation, CERN*, 1959.

[Howard y Kitchen, 1996a] Andrew Howard y Les Kitchen. Generating sonar maps in highly specular environments. In *Proceedings of the 4th International Conference on Control, Automation, Robotics and Vision*, pages 1870–1874, Singapore, December 1996.

[Howard y Kitchen, 1996b] Andrew Howard y Les Kitchen. Sonar mapping for mobile robots. Technical Report TR 96/34, Computer Vision and Machine Intelligence Laboratory, Department of Computer Science, University of Melbourne, 1996.

[Hu y Michael, 1996] Huosheng Hu y Brady Michael. A parallel processing architecture for senor-based control of intelligent mobile robots. *International Journal on Robotics and Autonomous Systems*, 17(4):235–257, 1996.

[Illingworth y Kittler, 1988] J. Illingworth y J. Kittler. A survey of the hough transform. *Computer Vision, Graphics, and Image Processing*, 1988.

[Ko y Simmons, 1998] Nak Yong Ko y Reid Simmons. The lane-curvature method for local obstacle avoidance. In *Proceedings of the 1998 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, pages 1615–1621, October 1998.

[Koening *et al.*, 1996] Sven Koening, Richard Goodwin, y Reid G. Simmons. Robot navigation with Markov models: a framework for path planning and learning with limited computational resources. In M. van Lambalgen L. Dorst y R. Voorbraak, editors, *Reasoning with uncertainty in robotics*, volume 1093 of *Lecture notes in artificial intelligence*, pages 322–337. Springer, 1996.

[Konolige y Myers, 1998] Kurt Konolige y Karen L. Myers. The Saphira architecture for autonomous mobile robots. In David Kortenkamp, R. Peter Bonasso, y Robin Murphy, editors, *Artificial Intelligence and Mobile Robots: case studies of successful robot systems*, pages 211–242. MIT Press, AAAI Press, 1998. ISBN: 0-262-61137-6.

[Konolige, 1997] Kurt Konolige. Improved occupancy grids for map building. *Autonomous Robots*, 4:351–367, 1997.

[Kosecká y Bajcsy, 1994] Jana Kosecká y Ruzena Bajcsy. Discrete event systems for autonomous mobile agents. *Robotics and Autonomous Systems*, 12:187–198, 1994.

[Kosecká y Bogoni, 1994] Jana Kosecká y Luca Bogoni. Application of Discrete Event Systems for modelling and controlling robotic agents. In *Proceedings of the 1994 IEEE Conference on Robotics and Automation ICRA'94*, pages 2557–2562, San Diego (CA-USA), May 1994.

[Kuc y Barshan, 1992] Roman Kuc y Billur Barshan. Bat-like sonars for guiding mobile robots. *IEEE Control Systems Magazine*, pages 4–12, August 1992.

[Laird *et al.*, 1987] John E. Laird, Allen Newell, y Paul S. Rosenbloom. SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[Laird y Rosenbloom, 1990] John E. Laird y Paul S. Rosenbloom. Integrating execution, planning and learning in Soar for external environments. In *Proceedings of the 8th AAAI National Conference on Artificial Intelligence*, pages 1022–1029, Boston (Massachusetts), July 1990.

[Laird y Rosenbloom, 1996] John E. Laird y Paul Rosenbloom. The evolution of the Soar cognitive architecture. In David M. Steier y Tom M. Mitchell, editors, *Mind Matters: A Tribute to Allen Newell*, pages 1–50. Lawrence Erlbaum Associates, Inc., Mahwah, New Jersey, 1996.

[Langer *et al.*, 1994] D. Langer, J.K. Rosenblatt, y M. Hebert. A behavior-based system for off-road navigation. *IEEE Journal of Robotics and Automation*, 10(6):776–782, December 1994.

[Leonard *et al.*, 1992] John J. Leonard, Hugh Durrant-Whyte, y Ingemar J. Cox. Dynamic map building for an autonomous mobile robot. *The International Journal of Robotics Research*, 11(4):286–298, aug 1992.

[Leonard *et al.*, 1995] John J. Leonard, Bradley A. Moran, Ingemar J. Cox, y Matthew L. Miller. Underwater sonar data fusion using an efficient multiple hypothesis algorithm. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1995.

[Lobato, 2003] David Lobato. Navegación local con ventana dinámica para un robot móvil. Proyecto fin de carrera, Universidad Rey Juan Carlos, February 2003.

[López de Mántaras *et al.*, 1997] R. López de Mántaras, J. Amat, F. Esteva, M. López, y C. Sierra. Generation of unknown environment maps by cooperative low-cost robots. In *Proceedings of the ACM International Conference on Autonomous Agents*, pages 164–169, Marina del Rey (USA), February 1997.

[López-Sánchez *et al.*, 1997] Maite López-Sánchez, R. López de Mántaras, y C. Sierra. Incremental map generation by low cost robots based on possibility/necessity grids. In *Proceedings of the 13th AAAI International Conference on Uncertainty in AI*, pages 351–357, Rhode Island (USA), August 1997.

[Lorenz, 1978] Konrad Lorenz. *Fundamentos de la etología*. Ediciones Paidós, 1978.

[Lozano-Pérez, 1983] T. Lozano-Pérez. Spatial planning: a configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[Ludlow, 1976] A. R. Ludlow. The behavior of a model animal. *Behavior*, 58(1-2):131–172, 1976.

[López de Teruel y Ruiz, 1998] Pedro E. López de Teruel y Alberto Ruiz. A probabilistic learning algorithm for real-time line detection. In *Proceedings of Learning'98*, Madrid (España), 1998. ISBN: 84-89315-11-6.

[Maes, 1989a] Pattie Maes. The dynamics of action selection. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Detroit, 1989. Morgan Kaufmann.

[Maes, 1989b] Pattie Maes. How to do the right thing. *Connection Science Journal*, 1(3):291–323, February 1989. Special issue on hybrid systems.

[Maes, 1990] Pattie Maes. Situated agents can have goals. *International Journal on Robotics and Autonomous Systems*, 6(1):49–70, 1990.

[Mallot y Franz, 1999] Hanspeter A. Mallot y Matthias O. Franz. Biomimetic robot navigation. *Autonomous Systems*, 20:133–153, 1999.

[Margaritis y Thrun, 1998] D. Margaritis y S. Thrun. Learning to locate an object in 3d space from a sequence of images. In *Proc. International Conference on Machine Learning*, pages 332–340, 1998.

[Martin y Moravec, 1996] Martin C Martin y Hans P. Moravec. Robot evidence grids. Cmu-ri-tr-96-06, The Robotics Institute, Carnegie Mellon University, March 1996.

[Martín, 2002] Félix San Martín. Comportamiento sigue pelota en un robot con visión local. Proyecto fin de carrera, Universidad Rey Juan Carlos, September 2002.

[Martínez Gil, 2003] Juanjo Martínez Gil. Equipo de futbol con jde para la liga simulada robocup. Proyecto fin de carrera, Universidad Rey Juan Carlos, September 2003.

[Martínez, 2003] Marta Martínez. Comportamiento sigue pelota con visión cenital. Proyecto fin de carrera, Universidad Rey Juan Carlos, July 2003.

[Mataric, 1991] Maja J. Mataric. Navigation with a rat brain: a neurobiologically-inspired model for robot spatial representation. In J-A. Meyer y S.W. Wilson, editors, *From Animals to Animats*, pages 169–175. MIT Press, 1991.

[Mataric, 1992a] Maja J. Mataric. Behavior-based control: main properties and implications. In *Proceedings of the IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, pages 46–54, Nice (France), May 1992.

[Mataric, 1992b] Maja J. Mataric. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, June 1992.

[Mataric, 1995] Maja J. Mataric. Designing and understanding adaptive group behavior. *Adaptive behavior*, 4(1):51–80, December 1995.

[Mataric, 1999] Maja J. Mataric. Behavior-based robotics. In Robert A. Wilson y Frank Keil, editors, *The MIT encyclopedia of cognitive sciences*, pages 74–77. MIT Press, 1999. ISBN 0-262-23200-6.

[Mataric, 2002] Maja J. Mataric. Situated robotics. In Lynn Nadel, editor, *Encyclopedia of Cognitive Science*. Nature Publishers Group, London UK, 2002.

[Matellán *et al.*, 1998] Vicente Matellán, Daniel Borrajo, y Camino Fernández. Using ABC^2 on the RoboCup domain. In Hiroaki Kitano, editor, *Robocup-97: Robot soccer world cup I, Lecture Notes in Artificial Intelligence*, pages 475–483. Springer-Verlag, 1998. ISBN: 3-540-64473-3.

[Matellán y Borrajo, 1998] Vicente Matellán y Daniel Borrajo. ABC^2: an architecture for intelligent autonomous systems. In *Proceedings of the 3rd IFAC Symposium on Intelligent Autonomous Vehicles IAV'98*, pages 57–61, Madrid, March 1998.

[Matía y Jiménez, 1998] F. Matía y A. Jiménez. Multisensor fusion: an autonomous mobile robot. *Journal of Intelligent and Robotic Systems*, 22:129–141, 1998.

[McFarland y Bösser, 1993] David McFarland y Thomas Bösser. *Intelligent behavior in animals and robots*. The MIT Press, 1993. ISBN-0-262-13293-1.

[Medeiros, 1998] Adelardo A. D. Medeiros. A survey of control architectures for autonomous mobile robots. *Journal of the Brazilian Computer Society*, 4(3), 1998. ISSN: 0104-6500.

[Meystel, 1987] A. Meystel. Planning/control architectures for master dependent autonomous systems with nonhomogeneous knowledge representation. In *Proceedings of the 1987 IEEE International Symposium on Intelligent Control*, pages 31–41, Philadelphia PA, USA, 1987.

[Meystel, 1998] A. Meystel. Multiresolutional autonomy. In *Proceedings of the 1998 IEEE International Symposium on Intelligent Control*, pages 516–519, Gaithersburg MD, USA, September 1998.

[Minksy, 1986] Marvin Minksy. *Society of Mind*. Simon & Schuster, Inc., New York (USA), 1986.

[Moravec, 1989] Hans Moravec. Sensor fusion in certainty grids for mobile robots. In *Sensor Devices and Systems for Robotics*, Nato ASIS Series, pages 253–276. Springer-Verlag, 1989.

[Mucientes *et al.*, 2001] M. Mucientes, M. Rodríguez, R. Iglesias, A. Bugarín, S. Barro, y C.V. Regueiro. Avoidance of mobile obstacles in real environments. In *Proceedings of the International Joint Conference on Artificial Intelligence IJCAI, Workshop on Reasoning with Uncertainty in Robotics*, pages 69–76, Seattle (USA), August 2001.

[Murphy *et al.*, 1999] Robin R. Murphy, Ken Hughes, Alisa Marzilli, y Eva Noll. Integrating explicit path planning with reactive control of mobile robots using trulla. *Robotics and Autonomous Systems*, 27:225–245, 1999.

[Murphy, 1998] Robin R. Murphy. Dempster-shafer theory for sensor fusion in autonomous mobile robots. *IEEE Transactions on Robotics and Automation*, 14(2):197–206, April 1998.

[Murphy, 2000] Robin R. Murphy. *Introduction to AI robotics*. MIT Press, 2000.

[Nehmzow, 1995] Ulrich Nehmzow. Animal and robot navigation. *Robotics and Autonomous Systems*, 15(1-2):71–81, 1995.

[Newell y Simon, 1976] Allen Newell y Herb A. Simon. Computer science as empirical enquiry: symbols and search. *Communications of the ACM*, 19:113–126, 1976.

[Nilsson, 1969] N.J. Nilsson. A mobile automaton: an application of artificial intelligence techniques. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence IJCAI*, pages 509–520, Washington, (USA), 1969.

[Ogata, 1990] K. Ogata. *Modern Control Engineering*. Prentice Hall, 1990.

[Oliveira Freire *et al.*, 1998] Eduardo Oliveira Freire, Teodiano Freire Bastos, et al. Development of an external sensing system for an agent-based controlled mobile robot. In *Proceedings of the 3rd IFAC Symposium on Intelligent Autonomous Vehicles IAV'98*, pages 243–248, Madrid, March 1998.

[Oller *et al.*, 2000] A. Oller, J.L. de la Rosa, R. García, J.A. Ramón, y A. Figueras. Micro-robots playing soccer games: a real implementation based on a multi-agent decision-making structure. *International Journal of Intelligent Automation and Soft Computing*, 6(1):65–74, 2000. Special Issue on Soccer Robotics: Micro-robot WorldCup Soccer Tournament'97.

[Ollero Baturone, 2001] Aníbal Ollero Baturone. *Robótica: manipuladores y robots móviles*. Marcombo, Boixareu Editores, 2001. ISBN 84/267-1313-0.

[Oriolo *et al.*, 1998] Giuseppe Oriolo, Giovanni Ulivi, y Marilena Venditelli. Real-time map building and navigation for autonomous robots in unknown environments. *IEEE Transactions on Systems, Man and Cybernetics*, 28(3):316–333, June 1998.

[O'Sullivan *et al.*, 1997] Joseph O'Sullivan, Karen Zita Haigh, y Greg D. Armstrong. Xavier's manual. Technical Report version 0.4, School of Computer Science, Carnegie Mellon University, April 1997.

[Owen y Nehmzow, 1998] Carl Owen y Ulrich Nehmzow. Map interpretation in dynamic environments. In *Proceedings of the 5th IEEE International Workshop on Advanced Motion AMC'98*, pages 340–345, Coimbra (Portugal), 1998.

[Pagac *et al.*, 1998] Daniel Pagac, Eduardo M. Nebot, y Hugh Durrant-Whyte. An evidential approach to probabilistic map-building. *IEEE Transactions on Robotics and Automation*, 14(4):623–629, August 1998.

[Passino, 1995] Kevin M. Passino. Intelligent control for autonomous systems. *IEEE Spectrum*, 32(6):55–62, June 1995.

[Payton *et al.*, 1990] David W. Payton, J. Kenneth Rosenblatt, y David M. Keirsey. Plan guided reaction. *IEEE Transactions on Systems Man and Cybernetics*, 20(6):1370–1382, 1990.

[Payton, 1990] David W. Payton. Internalized plans: a representation for action resources. *Robotics and Autonomous Systems*, 6:89–103, 1990.

[Peignot *et al.*, 1998] Cédric Peignot, Fabrice Wawak, Fernando Matía, y E.A. Puente. Integration of heterogeneous world mapping techniques in the navigation system of an autonomous mobile robot. In *Proceedings of the Computer Vision and Mobile Robotics Workshop CVMR*, pages 74–81, Santorini (Greece), September 1998.

[Peño del Barrio, 2003] Manuel Peño del Barrio. Comportamiento saque de banda para la robocup. Proyecto fin de carrera, Universidad Rey Juan Carlos, September 2003.

[Pirjanian *et al.*, 1998] Paolo Pirjanian, Henrik I. Christensen, y Jeffrey A. Fayman. Application of voting to fusion of purposive modules: an experimental investigation. *Journal of Robotics & Autonomous Systems*, 23(4):253–266, July 1998.

[Pirjanian, 1997a] Paolo Pirjanian. Behavior coordination mechanisms - state-of-the-art. Technical report, USC Robotics Research Laboratory, University of Southern California, October 1997.

[Pirjanian, 1997b] Paolo Pirjanian. An overview of system architectures for action selection in mobile robotics. Technical report, Laboratory of Image Analysis, Aalborg University, March 1997.

[Pirjanian, 2000] Paolo Pirjanian. Multiple objective behavior-based control. *Robotics and Autonomous Systems*, 31(1-2):53–60, 2000.

[Poloni *et al.*, 1995] M. Poloni, G. Ulivi, y M. Venditelli. Fuzzy logic and autonomous vehicles: experiments in ultrasonic vision. *Fuzzy Sets and Systems*, 69:15–27, 1995.

[Prassler *et al.*, 2000] Erwin Prassler, Jens Scholz, y Alberto Elfes. Tracking multiple moving objects for real-time robot navigation. *Autonomous Robots*, 8:105–116, 2000.

[Ramadge y Wonham, 1989] Peter J. G. Ramadge y W. Murray Wonham. The control fo discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[Rao y Georgeff, 1995] Anand Rao y Michael P. Georgeff. Bdi agents: from theory to practice. Technical Report 56, Australian Artificial Intelligence Institute, Melbourne, Australia, April 1995.

[Real World Interface, 1996] Inc. Real World Interface. System software and rai-1.2.2 documentation. Technical report, Real World Interface, Inc., 1996.

[Reyes *et al.*, 1999] Caridad Reyes, Sergio Alcalde, Isidoro Hernán, Jesús Reviejo, Ricardo García, y Teresa de Pedro. Controlador de vehículos sin conductor: simulación e instrumentación. In *Proceedings of the VIII Conference of Spanish Artificial Intelligence Association, CAEPIA'99*, pages 9–16, Murcia, 1999.

[Ribo y Pinz, 2001] Miguel Ribo y Axel Pinz. A comparison of three uncertainty calculi for building sonar-based occupancy grids. *Robotics and Autonomous Systems*, 35:201–209, 2001.

[Rodríguez *et al.*, 2000] M. Rodríguez, J. Correa, R. Iglesias, C.V. Regueiro, y S. Barro. Probabilistic and count methods in map building for autonomous mobile robots. In J.L. Watt y J.Deminis, editors, *Advances in robot learning*, volume 1812 of *Lecture notes on artificial intelligence*, pages 120–138. Springer, 2000.

[Rojas *et al.*, 2002] Raul Rojas, Felix von Hundelshausen, Sven Behnke, y Bernhard Frötschl. Fu-fighters omni 2001 (local vision). In S.Coradeschi A.Birk y S.Takodoro, editors, *Robocup-2001: Robot soccer world cup V, Lecture Notes in Artificial Intelligence 2377*, pages 575–578. Springer-Verlag, 2002.

[Rosenblatt y Pyton, 1989] Julio K. Rosenblatt y David W. Pyton. A fine-grained alternative to the subsumption architecture for mobile robot control. In *Proceedings of IEEE International Joint Conference on Neural Networks*, volume 2, pages 317–323, Washington D.C., June 1989.

[Rosenblatt, 1997] Julio Rosenblatt. *DAMN: A Distributed Architecture for Mobile Navigation*. PhD thesis, Carnegie Mellon University, Robotics Institute, 1997.

[Rosenblatt, 2000] Julio K. Rosenblatt. Optimal selection of uncertain actions by maximizing expected utility. *Autonomous Robots*, 9:17–25, 2000.

[Rosenschein y Kaelbling, 1995] Stanley J. Rosenschein y Leslie Pack Kaelbling. A situated view of representation and control. *Artificial Intelligence*, 73:149–173, 1995.

[Saffiotti y Wasik, 2003] Alessandro Saffiotti y Zbigniew Wasik. Using hierarchical fuzzy behaviors in the robocup domain. In D. Maravall C. Zhou y D. Ruan, editors, *Autonomous Robotic Systems*, pages 235–262. Springer, 2003.

[Saffiotti, 1997] Alessandro Saffiotti. The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1:180–197, 1997.

[Schneider Fontán, 1996] Miguel Schneider Fontán. *Planificación basada en percepción activa para la navegación de un robot móvil*. PhD thesis, Universidad Complutense de Madrid, 1996.

[Serradilla, 1997] Francisco Serradilla. *Arquitectura cognitiva basada en el gradiente sensorial y su aplicación a la robótica móvil*. PhD thesis, Universidad Politécnica Madrid, 1997.

[Simmons *et al.*, 1997a] Reid Simmons, Richard Goodwin, Christopher Fedor, y Jeff Basista. Programmer's Guide to tca. Technical Report version 8.0, School of Computer Science, Carnegie Mellon University, May 1997.

[Simmons *et al.*, 1997b] Reid Simmons, Richard Goodwin, Karen Zita Haigh, Sven Koening, y Joseph O'Sullivan. A layered architecture for office delivery robots. In *Proceedings of the ACM International Conference on Autonomous Agents*, pages 245–252, Marina del Rey (USA), feb 1997.

[Simmons y Koening, 1995] Reid Simmons y Sven Koening. Probabilistic navigation in partially observable environments. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, Montreal (Canada), July 1995.

[Simmons, 1994] Reid G. Simmons. Structured control for autonomous robots. *IEEE Journal of Robotics and Automation*, 10(1):34–43, February 1994.

[Simmons, 1996] Reid Simmons. The curvature-velocity method for local obstacle avoidance. In *Proceedings of the 1996 International Conference on Robotics and Automation*, Minneapolis-MN, April 1996.

[Smith y Brady, 1997] S.M. Smith y J.M. Brady. SUSAN: a new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78, May 1997.

[Stentz *et al.*, 2002] Anthony Stentz, Cristian Dima, Carl Wellington, Herman Herman, y David Stager. A system for semi-autonomous tractor operations. *Autonomous Robots*, 13:87–104, 2002.

[Stoytchev y Arkin, 2001] Alexander Stoytchev y Ronald C. Arkin. Combining deliberation, reactivity and motivation in the context of a behavior-based robot architecture. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA-2001*, pages 290–295, 2001.

[Sugeno, 1985] Michio Sugeno. An introductory survey of fuzzy control. *Information Sciences*, 36:59–83, 1985.

[Sugeno, 1999] Michio Sugeno. Development of an intelligent unmanned helicopter. In Hung T. Nguyen y Nadipuram R. Prasad, editors, *Fuzzy modeling and control, selected works of M. Sugeno*, pages 13–43. CRC-Press, March 1999. ISBN 0849328845.

[Sáez y Escolano, 2002] J.M. Sáez y F. Escolano. Localización global en mapas 3D basada en filtros de partículas. In *Actas del III Workshop de Agentes Físicos, WAF'2002*, pages 29–40, Universidad de Murcia, March 2002.

[Takahashi y Schilling, 1989] O. Takahashi y R.J. Schilling. Motion planning in a plane using generalized Voronoi diagrams. *IEEE Transactions on Robotics and Automation*, 5(2):143–150, April 1989.

[Thrun *et al.*, 1998] Sebastian Thrun, Jens-Steffen Gutmann, Dieter Fox, Wolfram Burgard, y Benjamin J. Kuipers. Integrating topological and metric maps for mobile robot navigation: a statistical approach. In *Proceedings of the 15th AAAI National Conference on Artificial Intelligence*, pages 989–995, Madison (Wisconsin), July 1998.

[Timberlake, 2000] William Timberlake. Motivational models in behavior systems. In S.B. Klein R.R. Mowrer, editor, *Handbook of contemporary learning theories*, pages 155–209. Hillsdale, NJ: Erlbaum Associates, 2000.

[Tinbergen, 1950] N. Tinbergen. The hierarchical organization of nervous mechanisms underlying instinctive behavior. *Symposia of the Society for Experimental Biology*, 4:305–312, 1950.

[Tinbergen, 1987] Nikola Tinbergen. *El estudio del instinto*. Siglo Veintiuno de España Editores, 1987.

[Turing, 1950] Alan M. Turing. Computer machinery and intelligence. *Mind*, 59(236):433–460, October 1950.

[Tyrrell, 1992] Toby Tyrrell. Defining the action selection problem. In *Proceedings of the Fourteenth Annual Conference on the Cognitive Science Society*. Lawrence Erlbaum Associates, 1992.

[Tyrrell, 1993a] Toby Tyrrell. *Computational mechanisms for action selection*. PhD thesis, University of Edinburgh, 1993.

[Tyrrell, 1993b] Toby Tyrrell. The use of hierarchies for action selection. *Journal of Adaptive Behaviour*, 1(4):387–420, 1993.

[Tyrrell, 1994] Toby Tyrrell. An evaluation of Maes' "bottom-up mechanism for behavior selection". *Journal of Adaptive Behavior*, 2(4):307–348, 1994.

[Uribe *et al.*, 1995] Juan P. Uribe, Jokin Mujika, y Reinhard Braunstingl. Controlador fuzzy optimizado mediante algoritmos genéticos para el seguimiento de paredes en un robot móvil. In *Actas del V Congreso Español sobre Tecnologías y Lógica Fuzzy (ESTYLF)*, pages 211–216, Murcia, España, September 1995.

[Velasco y Magdalena, 1996] Juan R. Velasco y Luis Magdalena. Controladores borrosos con aprendizaje. In *Actas de las I jornadas sobre Inteligencia Artificial, Control y Sistemas expertos*, pages 121–137, Universidad de Alcalá, 1996.

[Vera y Simon, 1993] Alonso H. Vera y Herbert A. Simon. Situated action: a symbolic interpretation. *Gognitive Science*, 17:7–48, 1993.

[Vogel y Angermann, 1974] Güntel Vogel y Hartmut Angermann. *Atlas de biología*. Ediciones Omega, 1974.

[Volpe *et al.*, 2001] Richard Volpe, Issa Nesnas, Tara Estlin, Darren Mutz, Richard Petras, y Ari Das. The CLARAty architecture for robotic autonomy. In *Proceedings of the IEEE Aerospace Conference IAC-2001*, volume 1, pages 121–132, Big Sky, MT, 2001.

[Wang y Thorpe, 2002] Chieh-Chih Wang y Chuck Thorpe. Simultaneous localization and mapping with detection and tracking of moving objects. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2918–2924, Washington DC, May 2002.

[Wasson *et al.*, 1999] Glenn Wasson, David Kortenkamp, y Eric Huber. Integrating active perception with an autonomous robot architecture. *International Journal on Robotics and Autonomous Systems*, 29:175–186, 1999.

[Wawak *et al.*, 1998] F. Wawak, F. Matía, C. Peignot, y E.A. Puente. A framework for the integration of perception and localisation systems over mobile platforms. In *Proceedings of the 3rd European Robotics, Intelligent Systems and Control Conference EURISCON*, Athens, June 1998.

[Yamauchi *et al.*, 1998] Brian Yamauchi, Alan Schultz, y William Adams. Mobile robot exploration and map-building with continuous localization. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven (Belgium), May 1998.

[Yen y Pfluger, 1995] John Yen y Nathan Pfluger. A fuzzy logic based extension to Payton and Rosenblatt's command fusion method for mobile robot navigation. *IEEE Transactions on Systems, Man and Cybernetics*, 25(6):971–978, June 1995.