UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
DEPARTAMENTO DE ELECTRÓNICA E COMPUTACIÓN

TESIS DOCTORAL

Arquitectura Fractal de Especialistas para robots móviles autónomos

Presentada por:
Carlos Vázquez Regueiro

Dirigida por:
Dr. Senén Barro Ameneiro

Santiago de Compostela, enero de 2002.

SENÉN BARRO AMENEIRO
Catedrático del Área de Ciencias de la
Computación e Inteligencia Artificial
de la Universidad de Santiago de Compostela

HACE CONSTAR

Que la memoria titulada "**Arquitectura Fractal de Especialistas para robots móviles autónomos**" ha sido realizada por D. Carlos Vázquez Regueiro bajo mi dirección en el Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela, y constituye la Tesis que presenta para optar al grado de Doctor en Ciencias Físicas.

Santiago de Compostela, 14 de enero de 2002.

Fdo: Senén Barro Ameneiro
Director de la Tesis y
Director del Departamento
de Electrónica y Computación

A mi mujer y
a mi familia

Agradecimientos

La realización de este trabajo ha sido posible gracias al apoyo de muchas personas, entre todas ellas quiero destacar:

A mi director Senén Barro Ameneiro, por la confianza que ha depositado en mi desde el principio, por los sabios consejos y enseñanzas que me ha brindado, y por su inestimable ayuda en todo momento.

Al Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela, especialmente a los miembros del Grupo de Sistemas Inteligentes, por su colaboración constante y desinteresada, por las discusiones y el intercambio de ideas, siempre beneficioso y constructivo.

Al Departamento de Electrónica y Sistemas de la Universidad de A Coruña, especialmente a todos los miembros del área de Tecnología y Estructura de Computadores, entre otras muchas cosas por las facilidades que me han brindado durante todo este tiempo para compatibilizar mis labores docentes e investigadoras.

A la Xunta de Galicia por la financiación recibida a través de los proyectos XUGA20608B97 y PGIDT99PXI20603A, y a un proyecto de infraestructura gracias al cual se ha podido adquirir el robot móvil *Nomad 200* utilizado en este trabajo.

Por último, pero sin duda los primeros, a mi mujer y también a mi familia por su infinita, a veces estoica, paciencia y por su apoyo inquebrantable durante todos estos años. Agradecimiento que hago extensivo a todos mis amigos, pocos pero de calidad.

Índice General

Introducción **1**

1 Introducción a los robots móviles autónomos **5**

1.1 Definiciones 5

1.2 Tipos de robots y su clasificación 8

1.2.1 Robots manipuladores 9

1.2.2 Robots móviles 10

1.3 Soporte físico 12

1.3.1 Sensores 14

1.3.2 Efectores 24

1.3.3 Computadores 26

1.4 Aplicaciones y usos de la robótica móvil 28

1.5 Problemas de la robótica móvil 30

1.6 Robot móvil *Nomad 200* 31

1.6.1 Entorno de desarrollo y simulador del *Nomad 200* 32

2 Aspectos funcionales de un RMA **37**

2.1 Robot móvil autónomo 37

2.2 Funciones de un RMA 39

2.2.1 Tipos de funciones 39

2.2.2 Robots e inteligencia artificial 42

2.3 Objetivos de un RMA 43

2.3.1 Tareas de supervivencia 44

2.3.2 Tareas proactivas 45

2.4 Tareas externas básicas 46
2.4.1 Navegación 47
2.4.2 Modificación del entorno 49
2.4.3 Comunicación con otros entes 51
2.5 RMA de propósito general 53
2.5.1 Elección del nicho ecológico 54
2.5.2 Selección de la tarea a implementar 54
2.5.3 Elección del sistema sensomotor 55

3 Arquitecturas de control para RMA 57
3.1 Arquitecturas de control en robótica 57
3.1.1 Definición 57
3.1.2 Requisitos 58
3.1.3 Criterios de evaluación 61
3.2 Principales paradigmas en robótica 65
3.2.1 Arquitecturas jerárquicas 66
3.2.2 Arquitecturas reactivas 72
3.2.3 Arquitecturas híbridas planificadoras/reactivas 79
3.3 Otros paradigmas software 91
3.3.1 Arquitecturas basadas en pizarra 91
3.3.2 Arquitecturas basadas en agentes 96
3.3.3 Una nueva propuesta: arquitectura basada en especialistas . . 100

4 Arquitectura fractal de especialistas 109
4.1 Principales características de AFE 109
4.2 Aplicación de AFE a la robótica móvil 111
4.2.1 Estructura de especialistas en AFE-Robótica 111
4.2.2 Estructura parcial de datos en AFE-Robótica 113
4.3 Arquitectura de un especialista 116
4.3.1 Arquitectura basada en pizarra 116
4.3.2 Componentes de un especialista 117
4.4 Comunicaciones entre especialistas 118

4.4.1 Protocolo de comunicaciones . . . 119
4.4.2 Tipos de comunicaciones . . . 120
4.4.3 Implementación de las comunicaciones . . . 122
4.5 Pizarra de datos . . . 127
4.5.1 Interfaz de pizarra . . . 127
4.6 Agentes terminales . . . 129
4.7 Agente de control . . . 131
4.7.1 Planes . . . 131
4.7.2 Componentes del agente de control . . . 135

5 Síntesis de AFE-Robótica 139
5.1 Tarea de navegación . . . 139
5.1.1 Prototipo implementado . . . 142
5.2 Interfaz de usuario DIÁLOGO . . . 144
5.3 Especialista NAVEGADOR . . . 145
5.3.1 Entradas y salidas . . . 145
5.3.2 Pizarra . . . 146
5.3.3 Agentes . . . 150
5.3.4 Síntesis de planes y control . . . 150
5.3.5 Evaluación . . . 155
5.3.6 Discusión . . . 161
5.4 Agente LOCALIZADOR . . . 161
5.4.1 Entradas y salidas . . . 162
5.4.2 Síntesis del agente . . . 162
5.4.3 Evaluación . . . 162
5.4.4 Soluciones existentes . . . 163
5.4.5 Discusión . . . 164
5.5 Agente PLANIFICADOR_RUTAS . . . 164
5.5.1 Entradas y salidas . . . 164
5.5.2 Síntesis del agente . . . 165
5.5.3 Evaluación . . . 166

5.5.4 Soluciones existentes . . . 166
5.5.5 Discusión . . . 169
5.6 Especialista PILOTO . . . 169
5.6.1 Entradas y salidas . . . 169
5.6.2 Pizarra . . . 171
5.6.3 Agentes . . . 173
5.6.4 Síntesis de planes y control . . . 174
5.6.5 Evaluación . . . 180
5.6.6 Discusión . . . 184
5.7 Agentes SENSORES . . . 184
5.7.1 Entradas y salidas . . . 185
5.7.2 Síntesis del agente . . . 186
5.7.3 Evaluación . . . 186
5.7.4 Soluciones existentes . . . 187
5.7.5 Discusión . . . 187
5.8 Agentes EFECTORES . . . 188
5.8.1 Entradas y salidas . . . 188
5.8.2 Síntesis del agente . . . 189
5.8.3 Evaluación . . . 190
5.8.4 Soluciones existentes . . . 190
5.8.5 Discusión . . . 191
5.9 Agente ELUDIR_CONTACTO . . . 191
5.9.1 Entradas y salidas . . . 191
5.9.2 Síntesis del agente . . . 192
5.9.3 Evaluación . . . 193
5.9.4 Soluciones existentes . . . 194
5.9.5 Discusión . . . 194
5.10 Agente MANTENER_DISTANCIA . . . 194
5.10.1 Entradas y salidas . . . 195
5.10.2 Síntesis del agente . . . 195
5.10.3 Evaluación . . . 198

5.10.4 Soluciones existentes 199
5.10.5 Discusión 201
5.11 Agente Mover_A 201
5.11.1 Entradas y salidas 201
5.11.2 Síntesis del agente 202
5.11.3 Evaluación 204
5.11.4 Soluciones existentes 204
5.11.5 Discusión 206
5.12 Agente Seguir_Contorno 206
5.12.1 Entradas y salidas 206
5.12.2 Síntesis del agente 207
5.12.3 Evaluación 209
5.12.4 Soluciones existentes 212
5.12.5 Discusión 212
5.13 Agente Creador_Mapa_Local 212
5.13.1 Entradas y salidas 213
5.13.2 Síntesis del agente 213
5.13.3 Evaluación 216
5.13.4 Soluciones existentes 216
5.13.5 Discusión 218
5.14 Agente Detector_Marcas 219
5.14.1 Entradas y salidas 219
5.14.2 Síntesis del agente 220
5.14.3 Evaluación 225
5.14.4 Soluciones existentes 226
5.14.5 Discusión 228
5.15 Agente Posicionador_Robot 229
5.15.1 Entradas y salidas 229
5.15.2 Síntesis del agente 230
5.15.3 Evaluación 237
5.15.4 Soluciones existentes 242

5.15.5 Discusión 243

6 Validación experimental **245**

6.1 Validación 245

6.1.1 Alcance de la validación experimental 247

6.2 Experimentos 249

6.2.1 Ejecución de una tarea en NAVEGADOR 249

6.2.2 Ejecución de una tarea en PILOTO 255

6.2.3 Respuestas reactivas 258

6.2.4 Ejecución de trayectos largos 261

6.2.5 Contingencias en la ejecución de un plan 264

6.3 Características de AFE-Robótica 270

6.3.1 Criterios de comparación 270

6.3.2 Otras propiedades 279

6.4 Discusión 286

6.4.1 Consideraciones finales 289

Conclusiones y principales aportaciones **291**

Glosario de abreviaturas **295**

Bibliografía **297**

Índice de Tablas

1.1 Clasificación de los robots según Knasel. 8

2.1 Primitivas de un robot definidas en términos de sus entradas y salidas. 40

3.1 Tabla comparativa de los principales paradigmas aplicados a robótica móvil. 107

5.1 Función de los agentes del especialista NAVEGADOR y los datos de su pizarra que necesita cada uno como entrada y como salida. 151

5.2 Función de los agentes del especialista PILOTO y los datos de la pizarra que necesita cada uno. 175

5.3 Tiempos (en ms) de adquisición y de recogida de los datos sensoriales básicos en función de la frecuencia. 187

5.4 Acciones que toma el agente ELUDIR_CONTACTO según dónde se produzca el contacto. 193

5.5 Acciones evasivas según el método del mínimo. 195

5.6 Acciones evasivas según el método por zonas. 197

5.7 Datos de los dos ejemplos de seguir contorno mostrados. 211

6.1 Medidas realizadas sobre distintas ejecuciones del trayecto *H14-H8-S3,1-H14.* 265

Índice de Figuras

1.1 Partes de un robot manipulador. 9
1.2 Tipos de robots manipuladores y su clasificación según el sistema LERT. 10
1.3 Ejemplos de robots móviles con patas. 11
1.4 Ejemplos de robots con ruedas y orugas. 13
1.5 Ejemplos de las distintas configuraciones de ruedas motrices y directrices en un robot móvil. 13
1.6 Patrón de emisión de un emisor típico de ultrasonidos. 17
1.7 Error que comete un sensor de ultrasonidos debido a la divergencia del pulso que emite. 17
1.8 Ejemplo de la reflexión especular en los sensores de ultrasonidos. . . . 18
1.9 La reflexión de una onda sobre una superficie depende del tamaño relativo entre la rugosidad de dicha superficie y su longitud de onda. . 18
1.10 Operación de un sensor láser basado en técnicas de triangulación. . . 19
1.11 Montaje para la visión: a) con dos grados de libertad, b) en estéreo y con cuatro grados de libertad, y c) omnidireccional. 21
1.12 Discos de codificadores ópticos absolutos: a) código Gray y b) código binario. 23
1.13 Robot móvil *Nomad 200*. 31
1.14 Interfaz gráfica del entorno de desarrollo del *Nomad 200*. 35

2.1 Posible organización de algunas de las tareas más importantes de un robot móvil autónomo. 43

3.1 Esquema general de una arquitectura de control jerárquica. 66
3.2 Organización de las funciones robóticas primitivas en una arquitectura jerárquica. 67

3.3 Descomposición de tareas de tipo horizontal o funcional propia de las arquitecturas jerárquicas. 67
3.4 Esquema interno de la arquitectura NHC. 71
3.5 Descomposición vertical o en base a comportamientos de las tareas de un robot móvil. 73
3.6 Organización de las funciones robóticas primitivas en una arquitectura reactiva. 73
3.7 Descomposición en comportamientos propia de una arquitectura reactiva. 74
3.8 Máquina de estados finitos aumentada. 77
3.9 Arquitectura de control del robot *Genghis*. 78
3.10 Esquema general de una arquitectura de control híbrido. 80
3.11 Relación entre las funciones robóticas primitivas en una arquitectura híbrida o de tipo (P, S-A). 81
3.12 Esquema de la arquitectura AuRA. 85
3.13 Elementos de la arquitectura 3T. 86
3.14 Componentes de la arquitectura Saphira. 88
3.15 Arquitectura en capas basada en TCA. 89
3.16 Esquema interno de la arquitectura AIS. 94
3.17 Esquema de la arquitectura DCS. 98
3.18 Esquema de la arquitectura fractal de especialistas. 100
3.19 Ilustración de la descomposición jerárquica de tareas y la abstracción de datos en la arquitectura basada en especialistas. 101

4.1 Esquema general de la arquitectura fractal de especialistas (AFE). . . 110
4.2 Arquitectura fractal de especialistas aplicada a la robótica móvil: AFE-Robótica. 112
4.3 Estructura parcial de datos en AFE-Robótica. 114
4.4 Estructura interna de un especialista en AFE-Robótica. 117
4.5 Estructura del Módulo de Comunicaciones de un especialista. 121
4.6 Envío de una orden de tarea. 123
4.7 Lectura de un dato por parte de un especialista en la pizarra de su especialista superior. 124

4.8 Inserción de un dato por parte de un especialista en la pizarra de su especialista superior. 125
4.9 Consulta de un dato en AFE-Robótica. 125
4.10 Esquema de funcionamiento de la reactividad externa en AFE. 126
4.11 Elementos de un dato almacenado en la pizarra de un especialista. . 128
4.12 Esquema interno de un agente terminal en AFE. 130
4.13 Plantilla de una tarea. 133
4.14 Ejemplo de un plan representado mediante un grafo. 134
4.15 Esquema interno del agente de control de un especialista de AFE-Robótica. 136

5.1 Estructura de AFE-Robótica implementada. 143
5.2 Estructura de datos en la pizarra del especialista NAVEGADOR. . . . 146
5.3 División de un corredor en lugares según las marcas detectadas. . . . 148
5.4 Mapa topológico que corresponde al entorno de la figura 5.3. 148
5.5 Mapa topológico de la sede del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela. 149
5.6 Etapas en las que se divide la tarea **Ir A**. 154
5.7 Vista parcial del Departamento de Electrónica y Computación. 156
5.8 Ruta entre el laboratorio 14 y el despacho 17. 156
5.9 Posición del robot después de cruzar la puerta del laboratorio 14. . . 157
5.10 Posición prevista del robot después de ejecutar la séptima orden de tarea y seguir el contorno izquierdo del corredor. 158
5.11 Ruta desde el despacho 18 al 17. 159
5.12 Ruta desde el inicio del corredor hasta el despacho 17. 159
5.13 Ejemplo de un corredor bloqueado parcialmente y trayectoria del robot cuando sigue el contorno izquierdo. 160
5.14 Mapa topológico donde se representa un corredor bloqueado. 160
5.15 Ejemplo de una ruta en un mapa topológico. 165
5.16 Cálculo de una ruta utilizando diferentes costes en las conexiones que representan puertas: a) 1000 y b) 500. 167
5.17 Alteración de una ruta al cambiar ligeramente su origen. 168

5.18 Estructura de los datos en la pizarra de PILOTO junto con los datos que se envían y reciben del especialista NAVEGADOR. 172
5.19 Grafo con el plan básico que utiliza PILOTO para ejecutar sus *OTs*. . 175
5.20 Fases para cruzar una puerta. 177
5.21 Fases para seguir un contorno. 178
5.22 Secuencia de acciones con las que PILOTO responde a un evento. . . 179
5.23 Cruce de una puerta: a) datos del láser y b) datos de US. 181
5.24 Cruce de una puerta con un marco estrecho: a) datos del láser y b) datos de US. 181
5.25 Ejecución de la tarea de seguir un contorno. 182
5.26 Integración de la reacción a eventos durante la ejecución de una tarea. 183
5.27 Los dos anillos de sensores táctiles en el robot móvil *Nomad 200* y las cuatro regiones en las que se dividen según la orientación de la base. 192
5.28 Funcionamiento del comportamiento ELUDIR_CONTACTO. 194
5.29 Regiones en las que se divide el entorno del robot en el método por zonas para mantenerse alejado de los obstáculos que lo rodean. 196
5.30 Diferencias de comportamiento del agente MANTENER_DISTANCIA según el tipo de algoritmo utilizado: (a) mínimo, (b) por zonas. . . . 198
5.31 Ejemplo de cómo el agente MANTENER_DISTANCIA puede quedarse atrapado cuando utiliza el método del mínimo. 198
5.32 Comportamiento del robot ante los huecos según se utilice el método del mínimo (izquierda) o por zonas (derecha). 199
5.33 Funcionamiento del comportamiento *mantener distancia* en un entorno reducido. 200
5.34 Ángulo θ para orientar el robot hacia la posición objetivo. 203
5.35 Dependencia de la velocidad de avance del agente MOVER_A con la distancia al objetivo. 203
5.36 Comportamientos *avanzar* del agente MOVER_A y *evitar obstáculos* del agente MANTENER_DISTANCIA. 205
5.37 Comportamientos *moverse siguiendo la orientación 0º* y *evitar obstáculos*. 205
5.38 Comportamiento *mover hacia una posición odométrica* del agente MOVER_A desde distintas posiciones y orientaciones iniciales. 205

5.39 Esquema del agente SEGUIR_CONTORNO. 207
5.40 Disposición de los sensores de US en el agente SEGUIR_CONTORNO. 208
5.41 Ejemplo 1 del agente SEGUIR_CONTORNO. 210
5.42 Ejemplo 2 del agente SEGUIR_CONTORNO. 210
5.43 Variación de la velocidad lineal a lo largo de la ruta del ejemplo 2. . 211
5.44 Confección del mapa de obstáculos. 214
5.45 Creación del mapa local: (a) medidas sensoriales, (b) rejilla instantánea y (c) rejilla de incertidumbre. 215
5.46 Modelo de incertidumbre del sensor de US. 215
5.47 Ejemplo 1 de rejilla de incertidumbre. 216
5.48 Ejemplo 2 de rejilla de incertidumbre. 217
5.49 Secuencia que ilustra la construcción de una rejilla de incertidumbre. 217
5.50 Ejemplos de marcas tipo pared, puerta y corredor. 220
5.51 Fusión de dos segmentos colineales y muy próximos entre sí. 221
5.52 Condiciones para que se detecte un borde. 222
5.53 Condiciones para que se detecte o no una puerta. 222
5.54 Detección de una pared con los US mientras el robot la sigue. 224
5.55 Condiciones para que se detecte un corredor: dos paredes paralelas o una pared a continuación de un corredor. 225
5.56 Detección de paredes y puertas moviendo el robot en línea recta y girando la torreta a velocidad constante. 225
5.57 Detección de paredes y puertas con los datos del láser, alineando la torreta con el contorno que sigue el robot (el de la derecha). 226
5.58 Detección de paredes (izquierda) y corredores (centro) a través de los datos de ultrasonidos (derecha). 227
5.59 Detección de paredes (izquierda) y corredores (centro) a través de los datos de ultrasonidos (derecha). Los mismos datos que en la figura 5.58, pero haciendo el recorrido de ida y vuelta. 227
5.60 Cálculo del error en la posición y orientación del robot según el tipo de marca: a) esquina y b) pared. 231
5.61 Cálculo del vector de desplazamiento para recalibrar la odometría. . . 232

5.62 Reposicionamiento de una marca después de recalibrar la posición odométrica un ángulo $\widetilde{\theta}$ y un vector d'. 233

5.63 Adquisiciones del láser sobre una pared recta larga cuando el ángulo de la base según la odometría no es correcto. 234

5.64 Relaciones geométricas entre dos adquisiciones láser de una pared recta cuando el ángulo de la base odométrico está mal calculado. . . . 235

5.65 Detección de paredes con datos del láser sin y con recalibración de la odometría y girando la torreta a velocidad constante. 238

5.66 Resultado de recalibrar sólo el ángulo de la odometría. 239

5.67 Resultado de utilizar corredores un metro más largos que los reales para la recalibración de la odometría. 239

5.68 Recalibración de la odometría usando datos de ultrasonido y moviendo el robot a unos 15 cm/s. 240

5.69 Resultado de la recalibración de la odometría utilizando únicamente datos de ultrasonidos, moviendo el robot a unos 25 cm/s. 240

5.70 Recalibración del ángulo de la base partiendo de un error de 30º: a) movimiento rectilíneo y b) siguiendo el contorno de la derecha. 241

6.1 Ilustración de los conceptos de validación y verificación. 246

6.2 Vista parcial de la sede del Departamento de Electrónica y Computación con obstáculos. 250

6.3 Etapas en las que se divide la tarea **Ir A** *destino*. 250

6.4 Órdenes de Tarea en la ejecución de una tarea de desplazamiento en AFE-Robótica. 251

6.5 Ruta encontrada entre los nodos *H14* y *H17* marcada en el mapa topológico de la sede del Departamento de Electrónica y Computación. 252

6.6 Trayectoria del robot al ejecutar la orden "**Ir A** *Habitación 17*". . . . 253

6.7 Secuencia de tareas generadas por NAVEGADOR para ejecutar la ruta mostrada en la figura 6.6. 254

6.8 Instanciación del plan asociado en NAVEGADOR a "**Ir A** *destino*" para la secuencia de tareas mostrada en la figura 6.7. 255

6.9 Plan P_1^7 asociado a OT_1^7 en PILOTO. 256

6.10 Ejemplos de la activación de la reactividad interna y externa en AFE-Robótica. 259
6.11 Ampliación de la región (a) de la figura 6.6 dónde se ilustra la integración de las respuestas reactivas con la ejecución de un plan. 260
6.12 Activación continua del evento OBSTÁCULO MUY PRÓXIMO buscando espacio libre cuando el robot está en un espacio muy reducido. . . 260
6.13 Datos del sensor láser y la trayectoria seguida por el robot en un trayecto circular con principio y fin en la habitación *14* y pasando por la habitación *8* y el aula A (*S3,1*). 262
6.14 Mapa métrico global construido a partir de los diferentes mapas de obstáculos locales creados durante el trayecto de la figura 6.13. 263
6.15 La trayectoria del robot primero desde la habitación *17* a la habitación *15* y después desde ésta última a la *9*. 265
6.16 Primera parte del trayecto entre la habitación *17* y la *15* mostrado en la figura 6.15. 266
6.17 Ejemplo de cómo el robot puede cruzar una puerta (no detectada) mientras PILOTO ejecuta la tarea de seguir contorno. 269
6.18 Ejemplo de cómo el especialista PILOTO es capaz de lograr que el robot cruce una puerta. 274
6.19 Integración en el especialista PILOTO del comportamiento voluntario AVANZAR y el reactivo MANTENER_DISTANCIA cuando un obstáculo está muy próximo al robot. 275
6.20 Posible modificación de AFE-Robótica con la incorporación al robot móvil de una cabeza robótica con cámaras de vídeo. 277
6.21 Uso de AFE-Robótica para controlar varios robots móviles no autónomos. 278
6.22 Ejemplo del tiempo de ejecución de las principales *OTs* del ciclo de control del especialista PILOTO permitiendo ejecutar dos ciclos de control concurrentemente. 280
6.23 Ejemplo del tiempo de ejecución de las *OTs* del ciclo de control de PILOTO cuando se espera a procesar todos los datos del ciclo anterior. 281
6.24 Posible distribución de los especialistas y agentes de AFE-Robótica en una red de computadores. 283

Introducción

La robótica móvil es un campo de investigación en constante desarrollo debido a sus múltiples aplicaciones. Destacan la realización de tareas en entornos peligrosos o no fácilmente accesibles para los seres humanos (exploración espacial o de fondos marinos, trabajo en centrales nucleares, desactivación de minas, etc.), la ejecución rutinaria de tareas monótonas y fatigosas (mantenimiento, almacenaje, reparto, fabricación, etc.), las visitas guiadas y los servicios de búsqueda y rescate.

El diseño de una arquitectura de control para un robot móvil plantea muchos e importantes retos. Los robots operan e interaccionan con entornos complejos, dinámicos e impredecibles a través de sensores y efectores imperfectos. El entorno en el que se mueve el robot no se puede modelar completamente y posee su propia dinámica, muchas veces desconocida. La información sensorial es siempre incompleta, ya que los sensores sólo pueden medir un número muy reducido de variables del entorno. Las medidas suelen ser imprecisas e incluso ambiguas. Todo ello implica que los datos sensoriales se tienen que monitorizar e interpretar constantemente, para poder disponer de un conocimiento suficiente del entorno.

Por otra parte, un robot debe ser capaz de planificar la consecución de múltiples objetivos, a veces contradictorios entre si. En general, los robots deben ser autónomos, es decir, actuar de forma independiente y sin ayuda externa, resolviendo las dificultades que surjan durante la interacción con el entorno. Y, por supuesto, deben operar en tiempo real, lo que significa que el tiempo para tomar decisiones es siempre limitado.

En general, estos requerimientos demandan una gran potencia de procesamiento, lo que entra en conflicto con las limitaciones propias de un robot móvil en cuanto a la capacidad de carga, a su volumen y a la energía disponible. Por todo ello, las distintas tareas que realiza internamente un robot móvil han de ser continuamente priorizadas según las necesidades y recursos disponibles en cada momento. Todos estos hechos hacen que la arquitectura de control de un robot móvil sea un elemento

crítico, que condiciona absolutamente la aplicabilidad del mismo a problemas reales.

Además de las consideraciones propias del dominio, en la implementación de una arquitectura de control también se han de tener en cuenta las propiedades exigibles a cualquier sistema software: modularidad, robustez, eficiencia, portabilidad, reutilización de componentes, etc. Por otra parte, es muy importante que la arquitectura de control sea suficientemente flexible como para facilitar los cambios que surjan durante la implementación con respecto a los requerimientos iniciales así como en futuras modificaciones. También es conveniente que la arquitectura permita comparar, con el menor número de cambios y en igualdad de condiciones, diferentes métodos y técnicas para resolver una misma tarea. Por último, es fundamental que la arquitectura se pueda escalar fácilmente, de forma que un sistema complejo se pueda implementar de forma gradual y progresiva, y se pueda adaptar a las necesidades de cada aplicación.

Teniendo en cuenta todos estos requisitos y condicionantes, nuestro objetivo ha sido el diseño y la síntesis de una arquitectura de control para robots móviles autónomos, con el interés puesto en incidir en algunos de los principales problemas y limitaciones de las soluciones existentes. El trabajo llevado a cabo se describe en esta memoria, de acuerdo con la estructura que a continuación se indica.

Estructura de la memoria

Capítulo 1. En este primer capítulo se repasan algunos de los conceptos básicos relacionados con la robótica y se describen los principales componentes físicos, campos de aplicación y limitaciones de los robots móviles. En particular, se describe el robot utilizado durante el desarrollo del trabajo al que se refiere en esta memoria.

Capítulo 2. Dado que nuestra investigación se centra en los robots móviles autónomos (RMA), es imprescindible explicar qué significa dicho concepto y estudiar cuáles de sus funcionalidades se pueden considerar básicas. A partir de ahí, se marcan los objetivos propios de cualquier RMA (sobre todo ser útil y mantenerse operativo) y se clasifican algunas de las tareas necesarias para conseguirlos. Por último, se fija la meta del presente trabajo y se comentan las simplificaciones adoptadas respecto al planteamiento inicial, es decir, dotar de contenido (inteligencia básica) el hardware de un RMA para que éste pueda

realizar tareas de interés. También se establecen el tipo de tareas a realizar, el nicho ecológico y el soporte físico de un robot de "propósito general".

Capítulo 3. Una vez vistos los principales aspectos relacionados con un RMA, pasamos a estudiar qué tipo de arquitectura se necesita para implementar su sistema de control. En primer lugar, definiremos qué entendemos por arquitectura en robótica y estableceremos qué requisitos debe cumplir en general. También fijaremos los criterios que a nuestro parecer se han de exigir a cualquier arquitectura de control para RMA y que también nos servirán para evaluar y comparar distintas arquitecturas entre sí. A continuación, pasaremos revista a las arquitecturas de control más importantes que se han propuesto en robótica móvil, indicando las principales ventajas y limitaciones de cada una. Se hace hincapié en las deficiencias que se observan en todas ellas, y se introducen las principales características de nuestro modelo de arquitectura de control basado en especialistas, que se ajusta mejor a los criterios fijados.

Capítulo 4. Después de indicar las principales propiedades de la nueva arquitectura de control propuesta, en este capítulo se realiza una descripción detallada de cómo puede ser implementada y de todos y cada uno de sus componentes. Aunque la arquitectura puede aplicarse a otros dominios, en este trabajo nos centraremos en su aplicación al dominio de la robótica móvil. Al resultado obtenido lo hemos denominado AFE-Robótica.

Capítulo 5. Una arquitectura es sólo un cascarón vacío si no se aplica a la resolución de una determinada tarea. Se ha elegido la tarea de navegación porque es fundamental para la operatividad y autonomía de un robot móvil y porque, a pesar de la gran cantidad de estudios sobre el tema, aún no es una tarea resuelta en entornos no estructurados, dinámicos y no totalmente modelables, precisamente aquellos en donde queremos que se desenvuelva nuestro robot. El primer paso ha sido desarrollar un prototipo básico que resulta de establecer un compromiso entre varios intereses: que su implementación sea factible en un plazo de tiempo razonable y con recursos limitados, que sirva para validar la arquitectura propuesta y que sea capaz de realizar tareas útiles, suficientemente complejas y usuales en un robot móvil. En este capítulo, además de la estructura general del prototipo implementado, también se describen con detalle todos los aspectos importantes de cada uno de los especialistas que lo componen y de su implementación y se presentan los resultados de su validación.

Capítulo 6. Aunque los distintos agentes desarrollados se han ido validando tras su diseño e implementación, es necesario demostrar el correcto funcionamiento de AFE-Robótica en su conjunto. Para ello se planificaron toda una serie de pruebas de validación de la arquitectura, que se describen en lo sustancial en este capítulo. Primero, se muestran los resultados obtenidos en pequeños desplazamientos y después en rutas más largas, siempre dentro de las posibilidades del entorno en donde se realizaron los experimentos. También se comentan las distintas incidencias encontradas durante dichos experimentos. Por último, sobre el prototipo implementado se estudiaron las principales propiedades que debe poseer una arquitectura de control, exponiendo las conclusiones más importantes que se pueden extraer de los resultados obtenidos.

Capítulo 1

Introducción a los robots móviles autónomos

En este capítulo se introducen las definiciones más importantes referentes a los robots móviles autónomos, se estudian sus distintos tipos y se describen sus principales componentes físicos, campos de aplicación y problemas más importantes. Por último, se describe el robot móvil utilizado durante nuestra investigación.

1.1 Definiciones

El propósito de esta sección consiste en introducir los conceptos y términos técnicos más importantes referentes al campo de la robótica móvil autónoma, que es el área en donde se inscribe el presente trabajo.

Robot. Según todas las investigaciones la palabra "robot" proviene de una obra de teatro llamada "R.U.R.", escrita en 1921 por el escritor checo Karel Capek. Capek derivó dicho nombre del término checo *robota*, que significa "trabajo forzado". En dicha obra, un robot era un trabajador mecánico (humanoide) sin ningún tipo de voluntad y totalmente esclavizado.

Una definición más técnica de lo que es o se entiende por un robot es la que nos proporciona la RIA (*Robotics Industry Association*). Para dicha asociación un robot es "un manipulador programable y multifuncional, diseñado para mover materiales, piezas, herramientas o dispositivos especializados mediante movimientos programados variables con objeto de realizar diversas tareas". Esta definición es bastante

restrictiva ya que, entre otras cosas, no considera la posibilidad de desplazamiento.

Para Brady (Brady, 1986) un robot es una conexión inteligente de percepción y acción. Esta definición es quizás demasiado amplia, aunque incluye los aspectos más importantes de un robot. Una definición más operativa podría ser la que utiliza Arkin (Arkin, 1998), según la cual un robot (inteligente) es una máquina capaz de extraer información del entorno y de usar el conocimiento que posee sobre el mundo para moverse de forma segura y con un propósito.

La palabra "robótica" apareció por primera vez en la novela de ciencia ficción "Runaround" escrita por el escritor y divulgador norteamericano Isaac Assimov. Con ella se designa a la ciencia que estudia los robots.

Robot móvil. La mayoría de los robots industriales de hoy en día son manipuladores ("robots de ensamblaje"), normalmente anclados, y que operan dentro de un espacio de trabajo bien delimitado y estructurado. Sin embargo, nosotros estamos más interesados en robots móviles, es decir, sistemas que se pueden mover y desplazar, más o menos libremente, por su entorno. Como veremos en el apartado siguiente, existen muchos tipos de robots móviles, tantos como tipos de vehículos y medios de transporte.

Los robots móviles más comunes actualmente son los "vehículos de guiado automático" (*Automated Guided Vehicle* o AGV). Dichos robots móviles operan en entornos especialmente acondicionados y generalmente realizan tareas de transporte a lo largo de rutas prefijadas. Debido a la necesidad de alterar de forma apropiada el entorno, estos robots son inflexibles y poco fiables, ya que cualquier cambio no previsto en el entorno les puede impedir funcionar correctamente. Por otra parte, la modificación del entorno siempre es costosa y problemática. Por lo tanto, la solución no consiste en adaptar el entorno al robot sino en diseñar sistemas flexibles que se adapten al entorno, es decir, robots móviles autónomos.

Autonomía. La autonomía completa no existe (Pfeifer y Scheier, 2000), ya que todos los sistemas dependen, de uno u otro modo, de su entorno. La autonomía es, por tanto, una cuestión de grado. Según (Merriam-Webster, 1993) existen dos definiciones de autonomía o, lo que es lo mismo, se pueden establecer dos grandes categorías: actuación sin control externo y capacidad de auto-gobierno.

En el primer caso, cualquier sistema que transporte su propia fuente de energía y su propio sistema de control se dice que es autónomo, ya que no depende del exterior,

es decir, es autosuficiente. Sin embargo, este tipo de autonomía se considera "débil", ya que cualquier alteración del entorno o situación no especificada generalmente conlleva un funcionamiento incorrecto del sistema.

La segunda definición implica un mayor grado de autonomía que podemos denominar "autonomía fuerte". En este nivel un sistema es autónomo cuando tiene capacidad para decidir, a partir de sus propios procesos de razonamiento, las acciones que ha de tomar en vez de seguir y ejecutar una secuencia fija e inmutable de pasos o instrucciones suministrada desde el exterior.

La característica más importante de un sistema autónomo es su capacidad para operar durante largos periodos de tiempo en un entorno "natural" (es decir, sin alteraciones que faciliten su funcionamiento) y sin ningún tipo de intervención humana directa. Esta cualidad conlleva, inevitablemente, la adaptación a los cambios que se produzcan en el entorno, una actuación aceptable ante situaciones no previstas, el aprendizaje y modificación del comportamiento a partir de la propia experiencia, y la construcción de representaciones internas del mundo para utilizarlas en los procesos internos de razonamiento, por ejemplo, la navegación.

Robot móvil autónomo. En resumen, podemos definir un robot móvil autónomo como un sistema físico capaz de moverse sin la intervención humana para realizar tareas o alcanzar objetivos en entornos reales no diseñados específicamente para él.

Agente. En latín, *agere* significa "hacer". En este trabajo se utiliza el término agente para designar cualquier elemento o entidad que produce un efecto. En particular, hablaremos de agentes cuando estemos considerando entidades o módulos "software" que generan un efecto o realizan una tarea específica. Como un robot produce un efecto y realiza tareas, también se le puede considerar un agente. Sin embargo, la diferencia estriba en que un robot es un sistema físico, mientras que para nosotros un agente sólo es un módulo software.

Inteligencia. El término de inteligencia es ampliamente utilizado en la bibliografía, aunque hasta ahora se ha resistido a una definición clara, útil y ampliamente aceptada por toda la comunidad científica. Por este motivo, establecer si un sistema se comporta de forma inteligente no deja de ser, hasta cierto punto, subjetivo y sujeto a la propia interpretación del observador y, por lo tanto, a su educación, entendimiento y punto de vista. Por todo ello, en este trabajo se evitará considerar

Generación	Nombre	Tipo de control	Grado movilidad	Usos frecuentes
1ª (1982)	*Pick & Place*	Fines de carrera.	Ninguno	Manipulación, servicio máquinas
2ª (1984)	Servo	Servocontrol, trayectoria continua.	Desplazamiento por vía	Soldadura, pintura
3ª (1989)	Ensamblado	Servos de precisión, visión, tacto.	AGV Guiado por vía	Ensamblado
4ª (2000)	Móvil	Sensores inteligentes	Patas Ruedas	Construcción Mantenimiento
5ª (2010)	Especiales	Controladores con técnicas de IA	Andante, Saltarín	Uso militar Uso espacial

Tabla 1.1: Clasificación de los robots según Knasel (Knasel, 1986).

si un sistema es o no inteligente, nos centraremos más en su grado de autonomía.

Triada robot-tarea-entorno. Al igual que la inteligencia, el comportamiento de un robot no puede ser evaluado independientemente de su entorno y de las tareas que realiza. Robot, tarea y entorno dependen unos de otros y se influyen mutuamente entre sí, de manera que no pueden ser considerados de manera aislada y forman lo que se conoce como triada robot-tarea-entorno. La consecuencia más importante de esta fuerte dependencia es que debemos evitar el plantear un robot de propósito general, es decir, un robot apto para ejecutar cualquier tarea en cualquier tipo de entorno. En realidad lo que ocurre es que las características del robot delimitan el conjunto de entornos en donde puede desenvolverse y el tipo de tareas que puede ejecutar en el mismo. O, lo que es lo mismo, las cualidades mínimas que debe poseer un robot vienen fijadas por el entorno y el tipo de tareas a ejecutar.

1.2 Tipos de robots y su clasificación

Una de las clasificaciones de robots más completa es la que propone T.M. Knasel (Knasel, 1986) y que los divide en cinco generaciones (tabla 1.1) según su evolución. En la actualidad, los robots más comunes son los manipuladores o de ensamblado y los móviles.

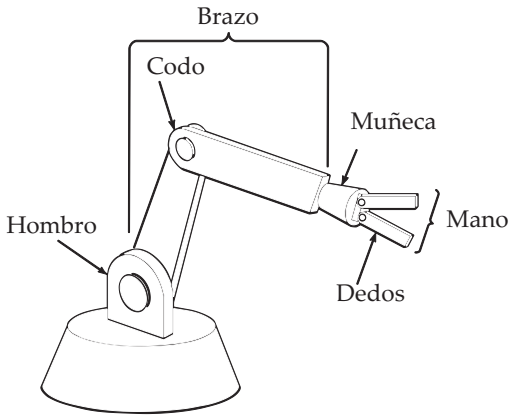

Figura 1.1: Partes de un robot manipulador.

1.2.1 Robots manipuladores

Un robot manipulador es, básicamente, un brazo articulado con un efector que puede modificar el entorno que le rodea, pero con una capacidad de desplazamiento muy reducida (generalmente están anclados en una posición) y un entorno de trabajo muy estructurado y delimitado. En general, un robot manipulador consta de varios elementos que se unen, rotan y deslizan entre sí, formando articulaciones o ejes de movimiento (Fuller, 1995) que reciben nombres análogos a las partes del cuerpo humano, tales como hombro, codo, muñeca y dedos (figura 1.1). La mayoría de los robots industriales sustituyen los dedos por una herramienta (p.e., un soldador).

Los robots manipuladores se pueden clasificar según el número y el tipo de sus articulaciones, el tipo de motores que utilizan, el peso que pueden cargar, el tipo de funciones para las que están diseñados, etc. Sin embargo, lo más usual es dividirlos según la forma de su espacio de trabajo, es decir, de los lugares a los que puede llegar con su efector final (figura 1.2). Una clasificación equivalente es el sistema LERT, acrónimo formado por las iniciales de los cuatro movimientos básicos: lineal (L), extensional (E), rotacional (R) y torsión (T).

Los manipuladores más flexibles y versátiles son aquellos que poseen "dedos" con los que pueden agarrar objetos. El tipo y la configuración de los "dedos" es muy diversa y constituye un activo campo de investigación tanto a nivel de diseño "hardware" como del "software" necesario para su control. Recientemente se han desarrollado "manos robóticas" con una flexibilidad y un número de grados de libertad comparables a las de una mano humana.

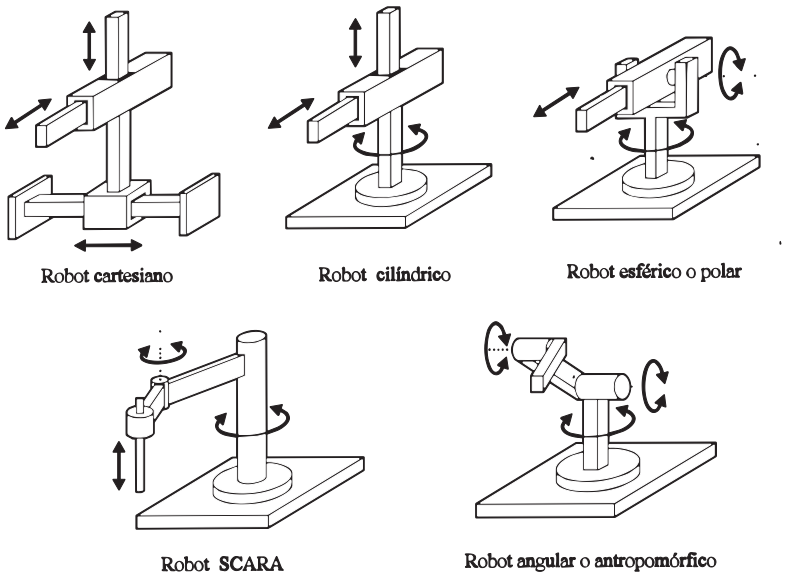

Figura 1.2: Tipos de robots manipuladores y su clasificación según el sistema LERT: (de izquierda a derecha y de arriba a abajo) coordenadas cartesianas (L^3), coordenadas cilíndricas (L^2R^1), coordenadas polares (L^1R^2), tipo SCARA (L^2R^1) y coordenadas de revolución (R^3).

1.2.2 Robots móviles

Los robots móviles se caracterizan por su capacidad para moverse por el entorno que los rodea. La forma en la que se realiza dicho desplazamiento establece el modelo de clasificación más intuitivo. En primer lugar, al igual que cualquier otro vehículo, los robots móviles se clasifican según el medio en dónde se desplazan. Podemos hablar entonces de robots espaciales, aéreos, acuáticos y terrestres. Por supuesto, también habría que incluir ciertos tipos mixtos, por ejemplo, los *hovercraft* y los hidroaviones.

Hasta ahora han escaseado los robots espaciales, es decir, aquellos que se desenvuelvan en el espacio exterior,[1] aunque sí existen ejemplos de robots acuáticos (principalmente submarinos) y de robots aéreos (principalmente helicópteros). Sin ninguna duda, los robots móviles son los que más se utilizan actualmente. En parte porque son más fáciles de controlar (en general, sólo poseen tres grados de libertad), y en parte porque su construcción y operación no requiere equipos complejos. Además, tienen un abanico mayor de aplicaciones.

[1] El *Mars Path Finder* no se puede considerar un robot espacial sino más bien un robot terrestre adaptado a la exploración planetaria.

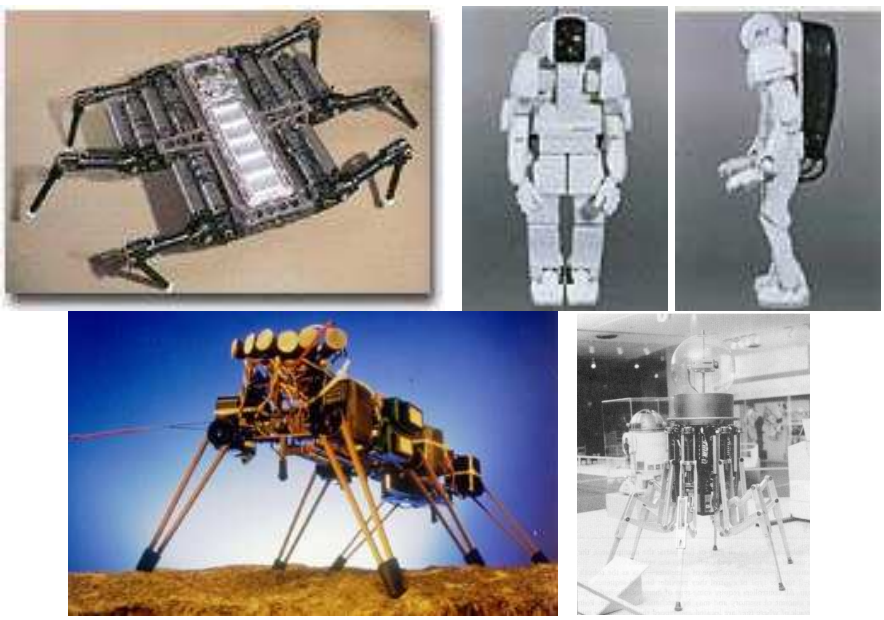

Figura 1.3: Ejemplos de robots móviles con patas. Arriba izquierda, *Ariel*, centro y derecha, robot humanoide de Honda *P3*. Abajo izquierda *Ghengis,* y derecha, *Odix*.

Los robots móviles terrestres se pueden clasificar según el sistema de transporte que utilicen: patas o ruedas.[2] Los robots basados en ruedas son más fáciles de controlar (cada rueda posee un grado de libertad y cómo máximo el robot suele tener sólo dos grados de libertad), son más robustos (hay menos partes móviles), energéticamente más eficientes, más rápidos, admiten mayores cargas, etc. Por el contrario, los robots basados en patas se adaptan mejor a entornos complejos y con muchos desniveles y, aunque son más complejos de controlar porque existen más grados de libertad, también permiten una mayor versatilidad de movimientos. También suelen ser más robustos a fallos (por ejemplo, si una pata deja de funcionar).

Los **robots con patas** se pueden clasificar según su número de patas. Los tipos más comunes son los humanoides, los insectoides y los funcionoides (figura 1.3). Los primeros se caracterizan por ser bípedos e inestables estáticamente. Los robots insectoides son los más comunes y suelen tener seis patas. Los robots funcionoides son aquellos que no están inspirados en la naturaleza, por ejemplo, *Ambler* (Bares et al, 1989) y *Odix* (figura 1.3) cuyas patas se disponen circularmente alrededor de la base.

[2]Las orugas se pueden considerar como una variante de las ruedas, por lo menos en lo que respecta a su control.

Igualmente, los **robots con ruedas** (figura 1.4) se pueden clasificar según cómo se disponen las ruedas motrices y directrices (Everett, 1995). Existen cinco configuraciones principales (figura 1.5): diferencial, Ackerman, síncrona, triciclo y omnidireccional.

- Diferencial. Es la más sencilla, ya que consta de dos ruedas que son al mismo tiempo motrices y directrices. La estabilidad se consigue gracias a ruedas adicionales que giran libremente. Para girar el robot basta girar ambas ruedas a distinta velocidad. Incluso es posible realizar giros de radio cero. A esta categoría pertenecen todos los robots con tracción de oruga.
- Ackerman. Posee dos ruedas motrices y otras dos directrices. Su principal característica es que cuando el robot gira las circunferencias que trazan las dos ruedas directrices tienen el mismo centro. Energéticamente es la más eficiente y permite mayores velocidades pero no es holonómica.
- Síncrona. Todas las ruedas giran simultáneamente sobre sí mismas y apuntan siempre en la misma dirección. En esta configuración es posible desacoplar los giros y los desplazamientos pero permite velocidades menores y exige una cuidadosa calibración.
- Triciclo. Es muy fácil de implementar y de controlar pero es algo inestable por disponer de sólo tres ruedas.
- Omnidireccional. Permite desplazamientos en cualquier dirección sin necesidad de girar. Son poco frecuentes por su baja eficiencia y fiabilidad.

1.3 Soporte físico

Un robot es una máquina que se desenvuelve en el mundo real, por lo tanto, gran parte de sus limitaciones, necesidades y capacidades vendrán impuestas por los distintos elementos físicos de los que se compone. El primer paso para conocer un robot es comprender cómo funcionan sus distintos componentes y cuáles son sus propiedades y sus limitaciones.

Los componentes de un robot se pueden dividir en tres apartados: sensores, efectores y computación. Los sensores recogen información sobre el entorno y sobre parámetros internos (sensores exteroceptivos y propioceptivos, respectivamente). Los módulos de computación realizan los cálculos necesarios a partir de dicha

Figura 1.4: Ejemplos de robots con ruedas y orugas. Arriba izquierda, *Flakey* (construido en el SRI, prototipo del *Nomad 200*); centro, *Xavier* (prototipo del *B21* de RWI-ISR); derecha, *Urban* (RWI). Abajo, *Pioneer 2 DX* y *AT* (ActivMedia).

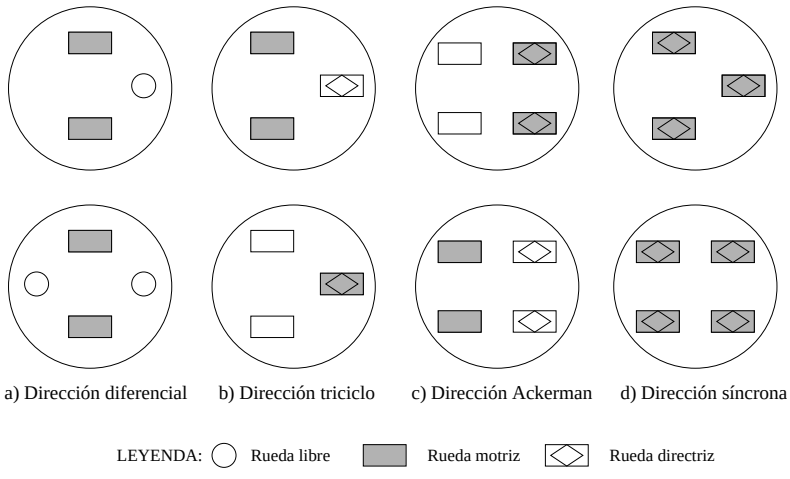

Figura 1.5: Ejemplos de las distintas configuraciones de ruedas motrices y directrices en un robot móvil.

información y toman las decisiones oportunas para que el robot realice las tareas encomendadas. Por último, los efectores generan las acciones o los movimientos que los módulos de computación han decidido.

1.3.1 Sensores

Los sensores son claves en robótica, y sobre todo en robótica móvil autónoma, porque para que un robot pueda moverse por su entorno de forma autónoma es imprescindible que tenga la capacidad de extraer del entorno toda la información que pueda ser relevante. Si el sistema sensorial es demasiado incompleto o imperfecto, el funcionamiento del robot se verá severamente limitado.

En realidad, los sensores son dispositivos que pueden medir una cierta magnitud física del entorno, tal como temperatura, luminosidad, distancia, tiempo, etc. Generalmente, existe una relación entre cada magnitud física y una característica del entorno que interesa al robot. Por ejemplo, una cámara mide el nivel de luminosidad en una determinada región desde una cierta posición y orientación. Pero realmente al robot le interesa saber, por ejemplo, si hay espacio libre, qué objetos hay en esa región, si alguno de los objetos se mueve, etc.

Los sensores generan información de bajo nivel, es decir, medidas sobre una cierta magnitud física. Esa información es imprecisa, con mucho ruido y, a menudo, ambigua y contradictoria. Por lo tanto, para que el robot realmente sea capaz de "percibir" su entorno, la información de sus sensores debe ser procesada apropiadamente para dotarla de significado para el robot y que pueda así serle de utilidad.

Existen muchos y variados tipos de sensores que se aplican en robótica móvil, aunque ninguno de ellos es perfecto y todos poseen sus inconvenientes. En esta sección sólo se van a comentar los más importantes y los más ampliamente utilizados, indicando sus principales aplicaciones y limitaciones. Para un estudio en mayor profundidad se puede consultar la bibliografía existente al respecto (Everett, 1995).

Sensores táctiles

Los sensores táctiles detectan el contacto físico del robot con un objeto de su entorno. Generalmente la detección se suele hacer mediante métodos indirectos, es decir, a través de los *efectos* que produce el choque en ciertos elementos del robot. Existen distintos tipos de sensores táctiles. Los más sencillos son los microinterruptores (Pio, 2000) y los "bigotes metálicos". Cuando un objeto toca uno de estos

sensores se cierra (abre) un circuito y se produce una señal eléctrica que puede ser fácilmente detectada (Jones y Flynn, 1993). Obviamente, sólo proporcionan información de tipo binario. Sin embargo, son muy sencillos de implementar y con ellos se puede "recubrir" todo el robot (p.e., mediante paneles (B21, 1995)).

Otros dispositivos bastante comunes están basados en materiales que cambian su resistencia a la corriente eléctrica cuando son deformados (generalmente, por compresión) mediante una fuerza externa. Midiendo la variación de la resistividad del material se puede calcular el punto de contacto (Nom, 1993). En las versiones más sofisticadas de este tipo de sensores, también se puede determinar la intensidad de la deformación y, por tanto, la fuerza aplicada.

El principal inconveniente de los sensores táctiles es que su activación significa un contacto físico con un objeto. En general, se suele evitar que el robot toque los objetos del entorno porque, aunque el contacto se realice a bajas velocidades y de manera controlada, puede producir desperfectos tanto en el robot como en los elementos del entorno.

Los sensores táctiles se pueden considerar el último recurso para detectar las colisiones del robot con los obstáculos que le rodean y evitar así males mayores. Si se disponen de tal manera que "recubran" totalmente el robot, entonces pueden detectar un choque desde cualquier posición.

Sensores de infrarrojos

El funcionamiento de los sensores de infrarrojos (IR) se basa en emitir un haz de luz en el espectro infrarrojo de las ondas electromagnéticas y en detectar la que es reflejada por los objetos cercanos al robot. En principio, cuanto más lejos está un objeto menos luz refleja. Por lo tanto, se puede calibrar el sensor para determinar a qué distancia está cada objeto según la intensidad de luz IR medida.

La principal característica de estos sensores es que son los más sencillos y fáciles de implementar, por lo que son los sensores más ampliamente utilizados en robots pequeños y sencillos. Otra cualidad interesante es que poseen una resolución angular muy alta, ya que el haz de luz IR se puede hacer muy estrecho y se puede emitir en una dirección con gran precisión. Por último, es muy difícil que los sensores de IR detecten objetos "fantasmas", es decir, que no existen o que se detecten en una posición distinta a la que en realidad ocupan.

Entre sus principales limitaciones destaca que la intensidad de luz IR reflejada por los objetos también depende de la reflectividad de la superficie del objeto. Como

los objetos poseen reflectividades a la luz IR muy diferentes es difícil medir con fiabilidad las distancias. De hecho, un número importante de objetos de color negro prácticamente no reflejan la luz IR y, por tanto, son casi imposibles de detectar con sensores IR.

Una segunda limitación es que la intensidad de la luz IR disminuye con el cuadrado de la distancia. Por lo tanto, el alcance de los sensores de IR es limitado aún en condiciones ideales, típicamente del orden de 50 a 100 cm También provoca que las medidas no puedan ser demasiado precisas. Una tercera limitación de los sensores IR es que son activos, es decir, para funcionar necesitan emitir energía.

Sensores de ultrasonidos

El funcionamiento de los sensores de ultrasonidos (US) se basa en emitir un fuerte pulso de sonido de una frecuencia o rango de frecuencias fijas y detectar las señales reflejadas por los objetos del entorno. A partir de dichas señales se puede calcular la posición de los distintos obstáculos, su forma y hasta su naturaleza. Para lograr extraer tal cantidad de información es imprescindible utilizar varios detectores y procesar directamente la señal (p.e., midiendo intensidades, diferencias de fase, etc.). El proceso es complejo, costoso y no siempre fiable, pero ya existen sistemas experimentales operativos (Chong y Kleeman, 1997).

Sin embargo, la técnicas más usada consiste simplemente en medir el tiempo que transcurre desde la emisión del pulso de US hasta que la intensidad de la señal reflejada supera un cierto umbral. Como la velocidad del sonido es conocida, se puede calcular la distancia a la que se encuentra el objeto que ha reflejado el pulso (Harris y Recce, 1998). Para simplificar el diseño y reducir costes en este tipo de sistemas, lo normal es utilizar el dispositivo emisor (transductor) también como receptor.

Los sensores de US poseen propiedades muy interesantes, ya que son bastante sencillos de implementar, pueden detectar objetos en un rango de distancias relativamente amplio (de 6 cm a 10 m), tienen una precisión aceptable (el error en las medidas suele ser inferior al 5%), consumen poca energía, son ligeros y fáciles de instalar. Por otra parte los datos que generan son fáciles de procesar en tiempo real. Por todo ello son los sensores más utilizados en robótica móvil.

Respecto a su alcance, reseñar que la distancia mínima de detección viene determinada por el periodo de reposo que necesita el dispositivo emisor antes de poder actuar como detector. Por su parte, la distancia máxima depende de la energía

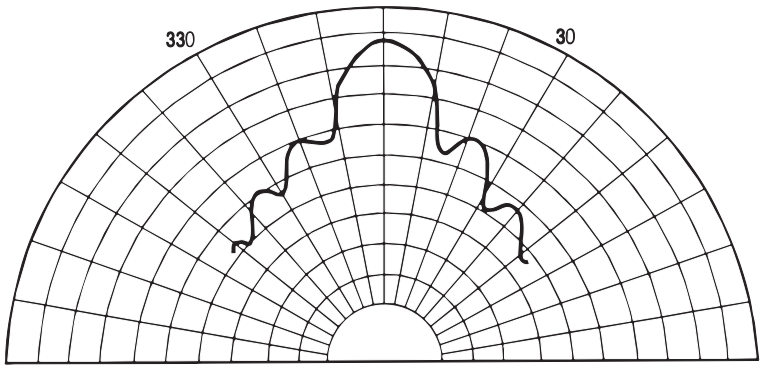

Figura 1.6: Patrón de emisión de un emisor típico de ultrasonidos (transductor electrostático Polaroid modelo 600 a 50 KHz).

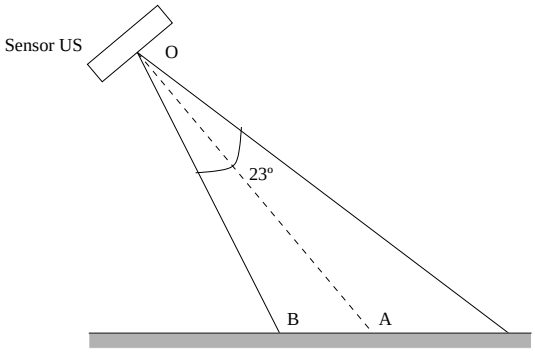

Figura 1.7: Error que comete un sensor de ultrasonidos debido a la divergencia del pulso que emite. La distancia real es OA, pero se mide OB.

emitida y de la sensibilidad del detector.

Sin embargo, los sensores de US no están exentos de ciertas limitaciones importantes. La primera, y más destacada, es su baja resolución angular debido a que los pulsos emitidos tienen una forma cónica (figura 1.6). Para concentrar la emisión y reducir el ángulo de abertura es imprescindible aumentar las dimensiones del transductor, lo que no siempre es factible. Por otra parte, la forma del pulso también afecta a las medidas, ya que las señales emitidas en los laterales del cono se pueden reflejar antes que aquellas que se emiten en el eje del cono (figura 1.7). Cuanto mayor sea el ángulo de incidencia del pulso de US respecto a la normal a la superficie del objeto, mayor será el error cometido en el cálculo de la distancia.

Otra limitación importante de estos sensores son las reflexiones especulares que se producen cuando el pulso de US incide sobre una superficie suave y/o con un ángulo muy agudo, de tal manera que el pulso no es reflejado de vuelta hacia al sensor.

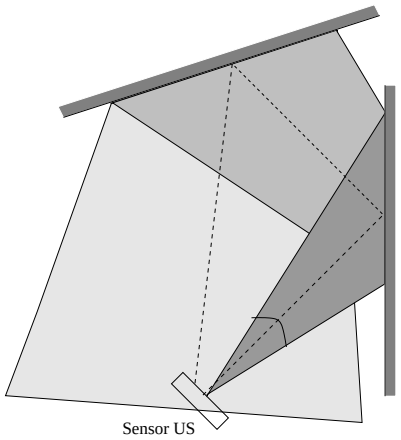

Figura 1.8: Ejemplo de la reflexión especular en los sensores de ultrasonidos.

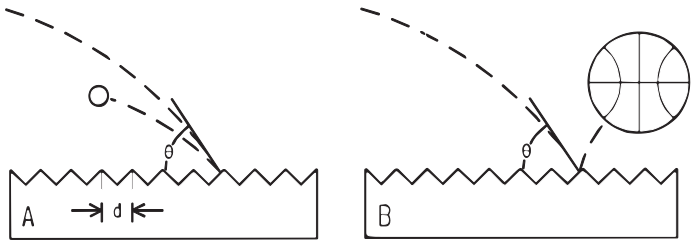

Figura 1.9: La reflexión de una onda sobre una superficie depende del tamaño relativo entre la rugosidad de dicha superficie y la longitud de onda utilizada, de un modo muy semejante a como rebotan pelotas de muy distinto tamaño sobre una superficie rugosa.

Por el contrario, si el pulso reflejado incide de nuevo en otro u otros obstáculos y se encamina hacia el sensor, entonces se detecta un objeto pero a una distancia superior a la real (figura 1.8).

El rango de frecuencias utilizadas está entre 20 y 100 KHz, muy altas para ser audibles por el oído humano pero relativamente sencillas de generar. Como la velocidad del sonido a 20 ^{o}C es de unos 344 m/s, a tales frecuencias le corresponden longitudes de onda de entre 3,5 y 17 milímetros. Objetos con superficies rugosas de ese tamaño reflejan el sonido en muchas direcciones y son, por tanto, fácilmente detectables. Por el contrario, si las superficies son suficientemente lisas se comportan como espejos y agudizan el problema de la reflexión especular (figura 1.9). Existen objetos que absorben completamente el sonido y que son indetectables para los sensores de US, pero son poco frecuentes.

Otra limitación de los sensores de US surge cuando se utilizan varios, lo que

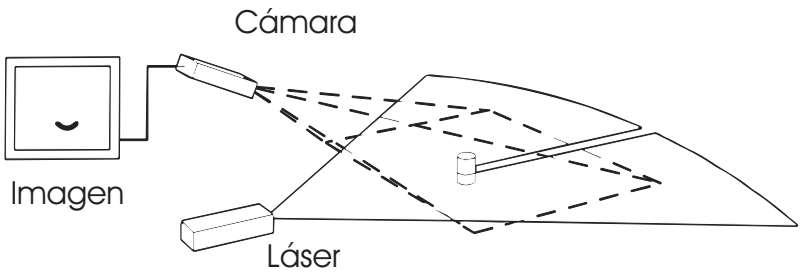

Figura 1.10: Operación de un sensor láser basado en técnicas de triangulación.

provoca interferencias entre ellos, es decir, un sensor detecta el pulso reflejado de otro sensor antes de detectar la reflexión del pulso que él ha emitido. Para evitar este problema los sensores se activan siguiendo patrones pensados para minimizar este problema.

Por último, señalar que existe cierta imprecisión en la medida de las distancias porque no se cumple la hipótesis de que la velocidad del sonido sea constante, sino que depende de la temperatura del aire y del grado de humedad. Sin embargo, este aspecto no es excesivamente relevante, ya que un cambio de 16 ^{o}C en la temperatura produce un error de 30 cm al medir una distancia de 10 m.

Sensores láser

Los sensores basados en láser se están haciendo muy populares en robótica móvil. Las características que más destacan de ellos son su alta precisión, tanto en ángulo como en distancia, y, en general, su alcance. Por otro lado, su principal limitación sigue siendo el precio, el alto consumo energético y su peso. Existen tres tipos básicos de estos sensores según estén basados en técnicas de triangulación, en el tiempo de vuelo o en el desplazamiento de fase.

En la figura 1.10 se puede observar cómo opera un sensor láser basado en técnicas de **triangulación** (Nom, 1995). El láser emite un haz de luz coherente que al pasar por un prisma se transforma en un haz plano. Una cámara de vídeo alineada con el emisor láser capta el haz reflejado en los objetos del entorno. Como la cámara no está en el mismo plano que el haz láser se puede establecer una equivalencia entre la posición del haz reflejado en la imagen captada por la cámara y la distancia a la que se encuentra el objeto.

Las principales limitaciones de este tipo de sensores láser son las mismas que cualquier método basado en triangulación. En primer lugar, se debe establecer un

compromiso entre alcance y precisión. Aumentando la separación entre la cámara y el láser se consigue una mayor precisión de las medidas. Por otra parte, para aumentar el alcance es necesario reducir el ángulo entre la dirección a la que apunta la cámara y el plano del haz láser.

En segundo lugar, el plano de visión de la cámara es constante (figura 1.10) lo que condiciona el rango de distancias y de ángulos en los que se puede detectar un objeto. Así, al aumentar el alcance máximo del sensor (disminuyendo el ángulo entre la cámara y el plano del haz) también se aumenta la distancia mínima de detección, y viceversa. Por último, sólo se pueden detectar aquellos objetos iluminados por el láser pero que "entran" dentro del plano de visión de la cámara. En general, un sensor láser basado en triangulación tiene severas limitaciones, ya que sólo detecta objetos entre 50 y 250 cm de distancia y a menos de 15º respecto al eje del sensor.

En tercer lugar, el sistema falla si la cámara no detecta el haz reflejado, bien porque existe algún objeto que se interponga entre la cámara y el haz reflejado, bien porque el haz no ilumina los objetos correctamente. Como el haz es cónico no se abre lo suficiente para iluminar objetos laterales muy próximos, mientras que objetos lejanos se iluminan débilmente (la energía por unidad de superficie disminuye con el cuadrado de la distancia). Como en todos los sensores activos el alcance depende estrechamente de la energía que emite el láser y de la reflectividad de las superficies a la longitud de onda utilizada.

El funcionamiento de los sensores láser basados en el **tiempo de vuelo** es similar a los sensores de US pero sustituyendo los pulsos de sonido por pulsos de luz (PMS, 2000). Este cambio conlleva una tecnología y una electrónica mucho más compleja, ya que la velocidad de la luz es muy superior a la del sonido. Por otra parte, como se utilizan longitudes de onda mucho más pequeñas, la probabilidad de que se produzca una reflexión total en una superficie suave es muchísimo menor, por lo que el problema de la reflexión especular es menos pronunciado.

Las principales características de este tipo de sensores son su alcance (entre 10 y 100 m), su amplio ángulo de percepción (típicamente 180º), el gran número de medidas por adquisición (entre 100 y 400) y su precisión (típicamente de milímetros).

En los sensores láser basados en el **desplazamiento de fase** la distancia se calcula midiendo la diferencia de fase entre el haz emitido y el reflejado. Para ello se modula la intensidad de la señal emitida, es decir, se cambia con el tiempo. La diferencia de fase es proporcional a la distancia recorrida por el haz y a la frecuencia de modulación de la señal.

Figura 1.11: Montaje para la visión: a) con dos grados de libertad, b) en estéreo y con cuatro grados de libertad, y c) omnidireccional.

Cámaras de vídeo

Las cámaras de vídeo (digitales) generan una matriz de números que se corresponde con la distribución del nivel de gris o del color que existe en una imagen. Usualmente la resolución de las cámaras (número de puntos por imagen) es de unos 800 x 600 y su velocidad es de unas 30 imágenes por segundo, lo que da lugar a una ingente cantidad de información, del orden de 14,4 millones de datos por segundo. El problema es que cada punto de la imagen (pixel) tomado de forma individual aporta muy poca información sobre la escena en su conjunto. La conversión de la inmensa cantidad de datos que produce una cámara en información útil de alto nivel es tarea de la visión artificial (Sonka y col., 1993).

La principal aplicación de las cámaras de vídeo es generar o detectar la información que sea de interés para el robot sobre el entorno que le rodea. Lo normal es utilizar una o dos cámaras montadas en un soporte fijo o con varios grados de libertad. El montaje fijo facilita la calibración de las cámaras y, por tanto, el procesado posterior de las imágenes obtenidas, mientras que el movimiento de las cámaras permite obtener más información sobre el entorno y desde distintos puntos de vista, por lo que el robot necesita desplazarse menos.

Una de las configuraciones típicas es la estéreo (figura 1.11.b) donde las cámaras se pueden focalizar en un mismo punto del espacio, lo que se denomina vergencia. Mediante técnicas de paralelaje se puede calcular la distancia a la que se encuentra un objeto que se observa con dos cámaras desde posiciones conocidas y ligeramente distintas. Otra configuración muy común es la omnidireccional (figura 1.11.c) donde la cámara recoge la imagen reflejada en un espejo obteniendo así una vista panorámica del entorno que rodea al robot. Conociendo el tipo de espejo (cónico,

parabólico, hiperbólico o semiesférico) se puede calcular cómo se distorsiona la imagen. Este montaje se utiliza principalmente en tareas de navegación, confección de mapas del entorno y detección de marcas u objetos en movimiento.

La principal virtud de las cámaras de vídeo es que, seguramente, son el dispositivo sensorial que refleja con mayor precisión y en tiempo real los detalles del entorno. No en vano es el sentido más ampliamente utilizado en los animales superiores. Esta "coincidencia" permite que los datos obtenidos a partir de estos sensores sean fáciles de asimilar y que puedan operar prácticamente en los mismos entornos en los que se desenvuelven los seres humanos. Por último, señalar que estos dispositivos se han hecho muy populares gracias a las aplicaciones multimedia.

La principal limitación de las cámaras de vídeo es que el procesado de las imágenes es extraordinariamente complejo y computacionalmente muy costoso. Extraer información de tipo local (p.e. bordes) es relativamente simple y rápido. La dificultad estriba en obtener información de tipo global y de alto nivel, tanto en cada imagen como en las secuencias de imágenes generadas, es decir, asociar los puntos de la imagen a objetos, identificar los objetos (aunque sólo se vean parcialmente), establecer las relaciones entre objetos, etc.

Para aumentar la potencia de cálculo existen en el mercado hardware específico para el procesamiento masivo y en paralelo de las imágenes. Con estas ayudas se pueden procesar más datos y reducir considerablemente el tipo necesario para extraer la información de alto nivel, pero suelen ser muy caras y consumir mucha energía. Otra limitación (o ventaja según se mire) de estos sensores es que son pasivos, es decir, necesitan iluminación exterior para funcionar. Este problema se minimiza utilizando cámaras térmicas y de visión nocturna.

Compases

Un compás sirve para medir la componente horizontal del campo magnético terrestre y establecer así una dirección absoluta, lo que es muy interesante para tareas de navegación. Existen diversos métodos para realizar esta medición: compases de *fluxgate*, de efecto Hall, magneto-mecánicos, magneto-resistivos o magneto-elásticos. Una discusión completa de todos ellos se puede encontrar en (Borenstein y col., 1996).

La principal limitación de los compases es que el campo magnético terrestre es muy débil y su medición se puede ver distorsionada por objetos metálicos cercanos o campos magnéticos creados por líneas de corriente eléctrica, motores eléctricos,

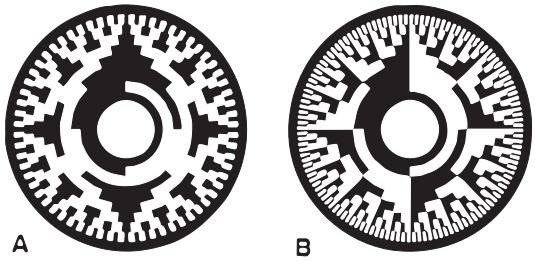

Figura 1.12: Discos de codificadores ópticos absolutos: a) código Gray y b) código binario.

etc. De todos modos, si dichas interferencias son estáticas las mediciones pueden servir como sistema de referencia local, ya que las mediciones serán más o menos constantes en las regiones afectadas.

Sensores codificadores

Los codificadores son los sensores propioceptivos más ampliamente utilizados y su función consiste en medir la posición y/o el movimiento de los distintos efectores del robot. Esa información es fundamental para cualquier robot. Existen diferentes dispositivos y mecanismos para realizar esta función (Borenstein y col., 1996), sin embargo, los más utilizados son los codificadores ópticos.

Los codificadores ópticos consisten en un pequeño disco acoplado al eje de giro del elemento móvil con una serie de líneas dibujadas para ser detectadas mediante células fotoeléctricas (de ahí su nombre). Existen dos tipos de codificadores ópticos: absolutos e incrementales. En los primeros las líneas del disco indican directamente el ángulo que ha girado el eje. Para mayor fiabilidad en vez del código binario se suele utilizar el código Gray (ver figura 1.12) donde cualesquiera dos valores consecutivos sólo se diferencian en un bit. El problema de los codificadores absolutos es que no son muy adecuados para integrar el movimiento a lo largo del tiempo. Para esta función son mejores los codificadores incrementales, en los cuales el disco sólo se compone de una hilera de marcas y que utilizan dos fotodetectores para detectar el sentido del giro. El ángulo girado se calcula contando las distintas marcas detectadas.

La principal utilidad de los codificadores en robótica móvil es calcular la posición del robot en un sistema de coordenadas absoluto. Para ello se mide el giro que realizan las ruedas del robot y, según el tipo de configuración que se esté utilizando, se determina el desplazamiento realizado. El problema estriba en que el sistema odométrico no puede tener en cuenta posibles errores que ocurren en el mundo real.

Los errores sistemáticos, tales como una mala medición del diámetro de las ruedas o de la distancia entre sus ejes, se pueden corregir calibrando el sistema correctamente.

Los errores no sistemáticos son más difíciles de detectar y de corregir. Si las ruedas resbalan o se deslizan el robot se mueve pero los codificadores no lo detectan. Cuando las ruedas se mueven sobre un terreno irregular, con altibajos, recorren más espacio del que realmente recorre el robot en el plano. Por otra parte, los errores de cuantización, inevitables cuando se miden ángulos pequeños, sobre todo cuando el robot realiza pequeñas maniobras, se van acumulando inexorablemente a lo largo del tiempo. En resumen, el cálculo de la posición del robot mediante técnicas odométricas sólo es fiable para cortas distancias, ya que los errores se van acumulando irremediablemente. Para reducirlos es preciso utilizar elementos del propio entorno del robot que sirvan como referencias y recalibrar el sistema constantemente.

Otros sensores

Además de los sensores mencionados existe una amplia variedad de dispositivos que son de interés en robótica móvil aunque tienen una menor difusión, bien por su complejidad, bien porque sólo son útiles en aplicaciones muy específicas. Así, en robots móviles de exteriores los dispositivos GPS (*Global Positioning System*) sirven para determinar la posición absoluta del robot prácticamente en cualquier parte del planeta. En sistemas avanzados tipo diferencial GPS (dGPS) la precisión puede llegar al orden de unos pocos centímetros. Sin embargo, sólo se pueden utilizar en zonas despejadas donde puedan llegar las señales de los satélites.

Por otra parte, para calcular el desplazamiento y la posición odométrica de un robot móvil en terrenos irregulares se necesitan sensores muy especializados y complejos: inclinómetros, sensores inerciales, acelerómetros, giróscopos, medidores de velocidad por efecto Dopler, etc. En sistemas dedicados a vigilancia se utilizan sensores de movimiento, de presencia, acústicos, de vibración, etc. También se utilizan sensores de calor, de humo, de ciertos gases peligrosos, etc. Un amplio resumen de todos estos y muchos otros sensores se puede ver en (Borenstein y col., 1996; Everett, 1995).

1.3.2 Efectores

Un efector es cualquier dispositivo que el robot utiliza para generar una acción sobre su entorno. El tipo de efector más común en robótica, y obviamente en robótica

móvil, son los actuadores, es decir, aquellos que generan algún tipo de movimiento. Pero también se debe considerar como efector a todo aquel dispositivo que modifique algún parámetro físico de su entorno. Por ejemplo, cualquier sistema de señalización: generadores de sonidos (p.e. altavoces y zumbadores), luces o imágenes (p.e., bombillas o monitores), ondas de radio, etc. En este apartado también se agrupan todos aquellos dispositivos especiales, por ejemplo, los que se utilizan para lanzar un balón en la *RoboCup* (Rob, 1997).

Aunque existen dispositivos capaces de producir movimientos, por ejemplo, los materiales con memoria que se pueden deformar pero que recuperan su forma original aplicando calor, los actuadores más frecuentes en robótica móvil son los motores. Éstos se clasifican según el tipo de energía que utilizan para operar en eléctricos, neumáticos e hidráulicos (Barrientos y col., 1997; Fuller, 1995; Nehmzow, 2000).

Los motores neumáticos funcionan inyectando aire comprimido en una cámara y moviendo un pistón. Son motores simples y baratos pero, debido a la compresibilidad del aire, no pueden ser controlados con precisión, por lo que normalmente sólo operan con dos posiciones. Los motores hidráulicos funcionan de manera similar a los neumáticos pero sustituyendo el aire por aceite, por lo que pueden generar movimientos más precisos y de mayor potencia. Como principales desventajas destacar que son más pesados, sucios y caros.

Los motores eléctricos son los más simples y fáciles de operar, producen torques moderados y se pueden controlar con gran precisión. A su vez, se pueden dividir en motores de corriente continua, motores paso a paso y servomotores. Los primeros son los más sencillos, ya que funcionan simplemente aplicando corriente eléctrica. El sentido del giro depende del signo de la corriente (son motores reversibles), mientras que la velocidad es proporcional a la intensidad y/o voltaje aplicados.

Los motores eléctricos paso a paso son aquellos que tienen su bobina dividida en secciones. En ellos el sentido de giro depende del orden en el que se activa cada sección del bobinado y la velocidad depende de la frecuencia y de la secuencia de activación de cada sección. Este tipo de motores son los más precisos. Los servomotores son motores de corriente continua con un potenciómetro y todos los engranajes integrados. Gracias al potenciómetro se puede medir y mantener con precisión un ángulo de giro del motor y su velocidad. Suelen ser motores con torques y corrientes de funcionamiento pequeñas, pero muy comunes en ciertas aplicaciones, como el modelismo.

1.3.3 Computadores

Los módulos de computación de un robot móvil constituyen su sistema de control y son los encargados de procesar toda la información generada por los distintos sensores y de tomar, en cada momento, las decisiones oportunas para que el robot realice las tareas que le han sido encomendadas. El número y tipo de los computadores, junto con su organización, marcan muchas de las limitaciones de un robot móvil, entre otras la potencia de cálculo, la flexibilidad, la modularidad, el tiempo de reacción, la robustez, la autonomía, etc.

El incremento de la potencia de cálculo de los computadores y la disminución del consumo energético han posibilitado que los robots actuales puedan realizar una mayor diversidad de tareas y que éstas sean de mayor interés que hace sólo unos pocos años. Además, la constante disminución de costes permite una utilización cada vez más amplia de sistemas más y más potentes. Estas tendencias se han mantenido, e incluso acelerado, en los últimos años.

Al igual que cualquier otro sistema de control, el control de un robot móvil se puede realizar mediante técnicas analógicas. Sin embargo, no se suele utilizar esta posibilidad porque es muy difícil diseñar sistemas de computación analógicos, son muy costosos y resultan muy poco flexibles. Lo más habitual por tanto es aplicar técnicas de control digital. Dentro de esta modalidad se pueden diseñar y construir circuitos y tarjetas específicas, aplicar microcontroladores y tarjetas de control comerciales, utilizar computadores de propósito general (ordenadores portátiles o de sobremesa) o una mezcla de todas ellas.

Un **microcontrolador** se puede definir como un ordenador reducido al tamaño de un simple circuito integrado, en donde, además de los elementos propios de un microprocesador (registros, unidad aritmético-lógica, unidad de control, etc.), encontramos componentes específicos para interaccionar con el entorno: líneas para controlar motores (p.e., PWM), conversores (analógico-digitales y digitales-analógicos), etc.

Gracias a esta enorme integración el control basado en microcontroladores utiliza muy pocos componentes externos adicionales, por lo que requiere poco espacio y consumo, al tiempo que permite una potencia de cálculo razonable. Por lo tanto, son capaces de realizar tareas sencillas de una forma muy eficaz, a bajo coste y en tiempo real. La mayoría de los robots móviles pequeños están basados en esta filosofía: *Didabot*, *Kephera*, *Rug Warrior* (Jones y Flynn, 1993), etc.

Cada vez existen herramientas más completas y sencillas para programar mi-

crocontroladores, sin embargo, aún se mantienen algunas desventajas importantes para su utilización en robótica móvil. Por ejemplo, microcontroladores de familias o líneas de producción diferentes suelen ser incompatibles entre sí, por lo que las aplicaciones desarrolladas para cada familia deben ser específicas y, por tanto, son poco portables. Por otra parte, los microcontroladores disponen de bastante menos memoria que un ordenador de propósito general, aunque suele ser suficiente para las aplicaciones más habituales.

El control de un robot móvil también se puede realizar mediante **computadores de propósito general** utilizando las interfaces y las tarjetas de adquisición de datos y de control necesarias. Las ventajas de este modelo es que se utiliza hardware y software ampliamente difundido, por lo que están disponibles una inmensa cantidad de herramientas y aplicaciones y a un precio muy razonable.[3] Realmente, para controlar un robot móvil se puede utilizar cualquier configuración de sobremesa, aunque teniendo en cuenta las lógicas limitaciones de espacio y consumo, y con la salvedad de que el control en tiempo real requiere de software específico, principalmente de sistemas operativos.

La configuración más habitual es la integración de los computadores de propósito general en un *rack* que proporciona la energía y las capacidades de comunicación necesarias para interaccionar con las tarjetas. Los *racks* más utilizados son los basados en los buses estándar: VME, PC104+, ISA, PCI, etc, por lo que el sistema se puede ampliar fácilmente y a un coste razonable según las necesidades que puedan surgir.

Ejemplos de esta estrategia son la familia de robots *Robuter* (Rob, 1994) y el humanoide *Isamu* (Kagami y col., 2001). El primero es un sistema basado en VME con el sistema operativo *Albatros*, un sistema operativo propietario desarrollado por Robosoft y que consiste en una variante del sistema operativo *Unix* con características de tiempo real. El segundo está basado en una plataforma Intel compatible y funciona con el sistema operativo *RT-Linux*, una variante del popular sistema operativo *Linux* pero también con características de tiempo real.

Un tercer tipo de control digital surge de integrar microcontroladores y computadores de propósito general en un mismo sistema con el objetivo de conjugar las ventajas de cada modelo de control. La idea es que los microprocesadores se encarguen de controlar a bajo nivel y en tiempo real todos los efectores y sensores del robot mientras que los computadores de propósito general se centran en las tareas

[3] Incluso con coste cero si se utiliza software libre GNU.

de más alto nivel. Con esta organización los computadores de propósito general se liberan de las tareas más repetitivas y engorrosas y ven rebajados sus requisitos para operar en tiempo real, es decir, pueden operar en 'tiempo real blando', lo que simplifica muy considerablemente su programación. La mayoría de los robots de mediana y alta complejidad utilizan este modelo de control, por ejemplo, el *Nomad 200* (Nom, 1993), el *B21* (B21, 1995) y el *Pioneer2* (Pio, 2000).

Por último, resaltar que las lógicas limitaciones de un robot móvil en cuanto a peso, espacio, y energía contribuyen a que cada vez sea más habitual distribuir el sistema de control de un robot entre varios computadores conectados en red y de los cuales sólo unos pocos están físicamente en el robot. El objetivo es conseguir mayor potencia de cálculo sin perder en ningún momento las características y la capacidad de operar en tiempo real.[4] Sin embargo, la distribución del control de un robot móvil entre varios computadores no es un problema sencillo y de fácil solución sino que exige técnicas especiales y que, hasta ahora, han sido muy poco estudiadas.

Con las tecnologías actuales las comunicaciones son cada vez más fiables, con mayor ancho de banda y de mayor alcance. Las comunicaciones se pueden establecer vía módem o, más recientemente, vía radioethernet, formando una red local clásica, aunque con un menor ancho de banda (típicamente entre 1 y 11 Mbs, según la tecnología y el protocolo utilizado).[5] A pesar de todas las mejoras realizadas, el cuello de botella de los sistemas de computación distribuida siguen siendo las comunicaciones. En el caso del control de robots móviles existe además el agravante de las zonas del entorno que puedan quedar fuera de cobertura.

1.4 Aplicaciones y usos de la robótica móvil

La habilidad de los robots móviles para desplazarse autónomamente y de forma robusta en su entorno determina en gran medida el tipo de aplicaciones posibles para tales robots. En general, son todas aquellas que conllevan transporte y suelen estar relacionadas con tareas de exploración, de inspección, de supervivencia, etc. En general, las aplicaciones de los robots móviles se pueden dividir en cuatro grandes áreas o grupos.

En primer lugar, los robots móviles se utilizan para realizar **tareas peligrosas**

[4] La parte crítica del control siempre debe realizarse en el propio robot móvil para evitar dificultades graves ante hipotéticos problemas en la red de computadores.

[5] Actualmente existen nuevos protocolos y dispositivos capaces de operar a 51 Mbs.

o que se realizan en entornos inaccesibles u hostiles al ser humano. Ejemplos de este tipo de tareas son la manipulación de explosivos y la desactivación de minas (p.e., robots *Pemex* y *Ariel*); el trabajo en centrales nucleares, astilleros, explotaciones mineras, submarinas y similares; la inspección del interior de conductos y canalizaciones; la exploración del fondo marino, volcanes (p.e., *Dante* (Bares y Wettergreen, 1999) y *Robovolc*), planetas (p.e., *Ambler* (Bares et al, 1989), *Sojourner* y *NOMAD*), cuevas subacuáticas, etc.

Un segundo tipo de aplicaciones muy importante es la realización de **tareas repetitivas y engorrosas**. Por ejemplo, la limpieza de grandes naves y superficies (p.e., la familia *AutoVacC* de la compañía Robosoft), los servicios de almacenaje y el reparto de mercancías, el repostaje automático de combustible (el sistema *OSCAR* de Robosoft), las ordeñadoras automáticas, etc.

El tercer grupo lo componen las **tareas de servicio**, por ahora muy poco explotadas pero con un futuro muy prometedor. Entre otros, podemos destacar los servicios de entretenimiento (p.e., juguetes como *Aibo* (Aib, 1997)), la ayuda a discapacitados (p.e., ciegos (Ulrich, 1997) o en silla de ruedas), las visitas guiadas (p.e., en museos (Burgard y col., 1998; Thrun y col., 1999)), el transporte de pasajeros (p.e., *CyCab* de Robosoft), los servicios de búsqueda y rescate (sobre todo en entornos urbanos), etc.

Por último, y no la menos importante, está la aplicación de la robótica móvil en **tareas de investigación** en múltiples campos científicos: inteligencia artificial, ciencia cognitiva, psicología, etología (Webb, 1995), etc. En todos ellos los robots móviles suelen utilizarse como plataforma de experimentación y validación (*testbed*) o refutación de las distintas hipótesis de trabajo. Por ejemplo, sobre el comportamiento inteligente (Pfeifer y Scheier, 2000), el comportamiento social (Breazeal, 2000), el estudio de las emociones, los procesos de percepción, la navegación de los animales (*Sahabot 2* (Sah, 1997)), estudio de robots antropomórficos (Adams y col., 2000) (*Isamu* (Kagami y col., 2001), *Cog* y *Kismet*).

La ventaja de experimentar los modelos sobre robots móviles es que sus programas de control pueden ser analizados con detalle. Por otra parte, las condiciones y los procedimientos experimentales pueden ser controlados cuidadosamente, lo que permite la replicación y la verificación independiente de los experimentos. Asimismo, es posible modificar y ajustar los parámetros del modelo de manera individual, ya que los experimentos se pueden repetir las veces que sean necesarias.

Dentro de este tipo de aplicaciones también resulta muy habitual utilizar los ro-

bots móviles para realizar tareas complejas en entornos reales con el objeto de aplicar y demostrar el funcionamiento de nuevas técnicas y metodologías en el campo de la inteligencia artificial (Murphy, 2000), entre otras, el aprendizaje, la planificación, sistema de representación, algoritmos de búsqueda, visión artificial, comprensión del lenguaje natural, etc.

1.5 Problemas de la robótica móvil

El principal problema con el que tienen que tratar los robots móviles es que operan en entornos reales sobre los cuales tienen muy poca o nula influencia, ya que poseen su propia dinámica independiente, muchas veces inaccesible o desconocida para el robot. Por todo ello, el entorno de robot móvil se torna complejo, cambiante y, muchas veces, impredecible.

Por otra parte, la información que percibe el robot a través de sus sensores es, en la mayoría de las ocasiones, bastante imprecisa, a menudo ambigua y casi siempre incompleta. Además, existen muy pocos sensores capaces de proporcionar al robot suficiente información acerca de todo lo que le rodea. Así, la gran mayoría de los sensores actuales (US, IR, láser) limitan su campo de detección a un plano paralelo al suelo.

Los pocos sensores que detectan en las tres dimensiones (cámaras de vídeo y láser que giran con dos grados de libertad) generan tantos datos que, en la práctica, es imposible procesarlos adecuadamente en tiempo real con los sistemas y técnicas de computación actuales. El resultado final de todas estas limitaciones es que el conocimiento que posee un robot móvil sobre su entorno es incompleto, incierto y aproximado. Por lo tanto, las decisiones que toma en cada momento pueden no ser las más acertadas, es decir, las óptimas.

Para complicar las cosas, las acciones que realiza el robot tampoco son del todo fiables y no se corresponden completamente con los comandos enviados a los efectores. Por ejemplo, cuando el robot realiza un desplazamiento los sensores propioceptivos (fundamentalmente codificadores ópticos) suelen cometer errores con frecuencia no despreciables. Más aún, el robot puede "moverse" de una manera no esperada o imprevisible. Por ejemplo, si choca violentamente o las ruedas patinan el robot se desplaza de una manera que no suele ser capaz de percibir. El resultado final es que el robot no siempre puede conocer completamente y con precisión los resultados y los efectos de todas las acciones que realiza.

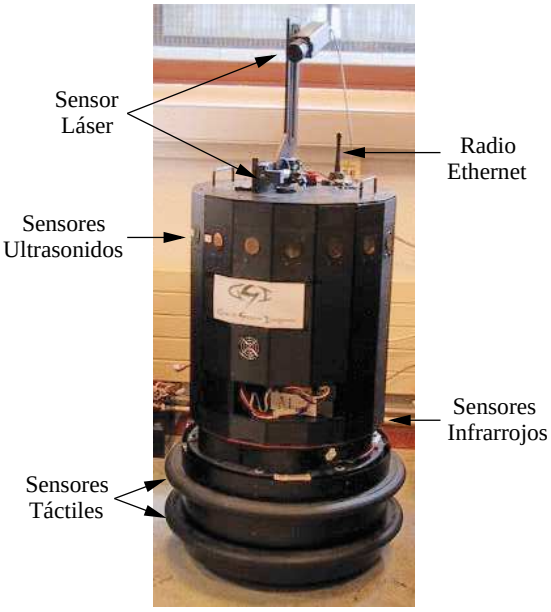

Figura 1.13: Robot móvil *Nomad 200*.

1.6 Robot móvil *Nomad 200*

El *Nomad 200* (Nom, 1993) (figura 1.13) es un robot móvil fabricado por la empresa americana Nomadic Technologies. Mide 116 cm de altura y está formado por una base y una torreta cilíndricas de 53 y 46 cm de diámetro, respectivamente. Es totalmente autosuficiente, ya que porta tanto su sistema de control como su fuente de energía, un conjunto de cinco baterías de plomo selladas de 850 Wh y que le proporcionan una autonomía de funcionamiento de entre 3 y 4 horas.

El *Nomad 200* es un robot modular que puede montar distintos tipos de sensores, cada uno de ellos controlado por su propio microcontrolador, en distintos "pisos" de la torreta. Un parachoques en la base hace de sensor táctil con una precisión de 18º (20 sensores que circundan la base del robot). La torreta no tiene una forma circular sino que está formada por 16 lados, en cada uno de cuales se monta un sensor de ultrasonidos (US) y otro de infrarrojos (IR). Los 16 sensores de cada tipo están equiespaciados 22,5º entre si y forman un plano paralelo al suelo.

El robot también viene equipado con un sensor láser (Nom, 1995) que funciona por triangulación y que consta de un emisor láser, una cámara y una tarjeta de adquisición y procesamiento de la información. Además, cada una de las ruedas tiene un codificador óptico acoplado que se utiliza para calcular la posición odométrica del robot mientras se desplaza.

En el *Nomad 200* los distintos tipos de sensores están dispuestos a una altura diferente respecto al suelo: los IR están a 36 cm, los US a 78 cm y el láser a 91 cm. El alcance de los IR es de 38 cm, mientras que el de los US está entre 15 cm y 6,5 m. Por su parte, el láser sólo puede detectar objetos que se encuentren a una distancia de entre 0,5 y 2,5 m y a menos de 15º del eje de la cámara del sensor.

Como efectores el *Nomad 200* posee tres ruedas dispuestas en una configuración síncrona, de tal forma que todas siempre apuntan a la misma dirección. El robot puede realizar giros con un radio cero, es decir, girar sobre su eje. Con esta configuración la base permanece fija aunque las ruedas giren. Por su parte, la torreta también puede girar de forma independiente respecto a la base. Por lo tanto, el *Nomad 200* tiene cuatro grados de libertad (x, y, θ_{base}, $\theta_{torreta}$). El robot se desplaza a un máximo de 51 cm/s, mientras que la base y la torreta giran a un máximo de 45º/s. Además de las ruedas, los efectores del *Nomad 200* son un altavoz y un sintetizador de voz con los cuales puede emitir sonidos y unas pocas palabras.

El sistema de control del *Nomad 200* está basado en un computador de propósito general tipo PC estándar que realiza todas las tareas de alto nivel y de una serie de microcontroladores y tarjetas de adquisición y de control que se encargan de operar, a bajo nivel, los distintos sensores y efectores del robot. El sistema es un multiprocesador de memoria compartida basado en una memoria RAM de doble puerto, sobre la cual pueden escribir tanto el computador principal como los distintos microcontroladores. Con esta técnica se disminuye muy considerablemente la carga computacional del computador principal y se facilita y agiliza el intercambio de información entre los distintos dispositivos del sistema de control.

Para poder acceder a una mayor capacidad computacional y mejorar la interacción con los usuarios el *Nomad 200* viene equipado con una radioethernet RangeLAN2 de Proxim (PRO, 1993). Con ella el robot se puede comunicar tanto con los computadores de una red ethernet estándar, a través de un punto de acceso, como con otros ordenadores que estén equipados con idénticos dispositivos (p.e., un portátil). Utilizando varios repetidores se aumenta la cobertura del enlace vía radio. El ancho de banda de la radioethernet llega hasta los 1.6 Mbs, es decir, entre uno o dos órdenes de magnitud menor que una red ethernet estándar actual.

1.6.1 Entorno de desarrollo y simulador del *Nomad 200*

Las ventajas de un simulador son evidentes (Nehmzow, 2000): velocidad y simplicidad de ejecución, bajo coste, repetibilidad y disponibilidad. Por otro lado,

facilita realizar múltiples pruebas y siempre bajo unas condiciones de experimentación muy controladas. También permite realizar todo tipo de modificaciones en los parámetros y contrastar y medir su influencia en el comportamiento final del robot. Además, se pueden experimentar todo tipo de cambios y parámetros e, incluso, en escenarios difíciles de conseguir, poco probables o que pueden suponer un peligro para la integridad física del robot o la de su entorno, se pueden realizar predicciones, etc. Por último, también facilita el entrenamiento y el aprendizaje.

Sin embargo, la simulación posee unas fuertes limitaciones y ciertos peligros, sobre todo si se utiliza indebidamente o si la extrapolación de los resultados obtenidos es excesiva o inadecuada. La principal limitación es que una simulación siempre se realiza sobre un *modelo* del entorno real en donde va a operar el robot. Sin embargo, la gran mayoría de dichos entornos son complejos, dinámicos y, hasta cierto punto, inmodelables. Por lo tanto, es imprescindible realizar una serie de simplificaciones, a veces nada despreciables. Si dichas simplificaciones no se hacen apropiadamente los resultados obtenidos con la simulación tendrán una dudosa utilidad.

Otra limitación de las simulaciones son los cálculos que se realizan sobre el modelo elegido. Lo usual es hacerlos de forma digital mediante aproximaciones numéricas, lo que conlleva errores tanto de aproximación como de redondeo. Otro problema es que en simulación pueden aparecer dificultades que no se dan en el mundo real (Nehmzow, 2000). Por ejemplo, que dos robots lleguen a una intersección exactamente al mismo tiempo. Otro riesgo aparece cuando el diseñador de la simulación es el mismo que diseña la aplicación del robot. En ese caso, las suposiciones erróneas cometidas durante el diseño del simulador no serán advertidas durante su aplicación.

Todos estos errores y limitaciones han llevado a algunos autores (Connell, 1990; Brooks, 1991b) a propugnar la completa eliminación de toda simulación y abogar por utilizar el mundo real como su propio modelo, es decir, realizar las pruebas únicamente en el entorno real. Actualmente la postura de la mayoría de los investigadores es menos radical y se considera que la simulación es interesante, bien como una primera aproximación, bien para evitar pruebas que entrañen un cierto riesgo para el robot o en condiciones que sean difíciles de conseguir y controlar. De todos modos, se sigue considerando que la única validación finalmente admisible es la que se realiza sobre el entorno real en donde va a operar el robot y ejecutando la tarea asignada.

Otra de las características más sobresalientes del *Nomad 200* es que, además de un simulador, viene acompañado de un entorno de desarrollo de software. Ambos

sistemas están integrados, lo que facilita enormemente el diseño, implementación y validación del software de control del robot y permite un ciclo de desarrollo mucho más corto. De hecho, un mismo programa de control puede operar en tres modos distintos. En el primero el programa puede conectarse directamente al robot real. En los otros dos modos, el programa se conecta a la interfaz gráfica del entorno de desarrollo que, a su vez, se puede después conectar al robot real o a un simulador del *Nomad 200*. Un mismo programa o aplicación se puede conectar de cualquiera de los tres modos sin necesidad de modificar ni una sola línea de código.

El simulador del *Nomad 200* es de tipo numérico y es capaz de modelar los principales parámetros que gobiernan sus acciones y percepciones sensoriales aunque, en general, utilizando modelos bastante simples (Nehmzow, 2000). Así, por ejemplo, se modela el pulso que emite un sensor de US mediante un segmento circular de 22,5 grados. Por otra parte, se supone que el ángulo de incidencia bajo el cual se genera el fenómeno de las reflexiones especulares es constante y, por lo tanto, que todos los objetos del entorno simulado poseen la misma textura superficial (lo que es totalmente falso). Algo similar ocurre con el color, ya que en el simulador se asume que todos los objetos tienen el mismo color y, por lo tanto, reflejan la luz infrarroja de igual manera.

Otra fuerte simplificación es utilizar un modelo lineal de la dinámica del robot, por lo que no se tienen en cuenta fuerzas, inercias ni retardos. El simulador tampoco tiene en cuenta las posibles irregularidades del terreno ni la posibilidad de que ocurran deslizamientos o derrapes de las ruedas. Por último, señalar que el simulador se limita a modelar entornos en dos dimensiones por lo que, entre otros efectos, presupone que todos los objetos del entorno pueden ser detectados por todos los sensores.

La interfaz gráfica del *Nomad 200* se denomina Nserver y es capaz de "soportar" varios robots simultáneamente (en la figura 1.13 se pueden ver dos). Consta de una ventana en la que se muestran los principales objetos del entorno y la posición de los distintos robots conectados. A su vez, cada robot posee su propia interfaz que consta de un mínimo de tres ventanas. En la principal se pueden ver de nuevo el entorno inmediato del robot y su posición en el sistema global de referencia. En esta ventana también se pueden visualizar las medidas sensoriales del robot y una serie de primitivas geométricas (segmentos y arcos). En las otras dos ventanas se representan los últimos datos sensoriales detectados, agrupados en sensores de corto alcance (táctiles e IR) y de largo alcance (US y láser). Además, cada robot puede ser controlado de forma remota a través de un *joystick* virtual o enviando

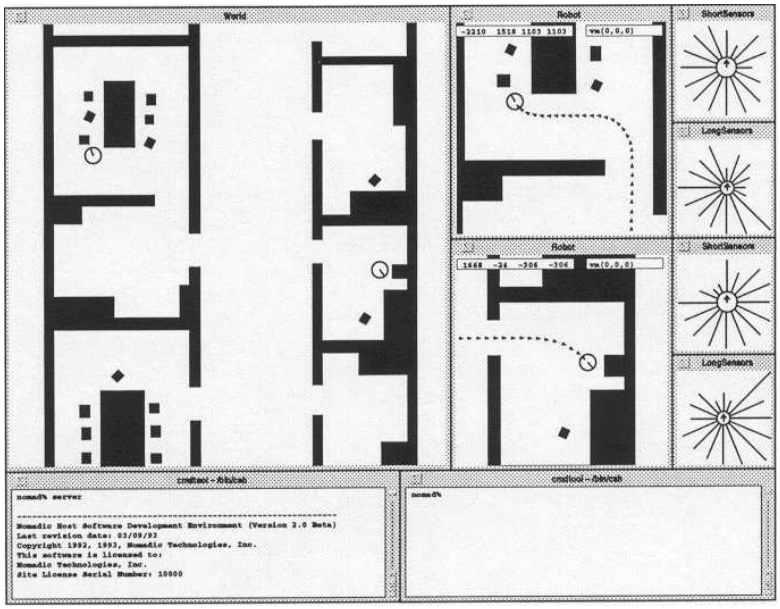

Figura 1.14: Interfaz gráfica del entorno de desarrollo del *Nomad 200*.

directamente los comandos básicos. Por último, la interfaz de cada robot puede almacenar y reproducir tanto los comandos enviados como los datos sensoriales percibidos y salvarlos en un fichero.

Capítulo 2

Aspectos funcionales de un RMA

En el capítulo anterior se han estudiado los distintos elementos que componen un robot móvil y se han visto algunas de sus principales aplicaciones. En el capítulo actual se explica el concepto de robot móvil autónomo (RMA) y se indica cuáles son algunas de sus principales funcionalidades y cómo se pueden implementar. En base a dicha definición, se marcan los dos objetivos que debe poseer un RMA y cómo se pueden clasificar algunas de las tareas necesarias para alcanzarlos. El objetivo no es tanto realizar un estudio exhaustivo sobre el tema, sino realizar una introducción al mismo. Por último, se fija con mayor detalle la meta del presente trabajo y se comentan las simplificaciones adoptadas respecto al planteamiento inicial.

2.1 Robot móvil autónomo

La principal característica de un robot móvil autónomo (RMA) es, por supuesto, su *movilidad*, es decir, su capacidad para desplazarse por un entorno por sus propios medios. La plataforma móvil de un robot depende del entorno y del tipo de desplazamiento que se necesite. En el capítulo anterior se han mencionado algunos ejemplos.

Los primeros robots móviles tenían restringida su movilidad, ya que sólo eran capaces de seguir los caminos marcados en su entorno, normalmente sobre o bajo el suelo. Como es evidente, este método implica modificar el entorno del robot, con todos los problemas que ello conlleva. Estas primeras implementaciones tienen serios problemas si el camino aparece cortado o si algún objeto lo bloquea.

Sin embargo, el entorno usual en el que se desenvuelve un robot móvil es comple-

jo, dinámico y, en bastantes ocasiones, impredecible. Además, el robot interacciona con su entorno a través de sensores y efectores no siempre fiables y, a menudo, con un alto grado de incertidumbre. Por lo tanto, no es posible definir *a priori* todos los problemas o contingencias que pueden surgir durante la ejecución de una determinada tarea. Es imprescindible dotar al robot de un nivel suficiente de *autonomía* en su operación, lo que en el capítulo anterior hemos denominado "autonomía fuerte", de manera que la ejecución de tareas sea más robusta y capaz de resolver pequeños contratiempos.

Para que un sistema sea autónomo debe estar integrado y ubicado en su entorno (Pfeifer, 1996; Pfeifer y Scheier, 2000). Por definición, un robot está integrado (*embodied*) en la medida en que es un sistema físico capaz de actuar en su entorno y está sujeto a sus leyes (por ejemplo, alguien puede mover o desplazar el robot, o éste puede deslizarse por una cuesta). Que el robot esté ubicado (*situated*) en su entorno significa que toda la interacción con el entorno siempre debe ser vista y entendida desde la perspectiva del robot, es decir, a través de sus propios sensores y actuadores.

El concepto de autonomía surge también en sistemas robóticos teleoperados, es decir, robots controlados a distancia por un operador humano, normalmente sin tener una visión directa del robot. En estos casos se ha demostrado que el operador no puede mantenerse concentrado mucho tiempo, sobre todo en tareas repetitivas (Murphy, 2000). Las principales causas son los retardos en las comunicaciones, la escasa y limitada información sensorial que proporciona el robot (fatiga cognitiva) y la discordancia entre el movimiento que se percibe a través del robot y la situación estática del propio operador.

Una forma de mitigar estos problemas consiste en utilizar sistemas semi-autónomos capaces de resolver por sí mismos ciertas tareas de bajo nivel. Existen dos tendencias: control compartido (*shared control*) y control delegado (*trading control*). En los primeros, el operador monitoriza el robot constantemente mientras éste ejecuta las tareas asignadas, retomando el control cuando lo considera conveniente. Sería el ejemplo clásico del control de un brazo articulado que necesitase realizar manipulaciones complejas.

En el control delegado, el operador sólo interacciona con el robot en el momento de enviar las órdenes, de cuya ejecución puede desentenderse completamente. Casos típicos serían aquellos donde las comunicaciones poseen grandes retardos (p.e., exploración espacial, control a través de Internet) o donde se intenta reducirlas al

máximo (p.e., misiones militares).

Autonomía significa capacidad para operar durante largos periodos de tiempo sin necesidad de ayuda externa. Esta cualidad conlleva la adaptación a los cambios que se produzcan en el entorno, en el propio robot y una actuación aceptable ante situaciones no previstas. Para conseguir tales objetivos, es conveniente incorporar un cierto grado de "*inteligencia*" a los robots. Es decir, capacidades de deliberación, planificación y aprendizaje que les permitan una mejor adaptación a su entorno y a la tarea que deben desempeñar.

El objetivo del presente trabajo es conseguir un robot móvil autónomo con la *inteligencia básica* necesaria para poder abordar un amplio espectro de tareas, sin ceñirse, por el momento, a ninguna aplicación específica. El reto es conseguir un RMA de *propósito general*, sacrificando la eficiencia y la simplicidad en aras de una mayor generalidad y flexibilidad. A partir de esta funcionalidad básica el robot podrá adaptarse, de forma relativamente sencilla, a la ejecución de distintos cometidos prácticos.

2.2 Funciones de un RMA

Las funciones que puede implementar un robot son muchas y diversas, pero las que necesita un RMA concreto dependen tanto del tipo de tareas que debe realizar como del entorno o "nicho ecológico" en dónde debe desenvolverse. Ambos aspectos, tareas y entorno, condicionan los componentes que debe incorporar el robot, es decir, el número, tipo y disposición de los sensores y efectores, la capacidad computacional, los elementos de interacción, etc. Al mismo tiempo, los componentes de un RMA, su soporte físico, condicionan las funciones que éste puede realizar y, sobre todo, la manera de desarrollarlas.

2.2.1 Tipos de funciones

Hay un consenso generalizado en robótica de que las funciones de un RMA se pueden clasificar en tres amplias categorías: percepción, control y actuación, también conocidas como percibir, pensar y actuar. A estas categorías se las suele denominar *primitivas* (Murphy, 2000). En la tabla 2.1 se las define en base a sus entradas y salidas.

Primitiva	Entradas	Salidas
Percepción	Datos sensoriales	Información percibida
Control	Información (percibida y/o cognitiva)	Directivas
Actuación	Información percibida o directivas	Comandos de actuación

Tabla 2.1: Primitivas de un robot definidas en términos de sus entradas y salidas (adaptado de (Murphy, 2000)).

Funciones de percepción

Las funciones de percepción son todas aquellas que recogen la información de los sensores y la traducen en datos que puedan ser utilizados por otras funciones. Como es fácil de imaginar, su operación es vital para cualquier sistema que necesite operar en entornos dinámicos o impredecibles, siempre cambiantes o difíciles de modelar. En general, en todos aquellos casos donde no sea posible predecir o anticipar con total exactitud la interacción entre el robot y el entorno. En esas situaciones es imprescindible que el robot recoja información de su entorno y perciba los detalles que le son de interés para su correcto funcionamiento y para la ejecución de sus tareas.

Las funciones de percepción son muy variadas y van desde la simple detección de un simple patrón sensorial (p.e., una medida muy pequeña de un conjunto de sensores de ultrasonidos), a la construcción y el mantenimiento de un modelo completo del entorno (p.e., mapas CAD en tres dimensiones), pasando por el reconocimiento y/o modelado de un objeto (p.e., mediante visión).

Sin embargo, una característica común en todas ellas es que siempre es necesario establecer algún tipo de hipótesis en el procesamiento de la información, ya que los sensores normalmente no permiten la obtención directa de justo las variables del entorno que nos interesan ni hacerlo con absoluta precisión. Así, por ejemplo, la medida del tiempo que tarda en volver el eco de un pulso de un sonar se interpreta como la presencia de un obstáculo a una determinada distancia. O también, que la falta de una medida en un láser se traduce en espacio libre. Por lo tanto, siempre es necesario algún tipo de abstracción y procesamiento en relación a la información aportada directamente por los sensores.

Funciones de actuación

Las funciones que producen comandos de salida para los efectores son las que se consideran funciones de actuación. Son, por ejemplo, las encargadas de generar los comandos necesarios para que un robot móvil pueda evitar un obstáculo o seguir un contorno, o para que la pinza de un brazo articulado pueda agarrar un objeto.

También se incluyen aquellas que mantienen el equilibrio del robot o las que ordenan los movimientos necesarios para lograr una postura corporal, una expresión "facial" o cualquier otro tipo de comunicación.

Funciones de control

Las funciones de control son todas aquellas que recogen información sobre su entorno, tanto percibida como introducida de forma previa, y producen una o más directivas que debe ejecutar el robot. Para simplificar, se puede decir que son todas aquellas tareas que no son ni de percepción ni de actuación. Existen tres tipos principales de funciones de control: reacción, deliberación y planificación.

El control reactivo toma decisiones considerando únicamente la información actual percibida a través de los sensores (generalmente, mediante un procesamiento muy simple). Este tipo de control se relaciona con tareas críticas, que poseen un tiempo de ejecución muy pequeño. Ejemplos de control reactivo son evitar un objeto, mantener el equilibrio (p.e., en un robot con dos patas) o seguir un objeto con la vista.

El control planificado utiliza toda la información posible: la actual, la pasada e, incluso, trata de predecir la futura. Su función consiste en decidir qué tareas se deben ejecutar en el futuro y en qué orden. Ejemplos de este tipo de control son la planificación de rutas para un robot móvil, la planificación de movimientos para un brazo articulado o la predicción del comportamiento de un colaborador o un competidor.

El control de tipo deliberativo toma sus decisiones utilizando no sólo la información actual, sino también la pasada. Es decir, utiliza la experiencia y los conocimientos adquiridos durante la interacción con su entorno. Se caracteriza por poseer memoria y capacidad de adaptación a los cambios en el entorno, recordando y evitando anteriores decisiones erróneas. Ejemplos de este tipo de control sería evitar objetos en movimiento, salir de un callejón sin salida, construir un mapa y cualquier tarea que implique aprendizaje.

2.2.2 Robots e inteligencia artificial

Para implementar cada una de las funciones de un RMA se aplican múltiples y diferentes técnicas. Sin embargo, la aplicación de la Inteligencia Artificial (IA) es de las más prometedoras (algunos autores consideran que es el único camino posible para lograr robots inteligentes) lo que da lugar a lo que se denomina "robots IA" (Murphy, 2000). Entre las diferentes líneas de investigación sobre IA que se pueden aplicar a la robótica (Angulo Usátegui y col., 1999) estarían:

1. Sistemas expertos: aquellos destinados a simular procesos intelectuales propios de expertos humanos y que, en general, requieren una capacidad deductiva.

2. Lenguaje natural: trata sobre el problema complejo de la comprensión del lenguaje, la síntesis y análisis de la voz, y el resumen, entre otros.

3. Visión artificial: se dedica a la identificación, localización y reconocimiento de objetos por medio de imágenes captadas por cámaras.

4. Aprendizaje: o el desarrollo de métodos para la adquisición de nuevo conocimiento a través de la percepción, la experiencia y los datos previos.

5. Programación automática: estudia la generación automática de programas para la resolución de un problema a partir de unas especificaciones dadas.

A esta lista habría que añadir muchas otras líneas de investigación, entre otras la lógica borrosa (Saffiotti y col., 1995; Mucientes y col., 1999), las redes neuronales (Iglesias y col., 1997) y las técnicas de la inteligencia artificial distribuida (IAD) (Vivancos y col., 1998).

Existen autores (Murphy, 2000) que defienden que, más que aplicar la IA a la robótica, lo que realmente ha ocurrido es que la robótica ha proporcionado el impulso necesario para que las distintas técnicas de la IA hayan madurado y avanzado.

Para estos autores, por ejemplo, los ingenios de búsqueda en Internet y los "agentes software" son programas autónomos que interactúan y se adaptan a su entorno igual que un animal o un robot inteligente. De hecho, constatan que el nombre que reciben, *web-bots* y *softbots*, denotan su origen en las técnicas desarrolladas para la robótica.

Sin llegar a tales extremos, sí se puede afirmar que en la robótica pueden y deben confluir muchos de los esfuerzos dedicados a la IA. La robótica se convertiría así en

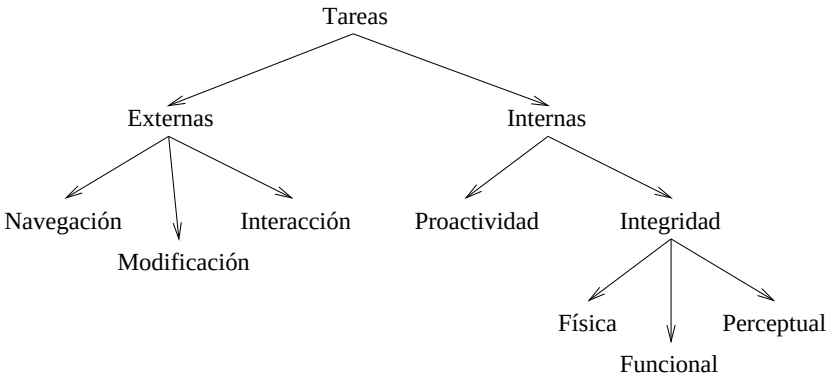

Figura 2.1: Posible organización de algunas de las tareas más importantes de un robot móvil autónomo.

un magnífico campo de pruebas de las teorías y métodos desarrollados en IA. En realidad, se considera que la IA se ha equivocado al despreciar históricamente la implementación de sistemas que tuvieran que desenvolverse autónomamente en un mundo real o virtual. Al perder la visión de conjunto, el resultado ha consistido en una progresiva subdivisión en campos más independientes entre sí e imposibles de aunar en la práctica.

2.3 Objetivos de un RMA

La principal razón para que existan RMAs es su aplicación a alguna tarea de interés. Bajo ese punto de vista, su principal cometido es ejecutar las tareas que le son "propuestas", generalmente por un usuario. No hay que olvidar que el sistema debe ser útil y responder a ciertas necesidades que justifiquen su construcción y puesta en servicio. Es lo que podemos denominar *objetivos externos* o tareas externas (figura 2.1).

Sin embargo, es imposible programar un robot para que pueda solucionar cualquier tipo de contingencia que se pueda presentar mientras ejecuta una tarea. Por lo tanto, es imprescindible dotar al robot con los mecanismos necesarios para que pueda reaccionar adecuadamente ante las dificultades que surjan durante su normal funcionamiento. Para conseguirlo se dota al robot de cierto nivel de autonomía. Cuando eso ocurre se puede decir que el robot trata de alcanzar una serie de *objetivos internos*.

Las tareas internas (figura 2.1) se pueden dividir a su vez en tareas de superviven-

cia y tareas proactivas. Las primeras son las que tratan de mantener la integridad del robot en diversos aspectos o bajo distintos puntos de vista. Las últimas posibilitan que el robot tenga iniciativa propia.

En definitiva, existen dos puntos de vista a la hora de considerar o valorar la actividad de un RMA. El primero consiste en verlo como una máquina flexible y robusta capaz de ejecutar tareas de una manera autónoma e inteligente. Bajo esta perspectiva los objetivos internos son una herramienta más que ayuda a la consecución de las tareas pedidas, y suelen permanecer semiocultos durante el funcionamiento normal del RMA. Esta visión de corte *ingenieríl* es la más común.

El segundo punto de vista considera un RMA como un sistema autónomo donde lo importante son las motivaciones que le guían en una correcta interacción con el entorno. Bajo esta perspectiva la consecución de objetivos externos es una motivación más, la de "ser útil", aunque fundamental para competir y sobrevivir en un mundo dominado por seres humanos. Este es el original enfoque planteado por McFarland y Bösser en su libro (McFarland y Bösser, 1993).

2.3.1 Tareas de supervivencia

Las tareas de supervivencia son todas aquellas tareas internas que velan por la integridad del propio robot. Son tareas que el diseñador del robot ha incluido de manera implícita y permanente. Se pueden dividir en tres grandes categorías, según velen por la integridad física, la integridad funcional o la consistencia perceptual del robot.

Las tareas de supervivencia suelen tener la prioridad de ejecución más alta en caso de conflicto entre varias tareas. Sin embargo, en ciertos casos especiales pueden ser inhibidas. Un ejemplo de tal inhibición ocurre cuando el robot debe empujar un objeto. Para ello necesita inhibir la tarea que trata de evitar que el robot se "golpee" contra cualquier objeto de su entorno.

Tareas de integridad física

Las tareas cuyo objetivo es mantener la integridad física del robot se activan para tratar de evitar que el robot sufra algún daño. Claramente pertenecen a esta categoría las tareas de evitar choques, sea con obstáculos fijos u objetos en movimiento.

Sin embargo, y desde un punto de vista más amplio, también se incluyen aquellas

tareas que preven con anticipación un posible daño físico. Por ejemplo, mantener alejado el robot de lugares críticos en relación con su capacidad motriz (agujeros, pendientes, terreno resbaladizo, etc.), a su forma (p.e., evitar la "decapitación" del robot por una mesa) o a su estructura (p.e. evitar lugares con agua o que no resistan el peso del robot).

Tareas de integridad funcional

El principal objetivo de este tipo de tareas es mantener la funcionalidad del robot. El caso más evidente es mantener un adecuado nivel de energía en la baterías para el correcto funcionamiento del robot.

También pertenecen a esta categoría aquellas tareas que minimizan el impacto de un posible fallo en algún sensor o efector en la operación del robot, por ejemplo, reordenando las funciones de percepción y de actuación.

Las funciones que anticipan posibles o reales deficiencias funcionales también formarían parte de esta categoría. Por ejemplo, evitar perder la comunicación con la estación base.

Tareas de consistencia perceptual

Las tareas de consistencia perceptual tratan de mantener la coherencia de la información que almacena el robot, tanto del entorno que le rodea como de sí mismo. Su principal cometido es cotejar constantemente la información almacenada con la que se está percibiendo en cada momento y, en caso de incongruencia, resolver la ambigüedad resultante. Ejemplos de este tipo de tareas son verificar continuamente la posición que ocupa el robot respecto al mapa interno o detectar e identificar los objetos conocidos del entorno.

Velar por la integridad perceptual también consiste en mejorar constantemente la calidad de la información que guarda el robot y en eliminar datos redundantes. Esta tarea es crucial en un robot móvil puesto que el espacio de almacenamiento es siempre limitado.

2.3.2 Tareas proactivas

La proactividad consiste en que el RMA tenga iniciativa y tome sus propias decisiones. No se consideran tareas proactivas aquellas que estén directamente su-

bordinadas a la realización de una tarea determinada. Este tipo de tareas suelen tener un claro matiz oportunista ya que, generalmente, sólo se ejecutan cuando el robot dispone de un tiempo muerto o no existe ninguna tarea (externa) urgente. En todo caso, el coste (pérdida de tiempo y energía) debe compensar el beneficio obtenido.

El ejemplo más claro de tarea proactiva es *explorar* el entorno, es decir, conocer más y mejor el entorno en el que habita el robot. No sería tanto la exploración inicial sino la que se realiza aprovechando la proximidad con un lugar desconocido.

Otro ejemplo de proactividad sería *intercambiar información* con otros agentes del entorno cuando surge la oportunidad, por ejemplo, si se "cruzan" en el camino. Éste sería un método para que varios robots pudiesen colaborar de forma espontánea para realizar una misma tarea, por ejemplo, la construcción de un mapa del entorno. Pedir ayuda para resolver una determinada tarea (externa) no se puede considerar proactividad, sino que dicha acción debería formar parte de la ejecución de cualquier tarea (externa).

2.4 Tareas externas básicas

Independientemente de todas las particularidades mencionadas, se pueden identificar un conjunto mínimo de funciones que cualquier RMA debería ser capaz de ejecutar para operar con normalidad en un amplio número de entornos, con diferentes configuraciones y ejecutando distintos tipos de tareas. Las tareas básicas de un RMA son tareas que generalmente se utilizan como apoyo a la ejecución de otras tareas, aunque tienen una utilidad evidente por sí mismas. Se pueden identificar las siguientes tareas básicas:

1. navegación o desplazamiento por el entorno del robot,
2. modificación del entorno del robot,
3. comunicación e interacción con los "entes" que pueblen el mismo entorno del robot: otros robots o seres humanos, por ejemplo.

Cada una de las tareas básicas requiere un *soporte físico mínimo*, es decir, que para su ejecución es necesario un conjunto mínimo (tanto en número como en calidad) de efectores, sensores y capacidad computacional, de lo contrario dichas tareas no podrían ser ejecutadas, o lo serían de forma restringida. A esta cualidad algunos

autores (Pfeifer, 1996) la denominan balance ecológico. Por ejemplo, de poco sirve disponer de una cámara si no se puede extraer información útil de las imágenes adquiridas, o no es posible hacerlo de forma fiable o en tiempo real.

Como es obvio, ese soporte físico mínimo depende del entorno en el que se desenvuelva el robot (su nicho ecológico). Por ejemplo, el valor de un anillo de ultrasonidos o un láser bidimensional en un plano paralelo al suelo es escaso si dicho suelo es extremadamente irregular o existen obstáculos indetectables a la altura del plano (p.e. mesas).

Recíprocamente, los componentes de un robot también influyen decisivamente en el tipo de representación y procesamiento de la información sensorial. P.e., no es lo mismo modelar un entorno utilizando ultrasonidos, un láser bidimensional o cámaras en estéreo.

2.4.1 Navegación

Por navegación entendemos mover el robot de forma segura, eficaz y fiable por su entorno. La seguridad se traduce fundamentalmente en evitar cualquier tipo de colisión con los objetos del entorno, tanto fijos como en movimiento. Eficaz en cuanto a reducir al máximo el tiempo y la energía necesarios para ejecutar el trayecto, lo cual significa extraer el máximo partido de la información que se posee del entorno y ejecutar la tarea desde una perspectiva lo más global posible. La fiabilidad se consigue mediante la seguridad y la robustez en el funcionamiento del sistema.

Una definición ligeramente diferente es la de (Franz y Mallot, 2000), dado que consideran la navegación como el proceso de determinar y mantener un curso o trayectoria hacia una localización objetivo. Esta definición trata de recoger y explicar las observaciones sobre el comportamiento de los animales durante sus desplazamientos.

Tipos de navegación

En robótica se distingue entre dos grandes tipos de navegación según la información (mapa) que se almacena del entorno (Borenstein y col., 1996): la métrica o cuantitativa y la topológica o cualitativa. En esta clasificación excluimos la navegación guiada, típica de los vehículos automáticos, limitada a los recorridos marcados artificialmente en su entorno.

Ciertos autores (Franz y Mallot, 2000; Nehmzow, 2000) establecen una clasifica-

ción diferente basándose en los comportamientos de navegación que se observan en el mundo animal. Por último, (Lee, 1996) distingue otro tipo de mapa basado en un conjunto de localizaciones que pueden ser reconocidas por el robot, pero de las que no se puede extraer ninguna relación entre ellas. Es dudoso que este tipo de mapas puedan ser útiles en general para navegar.

Navegación topológica. Se basa en representar el entorno mediante un conjunto de marcas relevantes junto con los caminos que las unen (Kuipers y Byun, 1991b; Nourbakhsh y col., 1995; Gutierrez-Osuna y Luo, 1996). El resultado final es un grafo en el que se almacenan las relaciones topológicas o de conectividad entre los lugares de interés del entorno. Por este motivo, a la navegación topológica también se la conoce como navegación basada en marcas (Wijk y Christensen, 2000).

A menudo también se incluye información métrica en los mapas topológicos para hacer su uso más eficiente. Por ejemplo, la distancia del camino que une dos marcas medida con el sistema odométrico del robot.

Navegación métrica. Consiste en representar el entorno mediante un conjunto de entidades geométricas (p.e. líneas o polígonos) referenciadas respecto a un mismo sistema de coordenadas. En este tipo de representación se suele almacenar información sobre todos los objetos detectables en el entorno. Existen dos tipos de mapas métricos: los basados en características y los mapas de área. Los primeros almacenan una serie de características geométricas primitivas del entorno (p.e. segmentos (Leonard y Durrant-Whyte, Kluwer; Edlinger y von Puttkamer, 1994; Castellanos y Tardós, 1999)) junto con sus propiedades. Los segundos dividen todo el entorno en regiones, normalmente celdas regulares (Elfes, 1989; Borenstein y Koren, 1991a; Rodríguez y col., 2000).

Subtareas de la navegación

Como es evidente, la tarea de navegación es la que más se ha estudiado hasta el momento, puesto que es vital para el funcionamiento de cualquier RMA. Es una tarea muy compleja, ya que prácticamente involucra todas las facetas de un RMA (percepción, actuación, planificación, arquitecturas de control, hardware, eficiencia computacional, resolución de problemas, ...), por lo que suele dividirse en varias subtareas. Una división muy común es la de (Zelek, 1996) o (Chatila, 1995) y que se corresponde a las siguientes preguntas:

1. ¿Qué posición ocupan el resto de objetos respecto al robot?, o ¿cómo se representa la información espacial que se tiene sobre el entorno?: confección de mapas.

2. ¿Dónde está el robot?: estimación de la posición. También se le conoce como el problema de la autolocalización; en base a la información que el robot percibe de su entorno ¿cómo relacionar la posición que ocupa actualmente el robot con el mapa que posee del entorno? Si no se conoce dicha relación ningún mapa sirve para nada.

3. ¿Por dónde debe ir el robot para llegar a su objetivo?, es decir, ¿qué ruta debe seguir?: planificación de rutas. La ruta suele minimizar algún parámetro, p.e. distancia recorrida, tiempo de ejecución o energía consumida.

4. ¿Cómo sigue el robot la ruta que, en teoría, le lleva a su objetivo?: ejecución de rutas. La traducción no siempre es sencilla y debería ser oportunista, es decir, aprovechar las ventajas que ofrece el medio (p.e., si se puede usar un atajo que se creía bloqueado).

Sin embargo, el consenso no es total. Así, por ejemplo, algunos autores (Murphy, 2000) añaden una quinta subtarea que trata de responder a la pregunta de ¿a dónde debe ir el robot?, es decir, establecer el destino del desplazamiento. Generalmente se considera que el objetivo es un parámetro de la tarea de navegación y que, por lo tanto, debe estar incluida en ella. Sin embargo, en robots totalmente autónomos la determinación del destino se corresponde con la tarea de explorar el entorno.

Otros autores (Lewitt y Lawton, 1990; Nehmzow, 2000) no tienen en cuenta la ejecución de una ruta. También hay quien sustituye la subtarea de ejecución de la ruta por la de exploración (Lee, 1996). Una postura más radical, actualmente poco defendible, es la de quienes proponen la eliminación total de los mapas, bajo el lema de que "el entorno es el mejor modelo de sí mismo"(Brooks, 1991b).

Por supuesto, la implementación de cada una de las subtareas mencionadas depende del tipo de navegación seleccionada (i.e, del tipo de mapa usado), del entorno y del soporte físico del robot.

2.4.2 Modificación del entorno

El robot modifica su entorno cuando provoca algún cambio en el mismo.

Tipos de modificación

En función de la intencionalidad y de los medios utilizados, se pueden clasificar las modificaciones o alteraciones que produce el robot en su entorno en tres tipos o categorías:

1. *Modificación pasiva*, o indirecta, consiste realmente en la propia interacción del robot con su entorno ya que la sola existencia del robot perturba de una manera u otra el medio en el que se desenvuelve. Es inevitable y se la considera más bien un efecto secundario no deseado.

2. *Modificación activa*, o directa, es la perturbación del entorno que tiene lugar de forma intencionada. Ejemplos de este tipo serían: bloquear un camino o una puerta parándose en el medio o empujar un objeto (p.e. una caja o una puerta medio abierta).

3. *Manipulación*, es la modificación que se realiza mediante efectores especializados, típicamente un brazo articulado con su pinza.

Subtareas de la manipulación

La manipulación se suele dividir en las mismas subtareas que la navegación, es decir,

1. confección y mantenimiento de un mapa del entorno del brazo,

2. localización de la posición y orientación del objetivo,

3. cálculo de la postura del brazo,

4. planificación de un ruta para alcanzar dicho objetivo,

5. ejecución de la ruta,

6. agarre.

Las principales diferencias con la tarea de navegación son las siguientes:

1. La localización del brazo es una tarea inmediata si se conoce la posición y la orientación de cada uno de sus "articulaciones" (cinemática directa). Por tanto, es muchísimo más simple que en navegación.

2. El mapa sólo se refiere al espacio de trabajo del brazo, es decir, a aquellas regiones a las que puede llegar. Sólo es comparable a los mapas de navegación utilizados en desplazamientos en tres dimensiones. Normalmente se confecciona a partir de información proporcionada por sensores en tres dimensiones: visión o láser 3DD.

3. El reconocimiento y la localización del objetivo es una tarea fundamental en manipulación, ya que a priori no se conoce la posición del objeto de interés o no se conoce con la precisión suficiente. Para esta etapa se suele utilizar un esquema de visión artificial, normalmente en estéreo.

4. Debido al elevado número de dimensiones del mapa y de grados de libertad del brazo, la planificación de rutas factibles, libres de obstáculos, es una tarea muy compleja. Por el mismo motivo, la ejecución de las rutas tampoco es nada sencilla. A veces se utilizan sensores de corto alcance (infrarrojos o ultrasonidos) para detectar colisiones durante el movimiento del brazo, pero no es lo más usual.

5. La fase de agarre no tiene una equivalencia en navegación. Consiste en seleccionar desde qué posición, orientación y modo se "coge" el objeto en cuestión. El modo de agarre depende del tipo de pinza: número de "dedos", disposición, grados de libertad, etc. La aproximación se puede realizar mediante visión, aunque la aproximación final requiere sensores táctiles, de fuerza o de corto alcance (p.e., barrera de infrarrojos). Esta subtarea crítica requiere una gran coordinación entre el brazo y la visión, muy similar a la existente entre el ojo y la mano humanos.

2.4.3 Comunicación con otros entes

Entendemos por interacción cualquier tipo de comunicación que tenga lugar entre el robot y cualquier otro "ente" que pueble su mismo entorno. Los entes pueden ser otros robots, seres humanos, animales, dispositivos automáticos, etc. La interacción es fundamental para que un robot pueda convivir e integrarse armónicamente en un espacio común con otros entes. Sería el equivalente en robótica a las "relaciones sociales".

La comunicación es fundamental para la coordinación en equipos o sociedades de múltiples robots, también denominados sistemas multiagentes (Dudek y col., 1996;

Cao y col., 1997; Arkin, 1998; Stone y Veloso, 2000; Murphy, 2000). El objetivo de estos sistemas es utilizar muchos robots sencillos para realizar la misma tarea que un único robot más grande y complejo. Alguno de sus beneficios son que cubren más área y trabajan más rápido (operan en paralelo), tienen un bajo coste y, fundamentalmente, operan de forma más robusta y redundante (si se "pierde" algún robot, el resto sigue operando). Actualmente este tipo de sistemas tienen un gran número de adeptos y poseen sus propias revistas, congresos y competiciones (p.e. *RoboCup*).

Medios de comunicación

Se pueden establecer tres medios fundamentales para el intercambio de información: el auditivo, el visual y el resto del espectro electromagnético. Aunque tampoco se pueden excluir otros medios más limitados como el táctil (p.e., pulsar botones o utilizar una pantalla especial).

El medio auditivo. Tiene que ver con la transmisión y recepción de todo tipo de sonidos. También incluye el lenguaje natural, con toda su complejidad. Es el preferido para la relación persona-robot por ser el más directo e intuitivo.

El medio visual. Por su importancia para los seres humanos se distingue el rango óptico del resto del espectro electromagnético. Es el asociado a la visión artificial. Se pueden establecer dos niveles diferentes en la comunicación visual. En un primer nivel estaría la comunicación diferida que se establece entre un conjunto de entes a través de una serie de *marcas artificiales* en el entorno, por ejemplo, señales, iconos y pictogramas.

En un segundo nivel estaría la comunicación directa mediante luces, gestos e, incluso, posturas. Este tipo de interacción es muy compleja de implementar (excepto en el caso de usar señales lumínicas u otros códigos específicos). Recientemente también se está dedicando mucha atención a la comunicación visual que puede *producir* un robot (p.e., expresiones "faciales" (Breazeal, 2000)). Esta característica ayuda a una mejor integración del robot en aquellas aplicaciones donde interacciona con humanos, p.e. robots-guías (Thrun y col., 1999) y robots-mascotas (Aib, 1997).

Las ondas electromagnéticas. Es el medio más utilizado para la comunicación entre diferentes robots. Se utilizan aparatos convencionales en el espectro infrarrojo y de ondas de radio (modems y tarjetas de radio ethernet) con un protocolo y

unas frecuencias predeterminadas. En este tipo de medios la comunicación está más controlada y se puede simplificar en gran medida (sobre todo los apartados de transmisión y recepción de la información). Recientemente incluso han aparecido pequeños robots móviles con los que se puede interaccionar, a un nivel muy simple, a través del mando a distancia de un televisor o vídeo.

Niveles de comunicación

Se pueden establecer tres niveles básicos en la comunicación entre un robot y otros "entes".

1. Nivel básico o de dependencia, del tipo maestro-esclavo. Por ejemplo, entre un usuario que ordena y el robot que obedece.

2. Nivel de colaboración, donde cada robot no es consciente de que existen los demás entes con los que interacciona (Angulo Usátegui y col., 1999). Por ejemplo, una granja de robots (Mataric, 1994). Si se quiere que los robots trabajen para lograr un objetivo común la coordinación debe correr a cargo del usuario o del diseñador.

3. Nivel de cooperación, donde cada robot tiene un conocimiento de los demás y juntos cooperan o compiten en el desarrollo de una o varias tareas. Un ejemplo sería la competición *RoboCup* (Rob, 1997), donde los robots de un mismo equipo cooperan entre sí (cada uno actuando según su propio papel o rol), compitiendo con los del equipo contrario.

2.5 RMA de propósito general

Como se ha podido observar, las habilidades que requiere un RMA de propósito general son muy amplias y complejas. Es necesario pues simplificar el esquema propuesto para que la implementación de un RMA pueda ser abordable. Eso sí, manteniendo siempre en el horizonte el objetivo final que se persigue: desarrollar el sistema de control de un RMA de *propósito general.*

2.5.1 Elección del nicho ecológico

La primera simplificación consiste en establecer un *nicho ecológico*. En el presente trabajo se ha optado por acotar el ámbito de actuación del RMA al interior de edificios (entornos *indoors*). Las principales razones que nos movieron a tomar esta decisión fueron, en primer lugar, que es relativamente fácil disponer de un recinto dentro de un edificio que sea válido para realizar las imprescindibles pruebas experimentales. La única limitación es el tamaño máximo del robot (p.e., para cruzar puertas).

En segundo lugar, las pruebas son más sencillas de realizar, las condiciones experimentales se pueden controlar más fácilmente y los experimentos pueden ser repetibles.

Esta opción no resta generalidad al trabajo presentado. Aunque, bien es cierto que la relativa estructuración o regularidad de un entorno de interiores reduce sensiblemente la complejidad del equipamiento sensomotor del RMA y, con ello, de sus funciones.

2.5.2 Selección de la tarea a implementar

Una segunda simplificación, y la más importante, se refiere al tipo de tareas seleccionadas para su implementación. Como se puede intuir, el completo desarrollo de un RMA requiere un esfuerzo extraordinario y durante un tiempo muy considerable. Para hacer abordable el problema, se ha decidido reducir las especificaciones de un RMA e implementar únicamente las habilidades relacionadas con la tarea de *navegación*.

La navegación es primordial en cualquier RMA y es la base para cualquier otra aplicación que se quiera desarrollar. Es una tarea compleja. Por un lado porque el robot opera en un entorno complejo, dinámico, impredecible y, muchas veces, no totalmente conocido. Por otro lado porque prácticamente involucra todas las facetas y funciones imprescindibles para un RMA.

Un beneficio colateral de centrarnos en la tarea de navegación consiste en la sencillez y en el relativamente poco equipamiento necesario para realizar pruebas, sobre todo si lo comparamos con la tarea de manipulación (donde se necesita un brazo con un número apropiado de grados de libertad y un completo sistema de visión artificial), o de colaboración entre robots (donde, obviamente, se necesitan varios robots móviles).

2.5.3 Elección del sistema sensomotor

La tercera y última simplificación ha consistido en la elección del *sistema sensomotor* del RMA. Concretamente, nos hemos centrado en las capacidades del robot móvil *Nomad 200* descrito en el capítulo anterior. La principal razón para esta decisión ha sido la adaptación de este robot tanto al nicho ecológico seleccionado como al tipo de tareas planteadas.

La adopción de un *Nomad 200* no resta generalidad al trabajo presentado. Más bien al contrario, la elección de los sensores (fundamentalmente el tipo de láser) condiciona y complica sensiblemente algunas de las funciones y tareas del RMA. Obviamente, la utilización de otro tipo de plataformas (p.e., robots con patas) y de sensores (p.e., cámaras) modificaría profundamente cierto tipo de funciones. Pero, como veremos, las funciones afectadas no son las principales del sistema, y los cambios inducidos son relativamente fáciles de integrar en un sistema de control válido para el robot elegido.

Por lo tanto, el planteamiento final del objetivo del presente trabajo quedaría de la siguiente manera: desarrollar el sistema de control de un robot móvil *Nomad 200* que le permita navegar por el interior de un edificio de forma autónoma y mostrando un comportamiento inteligente.

Capítulo 3

Arquitecturas de control para RMA

Establecido el tipo de tareas a realizar, el nicho ecológico y el soporte físico de un robot de "propósito general", pasamos a estudiar qué tipo de arquitectura es necesaria para implementar su sistema de control. En primer lugar definiremos qué entendemos por arquitectura de control en robótica y estableceremos qué requisitos debe cumplir en general. También fijaremos los criterios que nos guiarán en la evaluación de las principales propiedades exigibles para la arquitectura de control de un robot de las características reseñadas y que nos servirán para comparar las distintas arquitecturas propuestas.

A continuación pasaremos revista a algunas de las principales arquitecturas más significativas que se han utilizado en robótica hasta el momento, indicando sus principales ventajas y limitaciones. A la vista de las diferentes deficiencias que se observan en todas ellas, estableceremos un nuevo modelo de arquitectura basado en especialistas que se ajusta mejor a los criterios de interés fijados.

3.1 Arquitecturas de control en robótica

3.1.1 Definición

Existen múltiples y variadas definiciones de lo que se entiende por arquitectura de control en general y en el ámbito de la robótica móvil autónoma en particular. Entre todas ellas destacamos las realizadas por los siguientes autores:

Mataric (Mataric, 1992a): "Una arquitectura proporciona una forma pautada de organizar un sistema de control. Además de proporcionar la estructura, im-

pone unas restricciones en la forma en la que el problema de control puede ser solucionado".

Hayes-Roth (Hayes-Roth, 1995): "Una arquitectura se refiere al diseño abstracto de una clase de agentes: el conjunto de componentes estructurales (en los que se realiza la percepción, el razonamiento y la acción), la funcionalidad específica y la interfaz de cada componente, así como la interconexión topológica entre ellos".

Albus (Albus, 1995): "Una arquitectura es una descripción de cómo un sistema se construye a partir de componentes básicos y cómo dichos componentes encajan entre sí para formar el conjunto".

Gat (Gat, 1998): "Una arquitectura es un conjunto de restricciones en la estructura de un sistema software".

Arkin (Arkin, 1998): "La arquitectura en robótica es la disciplina que se dedica al diseño de robots individuales y altamente específicos a partir de una colección de bloques de construcción software comunes".

De todas estas definiciones, las que más se aproximan a nuestra percepción de arquitectura software de control son la de Hayes-Roth y la de Albus.

3.1.2 Requisitos

Los requisitos que debe cumplir una arquitectura software de control se pueden dividir en dos clases: requisitos de diseño y requisitos de control. Los primeros son los mismos que se utilizan en el diseño de cualquier aplicación software. Los segundos están más orientados a las necesidades de los sistemas de control y, más concretamente, a la robótica móvil autónoma.

Sin embargo, y contrariamente a cualquier otro software de aplicación, la arquitectura de control de un robot móvil autónomo no suele tener un conjunto de requisitos bien definido, sino que estos han de ser constantemente reevaluados y modificados (Liscano y col., 1995). Por ese motivo, la flexibilidad y modularidad en los sistemas de control es especialmente relevante.

Requisitos de diseño

Los requisitos de diseño de una arquitectura de control son los deseables y exigibles a cualquier aplicación software (Liscano y col., 1995; Joyanes, 1996):

1. *Modularidad*, el sistema debe ser divisible en subsistemas más pequeños que puedan ser diseñados, implementados y depurados por separado. La modularidad también es crucial para el diseño incremental, el mantenimiento y la detección y corrección de fallos.

2. *Flexibilidad*, facilita las modificaciones, resultado de los cambios en las especificaciones del sistema. P.e., tipos de representación de los datos, algoritmos de procesamiento, planes de actuación, etc. Esta propiedad es muy necesaria en dominios todavía en fase de investigación y donde estos componentes aún no están claramente definidos.

3. *Extensibilidad* o *escalabilidad*, facilita una implementación incremental de las funcionalidades, módulos o tareas que se necesita incorporar al sistema sin afectar gravemente a su rendimiento final y sin incurrir en un excesivo coste: computacional, de diseño, de mantenimiento, etc. Es importante para aumentar las capacidades del robot y sus ámbitos de aplicación.

4. *Fiabilidad*, o habilidad del sistema para operar sin fallos ni degradaciones de rendimiento a lo largo del tiempo.

5. *Eficiencia*, o capacidad del sistema para hacer un buen uso de los recursos que manipula.

6. *Portabilidad*, o la facilidad para ser ejecutado sobre diferentes sistemas físicos y lógicos.

7. *Reutilización de componentes*, o capacidad del sistema para ser reutilizado, en su totalidad o en parte, en nuevas aplicaciones.

8. *Generalidad*, el sistema debe ser usable en condiciones o bajo entradas diferentes a las utilizadas durante la etapa de diseño.

Requisitos de control

Denominamos requisitos de control a todas aquellas propiedades deseables en cualquier sistema que actúe en y sobre el mundo real (Fayek, 1992; Liscano y col., 1995).

1. *Predictibilidad*, o la capacidad de poder anticipar la salida del sistema si se conocen sus entradas y su estado interno. Es una característica de los sistemas determinísticos.

2. *Monitorizable* por un operador, lo que facilita un mejor entendimiento y depuración del sistema.

3. *Robustez*, o la capacidad para seguir operando incluso en situaciones anormales: entradas imperfectas y/o con incertidumbre, eventos inesperados, fallo de funcionamiento, etc. Los errores no deben provocar un colapso en el sistema, solamente una degradación suave en el rendimiento del mismo. Se trata de un requisito crucial en cualquier sistema que opere en el mundo real.

4. *Reactividad*, es decir, la capacidad del sistema para actuar apropiadamente y en un tiempo adecuado (tiempo real) ante cualquier modificación repentina que surja en su entorno. Es fundamental cuando se opera en entornos complejos, dinámicos, poco estructurados o no totalmente predecibles.

5. *Integración de comportamientos* de tipo reactivo, deliberativo y planificado, con el objetivo de obtener un comportamiento final del robot coherente e inteligente.

6. *Integración multisensorial*, imprescindible dados los grandes problemas que existen con los sensores actuales de un robot móvil: poca fiabilidad, baja precisión, reducido rango de aplicación, coste computacional, etc. Es necesario integrar información en el tiempo y proveniente de distintos sensores (en el mejor de los casos complementarios). También incluye el manejo de la incertidumbre en la información sensorial.

7. *Adaptabilidad*, permite que el sistema modifique su modo de operación a las distintas situaciones en las que se desenvuelve.

8. *Aprendizaje*, entendido como la capacidad para trabajar correctamente en situaciones nuevas y/o complejas a partir de las experiencias previas.

9. *Autonomía*, es decir, la nula dependencia con el exterior.

10. *Resolución de múltiples objetivos*, entendida como la capacidad de ejecutar varias tareas simultáneamente, y que muchas veces implican acciones contradictorias entre sí.

11. *Razonamiento global*, necesario para un conocimiento completo y global de la situación y que permita mantener la coherencia en el comportamiento del robot y mejore su eficacia.

12. *Razonamiento reflexivo*, que permita un mejor conocimiento sobre el propio sistema y sobre su actuación.

3.1.3 Criterios de evaluación

Diseñar una arquitectura consiste en tomar un determinado conjunto de decisiones, cada una de las cuales influye directamente en el resultado final. Sin embargo, no conviene olvidar (Murphy, 2000) que, al igual que en la construcción de una casa o de un coche, no existe una arquitectura "correcta", aunque en todos los casos se utilicen, básicamente, los mismos componentes. Existe un diseño más o menos adaptado a cada situación y a cada caso particular.

En realidad, lo más importante es la habilidad de evaluar en qué medida una arquitectura se adecua a una aplicación específica o a unos requisitos predeterminados. Visto de otro modo, en qué medida afecta al resultado final cada una de las decisiones tomadas durante el diseño de la arquitectura. Para ello es imprescindible establecer una serie de criterios de evaluación que, de una manera u otra, fijan una métrica con la que será posible comparar distintas versiones de una misma arquitectura y diferentes arquitecturas entre sí.

Como es de esperar, los criterios de evaluación dependen profundamente de las tareas del robot, de su soporte físico y de su entorno, es decir, de la triada robot-entorno-tarea. Se pueden imaginar multitud de criterios pero, de entre todos los posibles, hemos seleccionado de la bibliografía tres conjuntos de los que creemos más interesantes y después hemos fijado los criterios de evaluación utilizados en nuestro caso.

Criterios propuestos por Arkin y Murphy

R.C. Arkin (Arkin, 1998) propone un amplio conjunto de criterios para evaluar las características que considera más relevantes en una arquitectura de control en robótica. Se trata de un conjunto de criterios muy completo, pero algo sesgado hacia la evaluación de un determinado tipo de arquitecturas (concretamente las arquitecturas reactivas que veremos más adelante). Los criterios que propone son:

1. Soporte para la ejecución en *paralelo.*
2. *Adecuación al hardware*, tanto a nivel físico (sensores y efectores) como a la potencia de cálculo.
3. Adecuación al *nicho ecológico* en el que debe operar el robot.
4. Soporte para la *modularidad* de la implementación.
5. *Robustez.*
6. Facilidades de *desarrollo*, es decir, si la arquitectura es meramente una propuesta teórica o proporciona herramientas y métodos específicos para crear sistemas reales.
7. *Flexibilidad* en tiempo de ejecución. Por un lado, la capacidad de ajuste y reconfiguración durante el funcionamiento del sistema. Por otro lado, facilidad para incluir adaptación y aprendizaje en la arquitectura.
8. *Eficacia* en la ejecución de las tareas encomendadas. En algunos casos se puede definir un rendimiento a partir de variables como el tiempo empleado en ejecutar una tarea, la energía consumida, la distancia recorrida, etc.

R. Murphy (Murphy, 2000) propone una versión reducida del conjunto de criterios propuestos por Arkin que permite una evaluación más general y menos sesgada:

1. Soporte para *modularidad.*
2. *Adecuación al nicho ecológico* del robot.
3. Facilidad para su *portabilidad* a otros dominios de aplicación.
4. *Robustez.*

Criterios propuestos por Schumacher

En el libro de M. Shaw y D. Garlan (Shaw y Garlan, 1996) M. Schumacher propone un conjunto de cuatro requerimientos muy interesantes para comparar distintas aproximaciones arquitectónicas propuestas en robótica móvil:

1. La arquitectura debe acomodar *comportamientos de tipo deliberativo y reactivo*. Es decir, el robot debe coordinar las acciones que toma para conseguir el objetivo designado (p.e., coger una muestra de roca) con las reacciones a las que le fuerza el entorno (p.e., evitar un obstáculo).

2. La arquitectura debe tener en cuenta la *incertidumbre*. Las circunstancias de operación de un robot nunca son totalmente predecibles. La arquitectura debe proveer un marco en el cual el robot pueda actuar incluso con información incompleta o de poca confianza (p.e. medidas sensoriales contradictorias).

3. La arquitectura debe *tener respuesta a los peligros* inherentes a la operación del robot en su entorno. Teniendo en cuenta la tolerancia a fallos, la seguridad y el rendimiento, la arquitectura debe ayudar a mantener la integridad del robot, sus operadores y su entorno. Problemas como un reducido suministro de energía o puertas inesperadamente abiertas no deben conducir a un desastre.

4. La arquitectura debe proporcionar *flexibilidad* al diseñador. El desarrollo de aplicaciones para robots móviles frecuentemente requiere experimentación y reconfiguración. Más aún, los cambios en tareas requieren modificaciones regulares.

Criterios propuestos para el caso de un robot de "propósito general"

En nuestro caso el objetivo es diseñar la arquitectura de control necesaria para conseguir un robot móvil autónomo (RMA) de "propósito general", concretamente, con la "inteligencia básica" necesaria para realizar distintas tareas útiles en el interior de un edificio. Por ese motivo, los criterios de evaluación adoptados son ligeramente diferentes, frente a los anteriormente comentados, aunque se encuentran en clara sintonía con ellos.

Los criterios elegidos son un subconjunto de los requisitos que cualquier arquitectura de control debe poseer, al mismo tiempo que su número se ha mantenido lo suficientemente reducido para que sean manejables. Se han seleccionado aquellos

requisitos que, por un lado, inciden en las principales características que debe poseer el control de un RMA de "propósito general" y, por otro lado, resaltan las diferencias más importantes que existen entre las distintas propuestas.

Respecto a los requisitos de diseño consideramos que la flexibilidad y la escalabilidad son dos de los más importantes, ya que facilitan futuras modificaciones y ampliaciones de las funcionalidades implementadas. Para conseguir ambas cualidades es fundamental que el sistema sea modular. Por otra parte, es evidente que tanto la fiabilidad como la eficiencia son fundamentales para el control de cualquier sistema, más aún en el caso de un robot. Sin embargo, ambas propiedades dependen tanto o más de la implementación final de la arquitectura que de su diseño teórico. De igual forma, la portabilidad, la generalidad y la reutilización de componentes dependen fundamentalmente de la metodología utilizada para implementar la arquitectura.

En cuanto a los requisitos de control, para la arquitectura de control de un RMA son esenciales la robustez y la reactividad. Ambas cualidades dependen no tanto de la implementación como del diseño de la propia arquitectura. Además, son instrumentos eficaces para establecer comparaciones entre arquitecturas. Otros requisitos de control también son imprescindibles pero su utilidad como criterios de evaluación se puede considerar relativa. Así, un sistema de control debe ser predecible o, de lo contrario, es de escaso interés. Lo mismo ocurre con la autonomía.

Un requisito de control muy importante en el dominio específico de la robótica móvil es la integración de comportamientos de tipo reactivo, deliberativo y planificado. Si un sistema es capaz de realizar tal integración es legítimo considerar que también puede resolver múltiples objetivos e incorporar fácilmente mecanismos para el razonamiento tanto global como reflexivo. En un robot la integración multisensorial es imprescindible, ya que, en general, se utilizan diferentes tipos y número de sensores. Sin embargo, aunque un mal diseño de la arquitectura puede influir muy negativamente, la integración depende más bien de la representación de la información y del tipo de procesamiento utilizado. Por ese motivo, no se considera un buen criterio de evaluación.

El resto de requisitos de control son muy deseables pero no los consideramos decisivos, al menos por el momento, en una comparación entre arquitecturas. Así, las facilidades de monitorización facilitan el diagnóstico de errores, permiten una mejor comprensión de la arquitectura y simplifican su desarrollo y mantenimiento. En definitiva, ayudan a mejorar las características de la arquitectura. Por último, a

pesar del indudable interés de la adaptabilidad y el aprendizaje no tenemos constancia de que ninguna arquitectura incluya soporte específico para su implementación. Esta ausencia es, sin duda, uno de los puntos débiles de todas las arquitecturas de control actuales para robots móviles autónomos y una de las principales deficientes a paliar en un futuro próximo.

En resumen, como criterios de evaluación de la arquitectura de control de un RMA de "propósito general" hemos seleccionado los siguientes:

1. *Flexibilidad.*
2. *Escalabilidad.*
3. *Robustez.*
4. *Reactividad.*
5. *Integración de comportamientos.*

3.2 Principales paradigmas en robótica

Según (Murphy, 2000) un paradigma consiste en "una filosofía o un conjunto de suposiciones y/o técnicas que caracterizan una aproximación a una clase de problemas". Ello implica tanto la manera de ver el problema como el conjunto de herramientas necesarias para su solución. La misma autora recalca que ningún paradigma es correcto sin más, únicamente algunos problemas pueden ser resueltos más fácilmente por medio de unas aproximaciones que de otras, es decir, con el paradigma adecuado.

En robótica se han utilizado históricamente tres paradigmas: las arquitecturas jerárquicas, las arquitecturas reactivas o basadas en comportamientos y las arquitecturas híbridas planificadoras/reactivas. El ordenamiento cronológico es importante porque facilita la adquisición de una visión histórica de cómo han evolucionado los distintos paradigmas en las arquitecturas de control en robótica y, sobre todo, ayuda a entender cómo y porqué ha aparecido cada uno de ellos.

En (Murphy, 2000) se utilizan dos criterios básicos para diferenciar los distintos paradigmas:

1. la relación entre las funciones primitivas de la robótica: percepción, planificación y actuación.

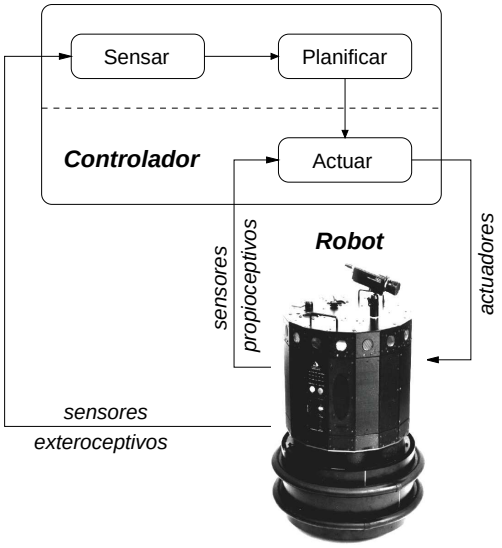

Figura 3.1: Esquema general de una arquitectura de control jerárquica (Saffiotti, 1997a).

2. la manera en la que los datos sensoriales son procesados y distribuidos a largo de la arquitectura.

A continuación veremos cuáles son las principales características de los distintos paradigmas, estudiaremos las ventajas y limitaciones de cada uno de ellos e ilustraremos su funcionamiento con alguna de las arquitecturas que pueden ser consideradas de referencia en cada caso. Finalmente, evaluaremos cada paradigma según los criterios que hemos establecido en la sección anterior.

3.2.1 Arquitecturas jerárquicas

Las arquitecturas jerárquicas han sido históricamente las primeras en ser aplicadas en robótica móvil (Nilsson, 1969), ya que surgen como una extensión natural de la cibernética (Nehmzow, 2000).

Principales características

Las arquitecturas jerárquicas se caracterizan por actuar según un ciclo cuyo objetivo es reducir un "error" que se define como la diferencia entre el estado actual del robot y el estado objetivo. El proceso se repite indefinidamente y de forma secuencial hasta que el objetivo es alcanzado. En la figura 3.1 se puede ver el

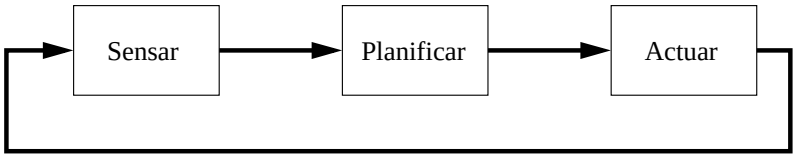

Figura 3.2: Organización de las funciones robóticas primitivas en una arquitectura jerárquica.

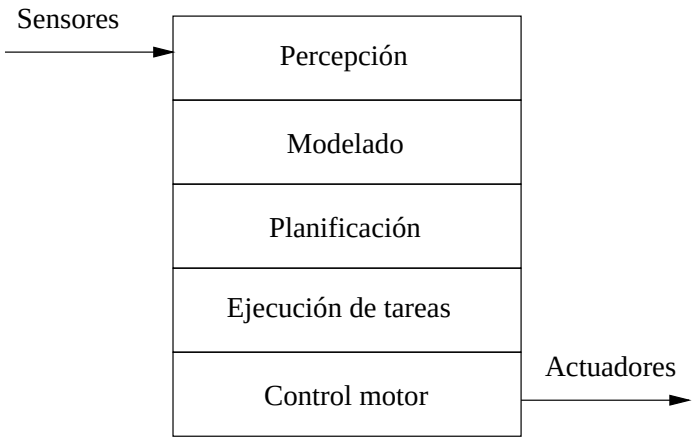

Figura 3.3: Descomposición de tareas de tipo horizontal o funcional propia de las arquitecturas jerárquicas.

esquema general de las arquitecturas jerárquicas, donde se puede apreciar que las primitivas de procesamiento se organizan formando el característico ciclo *Sensar-Planificar-Actuar* (figura 3.2) o, más abreviadamente, (S,P,A).

En general, las arquitecturas jerárquicas constan de 5 módulos funcionales que se ejecutan de manera secuencial: percepción o extracción de características, modelado del entorno, planificación de tareas, ejecución de tareas y control motor. Este tipo de descomposición de tareas se denomina horizontal o funcional (figura 3.3).

Otra característica sobresaliente de este tipo de arquitecturas es que la información sensorial se procesa secuencialmente utilizando un único y exhaustivo modelo interno del mundo, donde se incluyen tanto datos del entorno que rodea el robot como información del propio robot. Implícitamente se considera que se cumplen dos hipótesis: una, que el mundo realmente se puede modelar, es decir, se puede representar mediante símbolos; y dos, que el mundo es "cerrado", es decir, en el modelo se puede incluir todo lo que puede necesitar el robot y, por lo tanto, no puede haber sorpresas.

En general las arquitecturas jerárquicas están orientadas hacia la planificación y el razonamiento y suelen dar muy poca o nula importancia a la interacción del robot con su entorno. Por ese motivo, generalmente sólo se aplican en entornos altamente estructurados y estáticos. Suelen descomponer recursivamente las tareas en subtareas progresivamente más simples.

Ventajas

La principal ventaja de las arquitecturas jerárquicas es que proporcionan un orden en las relaciones entre la percepción, la planificación y la actuación. Proporcionan por tanto una buena organización estructural y funcional.

Las arquitecturas jerárquicas están muy bien adaptadas para operar en entornos altamente predecibles y estructurados (p.e., entornos industriales). En general son arquitecturas desarrolladas para una aplicación específica, más que para servir como una arquitectura general válida para nuevas aplicaciones. También son muy apropiadas para el control semi-autónomo, donde el operador humano se encarga de las funciones de más alto nivel.

Limitaciones

La primera limitación de las arquitecturas jerárquicas es que son extremadamente frágiles, ya que procesan la información de manera secuencial. De este modo, si cualquiera de los módulos falla, entonces todo el sistema falla completamente. Por tanto, no existe una degradación progresiva del rendimiento.

Sin embargo, el inconveniente más grave de estas arquitecturas es que ciertas tareas relativamente sencillas, pero muy importantes en robótica móvil, tales como esquivar un obstáculo o seguir paredes u objetos en movimiento, no se pueden realizar de forma satisfactoria, ya que la reactividad en estas arquitecturas es muy pobre. La razón es que la percepción y la actuación están siempre desconectadas, lo que elimina la posibilidad de ejecutar las acciones de tipo estímulo-respuesta que son necesarias para dichas reacciones. Por ese motivo las arquitecturas jerárquicas son inoperantes en entornos dinámicos y escasamente modelables o predecibles.

Este fracaso se debe a la propia concepción de la arquitectura. En ella las medidas sensoriales, independientemente de su importancia, deben recorrer toda la arquitectura, es decir, seguir el ciclo SPA, antes de que el robot pueda actuar. De esta manera, el tiempo de reacción aumenta prohibitivamente. Los módulos no están

diseñados para operar asíncronamente o con prioridades, lo que permitiría fijarse en las tareas o situaciones más importantes, es decir, no hay mecanismos para focalizar la atención y los recursos del sistema. Sin embargo, existen situaciones donde es más importante tomar una acción rápidamente (p.e. en una acción de evasión) que reflexionar acerca de las posibles opciones y seleccionar la óptima.

Por otra parte, el proceso de percepción o modelado general del entorno es extremadamente costoso y, en muchas ocasiones, innecesario. En muchas situaciones (sobre todo en momentos críticos donde hay que actuar rápidamente) el tratamiento sensorial puede y debe simplificarse muy considerablemente para agilizar la toma de decisiones. Mantener el modelo del mundo es un trabajo muy costoso y en muchos casos se presta atención a detalles irrelevantes para la tarea que se esté ejecutando, mientras que en otros no se consideran objetos que sí son importantes. En definitiva, cuanto más completo es el modelo, más costoso, ineficiente e improductivo es su mantenimiento, ya que un porcentaje cada vez mayor de los objetos modelados no se utilizan durante la ejecución de una tarea dada.

Respecto al modelado simbólico, algunos autores (Harnad, 1990; Nehmzow, 2000) señalan el problema de los sistemas de computación simbólica (*Symbol Grounding Problem*). Tal y como lo expresa Harnad, el comportamiento, aunque interpretable como controlado por reglas, no tiene porqué estar gobernado por reglas simbólicas, más al contrario, la mera manipulación de símbolos no es suficiente para el conocimiento. En lugar de eso, los símbolos tienen que estar basados en alguna entidad con significado. En otras palabras: los símbolos por si mismos no poseen significado, sólo su efecto (físico) sobre el robot es relevante para la operación del robot.

Otra importante limitación reside en el método de planificación utilizado: en cada ciclo el robot debe primero actualizar su modelo interno del mundo y después realizar algún tipo de planificación. Ello introduce un importante cuello de botella. Al mismo tiempo, la planificación es una tarea muy difícil y la ejecución de un plan en lazo abierto, tal y como tiene lugar en las arquitecturas jerárquicas, no es la mejor opción en entornos en donde domina la incertidumbre y la impredecibilidad (Gat, 1998).

Una limitación menos evidente de estas arquitecturas es su incapacidad para manejar adecuadamente la incertidumbre (Murphy, 2000). Y en robótica la incertidumbre surge en múltiples frentes: la indeterminación semántica (i.e, el significado real de las etiquetas), el ruido sensorial, los errores en los actuadores, el grado de cumplimiento de las tareas, etc.

Arquitecturas representativas

Posiblemente la arquitectura jerárquica más representativa es la NHC (*Nested Hierarchical Controller*) de Meystel (Meystel, 1990). Otras muy conocidas fueron las desarrolladas para controlar los primeros robots móviles: p.e. *Shakey* (Nilsson y col., 1984) y *Hilare* (Noreils y Chatila, 1995).

Otro conjunto importante de arquitecturas jerárquicas son las que corresponden al tipo BDI (*Belief, Desire, Intention*). Una implementación muy popular en robótica de esta clase de arquitecturas es PRS (*Procedural Reasoning System*) (Ingrand y col., 1992, 1996; Myers, 1996).

Utilizando NHC como modelo se han desarrollado otras arquitecturas jerárquicas, entre ellas destaca RCS (*Real Time Control System*) de Albus, una de cuyas versiones, NASREM (Albus y col., 1986), aún se utiliza en el JPL para teleoperar un brazo articulado.

Arquitectura NHC El esquema interno de la arquitectura NHC (Meystel, 1990) se muestra en la figura 3.4, donde se puede observar la clara separación entre los componentes *Sensar*, *Planificar* y *Actuar*. El módulo *Sensar* recoge la información sensorial y la integra en el modelo del mundo, que también contiene el conocimiento *a priori*. Actualizado el modelo del mundo, el robot lo utiliza para planificar las acciones a seguir y que posteriormente el módulo *Actuar* traduce en comandos a los actuadores.

La mayor contribución de NHC ha sido descomponer la planificación de la navegación en tres funciones o subsistemas: *Planificador de Misión*, *Navegador* y *Piloto*. Cada una de ellas tiene acceso al modelo del mundo, pero están organizadas de forma jerárquica. Así, el *Planificador de Misión* se encarga de traducir las órdenes de los usuarios a términos que el robot pueda manejar (p.e, la posición del destino). El *Navegador* se encarga de trazar una ruta desde la posición actual del robot a la de destino, normalmente fijando una serie de puntos de paso intermedios o subobjetivos (*waypoints*). *Piloto* recoge la ruta y determina qué acciones se deben ejecutar para seguir cada tramo.

Dos grandes ventajas de NHC respecto a las primeras arquitecturas jerárquicas (p.e. SHAKEY) son que durante la ejecución de las tareas se sigue actualizando el modelo del mundo y que intercala planificación y acción. Así, mientras existe un plan no se repite toda la planificación y *Piloto* se limita a comprobar si el robot sigue el camino fijado. Si aparece algún obstáculo *Piloto* avisa a *Navegador* para que genere

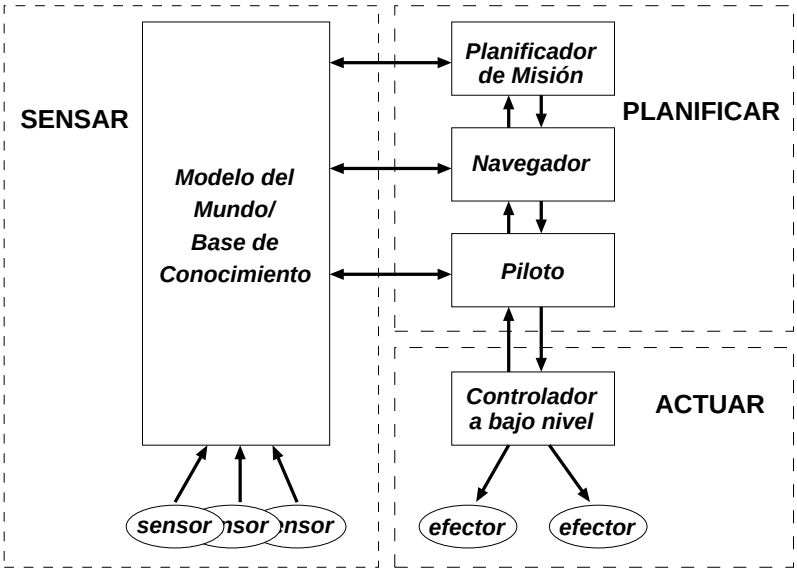

Figura 3.4: Esquema interno de la arquitectura NHC (Meystel, 1991).

una nueva ruta con el modelo del mundo actualizado. Si se ha alcanzado un punto de paso *Piloto* se lo comunica a *Navegador* para que éste le envíe el siguiente tramo de la ruta. Cuando el robot llega al destino *Navegador* se lo notifica al *Planificador de Misión*.

Como se puede apreciar, la descomposición de la planificación en la arquitectura NHC es totalmente jerárquica. Sin embargo, esta arquitectura sólo ha sido desarrollada para ejecutar la tarea de navegación y únicamente ha sido probada en simulación.

Evaluación

Respecto a los criterios de evaluación seleccionados ya se ha señalado que dos grandes limitaciones de las arquitecturas jerárquicas son su poca robustez y su nula reactividad. En el primer caso, dado que las funciones del sistema se ejecutan secuencialmente, un fallo en cualquiera de ellas provoca el colapso de todo el sistema. El método de planificación utilizado tampoco favorece la robustez, ya que los planes son una secuencia estricta de acciones donde todas las precondiciones deben cumplirse exactamente y todos los posibles errores o problemas en tiempo de ejecución deben ser contemplados.

Por otra parte, la activación de cualquier reacción implica que la información debe recorrer todo el ciclo de ejecución, dando lugar a tiempos de respuesta total-

mente prohibitivos que impiden la ejecución de cualquier comportamiento reactivo y, por tanto, su integración con otros comportamientos.

Por último, como las arquitecturas jerárquicas se organizan alrededor de un modelo interno del mundo, la flexibilidad y escalabilidad se ven fuertemente afectadas. Esta fuerte dependencia implica que cualquier cambio en el tipo de datos, en su forma de representarlos o, simplemente, en los algoritmos que se utilizan en su procesamiento conlleva una gran complejidad. Al mismo tiempo, la escalabilidad es muy reducida, ya que añadir nuevas tareas implica un modelo más completo del mundo y, por tanto, más complejo, más difícil de mantener y menos eficiente.

3.2.2 Arquitecturas reactivas

Como respuesta a los problemas de las arquitecturas tradicionales algunos autores apostaron por una vía radicalmente distinta, centrando su atención en la interacción con el entorno y primando la reactividad sobre la deliberación. Más que imitar las funciones cognitivas de los seres humanos, pretendían reproducir el comportamiento de seres vivos más simples, por ejemplo, los insectos. El cambio de enfoque fue tan profundo que, en la práctica, condujo a un nuevo paradigma en el diseño de las arquitecturas de control en robótica (Malcolm y col., 1989).

Principales características

La principal característica de las arquitecturas reactivas consiste en descomponer las funciones del robot no en módulos sucesivos y secuenciales, sino en módulos complementarios que se ejecutan en paralelo, dando lugar a una descomposición de tareas de tipo vertical (figura 3.5). Por ese motivo, también se las conoce como arquitecturas verticales (Brooks, 1986).

Cada módulo se centra en resolver una parte muy pequeña de las tareas que debe ejecutar el robot, independientemente del resto del sistema. Para ejecutar su tarea, cada módulo recibe directamente las medidas sensoriales y genera los comandos pertinentes para los efectores. De esta manera, y cómo se puede ver en la figura 3.6, existe un ciclo de percepción-acción (*Sensar-Actuar*) muy acoplado, lo que resulta fundamental para conseguir tiempos de reacción muy pequeños. Este modelo funcional se conoce como *Sensar-Actuar* o de tipo (S-A).

Generalmente, las acciones que genera un módulo conducen a que el robot realice un patrón de movimientos que un observador puede asociar a un comportamiento.

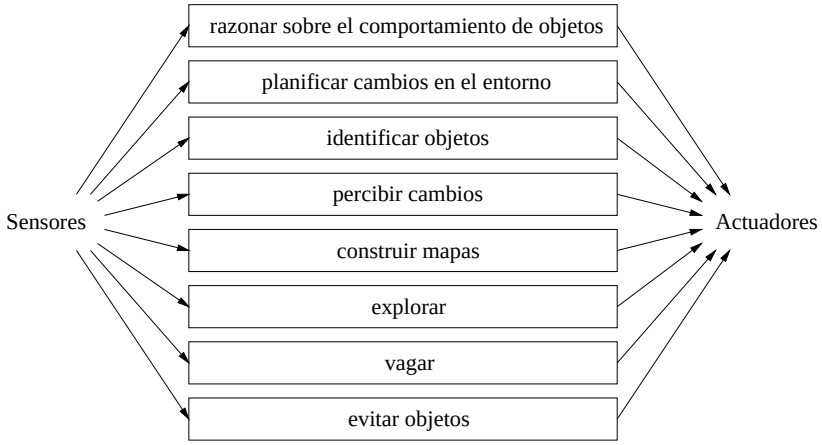

Figura 3.5: Descomposición vertical o en base a comportamientos de las tareas de un robot móvil (adaptado de (Brooks, 1986)).

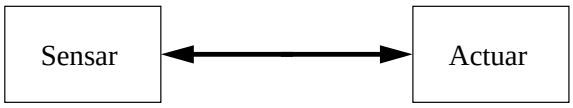

Figura 3.6: Organización de las funciones robóticas primitivas en una arquitectura reactiva.

Por extensión, cada módulo recibe el nombre de su comportamiento asociado aunque con la letra inicial en mayúsculas. Por este motivo, muy a menudo a las arquitecturas reactivas también se las conoce como arquitecturas basadas en comportamientos (Maes, 1993).

Otra característica relevante de las arquitecturas reactivas puras es que no almacenan ningún tipo de información sobre el entorno ni mantienen ningún modelo del mundo. Esta filosofía se enmarca dentro del lema "el entorno es el mejor modelo de sí mismo" (Brooks, 1991b). Como mucho se guarda información sobre el estado interno de cada comportamiento, pero siempre durante cortos intervalos de tiempo. Por lo tanto, se puede decir que los comportamientos implementados son muy simples y se limitan a actos reflejos donde cada patrón sensorial se asocia con una acción predeterminada.

Una consecuencia directa y muy importante de esta falta de información simbólica sobre el entorno es que se eliminan completamente todo tipo de tareas de deliberación y planificación.

Por otra parte, el control en las arquitecturas reactivas es estrictamente local ya que no existe ningún "controlador" central. La comunicación entre los Compor-

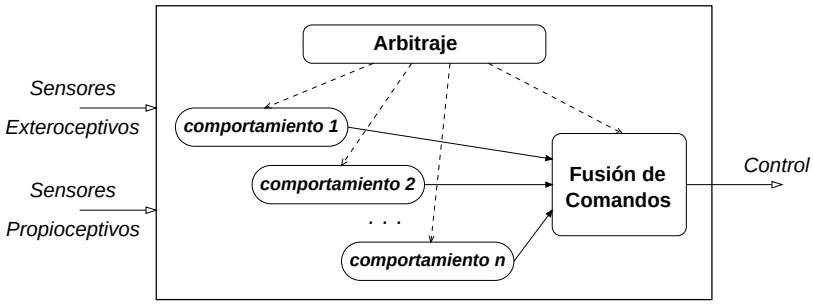

Figura 3.7: Descomposición en comportamientos propia de una arquitectura reactiva en donde se puede apreciar la fusión de comandos y el arbitraje entre comportamientos (adaptado de (Saffiotti, 1997a)).

tamientos o no existe y se realiza a través del propio entorno, o es asíncrona. Sin embargo, como varios Comportamientos se pueden ejecutar en paralelo, es necesario establecer tanto mecanismos de arbitraje entre los distintos Comportamientos activos como métodos de fusión entre los comandos generados. El comportamiento final del robot es emergente, es decir, es más que la simple suma de los comportamientos individuales.

Cada arquitectura que utiliza Comportamientos propone sus propios procedimientos de arbitraje y fusión que, a grandes rasgos, se pueden dividir en competitivos y cooperativos (Arkin, 1998). En los primeros se selecciona un Comportamiento entre todos los que están activos y su salida es el comando final de control del robot. En esta categoría están los métodos de subsunción (Brooks, 1986), de selección dinámica de acciones (Maes, 1989, 1990) y de votaciones (Hoff y Bekey, 1995; Rosenblatt, 1997).

Por el contrario, en los métodos cooperativos el comando final generado es el resultado de fusionar las salidas de todos los Comportamientos activos en cada momento. Destacan los métodos basados en campos de potencial (Arkin, 1989a, 1990b) y en lógica borrosa (Saffiotti y col., 1993; García-Alegre y col., 1995b,a; García-Alegre y Recio, 1998). Recientemente incluso han aparecido trabajos (Jung y Zelinsky, 1998) donde la selección de los comandos se realiza mediante planificación distribuida.

Ventajas

Las principales ventajas de las arquitecturas reactivas surgen de su modularidad y descomposición en varios Comportamientos independientes entre sí que operan en

paralelo. Cada módulo tiene acceso directo a los datos sensoriales y puede generar comandos para los efectores. En primer lugar, la ejecución en paralelo de varios Comportamientos, cada uno realizando su propia tarea, facilita de manera natural la ejecución de múltiples objetivos. Por lo tanto, las arquitecturas reactivas suelen ser más eficientes que las jerárquicas.

En segundo lugar, la utilización de Comportamientos independientes entre sí permite que cada uno se pueda diseñar, implementar y testear por separado. Teóricamente, se pueden incorporar nuevos Comportamientos a una arquitectura reactiva sin necesidad de modificar los Comportamientos ya existentes. Por lo tanto, estas arquitecturas son fáciles de diseñar, testear y extender.

En tercer lugar, la propia independencia entre los Comportamientos implica una mayor robustez y tolerancia a fallos: si uno de los Comportamientos deja de operar correctamente eso no influye en el resto del sistema. Como mucho, el robot dejará de mostrar dicho comportamiento y sufre la pertinente degradación en su funcionamiento.

Por último, debido a que el ciclo de percepción-acción dentro de cada Comportamiento está muy acoplado, su funcionamiento se adapta fácil y rápidamente a las cambiantes situaciones de su entorno. Todo ello permite que la arquitectura, en su conjunto, también posea unos tiempos de reacción muy pequeños (de ahí su nombre de arquitecturas reactivas).

Limitaciones

La principal limitación de las arquitecturas reactivas es su baja o nula capacidad de abstracción y de ejecución de tareas de alto nivel. Parte de la limitada capacidad de razonamiento se debe a que no se almacena información sobre el entorno a largo plazo (Gat, 1993) ya que, según la filosofía de diseño de estas arquitecturas, "el entorno es el mejor modelo de si mismo" (Brooks, 1991b), lo que hace que, por ejemplo, no se guarden mapas. De esa forma, aquellas tareas que requieren información de estado, memoria o planificación (p.e. navegación) se hacen de ejecución virtualmente imposible.

En este tipo de arquitecturas "la comunicación entre Comportamientos se realiza a través del propio entorno": cada Comportamiento coloca al robot en un estado tal que provoca el disparo del siguiente Comportamiento (Connell, 1989). El principal problema de esta aproximación es que estados muy similares pueden significar cosas muy diferentes según el contexto (Hartley y Pipitone, 1991). Sin embargo,

cada Comportamiento responde directamente a sus estímulos sensoriales y no posee memoria de estado interna, por lo que es imposible seguir secuencias de acciones especificadas externamente y es muy difícil establecer o expresar planes (Nehmzow, 2000).

En definitiva, la coordinación de tareas se hace muy compleja y costosa, por lo que se puede decir que en las arquitecturas reactivas no existe ningún soporte para la abstracción, la planificación o el manejo de recursos (Bonasso y col., 1997). Esta dificultad reduce la capacidad de conseguir objetivos a largo plazo e implica que las tareas que se pueden ejecutar son poco estructuradas y no muy complejas.

Más aún, las arquitecturas reactivas no son suficientemente modulares ya que, en la práctica, las capas superiores suelen interferir con las funciones internas de las capas inferiores, lo que dificulta que se puedan diseñar de forma independiente. También significa que incluso pequeños cambios en los Comportamientos de bajo nivel o en el propio robot implican un completo rediseño del controlador (Hartley y Pipitone, 1991).

Por último, los Comportamientos también son una fuente de problemas ya que, por ejemplo, es necesario seleccionar el conjunto básico de Comportamientos a implementar, estudiar cómo pueden interaccionar entre si, especificar los métodos de arbitraje y coordinación entre los Comportamientos activos en cada momento, etc.

Arquitecturas representativas

La arquitectura de subsunción (Brooks, 1986, 1990b, 1991b,a) y sus múltiples variantes son el clásico ejemplo de arquitecturas reactivas y es la que vamos a describir en más detalle. Sin embargo, existen muchas otras arquitecturas reactivas. En (Arkin, 1998) se puede ver una muestra de las más representativas.

Una arquitectura de tipo reactivo que destaca entre todas las demás es la arquitectura basada en esquemas (Arkin, 1989a) (Arkin, 1990b), prácticamente coetánea a la de subsunción. Además de sus interesantes soluciones a los principales problemas de las arquitecturas reactivas (división de los comportamientos en una parte de percepción y otra de acción, uso de campos de potencial para definir las acciones de los comportamientos, etc), esta arquitectura también es importante por ser la parte reactiva de la arquitectura híbrida AuRA.

En este punto también es conveniente comentar los esfuerzos que han realizado diversos autores para desarrollar lenguajes y formalismos que simplifiquen la representación (Arkin, 1998) e implementación de sistemas de control basados en compor-

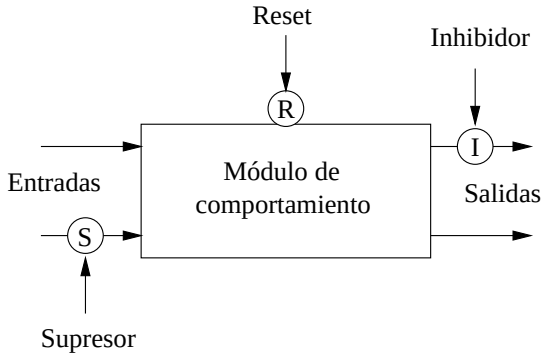

Figura 3.8: Máquina de estados finitos aumentada (adaptado de (Brooks, 1986)).

tamientos. Entre estos sistemas podemos destacar Colbert (Konolige, 1997), REX (Kaelbling, 1987), GAPPS (Kaelbling y Rosenchein, 1990), el Lenguaje L (Brooks y Rosenberg, 1995) y el Lenguaje del Comportamiento (*Behavior Language*) (Brooks, 1990a).

Arquitectura de subsunción La arquitectura de subsunción (Brooks, 1986, 1989, 1990b, 1991b,a) se considera como la primera y la más influyente de las arquitecturas de tipo reactivo y fue aplicada con gran éxito, con más o menos variantes, a multitud de pequeños robots (Brooks, 1990b): *Allen* (Brooks, 1986), *Genghis* (Brooks, 1989), *Atila*, *Herbert* (Connell, 1989, 1990), *Hannibal* (Ferrell, 1993), *Toto* (Mataric, 1992b), *Polly* (Horswill, 1993), etc.

En la mayoría de las implementaciones, los comportamientos se codifican directamente sobre hardware específico, o sobre pequeños microcontroladores, lo que permite que toda la computación se realice dentro del mismo robot. Las arquitecturas de subsunción fueron las primeras que lograron que los robots realizaran de forma eficaz y continua (sin pausas) las tareas más simples: andar, evitar colisiones, seguir contornos, etc.

Los comportamientos en la arquitectura de subsunción consisten en un conjunto de módulos de percepción de acción que son capaces de realizar una tarea. Se implementan por medio de máquinas de estados finitos aumentadas (AFSM, *Augmented Finite State Machines*, figura 3.8), es decir, autómatas de estados finitos que poseen contadores, registros y otras capacidades que les permiten interaccionar y comunicarse con otros módulos.

Los módulos se agrupan en capas de competencia que, a su vez, se organizan jerárquicamente: las inferiores incorporan las funciones básicas de supervivencia

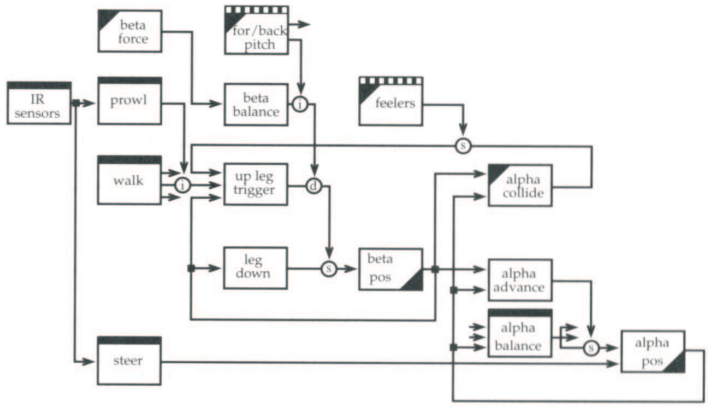

Figura 3.9: Arquitectura de control del robot *Genghis* (Brooks, 1989).

mientras que las superiores se encargan, progresivamente, de tareas de más alto nivel. Para ejecutar una tarea se activa la capa apropiada que se encarga de activar los módulos de las capas inferiores, y así sucesivamente. En caso de conflicto, los módulos de una capa superior pueden anular o subsumir las salidas e inhibir las entradas de los módulos de la capa inferior, de manera que siempre "gana" la capa activa más alta. La subsunción consiste en sustituir las salidas de un módulo inferior por las salidas de otro superior. En la figura 3.9 se puede ver una parte del sistema de control del robot *Genghis* (Brooks, 1989).

Quizás el mayor problema de las arquitecturas de subsunción es que, en la práctica, no son fáciles de reprogramar para realizar tareas distintas a aquellas para las que han sido diseñadas en un principio (Murphy, 2000). Esta falta de versatilidad y modularidad se debe a que los módulos superiores acceden a información interna de las capas inferiores, lo que los convierte en extremadamente dependientes de éstas últimas.

Evaluación

Las arquitecturas reactivas han sido diseñadas para ser extremadamente robustas y reactivas. Por un lado, los distintos Comportamientos se ejecutan en paralelo y de forma independiente. Por otro, tienen acceso directo a los datos sensoriales y pueden generar directamente comandos para los efectores con tiempos de retardo muy pequeños.

Son incapaces, sin embargo, de desarrollar comportamientos deliberativos o planificados, ya que no almacenan información sobre el mundo. Por lo tanto, única-

mente pueden mostrar comportamientos reactivos del robot.

La flexibilidad en las arquitecturas basadas en comportamientos es alta a nivel interno, es decir, dentro de cada Comportamiento. A nivel global los métodos de arbitraje y de fusión hacen que las arquitecturas sean bastante rígidas y poco dadas a las modificaciones.

Por último, en teoría, y por su propia organización, las arquitecturas reactivas deberían ser escalables. En la práctica el comportamiento emergente es más difícil de predecir y controlar conforme aumenta el número de Comportamientos y sus interacciones mutuas se tornan más complejas.

3.2.3 Arquitecturas híbridas planificadoras/reactivas

En los últimos años han aparecido nuevos modelos de arquitecturas que tratan de integrar los aspectos más positivos tanto de las arquitecturas jerárquicas como de las arquitecturas reactivas, ya que ambas son claramente complementarias y los defectos de unas se compensan con las virtudes de las otras. Estos modelos han supuesto en la práctica un cambio a un nuevo paradigma que se conoce con el nombre genérico de arquitecturas híbridas planificadoras/reactivas, aunque sus autores tienden a llamarlas deliberativas/reactivas.

Principales características

La característica identificativa de estas arquitecturas híbridas (figura 3.10) es que intentan conjugar en un mismo sistema dos componentes radicalmente distintos: uno planificador y otro reactivo. La parte reactiva se ocupa de la interacción con el entorno en tiempo real, de ejecutar y controlar los movimientos del robot y de adaptarse rápidamente, mediante un conjunto de respuestas preestablecidas, a las condiciones siempre cambiantes de un entorno dinámico y muchas veces impredecible.

La denominada parte planificadora incluye todo lo que no se considera reactivo. Es decir, funciones que requieren un conocimiento global del entorno (planificar tareas o rutas, crear mapas) o que necesitan conocer información "global" respecto al robot (y por tanto, fuera del alcance de funciones estrictamente reactivas): planificación de los comportamientos a usar, monitorización del rendimiento del sistema, diagnóstico y resolución de los problemas encontrados, etc.

En resumen, las arquitecturas híbridas se pueden definir como aquellas que siguen

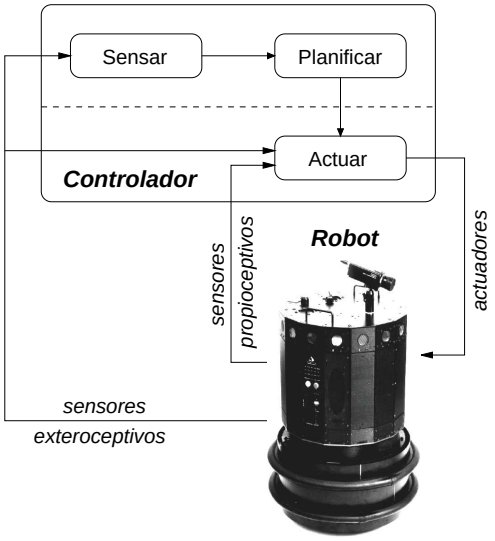

Figura 3.10: Esquema general de una arquitectura de control híbrido (Saffiotti, 1997a).

el modelo de *Planificar* primero y después *Sensar-Actuar*, tal como se muestra en la figura 3.11 o, de forma abreviada, (P, S-A). En estas arquitecturas *Planificar* consiste en instanciar o activar el conjunto necesario de Comportamientos (modelo S-A) necesarios para ejecutar una misión, una tarea o una parte de una tarea. El modelo (P, S-A) implícitamente supone que se cumplen dos hipótesis:

1. la planificación "cubre" un horizonte de tiempo amplio y requiere un conocimiento global, por lo que puede ser desacoplado de la ejecución en tiempo real. La planificación es interesante para fijar objetivos y seleccionar métodos, pero no para tomar decisiones de grano fino.

2. la planificación y el modelado global deben estar desacoplados de la ejecución en tiempo real, desde un punto de vista práctico, para no enlentecer los tiempos de reacción.

Una particularidad muy importante de los planes que se utilizan en las arquitecturas híbridas es que no constituyen un conjunto de órdenes estrictas que se deben cumplir ciegamente. Más bien son una guía de actuación que se debe interpretar y ejecutar en función de las condiciones del entorno y que, incluso, se puede modificar o actualizar mientras se están ejecutando. Es lo que se denomina planificación y ejecución reactiva (Agre y Chapman, 1990; Payton, 1990; Payton y col., 1990;

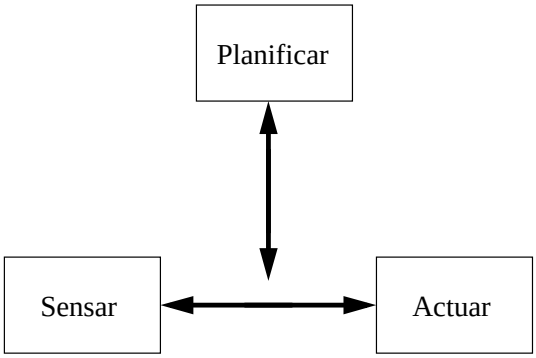

Figura 3.11: Relación entre las funciones robóticas primitivas en una arquitectura híbrida o de tipo (P, S-A).

Murphy y col., 1999). Cada arquitectura utiliza una técnica distinta para codificar e interpretar estos planes.

La mayor aportación de las arquitecturas híbridas consiste en los métodos de mediación entre sus componentes planificador y reactivo y que son los responsables de la integración de los diferentes tipos de comportamientos: planificado, deliberativo y reactivo. Normalmente esta mediación la realiza una capa intermedia que tiene dos funciones principales. En primer lugar, interpreta y ejecuta los planes generados en la capa planificadora coordinando y controlando los Comportamientos de la capa reactiva que son los realmente encargados de generar los movimientos del robot. Si surge algún problema, posee cierta capacidad para corregir y reorientar la ejecución del plan o, en casos extremos, invocar al planificador de la capa planificadora para que modifique el plan en marcha o genere un nuevo plan. En segundo lugar, la capa intermedia recoge todos los eventos relevantes que se generan en la capa reactiva y, en caso necesario, los reenvía a la parte planificadora para que ésta los procese.

Una última característica relevante es que la percepción en estas arquitecturas también es híbrida: por un lado cada Comportamiento sigue recibiendo los datos sensoriales y percibiendo únicamente lo que necesita; por otro lado, la planificación requiere modelos globales del entorno, que se construyen de forma independiente a los Comportamientos. Sin embargo, los componentes reactivos y planificadores pueden compartir sensores y percepciones: los Comportamientos pueden disponer de sensores virtuales que actúan sobre los modelos del entorno, y los modelos pueden incorporar las percepciones ya realizadas por los Comportamientos.

Ventajas

La principal ventaja de las arquitecturas híbridas planificadoras/reactivas es su organización en capas. Dicha organización facilita la abstracción tanto de datos como de tareas, de forma que las sucesivas capas manipulan información de más alto nivel y realizan tareas más generales según se sube en la estructura. Por otra parte, el uso de técnicas de procesamiento asíncronas (multitarea, hilos, etc.) permite la ejecución independiente tanto de los módulos reactivos como de los deliberativos y de planificación, cada uno con tiempos de ejecución muy diferentes.

Por último, la introducción de una interfaz entre capas permite que los detalles de la implementación de cada capa sean transparentes a las demás. Ello aumenta la modularidad del sistema y facilita que las arquitecturas híbridas se puedan adaptar mejor que los paradigmas anteriores a las diferentes aplicaciones. En definitiva, permite aprovechar las ventajas de las arquitecturas tanto reactivas como jerárquicas, así como reducir las limitaciones de ambas.

Limitaciones

La principal limitación de las arquitecturas híbridas es que no existe ningún tipo de especialización a nivel de capas. Esta carencia dificulta la implementación flexible y modular dentro de cada capa y, sobre todo, la ejecución distribuida de toda la arquitectura, lo que permitiría ampliar la capacidad computacional disponible y aumentar la robustez vía hardware.

Por otra parte, la falta de especialización también significa que, al añadir nuevas capacidades al sistema y descomponerlas en tareas más simples, aparecen cada vez más tareas para implementar en las capas inferiores lo que, a su vez, conlleva más interacciones entre tareas. El resultado final es que las capas inferiores se hacen cada vez más difíciles de implementar y su coordinación por parte de las capas superiores se vuelve más compleja y delicada. En definitiva, al aumentar el número de tareas a ejecutar las arquitecturas híbridas se vuelven menos eficientes y la reactividad también se puede resentir.

Estilos de arquitecturas híbridas

Las diferencias básicas entre las distintas arquitecturas híbridas surgen en tres áreas (Murphy, 2000):

1) ¿Cómo se distingue entre planificación, deliberación y reacción? Este aspecto

es crítico para construir con éxito una implementación orientada a objetos reutilizable. También determinará qué función se incluye en cada módulo, cuál será la información global y quiénes tienen acceso a esos datos globales.

2) ¿Cómo se organizan las responsabilidades en la parte planificadora? Una buena descomposición permitirá portabilidad y reutilización.

3) ¿Cómo emerge el comportamiento final? Los métodos utilizados en las arquitecturas reactivas (subsunción, campos de potencial, etc.) no son los mejores, por eso han aparecido otros nuevos asociados a este tipo de arquitecturas. Por ejemplo, los basados en votación (DAMN (Rosenblatt, 1997)), lógica borrosa (Saphira (Konolige y Myers, 1998)) o filtrado (SFX (Murphy y Mali, 1997)).

Siguiendo estos criterios, (Murphy, 2000) distingue tres estilos diferentes de arquitecturas híbridas: de tipo jerárquico (*managerial style*), de jerarquías de estados (*state hierarchies*) y orientadas a un modelo (*model-oriented style*).

De tipo jerárquico. Se reconocen por descomponer la capa planificadora según el ámbito de aplicación o la responsabilidad de cada función, siguiendo, por tanto, pautas muy similares a las arquitecturas jerárquicas, con las que comparten su estructura fuertemente jerárquica. En el nivel superior se planifica a alto nivel mientras que sus subordinados refinan dichos planes que, finalmente, se ejecutan en el último nivel, el reactivo. La implementación de este nivel reactivo es una de la principales diferencias con respecto a las arquitecturas jerárquicas. Cada nivel es responsable de ejecutar las directivas que le llegan y de identificar y solucionar localmente los problemas que surjan durante su ejecución. Sólo cuando no puede resolverlos pide ayuda al nivel superior, es lo que se conoce como "fallo hacia arriba". Los principales ejemplos de este estilo son las arquitecturas AuRA (*Autonomous Robot Architecture*) y SFX (*Sensor Fussion Efects*) (Murphy y Mali, 1997).

De jerarquías de estados. Utilizan el conocimiento sobre el estado del robot para distinguir entre las actividades reactivas, las deliberativas y las planificadoras según utilicen información PRESENTE, PASADA o FUTURA. Por lo tanto, suelen dividirse en tres capas donde cada una accede a las salidas de la capa superior y opera sobre las entradas de la capa inferior. Las arquitecturas denominadas de tres capas pertenecen a este estilo: 3T (Bonasso y col., 1995; Bonasso y Kortenkamp, 1996; Bonasso y col., 1997), ATLANTIS (Gat, 1992, 1993, 1998) y SSS (Connell, 1992).

Orientadas a un modelo. Su definición y clasificación es más nebulosa. Se suelen caracterizar por poseer comportamientos que acceden a porciones de un modelo del mundo (que actúa como un sensor virtual). En general, tienen una organización más *top-down* y son más simbólicas que el resto. El uso de un modelo del mundo parece un retorno a las arquitecturas jerárquicas (y conceptualmente lo es), pero existen cuatro diferencias prácticas. Primero, el modelo global monolítico suele ser menos ambicioso, más organizado y, normalmente, centrado en etiquetar sólo los objetos de interés (puertas, oficinas, vestíbulos, etc.). Segundo, la percepción es ejecutada de forma distribuida y asíncrona. Tercero, los errores e incertidumbres de los sensores se filtran mediante fusión sensorial a lo largo del tiempo. Cuarto, el incremento de la potencia de cálculo y la optimización de los compiladores ha mitigado el cuello de botella del procesamiento. Ejemplos de este estilo de arquitecturas híbridas son la arquitectura Saphira (Konolige y col., 1997; Konolige y Myers, 1998) y la arquitectura TCA (*Task Control Architecture*) (Simmons y col., 1997b,a; Koenig y Simmons, 1998).

Arquitecturas representativas

Recientemente han aparecido multitud de arquitecturas que propugnan su adscripción a este nuevo paradigma. Haremos una discusión más extensa de las arquitecturas que consideramos más representativas: AuRA (*Autonomous Robot Architecture*), 3T, Saphira y TCA (*Task Control Architecture*).

Existen muchas otras arquitecturas híbridas, pero entre todas ellas queremos destacar a SPOTT (Zelek, 1996; Zelek y Levine, 1996), AMOR (*Autonomous MObile Robot*) (Pirjanian y Blasvaer, 1994; Pirjanian y Christensen, 1995; Pirjanian, 1997), INTERRAP (Müller, 1996), ISR (Andersson y col., 1999) y *Animate Agent Architecture* (Martin y Firby, 1991; Firby y col., 1998). En (Arkin, 1998) se puede ver una muestra algo más extensa.

Arquitectura AuRA Muchos autores consideran que AuRA (Arkin, 1987, 1989b, 1990a) fue la primera arquitectura híbrida, ya que dicen que fue diseñada e implementada por Arkin al mismo tiempo que Brooks empezaba a publicar sus trabajos sobre la arquitectura de subsunción.

AuRA está basada en la teoría de esquemas y consta de cinco subsistemas (figura 3.12). Dos de ellos, el *Planificador* y el *Cartógrafo*, conforman la parte planificadora. El *Cartógrafo* incorpora todas las funciones relacionadas con la construcción de un

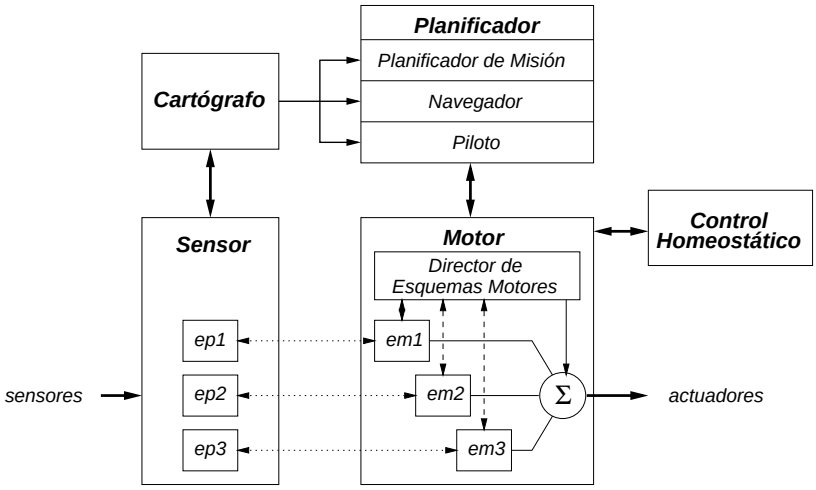

Figura 3.12: Esquema de la arquitectura AuRA donde se representan los cinco subsistemas que la componen (adaptado de (Arkin, 1990b,a)).

mapa del entorno, que será después utilizado por el *Planificador*. Éste se divide en los mismos componentes que la arquitectura NHC: *Planificador de Misión*, *Navegador* y *Piloto*. Las principales diferencias con NHC consisten en que *Navegador*, además de calcular las rutas para los desplazamientos, traduce cada segmento de la ruta en una subtarea. *Piloto* transforma cada subtarea en una lista de comportamientos cuya ejecución permite realizar el segmento de ruta correspondiente.

Todos los comportamientos del sistema se implementan en su parte reactiva. *Piloto* se encarga de indicarle al *Director de Esquemas Motores* qué comportamientos es necesario ejecutar y éste compone cada comportamiento a partir de los esquemas perceptuales que se incluyen en el subsistema *Sensorial* y los esquemas motores del subsistema *Motor*. Los esquemas motores sólo pueden representar acciones mediante campos de potencial, y el comportamiento final emerge a través de la suma de los vectores resultantes.

Sin embargo, y a diferencia de las arquitecturas reactivas, los comportamientos en AuRA no son únicamente de tipo reflejo, sino que también incluyen comportamientos que poseen memoria, representaciones y conocimiento específico. Además, los esquemas sensoriales y motores pueden compartir información a través de enlaces que establece el *Director de Esquemas Motores*.

El quinto subsistema de AuRA, el *Control Homeostático*, está entre la parte reactiva y la planificadora (figura 3.12). Su propósito consiste en modificar la relación entre los comportamientos en función de la "salud" del robot y otras consideraciones.

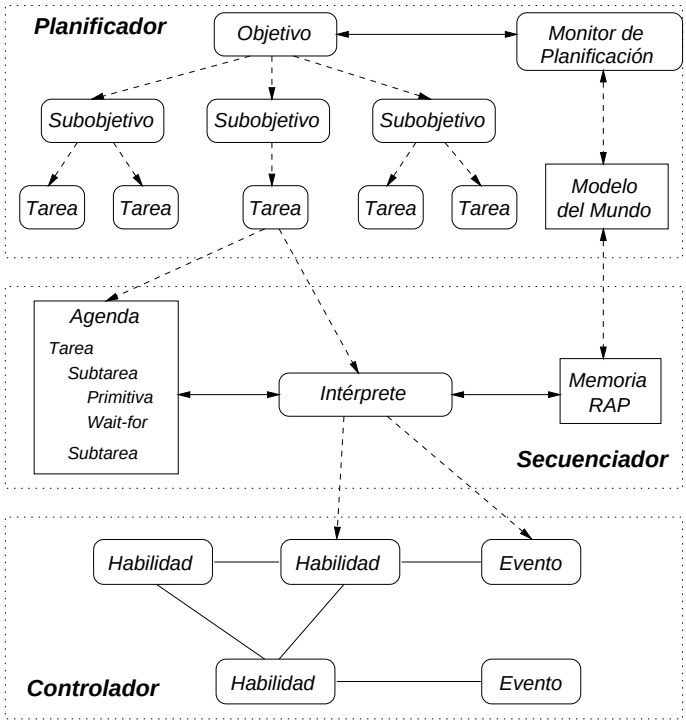

Figura 3.13: Elementos de la arquitectura 3T (Bonasso y col., 1995).

Está inspirado en la biología, donde parece que los animales modifican inconscientemente su comportamiento en todo momento como respuesta a sus necesidades internas. En la práctica, en AuRA esa modificación se traduce en la variación de los parámetros que establecen el campo de potencial con el que se define cada comportamiento.

Arquitectura 3T La arquitectura 3T (Bonasso y col., 1995; Bonasso y Kortenkamp, 1996; Bonasso y col., 1997) pertenece al conjunto de arquitecturas híbridas denominadas de tres capas, con las que comparte la mayoría de sus propiedades (Gat, 1998). Como su mismo nombre indica, su principal característica es que se dividen en tres capas (figura 3.13): una capa superior de planificación (*Planificador*), una capa inferior reactiva (*Controlador* o *Director de Habilidades*) y una capa intermedia que actúa de interfaz entre ambas (*Secuenciador*).

El *Planificador* se encarga de la planificación de las misiones y rutas, fija los objetivos y establece planes estratégicos. Los objetivos se pasan al *Secuenciador* que, utilizando técnicas de planificación reactiva (*Reactive Action Plan* o RAPs (Firby, 1989, 1995)), selecciona el conjunto de comportamientos primitivos y especifica la

secuencia en que se deben ejecutar para conseguir cada objetivo. En definitiva, el *Secuenciador* instancia una serie de comportamientos (o "habilidades") para ejecutar el plan. El cambio de nombre es para evitar confusiones con los comportamientos de las arquitecturas reactivas. Las habilidades son las que conforman la última capa de la arquitectura, el *Controlador*.

Un atributo importante del *Controlador* es que las habilidades tienen eventos asociados que sirven para comprobar si una acción ha conseguido el efecto buscado. Los eventos también se utilizan para indicar al *Secuenciador* si surge alguna dificultad o imprevisto. Si éste no es capaz de resolver el problema, entonces activa el *Planificador* para que establezca un nuevo plan.

Las tres capas representan, respectivamente, la planificación reactiva, la deliberación y el control reactivo. La organización es más en función del estado que de la responsabilidad. Así, el *Controlador* se compone de habilidades que sólo manejan información actual o PRESENTE (aunque se permite la persistencia de alguna información de estado). El *Secuenciador* opera tanto sobre datos del PRESENTE como del PASADO, por ejemplo recordando qué ha intentado el robot y qué ha conseguido realmente. Finalmente, el *Planificador* utiliza la información de estado, PRESENTE y PASADA, para predecir la FUTURA.

Sin embargo, en la práctica el reparto de las distintas funciones entre las tres capas está muy condicionada por los tiempos de ejecución: los algoritmos muy lentos se asignan al *Planificador* y los rápidos al *Controlador*. Por ejemplo, los algoritmos de visión, usualmente más lentos, se sitúan en la capa superior, a pesar de operar sobre datos sensoriales de bajo nivel.

Arquitectura Saphira La arquitectura Saphira (Konolige y col., 1997; Konolige y Myers, 1998) fue desarrollada en el SRI para controlar el robot *Flakey*, un descendiente de *Shakey*. Actualmente se vende junto con los robots *Pioneer* de ActivMedia y también es operativa en los robots *Khepera* y los de tipo *Bxx* de RWI (ahora ISR). En la figura 3.14 se pueden observar sus componentes principales.

Esta arquitectura se centra en tres aspectos claves para el correcto funcionamiento de un robot en el mundo real: coordinación, coherencia y comunicación (Konolige y Myers, 1998). La coordinación se refiere no sólo a nivel de actuadores y sensores sino que también incluye la coordinación de sus objetivos durante un periodo de tiempo. La coherencia es la habilidad para mantener un modelo global del mundo y Konolige y Myers la consideran esencial para el correcto funcionamiento de un robot

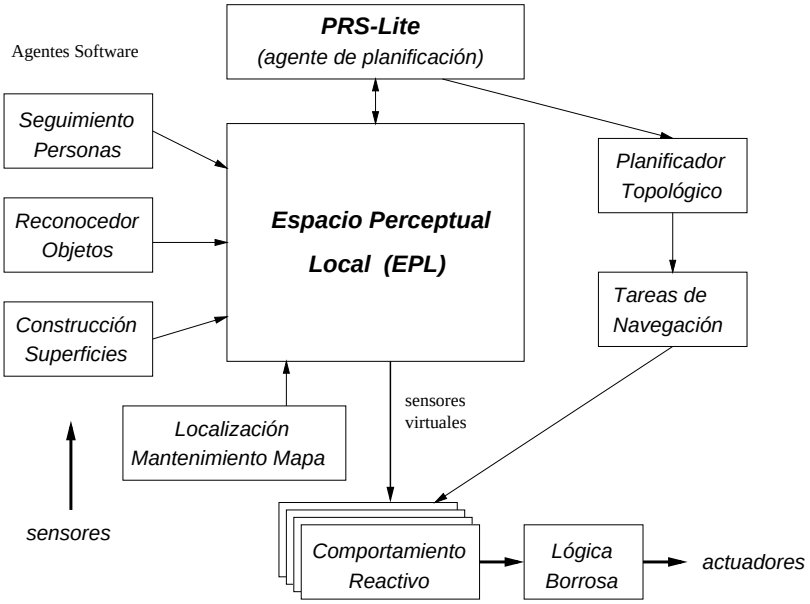

Figura 3.14: Componentes de la arquitectura Saphira (Konolige y Myers, 1998).

móvil (Konolige y Myers, 1998). Por último, la comunicación es vital en robots que interactúan con seres humanos y otros robots.

El grueso de la arquitectura es la parte concerniente a la planificación que corre a cargo de un planificador reactivo llamado PRS-lite (*Procedural Reasoning System*-lite (Myers, 1996)), una versión minimalista para su ejecución en tiempo real del más general PRS (Ingrand y col., 1992; Ingrand y Coutance, 1993; Ingrand y col., 1996). PRS-lite es capaz de entender sencillos comandos (incluso verbales) y de traducirlos a tareas de navegación y rutinas de reconocimiento perceptuales. Tanto la planificación como la ejecución se basan en un modelo central del mundo conocido como LPS (*Local Perceptual Space*) que se genera de manera incremental y progresiva a partir de los datos sensoriales, asignando etiquetas simbólicas a las diferentes regiones del entorno (corredores, puertas, intersecciones, etc.). Las tareas planificadoras en general, y las de percepción en particular, son independientes entre sí, por lo que se pueden ejecutar de forma distribuida y hacen que la arquitectura sea modular.

Al igual que en el resto de arquitecturas híbridas, la parte reactiva de Saphira está basada en comportamientos. La gran diferencia consiste en que las entradas de los comportamientos proceden del modelo central del mundo (LPS) a través de "sensores virtuales". Otra característica fundamental es que se utiliza la lógica borrosa para definir los comportamientos y para fundir los comandos generados por los distintos comportamientos activos en cada momento. Con esta técnica se

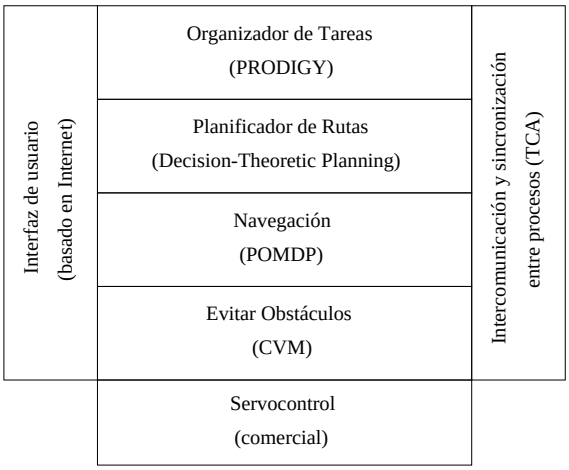

Figura 3.15: Arquitectura en capas basada en TCA (adaptado de (Simmons y col., 1997b; Koenig y Simmons, 1998)).

consigue un movimiento del robot y unas transiciones entre comportamientos más suaves, al mismo tiempo que permite integrar, dentro de un mismo esquema formal, los planes y el control basado en comportamientos (Saffiotti y col., 1995).

Una ventaja inmediata del LPS es que, gracias a la integración multisensorial tanto a lo largo del tiempo como del espacio, se minimizan los errores de las medidas sensoriales, lo que mejora la calidad del comportamiento final del robot. La contrapartida es el coste computacional que significa su mantenimiento. Sin embargo, gracias a los avances de la microelectrónica y a la cada vez mayor eficacia de los algoritmos utilizados, dicho coste se hace también cada vez más asumible.

Arquitectura TCA La arquitectura TCA (Simmons y Mitchell, 1989; Simmons, 1991, 1992, 1994) es la que controla el robot *Xavier* (Simmons y col., 1997b,a; Koenig y Simmons, 1998) y los robots de la NASA *Ambler* y *Dante*, entre otros. TCA, más que una arquitectura propiamente dicha, es un sistema operativo de alto nivel para robótica que provee las herramientas de control necesarias para la comunicación entre procesos, la descomposición y el secuenciamiento de tareas, la monitorización de la ejecución y el manejo de excepciones (Simmons, 1994).

En la figura 3.15 se puede ver la estructura en capas de TCA para realizar tareas de navegación. La capa de más alto nivel, el *Planificar de Tareas*, está basado en el planificador Prodigy y es el encargado de interaccionar con el usuario y determinar los objetivos y su orden de ejecución. Una vez fijada la tarea, el *Planificador de Rutas* calcula el mejor camino para llegar al destino a partir del modelo del mundo.

En TCA el modelo es un proceso de decisión de Markov parcialmente observable (*Partially Observable Markov Decision Process* o POMDP). *Navegación* utiliza el modelo del entorno para determinar, en cada momento, dónde está el robot, dónde ha estado y establecer la dirección que debe mantener el robot para seguir la ruta fijada.

La última capa, *Evitar Obstáculos*, recoge la dirección deseada que marca *Navegación* y trata de seguirla evitando los obstáculos que se encuentren a su paso. Para esta tarea utiliza el método curvatura-velocidad (*Curvature-Velocity Method* o CVM (Borenstein y Koren, 1991b)) que consigue que el robot trace trayectorias suaves mientras se adapta a los avatares del entorno. La percepción necesaria para detectar los obstáculos en tiempo real utiliza una rejilla de certidumbre basada en el método del histograma (Borenstein y Koren, 1991a), computacionalmente muy eficiente aunque algo impreciso en la determinación de espacio libre.

Evaluación

Las arquitecturas híbridas conjugan una parte reactiva y otra deliberativa-planificadora con el propósito de que las ventajas de una compensen las limitaciones de la otra. Algo similar ocurre con los criterios de evaluación propuestos: ambas partes se complementan.

En cuanto a la robustez se pueden contemplar dos aspectos. Por un lado, los comportamientos de la parte reactiva proporcionan la robustez y reactividad necesarias para operar en tiempo real en entornos dinámicos e impredecibles. Por otro, la planificación reactiva proporciona un método más flexible y robusto para operar en entornos abiertos y no totalmente modelables.

La reactividad de las arquitecturas híbridas es prácticamente la misma que en los sistemas reactivos, aunque quizás algo más lenta debido a la supervisión y el control que ejercen las capas superiores. A cambio, se consigue que las respuestas reactivas puedan ser modificadas o inhibidas según las circunstancias: el estado del sistema y del entorno, los objetivos a alcanzar, etc.

Por otra parte, la planificación reactiva está expresamente diseñada para adaptar la ejecución de los planes a las condiciones del entorno, lo que permite que la integración de los comportamientos reactivos, deliberativos y planificados sea sencilla e inmediata.

La flexibilidad de estas arquitecturas es mayor que en las jerárquicas pero sigue limitada debido a la fuerte dependencia con los modelos del entorno (aunque ya no

es uno sólo). Estas arquitecturas no están expresamente diseñadas para facilitar los cambios en el tipo de datos o en su procesamiento.

Por último, y al igual que ocurría con las arquitecturas jerárquicas y reactivas, se puede decir que la escalabilidad de estas arquitecturas no es muy alta. La razón principal es que, como no existe ningún tipo de especialización dentro de cada capa, cada vez que se añaden nuevas tareas se necesitan nuevos comportamientos en las capas inferiores. Este aumento hace que dichas capas se vuelvan más difíciles de implementar y, debido a las posibles interacciones indeseables entre comportamientos, su coordinación por parte de las capas superiores se torne más delicada. En definitiva, al aumentar el número de tareas a ejecutar, las arquitecturas híbridas se vuelven menos eficientes y su reactividad también se resiente.

3.3 Otros paradigmas software

Los paradigmas vistos en la sección anterior han sido específicamente diseñados para robótica, aunque algunos autores también han desarrollado arquitecturas de control para robótica a partir de paradigmas software más generales. Las más populares e interesantes son las arquitecturas basadas en pizarra y las arquitecturas basadas en agentes.

3.3.1 Arquitecturas basadas en pizarra

Principales características

Las arquitecturas basadas en pizarra (Nii, 1986b,a; Engelmore y Morgan, 1988; Jagannathan y col., 1989; Corkill, 1991; Carver y Lesser, 1992; Craig, 1995; Pfleger y Hayes-Roth, 1998a) se caracterizan, básicamente, por estar compuestas por un conjunto de módulos que comparten información a través de una memoria común, denominada pizarra, y estar gestionados por un único módulo de control.

En estas arquitecturas los módulos se conocen como bases de conocimiento (BC) ya que contienen una parcela independiente y separada del conocimiento dependiente de la aplicación. Su función consiste en chequear el estado de la pizarra para determinar cuando son aplicables, momento en el cual recogen sus datos de entrada de la pizarra, los procesan y devuelven los resultados obtenidos otra vez a la pizarra. Los nuevos datos pueden activar otras BCs y así, sucesivamente, los datos son procesados gradual e incrementalmente hasta llegar a la solución final. Como se puede

observar, la única interacción entre las BCs tiene lugar estrictamente a través de los datos de la pizarra, por lo que son muy independientes entre sí.

En la pizarra se almacenan, de forma estructurada, los datos necesarios para resolver el problema, normalmente organizados de forma jerárquica. Usualmente, dispone de una interfaz que se encarga de la interacción con las bases de conocimiento y con el módulo de control.

El módulo de control es el encargado de dirigir, supervisar y coordinar el funcionamiento de todo el sistema. La iniciativa de control posee una componente oportunista y otra planificada. La primera se manifiesta en el hecho de que las BC se activan cuando los cambios en los datos de la pizarra las hacen aplicables, es decir, su ejecución depende del estado del sistema. La planificación tiene lugar en el módulo de control donde finalmente se decide qué bases de conocimiento activas se deben ejecutar en función de los objetivos del sistema.

Ventajas

Las principales ventajas de las arquitecturas basadas en pizarra son su gran flexibilidad y su capacidad de adaptación. La adaptabilidad se manifiesta en varios niveles. En primer lugar, el procesamiento de los datos puede ser realizado utilizando distintas bases de conocimiento, según la información disponible y los criterios de rendimiento elegidos. En segundo lugar, si las tareas de razonamiento se implementan mediante BCs, su activación también depende de las condiciones percibidas e inferidas del entorno y de los objetivos demandados (que también pueden formar parte de la pizarra).

En tercer lugar, a través del módulo de control es posible modificar el orden de ejecución de los módulos activados, regulando de esa manera el grado de reactividad y de planificación del sistema. Como el módulo de control también tiene acceso a la pizarra, la regulación depende de las restricciones que impone cada objetivo y de la incertidumbre sobre el entorno.

La flexibilidad de estas arquitecturas también se observa en varios puntos. Es muy sencillo modificar el tipo de datos que se utilizan para resolver el problema o el tipo de representación utilizado. Cambiar el procesamiento de la información sólo consiste en sustituir o modificar las BCs apropiadas que únicamente necesitan mantener la misma interacción con la pizarra, es decir, los mismos datos de entrada y de salida. Por último, no existe ninguna restricción en la implementación de las BCs, por lo que cada una se puede implementar mediante técnicas diferentes: lógica

borrosa, redes neuronales, sistemas expertos, etc.

Respecto al control, como la activación de las bases de conocimiento depende de los datos y su ejecución es dirigida por el módulo de control, en estas arquitecturas es inmediata la integración de los comportamientos dirigidos por datos (p.e., los reactivos) y los dirigidos por objetivos (p.e., los planificados y los deliberativos).

Finalmente, reseñar que la centralización del control, generalmente a través de un plan de control explícito, simplifica la supervisión y coordinación entre los módulos, permite ejecutar tareas globales de manera más eficiente y obtener un comportamiento final del robot más coherente y armonioso.

Limitaciones

El principal inconveniente de las arquitecturas basadas en pizarra en su aplicación a la robótica móvil es la dificultad para conseguir bajos tiempos de reacción. La razón es que, aunque la activación de las BCs dependa de los datos, el control está centralizado y su funcionamiento suele ser lento y complejo, ya que normalmente se almacenan en colas las BCs activadas. En algunos sistemas se implementan mecanismos especiales para evitar que la activación de una reacción pase por el módulo de control (p.e., lazos reactivos (Liscano y col., 1995)). El problema es que, al quedar excluidas del ámbito del módulo de control, éste tampoco puede alterar su funcionamiento cuando así lo requieren las circunstancias (p.e., inhibiéndolas).

Otra limitación importante es el cuello de botella que se produce tanto en la pizarra como en el control, ya que todos los intercambios del sistema, bien de datos bien de información de control, tienen lugar en esos dos módulos. El problema se hace más evidente conforme aumenta el tamaño y la complejidad del sistema.

Ejemplos representativos

Las arquitecturas basadas en pizarra han sido ampliamente utilizadas en aplicaciones que requieren complejas interpretaciones en el procesamiento de señales, tales como el reconocimiento de voz y de patrones.

Debido a las interesantes propiedades de las arquitecturas basadas en pizarra, desde su aparición, son muchos los autores que las han aplicado (Shafer y col., 1986; Pang y Shen, 1990; Smieja y Beyer, 1994; Xu y van Brussel, 1997; Doty y Bou-Ghannam, 1995; Gómez Skarmeta y col., 1998; Liscano y col., 1995; Fayek, 1992; Fayek y col., 1993; Tigli y col., 1993), con mayor o menor éxito, al ámbito

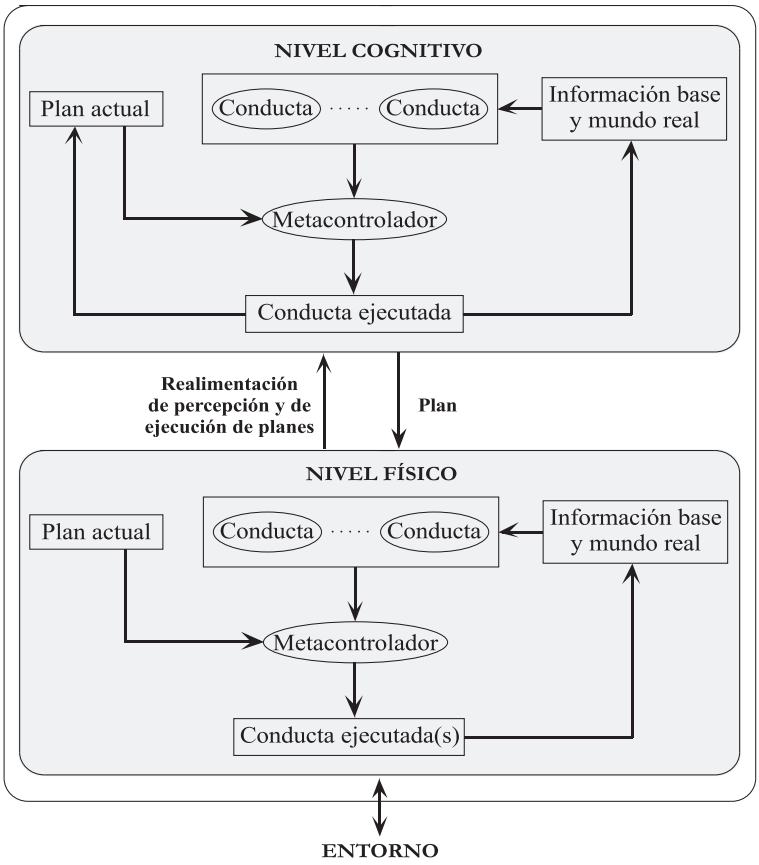

Figura 3.16: Esquema interno de la arquitectura AIS (Roth y col., 1995).

de la robótica móvil. En este apartado veremos una de las implementaciones más interesantes: la arquitectura AIS (*Adaptive Intelligent System*) (Hayes-Roth, 1995).

Arquitectura AIS La arquitectura AIS (*Adaptive Intelligent System*) (Hayes-Roth, 1995; Roth y col., 1995) está especialmente diseñada para operar en tiempo real en entornos dinámicos, complejos y, a menudo, incompletamente especificados. Se ha utilizado en monitorización inteligente en una unidad de cuidados intensivos (Hayes-Roth y col., 1992) y en tareas de vigilancia y mensajería con un robot móvil (Hayes-Roth y col., 1995; Pfleger y Hayes-Roth, 1998a).

La arquitectura AIS se compone de dos capas (figura 3.16): el nivel físico que se encarga de interaccionar con el entorno, y el nivel cognitivo responsable del razonamiento de alto nivel, como la planificación y la resolución de problemas. En caso de necesitar una resolución más fina, AIS se podría dividir en más capas, pero en principio dos son suficientes. Aunque siempre siguiendo una estructura jerárquica lineal

sin ningún tipo de especialización o "ramificación" y con la misma estructura interna para todas las capas. Concretamente, el nivel cognitivo se implementa mediante la arquitectura de pizarra BB1, mientras que el nivel físico utiliza una implementación más simple y rápida del mismo modelo.

Como se puede ver en la figura 3.16, la capa cognitiva es la encargada de generar los planes de control necesarios para ejecutar las tareas pedidas, planes que envía a la capa física para su ejecución. Por su parte, la capa física recoge los datos sensoriales, los procesa y traduce los planes de control en comandos a los efectores. Esta información perceptual y los resultados de la ejecución de las acciones se envían al nivel cognitivo.

En esta arquitectura los planes de control se consideran meras descripciones sobre el desarrollo deseado del comportamiento del robot (Pfleger y Hayes-Roth, 1996, 1998b). Por lo tanto, los comportamientos reactivos, deliberativos y planificados pueden coexistir dentro de cada capa. La diferencia entre niveles es esencialmente epistemológica y temporal: relativa al tipo de razonamiento utilizado (simbólico en el nivel cognitivo y métrico en el físico), al tiempo de reacción (más pequeño en el nivel físico que en el cognitivo), al grado de abstracción (mayor en el nivel cognitivo), etc.

Evaluación

Las arquitecturas basadas en pizarra no son excesivamente robustas, ya que poseen una enorme dependencia con la pizarra y el módulo de control; si uno de los dos elementos deja de funcionar, todo el sistema se colapsa. A nivel de bases de conocimiento el sistema sí es muy robusto: en primer lugar porque, en general, las BCs son redundantes; en segundo lugar, porque aunque falle una BC el sistema no se paraliza, simplemente pierde la funcionalidad de esa BC.

La reactividad es posible puesto que las BCs se activan a partir de los datos de la pizarra (datos sensoriales incluidos). Sin embargo, en las implementaciones usuales el tiempo de respuesta suele ser excesivamente grande. Se puede mejorar prestando especial atención al diseño del módulo de control para agilizar la ejecución de las BCs activadas y eliminar tiempos de espera innecesarios y potencialmente nocivos para la reactividad.

De todos modos, la reactividad en estas arquitecturas siempre estará limitada porque la activación de la respuesta reactiva debe estar supervisada por el módulo de control. La ventaja de esta supervisión es que, de esta manera se flexibilizan las

reacciones del sistema y se mantienen siempre bajo el control de los comportamientos deliberativos y planificados. Existe, por tanto, un claro compromiso entre tiempo de respuesta y "controlabilidad" de la reacción.

La integración de los distintos tipos de comportamientos se realiza de forma natural en el módulo de control, cuya función principal consiste en establecer qué bases de conocimiento activadas por los datos (comportamientos reactivos) se ejecutan finalmente en función de los objetivos del sistema (comportamiento planificado y deliberativo).

La flexibilidad de las arquitecturas basadas en pizarra es muy alta. En primer lugar, la independencia que existe entre las diferentes BCs permite que cualquier modificación en alguna de ellas sea transparente para el resto de la arquitectura. Sólo se necesita mantener la misma interfaz de interacción tanto con la pizarra como con el módulo de control. En segundo lugar, como todos los intercambios de datos entre las BCs se centralizan en la pizarra, se simplifican en gran medida tanto la modificación como la inclusión de nuevos datos y tipos de representación.

Por último, las arquitecturas basadas en pizarra no fácilmente son escalables, ya que, conforme se añaden nuevos datos, bases de conocimiento y tareas, se agudiza el problema del cuello de botella que se crea tanto en la interfaz de la pizarra como en el módulo de control.

3.3.2 Arquitecturas basadas en agentes

Principales características

Las arquitecturas basadas en agentes se caracterizan por una gran autonomía e independencia entre los distintos módulos o agentes que las componen. Los agentes interaccionan entre sí a alto nivel, utilizando complejos protocolos de comunicaciones (p.e., paso de mensajes o sistemas de contratos). Por norma general, los protocolos son asíncronos y aseguran que la estructura física de la red sea transparente para las comunicaciones.

Normalmente los datos y el control se encuentran totalmente distribuidos entre los distintos agentes del sistema y no existe ningún control centralizado en la arquitectura. La ejecución de las tareas es posible porque los distintos agentes se organizan y coordinan entre sí.

Ventajas

La ventaja más evidente de las arquitecturas basadas en agentes es que se pueden ejecutar de forma natural en diferentes procesadores. La ejecución distribuida facilita acceder a una mayor cantidad de recursos, sobre todo capacidad de cálculo, y permite un aprovechamiento más eficaz (p.e., balanceando la carga computacional).

La independencia entre los agentes trae consigo varias ventajas. La primera es que el sistema es muy robusto, ya que si falla un agente el sistema no se colapsa y únicamente se pierde la funcionalidad del agente en cuestión. La segunda es que los agentes se pueden implementar de forma heterogénea ya que las comunicaciones hacen que los detalles de las implementaciones permanecen ocultos al resto del sistema.

Limitaciones

Los principales inconvenientes de las arquitecturas basadas en agentes provienen de su naturaleza dispersa. La independencia entre los distintos módulos y la falta de un control central dificulta la ejecución de tareas globales y la obtención de un comportamiento final coherente. Para evitarlo, algunas implementaciones se separan de los principios fundamentales de estas arquitecturas y centralizan la información de control (p.e., mediante una pizarra).

Por otra parte, señalar que la selección de un protocolo de comunicación es extremadamente complejo y difícil, sobre todo si se tiene en cuenta que teóricamente debería ser común a todos los agentes de la arquitectura. Y no conviene olvidar que los agentes involucrados pueden ser de naturaleza y grado de abstracción muy diferentes.

Por último, comentar que a pesar de la teórica independencia entre los agentes, en la práctica un agente debe poseer cierto conocimiento sobre cada agente con el que interacciona. Como mínimo debe conocer su funcionalidad, es decir, lo que puede y lo que no puede hacer y cómo se puede dialogar con él. También se puede incluir información más detallada: datos de entrada, salidas que produce, recursos que necesita, tiempo aproximado de ejecución, precisión y grado de fiabilidad de los resultados, etc. Esta información crea cierta dependencia entre los agentes.

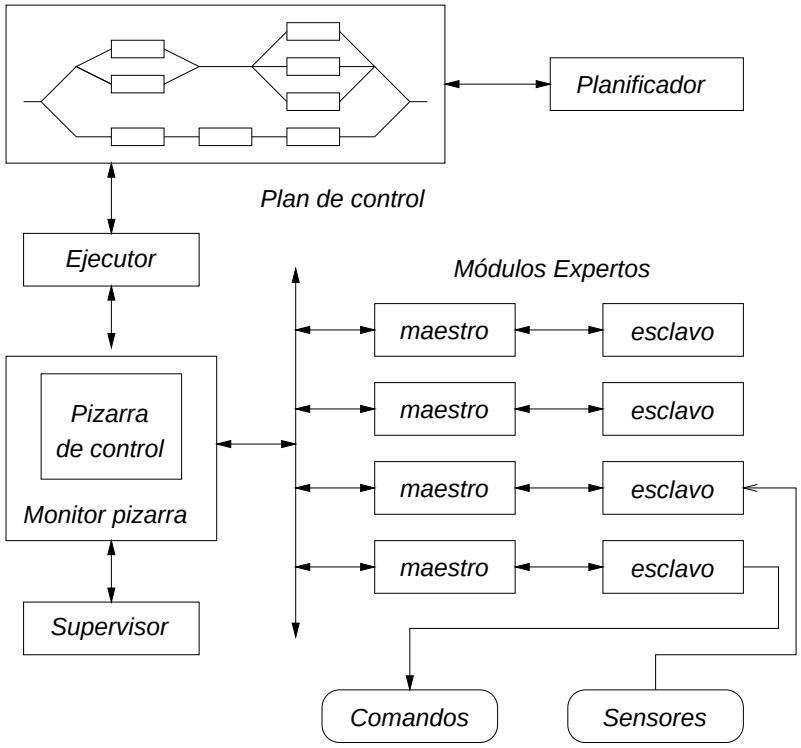

Figura 3.17: Esquema de la arquitectura DCS (Elfes, 1986).

Ejemplos representativos

No existen muchas implementaciones "puras" de arquitecturas basadas en agentes aplicadas a la robótica móvil. Sin embargo, sí hay alguna que se acerca bastante, p.e., la arquitectura LICA (*Locally Intelligent Control Agent*) (Hu y Brady, 1998) y la DCS (*Distributed Control System*) (Elfes, 1986, 1991).

Arquitectura DCS DCS (*Distributed Control System*) (Elfes, 1986, 1991) está formada por una comunidad de módulos expertos que se comunican por paso de mensajes y que se pueden ejecutar en varios procesadores. Una peculiaridad de esta arquitectura es que los módulos expertos comparten la información de control a través de una pizarra. Un módulo, denominado ejecutivo, extrae la información del plan de control (subtareas y restricciones en su ejecución) y se la facilita al resto de módulos expertos a través de esa pizarra.

Los módulos expertos son los encargados de monitorizar los sensores y controlar los actuadores, interpretar los datos sensoriales, construir el modelo interno del entorno del robot, planificar estrategias para ejecutar tareas propuestas, supervisar

la ejecución del plan de control y administrar el sistema en su conjunto. Cada módulo experto consta de un proceso maestro y de un proceso esclavo que se ejecutan en paralelo. El primero controla la ejecución del segundo, actúa de interfaz con el resto de módulos, extrae información de la pizarra y guarda en ella los resultados que produce el esclavo. El proceso esclavo es el que realmente procesa los datos.

La ventaja de DCS respecto a una arquitectura tradicional basada en pizarra es que los módulos expertos intercambian datos directamente entre sí, sin pasar por la pizarra. El intercambio de información entre módulos expertos reduce el problema de las comunicaciones pero aumenta la dificultad a la hora de especificar y descomponer una tarea es subtareas más simples, así como el control de su ejecución.

La principal diferencia de DCS respecto a una arquitectura clásica basada en agentes es que utiliza una pizarra para compartir la información de control, lo que conlleva ventajas e inconvenientes: simplifica la coordinación entre los módulos expertos y permite la ejecución eficiente de tareas globales a costa de enlentecer las comunicaciones y las acciones de tipo reactivo. Aunque la pizarra se divide en varios niveles de abstracción, no existe ningún tipo de especialización en la misma que favorezca la escalabilidad de la arquitectura.

Finalmente, mencionar que la arquitectura DCS no tiene ningún soporte para implementar comportamientos reactivos ni tampoco se especifica cómo se pueden integrar los comportamientos planificados y los reactivos. Parece que todo ello sólo es posible a nivel local, dentro de cada módulo experto.

Evaluación

Las arquitecturas basadas en agentes son robustas en el sentido de que pueden existir múltiples agentes para realizar una misma función. En caso de fallar uno de los agentes, como mucho se pierde su funcionalidad. El talón de Aquiles de estas arquitecturas son las comunicaciones: si fallan todo el sistema queda inutilizado.

Al igual que ocurre en la arquitectura DCS, las arquitecturas basadas en agentes no tienen un soporte específico para implementar la reactividad, que parece restringirse al ámbito local de cada agente. Tampoco existe soporte para la integración de los comportamientos planificado, deliberativo y reactivo, salvo la colaboración normal entre los agentes de la arquitectura.

Las arquitecturas basadas en agentes son flexibles, en el sentido de que los agentes pueden ser modificados y que los protocolos de comunicaciones suelen ser abiertos. Ahora bien, los nuevos agentes y las variaciones en el protocolo deben ser “compati-

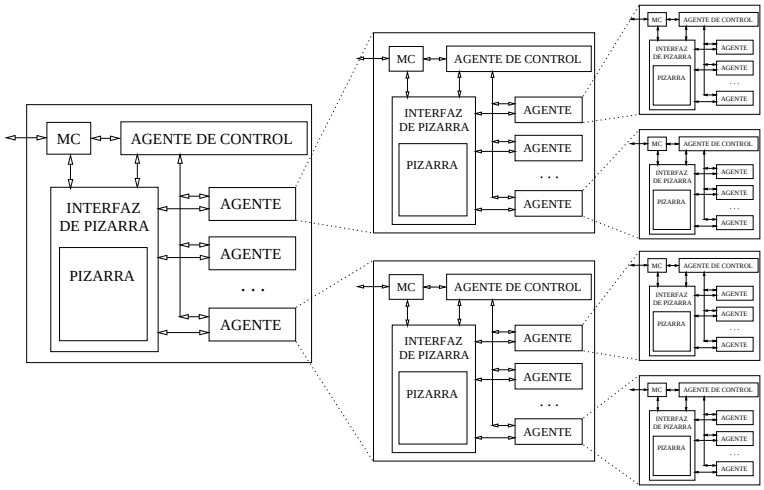

Figura 3.18: Esquema de la arquitectura fractal de especialistas.

bles" con los ya existentes, lo que limita en gran medida las posibles modificaciones.

La escalabilidad de estas arquitecturas es reducida en la práctica, ya que añadir nuevos agentes implica aumentar el protocolo de comunicaciones y hacerlo más complejo. El problema consiste en que, en conjunto, cada agente debe almacenar más información sobre el funcionamiento de los demás agentes del sistema para poder interaccionar con ellos.

3.3.3 Una nueva propuesta: arquitectura basada en especialistas

Como se ha podido observar, la mayoría de las arquitecturas tienen inconvenientes cuando se utilizan para implementar el sistema de control de un robot móvil autónomo de "propósito general", algunas de ellas muy importantes. En esta memoria se propone un nuevo modelo de arquitectura basada en especialistas que trata de incidir sobre las limitaciones apuntadas en cada una de las arquitecturas que se han utilizado hasta ahora en robótica móvil.

Principales características

La idea principal de la arquitectura basada en especialistas (Fraga y col., 1998; Lama y col., 1999; Lama, 2000; Regueiro y col., 1999, 2002) consiste en "encapsular" en bloques denominados especialistas aquellas tareas estrechamente relacionadas entre sí, junto con el control y los datos necesarios para su ejecución. Cada especialista

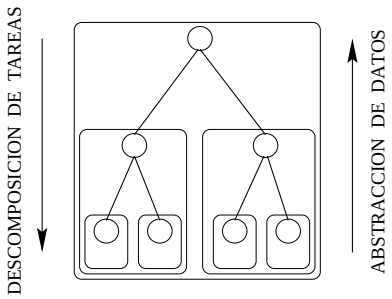

Figura 3.19: Ilustración de la descomposición jerárquica de tareas y la abstracción de datos en la arquitectura basada en especialistas.

es responsable de ejecutar y controlar las tareas que le han sido encomendadas, de procesar los datos necesarios para realizar dichas tareas y, en general, de resolver los problemas que surjan durante la ejecución de las mismas.

Cada especialista puede constar de varios módulos internos o agentes (figura 3.18) diseñados para sintetizar algunas de sus funciones o competencias, es decir, ejecutar y controlar algunas de sus subtareas y procesar parte de sus datos. A su vez, un agente se puede configurar como un especialista, con sus propios agentes a nivel interno, lo que significa que el esquema de la arquitectura se puede replicar en cada agente y, en ese sentido, también la denominaremos arquitectura fractal de especialistas (AFE). La referencia a especialistas o agentes será en función del punto de vista seguido en cada caso.

Los especialistas y sus agentes forman una estructura en la que se parte de una organización jerárquica de las tareas, donde las tareas complejas se van descomponiendo progresivamente y de manera natural conforme se va profundizando (figura 3.19), hasta que son suficientemente simples para que agentes de los niveles inferiores puedan ejecutarlas. Recíprocamente, los datos asociados a las tareas se vuelven progresivamente más abstractos en sentido contrario. El resultado final es una organización en árbol que se puede ir refinando según las necesidades (p.e. tareas complejas), dando lugar a una arquitectura de tipo jerárquico en varios niveles funcionales y de abstracción.

Al igual que en las arquitecturas híbridas planificadoras/reactivas se entiende que los especialistas de los niveles superiores se encargan de las tareas de más alto nivel, utilizando principalmente información abstracta de tipo simbólico, mientras que los especialistas de los niveles inferiores, más próximos a los sensores y efectores, operan de una manera más reactiva y con un procesamiento más simple y específico

de los datos.

Cada especialista se puede implementar de forma diferente, aunque, en aras a una mayor uniformidad y simplicidad del sistema, se utiliza la misma arquitectura para todos ellos. Como dicha arquitectura debe ser flexible y facilitar la adaptación de cada especialista a sus requisitos y necesidades, se ha elegido una arquitectura basada en pizarra (figura 3.18), aunque con dos diferencias fundamentales. La primera es que la pizarra de un especialista en vez de ser un mero almacén de datos, más o menos estructurado, es un sistema activo capaz de generar **eventos** cada vez que se actualiza la información que contiene. Estos eventos son fundamentales para mejorar tanto la reactividad del especialista como la del sistema en su conjunto. La segunda diferencia es que los agentes de un especialista se pueden implementar a su vez como especialistas, dando lugar a que la misma arquitectura se replique sea cual sea el nivel de detalle, hasta el nivel inferior, en el cual los especialistas son simples procesos independientes, que denominamos agentes terminales.

Como en cualquier arquitectura basada en pizarra, el agente de control de un especialista (figura 3.18) controla y supervisa el funcionamiento de todos sus agentes sobre los cuales delega parte de sus tareas y funciones mediante órdenes de tarea. El agente de control debe responder a las órdenes de tarea que le llegan de su especialista superior y a los eventos, tanto internos (generados en su propia pizarra) como externos (generados por otros especialistas), que afectan al comportamiento del especialista. Por lo tanto, es el encargado de integrar la respuesta (comportamiento reactivo) con el resto de tareas en curso (comportamiento planificado) para generar un comportamiento global coherente. Como se puede observar, y al igual que en las arquitecturas clásicas basadas en agentes, una característica interesante de AFE es que la actividad se puede iniciar en cualquier agente/especialista del sistema.

Otra característica importante de AFE es que los especialistas dialogan mediante paso de mensajes, es decir, utilizando protocolos de comunicaciones de alto nivel, flexibles e independientes de la implementación de cada especialista. En un especialista las comunicaciones se dividen en dos categorías: internas y externas. Las primeras consisten en el intercambio de datos que realizan todos los agentes (incluido el agente de control) con la pizarra y de información de control con el agente de control (figura 3.18).

Todas las comunicaciones con el exterior de un especialista se canalizan a través del módulo de comunicaciones (MC en la figura 3.18). Para mantener la estructura jerárquica de las tareas y de la arquitectura, el intercambio de información de control

únicamente tiene lugar entre un especialista y el agente de control de su especialista superior (para el cual es un agente más). Sin embargo, para aumentar la flexibilidad de todo el sistema, el intercambio de datos no se ve restringido por la jerarquía de tareas y cualesquiera dos especialistas pueden intercambiar directa y libremente los datos de sus pizarras a través de sus MCs.

Ventajas

Las ventajas más importantes de la arquitectura basada en especialistas tienen su origen en tres cualidades: la especialización, es decir, la encapsulación de tareas, datos y control en especialistas; la implementación de los especialistas alrededor de una pizarra; y la comunicación entre especialistas mediante paso de mensajes.

Como ventajas de la especialización se pueden mencionar las siguientes:

1. Hace transparente para el resto del sistema los detalles del diseño, implementación y ejecución de las tareas asignadas a un especialista y facilita la implementación independiente de cada uno. Cada especialista se puede definir mediante su funcionalidad (las tareas que puede realizar) y sus datos de entrada y de salida, lo que constituye su "interfaz".

2. Aumenta la modularidad del sistema y permite su implementación utilizando métodos avanzados de programación del tipo orientado a objetos y, por tanto, la reutilización del código.

3. Procura una mayor estructuración en el diseño de la arquitectura, de manera que se acomoda de forma directa a la existencia de una jerarquía, tanto a nivel de tareas como de datos. La organización jerárquica facilita el diseño y la síntesis de tareas complejas y reduce las necesidades de intercambio de información de control.

4. Simplifica el diseño del control, ya que la ejecución y el control de las tareas se distribuye entre los especialistas.

5. Mejora la respuesta reactiva del sistema y la focalización de los procesos perceptuales gracias a que los datos y el control relacionados con cada tarea se "encapsulan" en un mismo especialista.

6. Minimiza las comunicaciones puesto que, en general, el "micromundo" que afecta y que es afectado por un especialista está integrado en él, lo que reduce

considerablemente la necesidad de intercambio de información, tanto de datos como de control.

La mayoría de las ventajas de utilizar una arquitectura basada en pizarra son conocidas, aún así podemos destacar las siguientes:

1. Permite la implementación heterogénea de los agentes de cada especialista y "mezclar" en un mismo especialista distintas técnicas: sistemas basados en conocimiento, redes neuronales, lógica borrosa, etc.

2. Aumenta la flexibilidad del sistema, ya que se facilita la inclusión de nuevos datos y agentes y la modificación de los ya existentes. Esta cualidad es muy importante en dominios donde el desarrollo de nuevos métodos y algoritmos es continuo y donde la información aún no está claramente estructurada ni las dependencias completamente establecidas. La flexibilidad también incluye al agente de control, que puede ser reimplementado con diferentes técnicas de planificación reactiva y de lenguajes de control: PRS-Lite, RAP, Colbert, etc.

3. Mejora la robustez del sistema, ya que si un agente deja de operar, o lo hace incorrectamente, el especialista no deja de funcionar, simplemente pierde la funcionalidad asociada a dicho agente.

4. Facilita la reutilización del código, puesto que todos los especialistas se implementan siguiendo el mismo patrón software.

5. Simplifica la integración, en todos los niveles de la arquitectura, de los comportamientos reactivos, deliberativos y planificados que se desarrollan dentro de cada especialista. Como ya se ha comentado, esta tarea se realiza de manera natural en el agente de control de cada especialista.

La utilización de protocolos basados en paso de mensajes para las comunicaciones entre especialistas conlleva varias ventajas:

1. La implementación independiente de cada agente/especialista, del cual únicamente es necesario conocer su "interfaz" y no los detalles de su implementación.

2. La ejecución distribuida de la arquitectura, lo que permite acceder a una mayor potencia de cálculo y procesar varias tareas en paralelo, incluso de forma redundante. Todo ello abunda en una mayor robustez y en un menor tiempo de respuesta para todo el sistema.

3. Facilita la utilización de diferentes plataformas lógicas y de programación, tanto para la ejecución como en el desarrollo de la arquitectura.

4. Junto con la especialización, aumenta la escalabilidad de la arquitectura.

Limitaciones

La organización jerárquica de la arquitectura AFE impone cierta rigidez al sistema, limitación que, como se verá en lo sucesivo, se ve ampliamente compensada con las ventajas que ofrece.

Por otra parte, y al igual que las arquitecturas basadas en agentes, señalar que las comunicaciones pueden condicionar el rendimiento de la arquitectura y convertirse en un cuello de botella. Para evitarlo, el protocolo de comunicaciones debe ser rápido y sencillo.

Evaluación

La arquitectura basada en especialistas es, en su conjunto, robusta, porque si falla uno de los especialistas o uno de sus caminos de comunicación, el sistema sigue funcionando y únicamente pierde la funcionalidad del especialista afectado. Sin embargo, a nivel de especialista existe una fuerte dependencia con su única pizarra y su único módulo de control, lo que reduce, en gran medida, su robustez. En cuanto a las comunicaciones, en AFE se implementan de forma totalmente distribuida y se centran mayoritariamente en los intercambios entre un especialista y su especialista superior. Esta preferencia por los intercambios punto a punto simplifica las comunicaciones y aumenta su robustez.

Respecto a la reactividad, en AFE se ha prestado especial atención al diseño e implementación del módulo de control para evitar los inconvenientes habituales que existen en las arquitecturas basadas en pizarra. La gran ventaja de la especialización es que permite una mayor "focalización de la atención" tanto en la detección de los eventos de interés dentro de cada especialista como en la generación de la respuesta adecuada. La "parcelación" de la información sobre el mundo en distintos especialistas no afecta a la reactividad del sistema en su conjunto, porque se han implementado mecanismos que permiten que cualquier evento detectado en un especialista de AFE se pueda reenviar al especialista en donde se genera la respuesta.

La integración de los comportamientos planificado, deliberativo y reactivo es muy similar a la que tiene lugar en una arquitectura de pizarra y, por lo tanto,

es más flexible que los métodos utilizados en las arquitecturas híbridas planificadoras/reactivas actuales. Otra ventaja de AFE es que la integración tiene lugar dentro de cada especialista y en todos los niveles de la arquitectura. De nuevo, este "reparto" de las funciones de integración de los comportamientos simplifica su diseño e implementación.

A nivel de especialistas la flexibilidad de AFE es muy similar a las arquitecturas basadas en pizarra. Sin embargo, esa misma flexibilidad se refleja en el conjunto de la arquitectura ya que, no lo olvidemos, en AFE cada especialista no deja de ser un agente más de su especialista superior. Por lo tanto, la flexibilidad de AFE es superior tanto a las arquitecturas basadas en agentes como a las arquitecturas híbridas planificadoras/reactivas.

De todas las propiedades de AFE la que sin ninguna duda destaca más es su alto grado de escalabilidad, producto de varias decisiones de diseño de la arquitectura. En primer lugar, la encapsulación en especialistas provoca una división jerárquica de la arquitectura que reduce considerablemente la necesidad de intercambios y mitiga, en gran medida, el cuello de botella de las comunicaciones. En segundo lugar, la implementación recursiva de los agentes/especialistas mediante arquitecturas basadas en pizarra permite una implementación regular, independiente y heterogénea de todos sus componentes. En tercer lugar, la utilización de comunicaciones basadas en paso de mensajes facilita la deslocalización de los especialistas, la independencia respecto a las plataformas de computación y, por consiguiente, la ejecución distribuida de la arquitectura. De esta forma, y gracias a la organización jerárquica y recursiva de la arquitectura, la inclusión de nuevos especialistas no afecta significativamente al rendimiento del sistema.

Comparativa

A lo largo de este capítulo se han descrito los principales paradigmas que se han venido utilizando en robótica móvil hasta el momento actual y se han indicado sus principales ventajas y limitaciones. Como el objetivo del presente trabajo consiste en desarrollar un robot móvil autónomo de "propósito general", la comparación entre los distintos paradigmas se ha centrado en los criterios propuestos en la sección 3.1.3: robustez; reactividad; integración de comportamientos planificado, deliberativo y reactivo; flexibilidad y escalabilidad. Para conseguir un enfoque más general, las valoraciones se han realizado teniendo en cuenta únicamente las principales características de cada paradigma y no las particularidades de las distintas

Arquitectura	Robusta	Reactiva	Integra comportamientos	Flexible	Escalable
jerárquica	–	–	–	-	–
reactiva	++	++	–	+-	+-
híbrida	+	+	+	+-	-
de pizarra	-	+	++	++	-
de agentes	+	+-	+-	+-	+-
AFE	+	+	++	++	++

Tabla 3.1: Tabla comparativa donde se muestra el grado de cumplimiento de los criterios de evaluación propuestos para cada uno de los principales paradigmas aplicados a robótica móvil. La graduación es: poco (-), medio (+-), bastante (+) y mucho (++).

implementaciones.

Existen muchas dificultades para medir empíricamente en qué medida una arquitectura posee una cierta propiedad o cumple un criterio específico. Por lo tanto, en la mayoría de las ocasiones las comparaciones entre arquitecturas son de carácter cualitativo. El método más utilizado consiste en realizar una valoración relativa sobre cada criterio, es decir, considerar si una arquitectura cumple un criterio de evaluación en mayor o menor medida que otra. En la tabla 3.1 se resumen los resultados obtenidos mediante este método.

De los datos que contiene la tabla 3.1 se pueden extraer dos conclusiones. En primer lugar, la arquitectura basada en especialistas es la que *a priori* más fácilmente se puede escalar. En segundo lugar, aunque no maximiza todos los criterios de evaluación planteados, AFE consigue una muy buena valoración en todos ellos y, lo más importante, no posee ninguna deficiencia grave dentro del conjunto de criterios de evaluación considerados.

De la comparación también se deduce que las arquitecturas híbridas planificadoras/reactivas están bien compensadas, ya que no poseen ninguna deficiencia grave (excepto la escalabilidad), aunque tampoco maximizan ningún criterio. Por otra parte, las arquitecturas basadas en pizarra tienen un comportamiento bueno o muy bueno en cuatro de los criterios seleccionados, pero no son ni robustas ni escalables.

Capítulo 4

Arquitectura fractal de especialistas

En el capítulo anterior se han enunciado las principales propiedades y ventajas de la arquitectura fractal de especialistas en comparación con otros paradigmas aplicados al dominio de la robótica móvil autónoma. En este capítulo se procede a una descripción detallada de esta nueva arquitectura y de cada uno de sus componentes en su aplicación a dicho dominio y que denominaremos AFE-Robótica.

4.1 Principales características de AFE

La arquitectura fractal de especialistas (AFE) se articula sobre cuatro principios fundamentales. El primero de ellos es la *especialización*, obtenida a través de la "encapsulación", dentro de módulos denominados especialistas, de aquellas tareas estrechamente relacionadas entre sí, junto con el control y los datos necesarios para su ejecución. De esta manera, cada especialista es responsable de ejecutar y controlar las tareas que le han sido encomendadas, de procesar los datos necesarios para realizar dichas tareas y, en general, de resolver los problemas que surjan durante la ejecución de las mismas.

El segundo principio básico es la *recursividad*. Un especialista se puede dividir en varios módulos internos o agentes (figura 4.1) diseñados para sintetizar algunas de sus funciones o competencias, es decir, ejecutar y controlar algunas de sus subtareas y procesar parte de sus datos. Sin embargo, cada agente se puede configurar, a su vez, como un especialista con sus propios agentes a nivel interno. En definitiva, el esquema de la arquitectura se puede replicar en cada agente y, en ese sentido, la arquitectura se puede entender como fractal o recursiva.

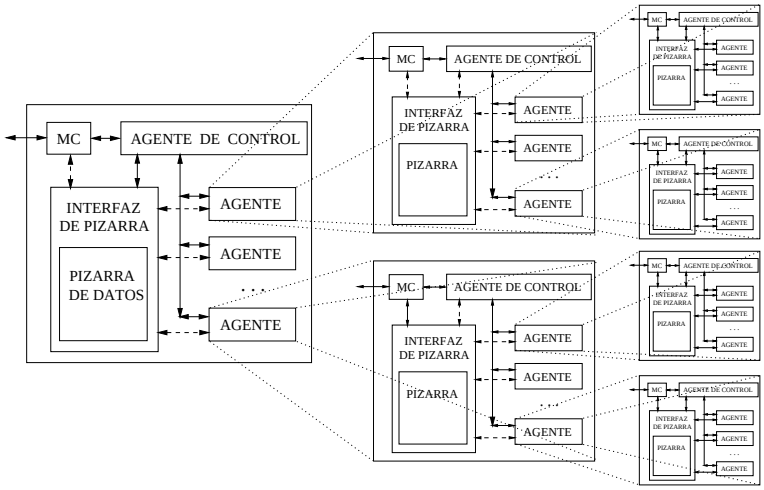

Figura 4.1: Esquema general de la arquitectura fractal de especialistas (AFE).

El tercer principio básico de AFE consiste en que los especialistas dialogan entre sí mediante *paso de mensajes*. De esa forma, las comunicaciones son independientes del lugar de ejecución de cada especialista. En segundo lugar, se ajustan perfectamente a la naturaleza fundamentalmente asíncrona del intercambio de información (órdenes de tarea, eventos y datos). Estos intercambios son, salvo casos específicos, no muy habituales, poco intensos y requieren un ancho de banda reducido. Por último, y quizás más importante, permite utilizar protocolos flexibles y en base a especificaciones de alto nivel de abstracción, de modo que las comunicaciones son fáciles de interpretar y las modificaciones, por otra parte inevitables, son sencillas de implementar.

El cuarto principio es la implementación de cada especialista mediante una *arquitectura de pizarra* (figura 4.1). Esta característica no es crítica en AFE, pero con esta restricción en la implementación de cada especialista se pretende una implementación más uniforme y simple de todo el sistema, así como una mayor facilidad para la reutilización del código. La elección de la arquitectura basada en pizarra se debe a sus conocidas propiedades de flexibilidad y capacidad de adaptación, lo que facilita que cada especialista se ajuste a sus requisitos y posibilidades, por muy diferentes que éstos sean.

Estos cuatro principios básicos son los que definen la arquitectura basada en especialistas AFE, la especificación concreta de la estructura de sus elementos no debería considerarse como parte de la caracterización de la arquitectura, sino como una instanciación particular de esos elementos para adaptarse a un dominio deter-

minado. En este sentido, la flexibilidad de AFE es avalada por su exitosa aplicación a dominios tan diferentes y exigentes como el de una arquitectura de control general para un robot móvil autónomo (Regueiro y col., 1999, 2002), un especialista terapéutico en Unidades de Cuidados Intensivos Coronarios (Lama y col., 1999; Lama, 2000) o un especialista en percepción de señales en el mismo dominio (Fraga y col., 1998).

4.2 Aplicación de AFE a la robótica móvil

El motivo del presente trabajo es desarrollar e implementar el sistema de control de un robot móvil autónomo (RMA) de *propósito general*, es decir, con la funcionalidad básica necesaria para realizar un amplio conjunto de tareas sin ceñirse, por el momento, a ninguna aplicación específica. En esta línea, en el capítulo sobre los aspectos funcionales del RMA se han descrito las principales habilidades que debe incorporar dicha arquitectura. También se han indicado las simplificaciones propuestas para que su implementación pueda ser más abordable, concretamente tres: restringir los entornos de operación del robot al interior de edificios (*entorno "indoors"*), implementar únicamente las tareas relacionadas con la navegación y utilizar como plataforma física un *Nomad 200*. En el mencionado capítulo se argumentan en detalle la razones para adoptar cada una de estas decisiones.

Como se ha visto en el capítulo anterior, los paradigmas utilizados hasta el momento en el dominio de la RMA no cumplen adecuadamente todos los requisitos planteados, por lo que se ha propuesto una nueva arquitectura basada en especialistas. La aplicación concreta de dicha arquitectura al dominio de la RMA es lo que denominamos AFE-Robótica. En esta sección se dan las líneas maestras del diseño de un prototipo básico de AFE-Robótica, lo suficientemente sencillo para que su implementación sea factible pero, al mismo tiempo, lo bastante completo para que se pueda validar adecuadamente la arquitectura propuesta y también para que el robot alcance un mínimo de inteligencia básica, concretamente en tareas relacionadas con la navegación.

4.2.1 Estructura de especialistas en AFE-Robótica

La división de tareas en AFE-Robótica es similar a la utilizada por la mayor parte de la bibliografía y que ya fue expuesta claramente por Meystel en su arqui-

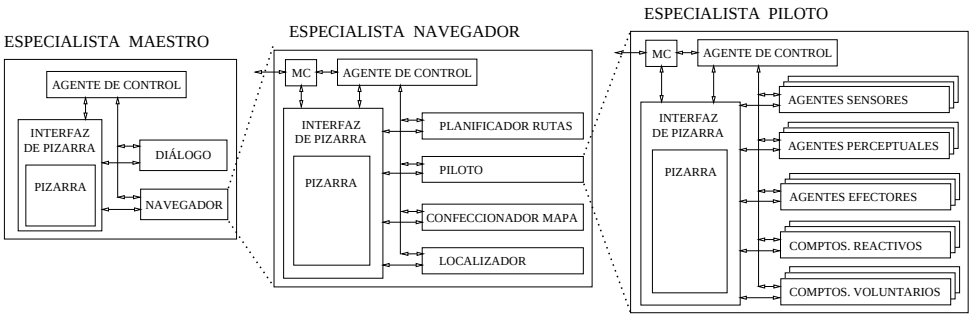

Figura 4.2: Arquitectura fractal de especialistas aplicada a la robótica móvil: AFE-Robótica.

tectura NHC (Meystel, 1990). Dicha división consta de tres niveles o capas: un "planificador de misión", un "planificador de rutas" o navegador y un "ejecutor de rutas" o piloto. Los nombres pueden variar de una a otra arquitectura, pero las funciones prácticamente son las mismas.

Una de las principales cualidades de AFE es permitir que en cada capa se puedan agrupar en especialistas aquellas tareas estrechamente relacionadas entre si junto con los datos y el control necesarios para su ejecución. Esta especialización simplifica todo el proceso de diseño e implementación, aún a costa de imponer una cierta rigidez al sistema. Por lo tanto, la estructura final de especialistas viene en gran medida impuesta por la jerarquía de tareas. En robótica, los datos no son tan decisivos porque sus dependencias son muchas, muy complejas y no se adaptan estrictamente a ninguna jerarquía de tareas.

La estructura de especialistas de AFE-Robótica para realizar tareas de navegación se recoge parcialmente en la figura 4.2, y en ella se puede apreciar la descomposición de las tareas y la abstracción de los datos que conlleva.

El especialista MAESTRO se encarga de planificar la ejecución de las tareas globales encomendadas por un usuario, pero delega en NAVEGADOR la planificación y ejecución de los desplazamientos y en DIÁLOGO la interacción con otros sistemas, principalmente, seres humanos y otros robots. Los detalles de la ejecución de las tareas que MAESTRO asigna a sus agentes son transparentes para él, ya que sólo su funcionalidad es importante. Como se puede intuir, la incorporación de nuevas funcionalidades del robot, p.e. tareas de manipulación, se realiza de una manera muy natural, simplemente poniendo a disposición de MAESTRO nuevos agentes.

Por su parte, el especialista NAVEGADOR utiliza a su agente PILOTO para mover el robot en su entorno local y extraer información de los datos sensoriales. A partir

de dicha información local, el agente CONFECCIONADOR_MAPA va confeccionando el mapa global del entorno. Dicho mapa es utilizado por un tercer agente, LOCALIZADOR, para determinar o verificar en qué región del mapa se encuentra el robot, según la posición del robot calculada y los objetos del entorno (marcas) percibidos por PILOTO. Esta tarea debe ser muy robusta ya que la posición del robot no siempre se puede determinar con la precisión deseable e, incluso, necesaria.

Por último, el agente PLANIFICADOR_RUTAS utiliza el mapa global del entorno para trazar una ruta que pueda seguir el robot para llegar al destino elegido desde su posición actual. La ruta obtenida es posteriormente traducida a una serie de pasos o tareas ejecutables por los agentes de NAVEGADOR, principalmente PILOTO. La traducción de la ruta y la supervisión y coordinación de su ejecución corre a cargo del módulo de control de NAVEGADOR.

Como ya se ha comentado, cada uno de los agentes de un especialista se puede implementar a su vez como otro especialista. Así, PILOTO procesa toda la información sensorial y ejecuta las tareas que le ordena su especialista superior, NAVEGADOR, mediante comportamientos voluntarios e incluye comportamientos reactivos para mantener la integridad del robot. Cada comportamiento es implementado por un agente que calcula un comando de movimiento en función de los datos recientemente percibidos del entorno. Los AGENTES EFECTORES integran los comandos generados por los distintos comportamientos activos (puede haber más de uno) y envían el comando final al actuador del robot que corresponda.

Los comportamientos voluntarios son excluyentes entre sí. Para ejecutar desplazamientos en el interior de edificios sólo se necesitan dos:[1] MOVER_A y SEGUIR_CONTORNO (Iglesias y col., 1997, 1998a; Mucientes y col., 2002b). Los comportamientos reactivos necesarios son EVITAR_CHOQUE, MANTENER_DISTANCIA (entre el robot y los objetos que lo rodean) y EVITAR_OBJETO_MÓVIL (Mucientes y col., 1999, 2001a,b). Los agentes perceptuales se describen en la siguiente sección.

4.2.2 Estructura parcial de datos en AFE-Robótica

En la figura 4.3 se indican parte de los datos que necesitan los principales especialistas y agentes de AFE-Robótica. En dicha figura las flechas continuas marcan las dependencias mientras que las punteadas indican indicios, útiles para focalizar

[1] Es necesario un tercer comportamiento para que el robot cruce angosturas (p.e., puertas) pero, en la práctica, se puede implementar como tres tareas MOVER_A, encadenadas y regulando apropiadamente los comportamientos reactivos.

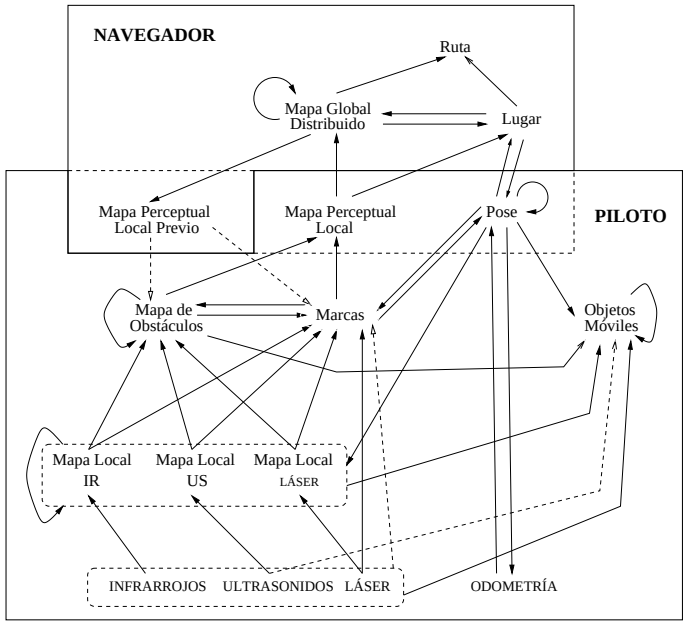

Figura 4.3: Estructura parcial de datos en AFE-Robótica.

la atención del robot. Por último, los rectángulos en línea continua indican a qué especialista pertenece cada dato y en discontinua muestran los datos que se comparten.

Los datos que maneja el especialista NAVEGADOR son principalmente tres (figura 4.3), todos de un alto grado de abstracción: el mapa global del entorno (MGD o MAPA GLOBAL DISTRIBUIDO), la última ruta planificada (RUTA), aunque podría haber otras, y la región (LUGAR) que ocupaba el robot la última vez que fue localizado de manera global (típicamente, antes de planificar una ruta). Los otros datos que se almacenan son las entradas y salidas de PILOTO.

El MGD se confecciona en base al conocimiento previo inyectado en el sistema (p.e. un plano del entorno) y, sobre todo, a partir de la información que va extrayendo PILOTO de los datos sensoriales: MAPA PERCEPTUAL LOCAL (MPL) y POSE. El primero incluye tanto la posición de los obstáculos (MAPA OBSTÁCULOS) como la posición y características de los objetos característicos (MARCAS) detectados en una cierta región del entorno. La posición del robot (POSE) es imprescindible para relacionar los distintos y sucesivos mapas locales entre sí (tarea que realiza el agente CONFECCIONADOR_MAPA). Para lograrlo es imprescindible que PILOTO tenga en todo momento razonablemente bien posicionado el robot con respecto a su entorno local (MPL).

En el especialista PILOTO sólo se dispone de la información local y reciente que proporcionan los sensores del robot y que los distintos AGENTES SENSORES se encargan de volcar en la pizarra (ODOMETRÍA, ULTRASONIDOS, INFRARROJOS y LÁSER en la figura 4.3, a los que habría que añadir TÁCTILES y VOLTAJES, no mostrados en la figura). Cada sensor posee, en general, su propia frecuencia de adquisición. Así, los datos de VOLTAJES se toman cada 2 minutos, puesto que son medidas que no cambian bruscamente. La información del sensor láser se adquiere cada segundo, tiempo suficiente para su procesamiento y aconsejable para evitar colapsar la percepción con información repetitiva. El resto de datos sensoriales, los denominados datos *Básicos* (ODOMETRÍA, ULTRASONIDOS, INFRARROJOS y TÁCTILES), se recogen a la mayor frecuencia que permite el hardware: 0.2 Hz.

Los agentes perceptuales se encargan de procesar e integrar dichos datos sensoriales junto con los datos de la misma región del entorno previamente almacenada en NAVEGADOR (MPLP o MAPA PERCEPTUAL LOCAL PREVIO) para obtener la información de interés para el robot: el MAPA PERCEPTUAL LOCAL (MPL). La percepción es parte de las tareas consideradas internas. Hay que tener en cuenta que la percepción es, o debería ser, un proceso activo y, por tanto, los agentes perceptuales pueden generar comandos de movimiento siempre que lo consideren necesario para mejorar la percepción y obtener más o mejores datos.

Actualmente, existen cuatro agentes de percepción implementados en PILOTO. El agente CREADOR_MAPA_LOCAL confecciona y mantiene un mapa a corto plazo con los obstáculos que rodean el robot (MAPA DE OBSTÁCULOS) (Rodríguez y col., 2000). El agente DETECTOR_MÓVILES detecta los objetos en movimiento cerca del robot (OBJETOS MÓVILES). Un tercer agente, DETECTOR_MARCAS, percibe los elementos de interés en el entorno (MARCAS), fundamentales para la relocalización del robot. Esta tarea de actualizar constantemente la posición real del robot (POSE) es la que realiza el cuarto agente de percepción: POSICIONADOR_ROBOT.

Como comentario final indicar que en AFE existe una gran flexibilidad en cuanto al tipo de datos que se pueden almacenar en las pizarras y a su representación. Por lo tanto, no es relevante cómo se representan los mapas del entorno, ni las marcas, ni la posición del robot, por poner algunos ejemplos. Estos aspectos no influyen decisivamente en la jerarquía de tareas y sólo son importantes para la implementación de los distintos agentes de la arquitectura. En realidad, no habría ningún problema si algún dato se almacena siguiendo dos representaciones diferentes. Únicamente habría que asegurarse de que cada agente que manipule el dato posea las funciones necesarias para operar con cada representación.

4.3 Arquitectura de un especialista

Cada especialista de AFE se puede implementar de forma diferente. Sin embargo, y en aras a una mayor uniformidad y simplicidad, se ha optado por utilizar una misma arquitectura software para todos ellos. Por supuesto, dicha arquitectura debe ser relativamente sencilla de implementar y suficientemente flexible para adaptarla a las necesidades y requisitos de cada especialista, a veces muy diferentes entre si.

4.3.1 Arquitectura basada en pizarra

Cada especialista en AFE se implementa, básicamente, mediante una arquitectura basada en pizarra , es decir, está conformado por un conjunto de agentes que comparten datos a través de una memoria común (pizarra) y que están coordinados por un mismo agente de control.

Las ventajas de utilizar una arquitectura basada en pizarra respecto a otras arquitecturas software son que facilita la implementación heterogénea de los agentes, la inclusión de nuevos datos y agentes y la modificación de los ya existentes. Esta flexibilidad es muy importante en dominios, como el que aquí estamos tratando, donde el desarrollo de nuevos métodos y algoritmos es continuo y donde la información aún no está claramente estructurada ni las dependencias completamente establecidas.

Por otra parte, la ejecución de las tareas en un especialista se delega en sus agentes, cuya operación es coordinada y supervisada a través del módulo de control. Si uno de tales agentes deja de operar, o lo hace incorrectamente, el especialista no deja de funcionar, simplemente pierde la funcionalidad asociada con tal agente.

Sin embargo, existen tres diferencias fundamentales entre un especialista y una arquitectura clásica de pizarra. La primera es que la pizarra de un especialista no es un mero almacén de datos, más o menos estructurados, sino un sistema activo capaz de generar **eventos** cada vez que se actualiza la información que contiene. Con ello se establecen los mecanismos necesarios para que el especialista, y la arquitectura en su conjunto, pueda reaccionar a tiempo ante cualquier cambio en el entorno.

La segunda diferencia es que los agentes de un especialista se pueden implementar como especialistas, dando lugar a que la misma arquitectura se replique sea cual sea el nivel de detalle, hasta el nivel inferior, en el cual los agentes son procesos simples, que denominamos terminales.

La tercera diferencia es que, debido a la "distribución" de la arquitectura, es

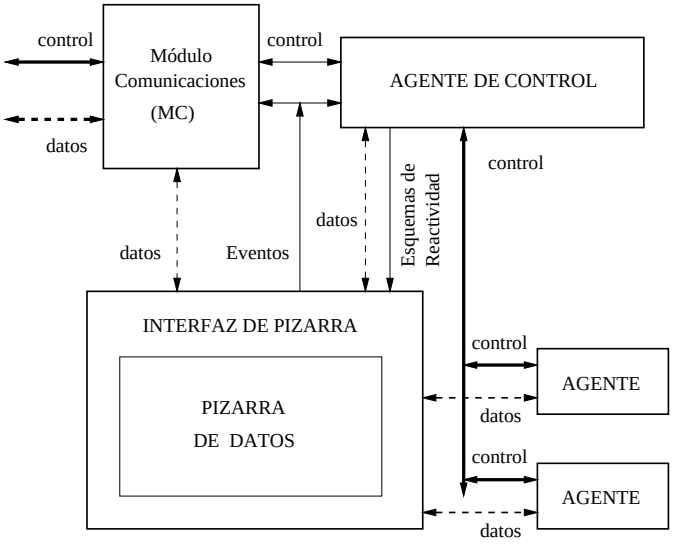

Figura 4.4: Estructura interna de un especialista en AFE-Robótica.

imprescindible desarrollar un protocolo de comunicaciones entre los especialistas. Todas estas funciones se centralizan en un nuevo elemento que no existe en la arquitectura clásica de pizarra: el módulo de comunicaciones.

4.3.2 Componentes de un especialista

En la figura 4.4 se pueden apreciar los distintos componentes de un especialista de AFE-Robótica:

Pizarra de datos. Almacena los datos que comparten los agentes del especialista, implementa los mecanismos necesarios para su intercambio y genera los eventos oportunos cada vez que se actualiza la información que contiene. La activación de los eventos puede ser regulada a través de los *Esquemas de Reactividad*.

Agentes. Se corresponden con las bases de conocimiento de las arquitecturas de pizarra y son los que poseen el conocimiento necesario para realizar parte de las tareas del especialista. En AFE, los agentes se pueden implementar, a su vez, como especialistas o bien pueden ser agentes terminales que se corresponden con procesos simples.

Agente de control. Al igual que en cualquier arquitectura basada en pizarra, es el encargado de controlar y supervisar el funcionamiento de todos sus agentes,

sobre los cuales delega parte de sus funciones. El agente de control debe responder a las órdenes de tarea que le llegan de su especialista superior y a los eventos, tanto internos (generados en su propia pizarra) como externos (generados por otros especialistas), que afectan al comportamiento del especialista. Por lo tanto, es el encargado de integrar la respuesta (comportamiento reactivo) con el resto de tareas en curso (comportamiento planificado) para generar un comportamiento global coherente.

Módulo de comunicaciones. Es un elemento específico de un especialista de AFE y sobre él recaen todas las funciones relacionadas con las comunicaciones del especialista con el exterior. La centralización modulariza el diseño y simplifica la implementación del sistema de comunicaciones.

4.4 Comunicaciones entre especialistas

Los especialistas de AFE son procesos autónomos e independientes aunque intercambian información tanto de control como de datos. Una característica muy importante de AFE es que los especialistas dialogan mediante paso de mensajes, es decir, utilizando protocolos de comunicaciones de alto nivel, flexibles e independientes de la implementación de cada especialista.

Como ya se ha comentado en el capítulo anterior, la utilización de protocolos basados en paso de mensajes conlleva varias ventajas. Por un lado, permite la implementación independiente de cada especialista, del cual sólo se necesita conocer su "interfaz" y no los detalles de su implementación. También posibilita la utilización de diferentes plataformas lógicas y de programación, tanto para la ejecución como para el desarrollo de la arquitectura. Por otro lado, facilita la ejecución distribuida de la arquitectura, lo que permite acceder a una mayor potencia de cálculo y procesar tareas en paralelo, incluso de forma redundante. Por último, en conjunción con la especialización, mejora la eficiencia en la gestión de los recursos computacionales y, sobre todo, aumenta la escalabilidad de la arquitectura.

Como inconvenientes se podría señalar que, al igual que las arquitecturas basadas en agentes, las comunicaciones pueden condicionar el rendimiento de la arquitectura y convertirse en un cuello de botella. Sin embargo, la encapsulación inherente a AFE minimiza el impacto de las comunicaciones, ya que, en general, el "micromundo" que afecta y es afectado por un especialista está integrado en él, lo que reduce considerablemente la necesidad de intercambiar información.

4.4.1 Protocolo de comunicaciones

Entendemos por protocolo de comunicaciones (Lama, 2000) tanto la especificación de las condiciones en que tiene lugar la interacción entre especialistas como la descripción de la forma en que se comunican y el soporte físico utilizado para ello. Así, cuando hablamos de interacción entre agentes distribuidos en una red de computadores, estamos asumiendo la existencia de un lenguaje de comunicación entre ellos, que puede ser de dos tipos (Genesereth y Ketchpel, 1994):

- *Procedimental*, en su contenido los mensajes llevan asociada la ejecución de un procedimiento o programa en el agente receptor. La comunicación entre los agentes, por tanto, se modela a través del trasvase de directivas procedimentales que pueden ser ejecutables directamente en los agentes receptores de los mensajes.

- *Declarativo*, permite establecer un intercambio de información entre los agentes, los cuales, dependiendo de la semántica de los mensajes recibidos, decidirán ejecutar ciertas acciones para darle respuesta.

El protocolo de comunicaciones debe cumplir el tercer principio fundamental de AFE: ser independiente del lugar de ejecución de cada especialista; estar orientado a un intercambio de información fundamentalmente de naturaleza asíncrona y no demasiado intenso; y, sobre todo, ser flexible y dialogar en base a especificaciones de alto nivel de abstracción para que las comunicaciones sean fáciles de interpretar y las modificaciones sencillas de implementar.

El lenguaje de comunicaciones declarativo KQML (*Knowledge Query and Manipulation Language*) (Finin y col., 1993; Mayfield y col., 1996; Labrou y Finin, 1997; Finin y col., 1997; Labrou y col., 1999) cumple perfectamente todos estos requisitos. Con él se puede definir una sintaxis compleja, abstracta y modular de las órdenes de tarea, se pueden intercambiar datos independientemente de su naturaleza y se pueden definir y enviar eventos, motivos por los cuales lo hemos elegido para nuestra implementación. Además, dado que los mensajes están codificados en ASCII, resulta muy portable y general.

El inconveniente es que KQML está orientado al intercambio de datos y no se ajusta bien a las necesidades de AFE referentes a las órdenes de tarea. Por lo tanto, ha sido necesario incluir nuevas *performativas* (clases de mensajes) de tipo "imperativo" (Lama, 2000), específicas para enviar la información de control de un

especialista a otro, aunque esto haya significado separarse ligeramente del estándar establecido.

Un protocolo de comunicaciones de tipo procedimental muy utilizado últimamente es el basado en el estándar CORBA. Con dicho protocolo dos objetos pueden intercambiar información de forma transparente para el programador e independientemente del lenguaje en el que estén definidos y de la plataforma y del sistema operativo en donde se ejecuten. Aunque la adopción de CORBA sería muy sencilla, dado que todas las dependencias con las comunicaciones se concentran en los MC, su menor eficiencia, mayor consumo de recursos computacionales y menor portabilidad respecto al KQML, nos han hecho descartarlo, al menos de momento.

4.4.2 Tipos de comunicaciones

Las comunicaciones en AFE se realizan en dos niveles: en el interior de cada especialista y entre un especialista y su exterior. En las primeras, el agente de control se encarga del intercambio de información de control con los agentes y la interfaz de pizarra es la responsable del intercambio de datos entre los agentes, incluido el de control, y la pizarra (figuras 4.4 y 4.5). Sin embargo, si un agente de un especialista se implementa a su vez como especialista, entonces las comunicaciones internas del primero serán las externas del segundo.

A nivel externo todas las comunicaciones se centralizan en el módulo de comunicaciones (MC en las figuras 4.4 y 4.5). Como las comunicaciones se dividen en dos categorías, claramente diferenciadas entre sí, esa misma división se refleja en la estructura interna del MC (figura 4.5). Así una parte opera sobre las comunicaciones relativas al control (Módulo de Comunicaciones de Control o MCC) y otra sobre las comunicaciones referidas a los datos (Módulo de Comunicaciones de Datos o MCD).

Comunicaciones de Control

La información de control la forman las Órdenes de Tarea (OTs) que envía el agente de control de un especialista a sus agentes y las respuestas de éstos a dichas órdenes ($ROTs$). Las OTs consisten, principalmente, en órdenes relativas a la ejecución de una tarea o al procesamiento de datos. También se incluyen una serie de comandos que actúan sobre OTs previamente emitidas y que sirven para anularlas, suspenderlas, reanudarlas, o modificar sus parámetros de funcionamiento: prioridad, tiempo máximo de ejecución, recursos utilizables, etc.

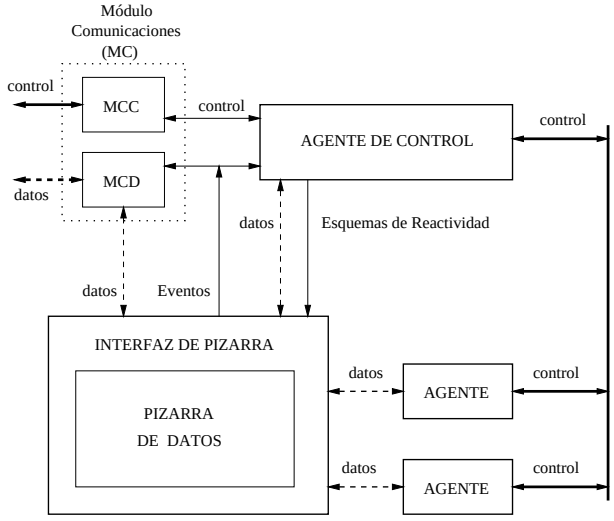

Figura 4.5: Estructura del Módulo de Comunicaciones de un especialista.

Las Respuestas a una Orden de Tarea ($ROTs$) indican si la ejecución ha sido correcta (OK) o no (No OK). En éste último caso se suele indicar el tipo de error encontrado durante la ejecución de la OT y aquellos datos que puedan resultar de interés para la resolución del problema, es decir, el contexto.

Debido a la estructura jerárquica de las tareas y de la arquitectura, el intercambio de información de control únicamente tiene lugar entre un especialista y el agente de control de su especialista superior.

Comunicaciones de Datos

Para aumentar la flexibilidad de la arquitectura AFE, el intercambio de datos no se ve restringido por la jerarquía de tareas, tal y como ocurre con la información de control. Por lo tanto, cualesquiera dos especialistas pueden intercambiar directa y libremente los datos de sus pizarras. En total, se distinguen tres tipos de comunicaciones a nivel de datos:

- **Intercambio de datos** entre la pizarra propia y la de su especialista superior. Este intercambio está regulado por las órdenes del agente de control: por un lado pide los datos necesarios para ejecutar las OTs enviadas, por otro lado envía los resultados obtenidos de tales ejecuciones. Un ejemplo de esta situación en AFE-Robótica se produce cuando el robot pasa a una nueva región espacial, momento en el cual PILOTO escribe en la pizarra de NAVEGADOR el

mapa (MPL) generado sobre la región anterior y recoge el mapa almacenado sobre la nueva región (el MPLP).

- **Consultas** entre especialistas. Una consulta es la petición de un dato y su envío entre dos especialistas de la arquitectura. Gracias a las consultas cualquier agente de AFE puede acceder a cualquier dato generado en la arquitectura y almacenado en una pizarra, sin las restricciones impuestas por la estructura de especialistas definida. Ejemplos de consultas son los datos que el especialista DIÁLOGO pide a todos los especialistas para mostrárselos a los usuarios.

- **Eventos externos**. Son idénticos a los eventos internos, excepto que se generan en la pizarra de un especialista pero se envían a otro para producir la respuesta adecuada. De esta manera, un evento puede tener respuesta en cualquier especialista de la arquitectura. Es otro aspecto del desajuste entre la estructura de datos y la jerarquía de tareas. Un ejemplo en AFE-Robótica es el evento "baterías descargadas": se detecta en PILOTO pero la respuesta se produce en MAESTRO (p.e., alterando la secuencia de tareas prevista) o en DIÁLOGO (p.e., mediante la presentación de un aviso).

4.4.3 Implementación de las comunicaciones

En esta sección se comentan los detalles más importantes y el funcionamiento de los mecanismos que implementan los distintos tipos de comunicaciones e intercambios utilizados en AFE-Robótica.

Intercambio de información de control

Es el mecanismo más frecuente e importante de comunicación entre un especialista y sus agentes, en virtud del cual el especialista remite tareas a sus agentes (OT) y recoge sus respuestas (ROT). Este intercambio se realiza entre el agente de control del especialista y el MC de cada uno de sus agentes, utilizando un protocolo común.

En principio este protocolo es propio o específico de cada especialista e independiente del usado en los demás especialistas. Sin embargo, en aras de una mayor uniformidad y simplicidad, tanto en el diseño como en la implementación de la arquitectura, se propone un esquema regular para todos los especialistas.

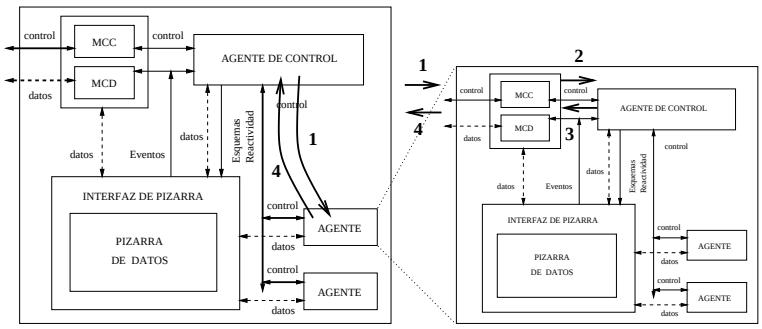

Figura 4.6: Envío de una orden de tarea.

El funcionamiento del intercambio de información de control es muy sencillo y se resume en la figura 4.6. Internamente, el agente de control del especialista codifica la orden y la envía al agente apropiado[2] (1). El MC (realmente el MCC) recoge el mensaje, lo descodifica y manda la correspondiente *OT* a su agente de control (2), que pasa a ejecutarla. Finalizada la tarea, o ejecutado el comando correspondiente, el agente de control manda la respuesta a su MCC (3) que la codifica y devuelve al agente de control del especialista superior (4).

Intercambio de datos

Además de información de control, un especialista y sus agentes intercambian datos por medio de su MCD. Una característica de este intercambio es que está subordinado a la ejecución de las tareas. La iniciativa puede correr a cargo, bien del agente de control, bien del propio agente. En el primer caso, antes del intercambio propiamente dicho, el agente de control envía la *OT* apropiada siguiendo el procedimiento visto en la sección anterior. El resto del proceso es el mismo y sólo depende de si el dato se lee o se escribe.

Si el agente es terminal (es decir, no posee a su vez una pizarra ni un agente de control) el proceso se simplifica. Según le interese leer o escribir, el agente envía al MCD de su especialista un mensaje para pedir o insertar el dato que le interesa. Éste descodifica el mensaje y, según el caso, recoge o inserta el dato en la pizarra a través de la interfaz, codifica un nuevo mensaje y se lo envía de vuelta al agente donde es descodificado. Un ejemplo de este intercambio es cuando el agente PLANIFICADOR_RUTAS solicita el mapa del entorno a la pizarra de NAVEGADOR o escribe en la misma pizarra la ruta calculada.

[2]Al explicar el agente de control se darán más detalles sobre cómo se realiza este paso.

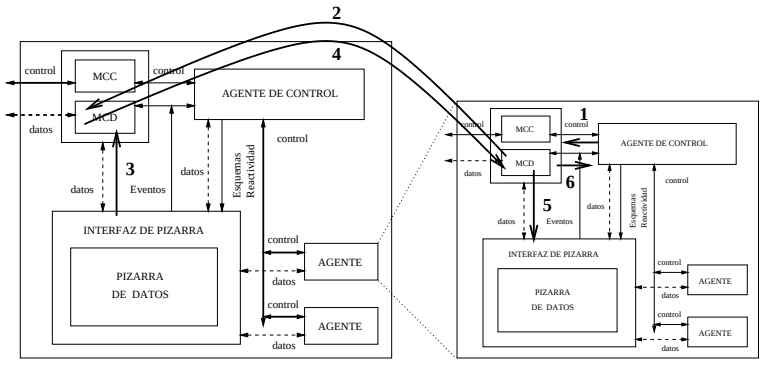

Figura 4.7: Lectura de un dato por parte de un especialista en la pizarra de su especialista superior.

Si el agente es un especialista el proceso es algo más complejo. En la figura 4.7 se indica cómo se realiza la lectura de un dato. En primer lugar, el agente de control del especialista comunica a su MCD qué dato le interesa (1). Éste codifica la petición y se la envía al MCD del especialista superior (2). Una vez descodificada, el MCD del especialista superior recoge el dato de la pizarra (3), codifica el mensaje apropiado y se lo envía de vuelta al MCD del especialista (4). Éste lo descodifica, inserta el dato en la pizarra del especialista (5) y avisa al agente de control que el dato ha llegado (6). Este sería el proceso mediante el cual PILOTO lee en la pizarra de NAVEGADOR.

El proceso inverso de escribir un dato es muy similar, tal y como se muestra en la figura 4.8. En primer lugar, el agente de control del especialista le indica a su MCD (1) que debe enviar un dato al especialista superior. A través de la interfaz de la pizarra, el MCD recoge el dato en cuestión (2), codifica un mensaje y se lo envía al MCD del especialista superior (3). Éste descodifica el mensaje e inserta el dato en la pizarra del especialista superior (4), también a través de la interfaz. Por último, envía al MCD del especialista un mensaje indicando que se ha insertado el dato (5). Cuando éste lo recibe, manda un aviso a su agente de control (6). Nótese que en las figuras 4.7 y 4.8 se han omitido algunas flechas para una mayor claridad de las mismas.

Consulta de datos

El intercambio de datos estudiado en la sección anterior es el método más habitual para que un especialista obtenga información. Sin embargo, existen casos en donde a un especialista le interesan datos que no son parte ni de su pizarra ni de

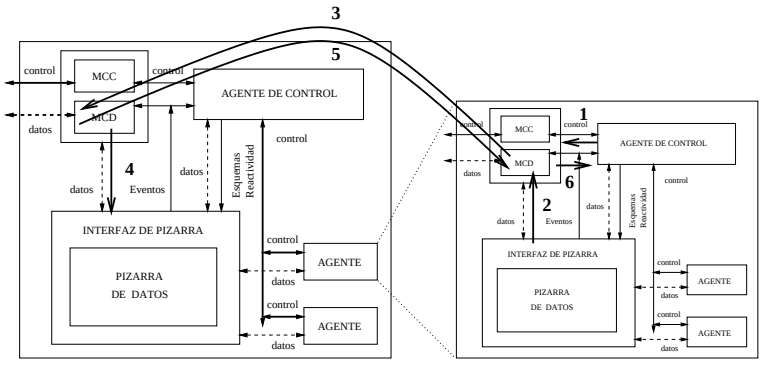

Figura 4.8: Inserción de un dato por parte de un especialista en la pizarra de su especialista superior.

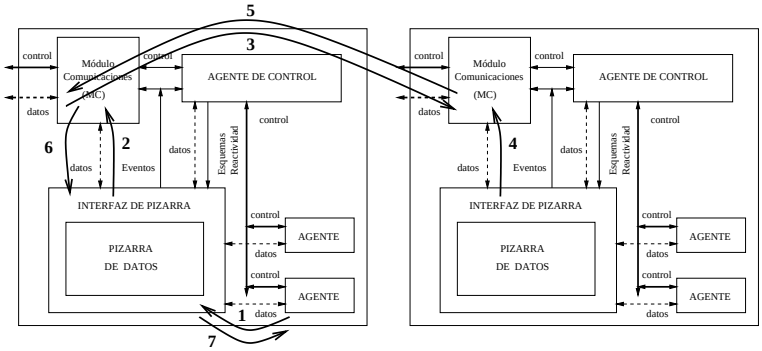

Figura 4.9: Consulta de un dato en AFE-Robótica.

la pizarra de su especialista superior. Para resolver esta dificultad AFE utiliza las consultas.

El primer paso en la implementación de las consultas consiste en disponer una copia enlace en la pizarra de cada uno de los especialistas que quieran disponer de un dato que no se genera en el propio especialista. Si un agente intenta acceder a una copia enlace (1 en la figura 4.9), la interfaz de pizarra lo detecta y, antes de servirlo, ordena al MCD (2) que pida el dato al especialista en donde se genera (3). El MCD de dicho especialista descodifica el mensaje y recoge el dato original de su pizarra a través de la interfaz (4). Después codifica otro mensaje y lo envía de vuelta al MCD origen de la petición (5). Cuando éste lo recibe, lo descodifica e inserta el dato en la pizarra (6). Por último, la interfaz de pizarra sirve el dato actualizado al agente que lo ha pedido (7).

Como se puede apreciar, el mecanismo de consultas se activa únicamente cuando se lee un dato, de esta manera es más fácil mantener la coherencia de toda la

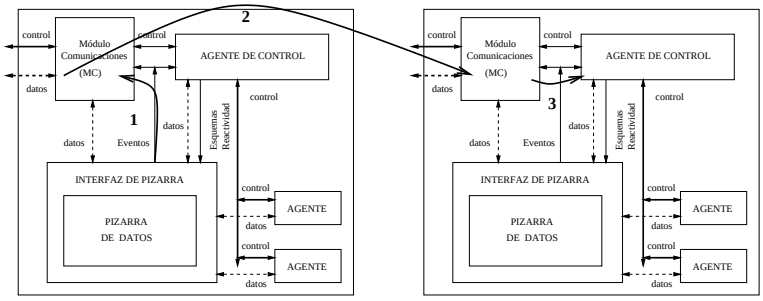

Figura 4.10: Esquema de funcionamiento de la reactividad externa en AFE.

información almacenada en la arquitectura: sólo se actualizan los datos cuando algún agente los necesita y accede a una copia enlace. Otro método de activación de las consultas podría ser la escritura o inserción de un dato: las copias enlaces de dicho dato se actualizarían cada vez que éste se modificase. Este sistema es, sin embargo, menos general, porque de esta manera es el dato original el que necesita guardar información sobre dónde están sus copias enlace. También es menos eficiente porque las copias se actualizan aunque ningún agente las necesite, lo que normalmente ocurrirá con una frecuencia mucho mayor.

Envío de eventos externos

Un robot móvil debe responder en tiempo real a cualquier cambio significativo en su entorno. En la arquitectura propuesta las pizarras son almacenes activos de información, capaces de generar eventos ligados a ciertos cambios en los datos dispuestos en la pizarra, y que han de tener una respuesta adecuada y suficientemente rápida. Cuando la reacción se ejecuta en el mismo especialista en el que se produjo el evento, se habla de reactividad interna.

A veces la respuesta reactiva no se adecua a la jerarquía que sigue la descomposición de tareas y la abstracción de datos, dado que las dependencias entre los datos y su utilización en un robot móvil son múltiples y complejas (figura 4.3). Esto hace que, en esas ocasiones, un evento producido en un especialista haya de tener respuesta en otro especialista. Para soslayar esta dificultad, la arquitectura envía el evento desde donde se genera hasta donde puede encontrar una respuesta, de forma que el sistema posee la capacidad de obtener respuestas reactivas a nivel global implicando a más de un especialista (reactividad externa).

Tal y como se muestra en la figura 4.10, la implementación de la reactividad externa corre a cargo de las pizarras y de los MCD de los dos especialistas ligados

por el evento. Cuando en un especialista se dispara un evento externo, la interfaz de pizarra avisa a su MCD y le indica a qué especialista debe dirigirlo (1). El MCD del especialista donde se origina el evento externo codifica un mensaje con sus datos y lo remite al MCD del especialista al que debe ir destinado (2). Éste descodifica el mensaje y manda el correspondiente evento a su agente de control (3), igual que si fuera un evento interno. El evento externo se puede enviar a más de un especialista, simplemente mandando más mensajes en el paso (2).

4.5 Pizarra de datos

La pizarra de un especialista está compuesta por los datos que los distintos agentes del especialista comparten entre sí. Los datos de la pizarra se implementan como clases C++ y todos poseen la misma estructura (figura 4.11), compuesta tanto por información de los propios datos como por información de control. La primera es la que realmente manejan y manipulan los agentes del especialista. La segunda se usa como apoyo a las operaciones de la interfaz de pizarra y del agente de control.

4.5.1 Interfaz de pizarra

La interfaz de pizarra la constituyen, básicamente, las funciones o métodos de lectura y escritura de las clases con las que se implementan los datos de la pizarra y cuyos atributos o propiedades constituyen los valores de dichos datos. Además de permitir el acceso de los agentes del especialista a los datos de la pizarra, la interfaz también se encarga de la generación de eventos y de la resolución de las consultas.

Como ya se ha comentado, una de las características más importantes de la pizarra de un especialista es su carácter activo, que le permite detectar aquellas situaciones de interés para el especialista, es decir, los eventos. Para ello, cada vez que se modifica un dato, la propia pizarra (realmente la interfaz de pizarra), automática y sistemáticamente, comprueba si alguno de sus eventos habilitados (si hay alguno) cumple las condiciones de activación. En caso afirmativo, la interfaz de pizarra genera el evento y lo envía, bien al propio agente de control (evento interno), bien al agente de control del especialista apropiado (evento externo).

Las consultas se implementan utilizando "copias enlace" de aquellos datos que se generan en otro especialista, indicando en su campo de control "*Especialista*" en qué especialista se genera realmente el dato. Cada vez que algún agente quiere ac-

Información Dato

Nombre o identificador unívoco del dato que será utilizado por los componentes del especialista para referirse a él en sus operaciones.

Valor del dato (puede ser uno sólo o un histórico de valores en puntos discretos de tiempo).

Tiempo almacenamiento del dato en la pizarra.

Tiempo adquisición de los últimos datos sensoriales utilizados para la actualización del dato. Se utiliza como referencia.

Información Control

IdDato o identificador numérico unívoco de la variable que facilita la automatización de los intercambios.

Número Datos indica la profundidad del histórico de pares valor-tiempo asociados al dato.

Especialista indica a qué especialista se debe pedir el dato si no se genera en el propio especialista al que pertenece la pizarra.

Evento de Reactividad (puede ser una lista, i.e., más de uno)

- **IdEvento** o identificador unívoco del evento reactivo en las operaciones de control.
- **Habilitación** indica si el evento de reactividad se encuentra habilitado (1) o no (0) por el agente de control del especialista.
- **Especialista** indica a qué especialista se debe comunicar la posible ocurrencia del evento de reactividad.
- **Condiciones activación** que debe verificar el dato para activar el evento de reactividad. Constituyen los *Esquemas de Reactividad* de la pizarra y pueden ser modificados en cualquier momento por el agente de control.

Figura 4.11: Elementos de un dato almacenado en la pizarra de un especialista.

ceder a uno de tales datos "externos", la interfaz de pizarra activa el mecanismo de consulta, ya explicado en la sección 4.4.3. De este modo, se proporciona una completa sensación de *localidad*, es decir, que todos los datos que necesitan los agentes del especialista se generan en el propio especialista. Esta operación es completamente transparente a los agentes, lo que aumenta la modularidad y flexibilidad del sistema, facilita la inclusión o modificación del conocimiento usado y permite la implementación heterogénea de los componentes del especialista.

Bajo esta perspectiva, podemos asumir que las pizarras de los especialistas están estrechamente relacionadas entre sí y que, en realidad, conforman una sola (Lama, 2000). Este carácter distribuido de la pizarra (Lesser y Corkill, 1983; Corkill, 1989; Gilmore y col., 1989; Jagannathan, 1989; Durfee y Lesser, 1991) constituye una de las propiedades intrínsecas de la arquitectura AFE en general y de AFE-Robótica en particular.

Ejemplos de datos "externos" en AFE-Robótica son las entradas y salidas del especialista PILOTO (figura 4.3). Como datos de entrada, PILOTO toma de la pizarra de NAVEGADOR el MAPA PERCEPTUAL LOCAL PREVIO (MPLP) que ha sido generado por otro de los agentes de NAVEGADOR (concretamente el agente CONFECCIONADOR_MAPA). Por su parte, PILOTO se encarga de procesar los datos sensoriales y escribe, siempre que se le ordena o lo crea oportuno, en la pizarra de NAVEGADOR la posición del robot (POSE) y el MAPA PERCEPTUAL LOCAL (MPL). En la figura 4.3 estos datos se muestran como "compartidos" aunque en la práctica sólo se generan en una de las pizarras mientras que en la otra son una mera "copia".

4.6 Agentes terminales

Los agentes de un especialista son los que se encargan de ejecutar las tareas encomendadas a dicho especialista y de procesar los datos que se requieren para ello. Aquellos agentes que no se implementan como un especialista se implementan como procesos independientes y se denominan *agentes terminales*. Son mayoría en AFE-Robótica.

En general se considera que los agentes terminales tienen a su cargo la realización de tareas suficientemente sencillas para poder ser implementados mediante simples procesos. Esta simplicidad también se manifiesta en su estructura interna, ya que, tal y como se ve en la figura 4.12, un agente terminal consta de tres bloques funcionales (Lama, 2000):

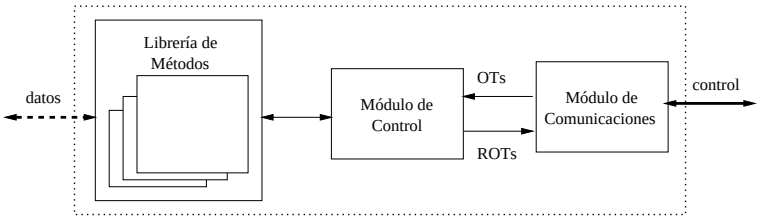

Figura 4.12: Esquema interno de un agente terminal en AFE (adaptado de (Lama, 2000)).

- *Módulo de Comunicaciones de Control* (MCC); al igual que el MCC de un especialista se encarga de detectar y descodificar las *OTs* procedentes del controlador. Estos mensajes tienen un carácter claramente imperativo, ya que, de acuerdo con la jerarquía de tareas en AFE-Robótica, un agente no puede deliberar sobre la oportunidad o no de las acciones de control que se le encomiendan.

  El MCC también se encarga de remitir el mensaje de respuesta (ROT) al agente de control, indicando si la ejecución de la tarea ha tenido éxito o no. En este último caso, también se puede adjuntar información adicional acerca de los posibles problemas o errores encontrados (información de contexto).

- *Módulo de Control*; es el responsable de dar respuesta a cada una de las acciones de control recibidas a través del MC. En el caso de ejecutar una nueva tarea la misión del módulo de control del agente consiste en seleccionar el mejor método para su resolución y en ejecutarlo como una hebra. En las tareas más complejas puede activar y secuenciar la ejecución de varios métodos diferentes. En el caso de anular, suspender o reanudar una tarea, el módulo de control se limita a enviar la señal de control pertinente a la hebra encargada de ejecutar la tarea en cuestión.

- *Librerías de Métodos*; es el conjunto de todos los procedimientos que posee el agente terminal para realizar cada una de las tareas asignadas. Por ejemplo, el agente MOVER_A utiliza técnicas diferentes para ejecutar los distintos tipos de movimientos en los que está especializado: girar, avanzar recto, seguir una dirección o alcanzar una posición odométrica. En el caso de que exista más de un método para resolver una misma tarea, la selección dependerá de las condiciones del sistema, del conocimiento del entorno y de las características de cada método (tiempo de ejecución, fiabilidad, precisión, etc.).

Ilustraremos el funcionamiento de un agente terminal describiendo el comportamiento del agente PLANIFICADOR_RUTAS cuando recibe de su agente de control la OT "Planificar una ruta desde un origen a un destino". Una vez descodificado el mensaje en el MCC, el Módulo de Control determina cuál es el mejor método para la tarea. En este ejemplo la decisión depende del tipo de representación del entorno (mapa topológico) y de los criterios que debe cumplir la ruta (minimizar la distancia). Por lo tanto, el Módulo de Control selecciona el algoritmo de Dijkstra, lo ejecuta como una hebra independiente y se queda esperando una nueva orden.

La hebra accede directamente a la pizarra a través de la interfaz para recoger los datos (el grafo que representa el entorno) y calcula el camino entre el origen y el destino dados que mejor satisfaga el criterio fijado. Finalizada su ejecución, si existe una ruta el método la escribe en la pizarra y avisa al módulo de control que, a su vez, envía la correspondiente respuesta (ROT) al agente de control de NAVEGADOR.

Como se puede ver, los agentes terminales se implementan siguiendo una gran modularidad funcional, que favorece en buena medida la inclusión de nuevos métodos o la modificación de los ya existentes. Esta cualidad es muy importante para obtener sistemas flexibles capaces de adaptarse a continuos cambios, tanto en el tipo y representación de los datos que utilizan como en su tratamiento.

4.7 Agente de control

El objetivo del agente de control es coordinar y supervisar el funcionamiento del resto de elementos del especialista. Su función es doble. Por un lado esperar las órdenes de tarea (OTs) que le envíe su especialista superior, ejecutarlas y devolverle el resultado de la ejecución. Por otro lado, responder adecuadamente cuando se genera un evento que sea de su competencia.

Por otra parte, un especialista debe ser autónomo, por lo que el agente de control debe tratar de responder a todos los problemas que se le presentan durante la ejecución de una tarea. Tampoco conviene olvidar que un especialista debe operar en tiempo real, por lo que el tiempo para tomar decisiones y ejecutarlas es limitado.

4.7.1 Planes

El conocimiento que necesita el agente de control para ejecutar una OT o para responder ante un evento conviene expresarlo explícitamente y de manera que sea

fácil de interpretar, modificar y ampliar. Sobre todo porque en dominios como el de la robótica móvil los cambios en los algoritmos y en los métodos son constantes e inevitables. Para conseguirlo, el agente de control divide cada *OT* y cada respuesta a un evento en un conjunto de tareas organizadas bajo un modelo de plan.

En un plan se especifican los recursos que necesita cada tarea, a qué agente se delega y que prioridad posee, todo ello determina en qué momento y de qué modo se ejecutará dicha tarea. Entre otras cosas, un plan también fija las condiciones bajo las cuales se generan los eventos en la pizarra del especialista (*Esquemas de Reactividad* en la figura 4.15). Por ejemplo, para inhibirlos. Los elementos que constituyen y definen una tarea se muestran en la figura 4.13.

Un *plan marco* se puede definir como la secuencia de pasos genérica que hay que ejecutar para realizar una *OT* o responder a un evento. En cada caso, el plan marco se adapta o particulariza según los parámetros enviados y el estado del especialista. Al conjunto de todos los planes marcos instanciados en un instante dado en el agente de control se le denomina *plan de control*. Este plan es el que regula el comportamiento del especialista en cada momento y varía según surgen nuevos eventos y *OTs*.

Por simplicidad, en AFE-Robótica un plan se representa mediante un grafo (figura 4.14), aunque no habría ningún inconveniente en utilizar otras representaciones, incluso métodos de planificación reactiva del tipo de RAPs (Firby, 1989, 1995), PRS-Lite (Myers, 1996; Konolige y col., 1997; Konolige y Myers, 1998), Teleo-Reactive+ (Zelek, 1996; Zelek y Levine, 1996), etc. Los nodos del plan lo constituyen las distintas acciones de control, mientras que las conexiones o arcos establecen las dependencias entre dichas acciones. Las posibles acciones de control que se pueden especificar en un plan se agrupan en las siguientes clases:

- El chequeo de una *condición* definida sobre un dato del entorno, el estado de un agente, o una variable de control. Generalmente siempre se definen unas condiciones de caducidad (*"¿Cond. Cad.?"* en la figura 4.14) y otras de finalización (*"¿Cond. Fin.?"*). Las primeras marcan si una tarea no se ha podido ejecutar y las segundas indican cuándo se puede considerar concluida una tarea. También se pueden especificar otras condiciones para distinguir entre las distintas etapas de un plan.

- Una lectura o modificación de una variable interna de control o de cualquier dato de la pizarra.

Información Tarea

Nombre o identificador unívoco de la tarea.

Argumentos específicos de la tarea. P.e., destino y origen de una ruta, velocidad media de un desplazamiento, etc.

Condiciones Caducidad que sirven para identificar cuando la tarea no se ha podido ejecutar correctamente. P.e., tiempo máximo de ejecución, o máximo número de intentos.

Condiciones Finalización que se deben cumplir para finalizar la tarea.

Resultados que después de finalizar la tarea se deben enviar al especialista superior.

Incompatibilidades o posibles conflictos con otras tareas. P.e., no se pueden seguir dos rutas diferentes simultáneamente.

Información de Control

Origen la tarea puede provenir del especialista superior, ser una subtarea de otra tarea ya en ejecución o una tarea interna respuesta a un evento.

Plan seleccionado para la ejecución de la tarea.

Estado de la tarea: en ejecución o en suspenso. Aunque también existen otros estados transitorios: nueva, finalizada, anulada, suspendida y reanudada.

Prioridad asociada, para ayudar a resolver los posibles conflictos entre tareas. Puede ser dinámica y depender del estado del especialista. P.e., aumentar según se consume el tiempo de ejecución establecido o disminuir conforme se agotan las baterías del robot.

Figura 4.13: Plantilla de una tarea.

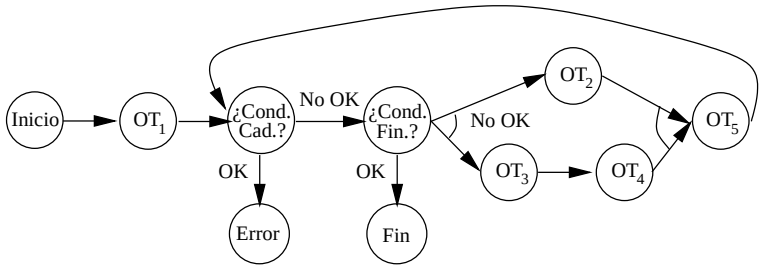

Figura 4.14: Ejemplo de un plan representado mediante un grafo.

- Una *acción de control* sobre el MCD para escribir o leer un dato de la pizarra del especialista superior (sección 4.4.3).

- La modificación de los *Esquemas de Reactividad* de la pizarra, es decir, qué eventos están habilitados y bajo qué condiciones se activan. Se suelen modificar al principio del plan (nodo "*Inicio*" en la figura 4.14), aunque se puede hacer en cualquier momento.[3]

- Un *comando interno de control* (OT) al propio agente de control (módulo GESTOR_OT), que puede significar la ejecución de una nueva tarea o la modificación del estado o los parámetros de alguna tarea ya existente. Con este mecanismo los planes son más generales y dinámicos. Un ejemplo lo encontramos en NAVEGADOR: la localización del robot es una de sus tareas y requiere un plan para su ejecución. Al mismo tiempo, también constituye la primera tarea del plan para planificar una ruta a un destino o realizar cualquier desplazamiento.

- Un *comando de control* para enviar una OT a un agente del especialista.

Las conexiones entre los nodos sirven para implementar las ramificaciones, bucles y demás elementos de control necesarios para la correcta secuenciación y sincronización de la ejecución de un plan. Las conexiones están etiquetadas en función de las condiciones que se establecen dentro del propio plan y de los mensajes (ROT) que se reciben de los agentes en respuesta a las Órdenes de Tarea (OT) enviadas. Cuando no se etiquetan las conexiones, los nodos del plan se ejecutan de forma secuencial.

Generalmente las etiquetas sólo contemplan dos opciones: *Verdadero* o *Falso*, es decir, se cumple o no la condición chequeada, la ejecución de una OT ha sido

[3]Por ejemplo, cuando se cruza una puerta se cambia tres veces la condición de activación del evento *Obstáculo_Próximo:* una por cada tarea MOVER_A.

correcta o hubo algún error. En este último caso la *ROT* suele incluir información de contexto sobre el problema encontrado, que es utilizada para elegir entre las posibles respuestas. Si no es así, se activa una secuencia genérica que se limita a anular o abortar el plan.

Cuando varias conexiones que salen de un nodo están unidas mediante un arco (ver figura 4.14) significa que se ordenan ejecutar simultáneamente las acciones de control asociadas con los nodos destino de cada una de ellas. Si las conexiones entran en un nodo significa que, antes de ejecutar la acción asociada al mismo, es necesario que hayan finalizado correctamente todas las acciones de control asociadas con los nodos origen de cada una de ellas (operación $\bigwedge$).

4.7.2 Componentes del agente de control

Para cumplir la triple funcionalidad que le ha sido encomendada: atender órdenes de tarea, responder a eventos y ejecutar planes, el agente de control necesita tres módulos que operen concurrentemente: el GESTOR_OTS, el GESTOR_EVENTOS y el EJECUTOR_PLAN. Además, necesita un módulo encargado de la gestión de las comunicaciones con sus agentes. En la figura 4.15 se muestran estos cuatro componentes, cómo se relacionan entre sí y cómo interaccionan con el resto de componentes del especialista: la pizarra, los agentes y el módulo de comunicaciones.

Gestor de órdenes de tarea

Una Orden de Tarea (*OT*) es un mensaje de control que se recibe del agente de control del especialista superior. Puede ser de tres tipos:

1. Ejecución de una nueva tarea con sus argumentos y parámetros.
2. Modificación del estado de una *OT* previa: anulación, suspensión o reanudación.
3. Modificación de los parámetros de una *OT* previa (p.e. la prioridad o las condiciones de caducidad o de finalización).

El GESTOR_OT espera la llegada de una *OT* del especialista superior y, según su contenido, realiza la acción de control específica. En el caso de modificar el estado o los parámetros de una tarea ya existente, se limita a identificar la tarea en cuestión y a realizar el cambio pertinente. Si la orden es reanudar una *OT,* antes comprueba

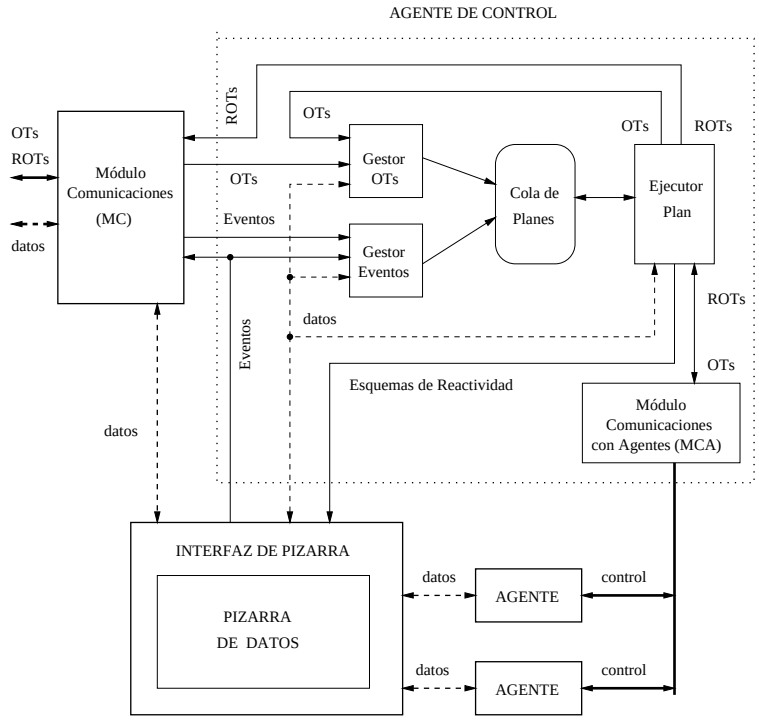

Figura 4.15: Esquema interno del agente de control de un especialista de AFE-Robótica.

si se cumplen las condiciones necesarias para proseguir con su ejecución o, por el contrario, se debe anular o mantener en suspenso.

Si se ejecuta una nueva tarea, el GESTOR_OT busca en una LIBRERÍA OT-PLANES el plan (puede haber más de uno) que mejor se adapte en ese momento al estado del sistema y al conocimiento del entorno. También se debe comprobar si es compatible con el resto de planes en ejecución. Si no existe alternativa y el plan seleccionado es incompatible con otro ya en ejecución, entonces se ejecuta aquel que posea una mayor prioridad.

Con los planes incompatibles de menor prioridad GESTOR_OT puede decidir dos cosas: anularlos o suspenderlos. Esta segunda opción implica una mayor gestión y complejidad de coordinación, ya que en algún momento será necesario reanudarlos. Por otra parte, no siempre es la más eficiente, ya que, en general, no es sencillo establecer bajo qué condiciones se puede reanudar la ejecución de un plan de forma segura. En realidad, GESTOR_OT únicamente fija un estado temporal del plan: "anulado", "suspendido" o "reanudado", GESTOR_PLAN es quién, como veremos más adelante, ejecuta y comprueba los pasos necesarios para obtener el cambio deseado.

Gestor de eventos

El módulo GESTOR_EVENTOS es el encargado de asociar un plan a cada evento recibido, bien interno, procedente de la propia interfaz de pizarra del especialista, bien externo, procedente de otro especialista y que se recibe vía MCD. Todos los planes que se pueden asociar con cada uno de los eventos a los que debe responder el especialista se almacenan en una LIBRERÍA EVENTOS-PLANES. Cuando se recibe un evento, el módulo GESTOR_EVENTOS selecciona el plan apropiado según el estado del especialista y el conocimiento que se posee del entorno en ese momento. Es responsable, en gran medida, de la respuesta del especialista.

Ejecutor de planes

El módulo EJECUTOR_PLAN tiene dos funciones principales. En primer lugar, interpretar la información del plan de control en función del estado del especialista y de la pizarra. En segundo lugar, actualizar la prioridad y modificar el estado de los planes integrados en el plan de control, asociados cada uno a una OT recibida o a un evento activado. Es, indiscutiblemente, uno de los elementos más importantes del agente de control y, por extensión, del especialista.

El EJECUTOR_PLAN se activa de forma tanto asíncrona como síncrona. La primera se utiliza para implementar la interpretación y ejecución del plan de control y ocurre cada vez que se incluye un nuevo plan o se recibe una respuesta (ROT) de un agente del especialista. La ejecución del plan de control consiste en determinar, en base a las $ROTs$ recibidas y a las condiciones chequeadas, los siguientes nodos en ser ejecutados y en enviar las acciones de control pertinentes. Cuando se alcanza un nodo final, el EJECUTOR_PLAN manda al MCC que envíe una ROT al agente de control de su especialista superior (figura 4.15) indicando qué OT ha finalizado y si lo ha hecho con éxito o no.

Si alguna OT enviada a uno de los agentes del especialista no ha podido ser ejecutada correctamente, el EJECUTOR_PLAN busca en el plan si hay definida una solución al problema, en general, utilizando la información de contexto que se devuelve en la propia ROT. Si no existe, entonces el EJECUTOR_PLAN manda enviar, a su vez, una respuesta (ROT) a su especialista superior indicando que la ejecución de la OT que le ha sido encomendada no ha sido posible y, si es factible, añade información adicional sobre el problema encontrado (contexto).

La activación periódica o síncrona sirve para detectar cuándo la ejecución del plan

ha quedado bloqueada en algún punto, generalmente esperando que algún agente ejecute una orden de tarea. En tales situaciones lo primero es tratar de obtener información adicional e interrogar al agente por el estado de la *OT* en cuestión. Si el agente no responde se reinicializa. Si no funciona, se debe adaptar el funcionamiento del agente de control y evitar todas las tareas que necesiten a dicho agente.

Como ya se ha comentado, un plan puede estar en ejecución o suspendido, pero también existen varios estados temporales por los que pasa mientras EJECUTOR_PLAN ejecuta los pasos pertinentes para el cambio de estado definitivo. Así, un plan puede ser: nuevo, finalizado, suspendido, reanudado o anulado. En los tres últimos, EJECUTOR_PLAN se limita a enviar las órdenes pertinentes para modificar el estado de las *OTs* a los agentes, que han sido enviadas y no han tenido respuesta. Cuando se reciban las *ROTs* confirmando el cambio de estado de todas ellas, entonces se cambia el estado del plan y se pone en suspenso, se ejecuta o se elimina, según corresponda. Por otra parte, si un plan finaliza o es nuevo, entonces EJECUTOR_PLAN pasa a ejecutar los pasos necesarios antes de borrar o inicializar el plan (p.e., modificar el *Esquema de Reactividad*).

Módulo de comunicaciones con los agentes

El MÓDULO_COMUNICACIONES_AGENTES (MCA) es el encargado de remitir las Órdenes de Tarea (*OTs*) que lanza el EJECUTOR_PLAN a cada uno de los agentes del especialista, de recibir las respuestas (*ROTs*) que envían los agentes al agente de control y de hacerlas llegar de nuevo al EJECUTOR_PLAN. El MCA dialoga directamente con los módulos de comunicaciones de los agentes, tanto terminales como especialistas.

Capítulo 5

Síntesis de AFE-Robótica

Este capítulo se dedica a la descripción en detalle de la síntesis de AFE-Robótica para la ejecución de la tarea de navegación sobre un robot *Nomad 200.* También se incluyen los aspectos importantes de los distintos especialistas que lo componen junto con la implementación de cada uno y los resultados de su validación.

5.1 Tarea de navegación

Una arquitectura es sólo un armazón, un mero cascarón vacío, si no se aplica a la resolución de una determinada tarea. La especificación e implementación de dicha tarea puede condicionar el rendimiento de la arquitectura. Y viceversa, una mala arquitectura puede limitar el desarrollo de una tarea hasta el extremo de hacerla inviable. Estas y otras razones ya comentadas anteriormente nos han llevado a la conclusión de que la primera tarea a la que aplicar nuestra arquitectura debía ser la navegación.

La navegación es fundamental para la operatividad y autonomía de un robot móvil. Sin embargo, a pesar de la gran cantidad de estudios sobre el tema aún no esta resuelta en entornos no modificados, dinámicos y no totalmente previsibles, precisamente el tipo de entornos donde queremos que se desenvuelva nuestro robot. En general se distinguen dos tipos fundamentales de navegación: geométrica y topológica o basada en características (marcas), aunque algunos estudiosos de los comportamientos de navegación de los animales en la naturaleza (Nehmzow, 1995; Franz y Mallot, 2000) manejan clasificaciones ligeramente diferentes.

Los métodos geométricos tienen como objetivo construir una mapa bidimensio-

nal del entorno a partir del cual se puedan deducir sus propiedades geométricas. En general, los mapas se construyen a partir de una serie de medidas sensoriales discretas, normalmente obtenidas de forma directa a través de sensores de ultrasonidos y láser o deducidas a partir de imágenes. Los mapas geométricos más populares se basan en una rejilla de incertidumbre (Elfes, 1987; Moravec, 1988; Elfes, 1989; Borenstein y Koren, 1991a; Konolige, 1996; Pagac y col., 1998). La aproximación de usar sensores imperfectos combinados con métodos probabilísticos en el marco de mapas geométricos ha producido algunos sistemas de navegación exitosos (Thrun, 1995; Simmons y Koenig, 1995).

Un avance importante en la construcción de mapas geométricos se refiere al uso de conocimiento general *a priori* sobre las simetrías de los entornos y que proporcionan restricciones en el proceso de generación del mapa. Varios investigadores (Castellanos y col., 1997; Thrun y col., 1998b) han demostrado que con estos métodos se puede incrementar sustancialmente la precisión del mapa final.

Existe, sin embargo, una desventaja fundamental en la aproximación geométrica y es que al basarse en muestras de medidas discretas cualquier entorno cuya geometría subyacente sea dinámica significará un reto formidable. Estas condiciones se producen cuando existen cambios drásticos en la topología del entorno o cuando el entorno está poblado por una gran cantidad de personas, cajas, sillas, en definitiva, de cualquier elemento en movimiento. Otra dificultad importante es que la construcción de un mapa detallado junto con la imprescindible corrección de la posición conllevan un fuerte coste computacional y son difícilmente escalables. Aunque hay trabajos que tratan de resolver estos problemas, lo cierto es que la mayoría de las implementaciones construyen el mapa del entorno de forma monótona y que las validaciones se realizan bajo condiciones muy controladas y frecuentemente sólo en pequeñas subregiones de un entorno navegable mucho mayor.

En contraste con las aproximaciones geométricas, los métodos basados en características identifican localmente áreas u objetos únicos en el entorno (marcas) y determinan las posiciones relativas entre ellas (Kuipers y Byun, 1991a,b; Nourbakhsh y col., 1995; Gutierrez-Osuna y Luo, 1996; Castellanos y Tardós, 1999; Dedeoglu y col., 1999). La ventaja de este tipo de mapas es que los errores transitorios inherentes a los sensores de medida tienen menos impacto en el mapa final. La filosofía que subyace a esta aproximación es que el robot no necesita un mapa geométrico detallado de su entorno para operar correctamente. Por el contrario, los investigadores que utilizan mapas basados en características tratan de identificar la mínima información que requiere una navegación robusta y después la capturan en un mapa

topológico que, frecuentemente, también incluye una limitada cantidad de información geométrica.

El principal problema de los mapas basados en características es precisamente la detección de los objetos y lugares significativos del entorno. Lo más usual es tratar de agrupar las medidas sensoriales en segmentos a partir de los cuales se construyen marcas más complejas: paredes, corredores, esquinas, etc. Este método se ha aplicado con éxito a medidas de ultrasonidos (Konolige y col., 1997; Konolige y Myers, 1998), de láser (Edlinger y von Puttkamer, 1994) o de láser e imágenes (Castellanos y Tardós, 1999). También se han probado técnicas basadas en triangulación (Wijk y Christensen, 2000), filtrado (Nourbakhsh, 1998; Konolige y Myers, 1998) e, incluso, sensores especialmente diseñados para detectar ciertas características del entorno (p. e. esquinas (Chong y Kleeman, 1997)).

Por otra parte, algunos investigadores en vez de detectar objetos o características individuales se centran en identificar y clasificar los lugares representativos del entorno a partir de las medidas sensoriales que percibe el robot en cada uno de ellos. Su objetivo es pues asociar cada punto con uno de los lugares significativos del entorno, normalmente definidos en una fase previa de aprendizaje. Hasta ahora lo habitual era usar sensores de ultrasonidos o láser, aunque recientemente se están popularizando las cámaras omnidireccionales. El principal problema de este tipo de métodos (Nehmzow, 2000) es el *perceptual aliasing*, es decir, lugares diferentes se perciben como iguales. Para evitarlo o reducirlo algunos autores utilizan información métrica (generalmente odométrica) o los comportamientos del propio robot.

De todos modos, la misma limitación fundamental se aplica a todos los trabajos que utilizan mapas basados en características y es que todos se basan en la suposición de que el entorno es estático. Por otra parte, aunque en teoría este tipo de mapas se escalan fácilmente, en la práctica la evidencia experimental se limita en tests a pequeña escala sobre entornos artificiales. Una posible explicación de este fenómeno puede ser el pequeño e incompleto conjunto de características seleccionadas para crear los mapas, insuficiente para abordar la inherente complejidad de los entornos reales. La consecuencia más inmediata es que los detectores de características aún no son suficientemente robustos para operar en entornos y condiciones normales.

Sin embargo, los mejores resultados se obtienen integrando los dos tipos de mapas, geométricos y topológicos, dando lugar a mapas híbridos (Edlinger y von Puttkamer, 1994; Lee, 1996; Fusiello y Caprile, 1997; Thrun, 1998; Thrun y col., 1998a; Burgard y col., 1998; Lemon y Nehmzow, 1998; Koenig y Simmons, 1998; Howard y

Kitchen, 1999). Ésta es la línea que hemos seguido en el presente trabajo: utilizar un único mapa global topológico y mapas locales geométricos. Los mapas topológicos consumen pocos recursos, son fáciles de ampliar, son muy intuitivos para los usuarios y en ellos se puede integrar fácilmente conocimiento previo sobre el entorno. Por su parte, los mapas geométricos son sencillos de crear y se pueden utilizar para mejorar la percepción del robot (p.e., filtrar datos espúreos).

Por otra parte, en la navegación nos hemos centrado en comportamientos basados en la topología del entorno (p.e., seguir un contorno o alcanzar una marca) o en el propio robot (p.e., avanzar y girar), ya que son más robustos, generales e intuitivos. Hemos evitado usar, en la medida de lo posible, comportamientos basados en algún sistema de coordenadas (p.e., la odometría), ya que son poco fiables en desplazamientos grandes. De igual modo, preferimos manejar condiciones de finalización de los desplazamientos basadas en características del entorno (p.e., número de puertas detectadas) y no en datos odométricos (p.e., distancia recorrida), aunque a veces estos últimos son inevitables.

Para finalizar, indicar que AFE-Robótica es suficientemente flexible y modular para adaptarse a cualquier estrategia de navegación que se diseñe. Más aún, las especiales características de la arquitectura propuesta permiten implementar simultáneamente varios métodos de navegación, conmutar entre ellos durante la operación del sistema e, incluso, ejecutarlos en paralelo. Esta cualidad es especialmente útil cuando se quieren investigar, comparar y validar distintas estrategias de navegación.

5.1.1 Prototipo implementado

En el capítulo anterior se ha descrito la composición general de AFE-Robótica que se considera necesaria para desarrollar tareas de navegación y se ha especificado una estructura tanto de especialistas como de datos. Sin embargo, para la validación se ha optado por un prototipo algo más simplificado para permitir una implementación más sencilla y rápida y, al mismo tiempo, suficientemente compleja para que el robot alcance un mínimo de inteligencia básica en tareas de navegación y obtener así resultados relevantes.

Las diferencias entre el prototipo implementado (figura 5.1) y la propuesta presentada en el capítulo anterior se resumen en dos. En primer lugar, como sólo se contemplan tareas de navegación (desplazamientos) no se ha creído necesario implementar el especialista MAESTRO, ya que, en general, no es necesario establecer prioridades entre las distintas tareas encomendadas por el usuario ni decidir cómo

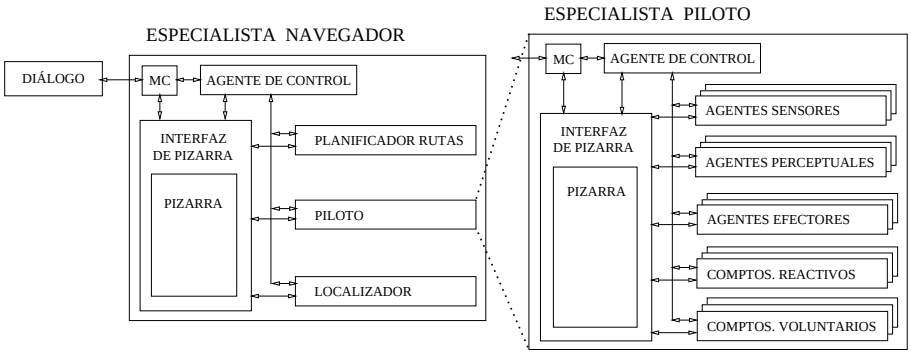

Figura 5.1: Estructura de AFE-Robótica implementada.

se van a ejecutar. En caso de conflicto, sencillamente se ejecuta la última tarea y se abortan las anteriores. Sin embargo, como la interacción con el usuario sigue siendo imprescindible, se mantiene el especialista DIÁLOGO. Para alterar lo menos posible la estructura planteada inicialmente se ha situado a NAVEGADOR como un agente de DIÁLOGO.

La segunda diferencia entre el prototipo y la propuesta inicial es que únicamente se utiliza información *a priori* del entorno en el que se va a desenvolver el robot. De esa manera se simplifican en gran medida las tareas del robot y, por lo tanto, se hace innecesaria la utilización del especialista CONFECCIONADOR_MAPA_GLOBAL. En realidad sólo se almacenan aquellas marcas que se consideran más importantes para el correcto funcionamiento del robot: corredores y paredes. Sin marcas sería imposible mantener el robot localizado con respecto al mapa interno durante largos desplazamientos y sin una correcta posición del robot el mapa del entorno es inútil y todas las funciones que se basan en él (principalmente la planificación de rutas) inviables.

La mejor estrategia para mantener el robot bien localizado es centrarse en las regiones más importantes para los desplazamientos en el interior de un edificio: los corredores. Dentro de una habitación los movimientos del robot están mucho más limitados, por lo que los errores en la odometría están acotados. Además, cuando se abandona una habitación se puede conocer la posición del robot con gran precisión, ya que para ello en general es necesario cruzar una puerta.

Por último, destacar que en un entorno usual el número de marcas (corredores y paredes) es reducido, por lo que en la presente implementación hemos optado por agruparlas en un único mapa. En general la información sobre dichas marcas es relativamente fácil de generar a mano o de hacer que lo haga el propio robot en una

etapa previa de exploración y filtrarla después manualmente para evitar errores e incongruencias. Es más difícil extraer y validar los datos de todas las puertas pero, según los datos obtenidos, no son imprescindibles para acotar razonablemente el error en cuanto a la posición del robot.

5.2 Interfaz de usuario DIÁLOGO

Como ya se ha comentado, en el prototipo implementado DIÁLOGO actúa fundamentalmente como una interfaz entre el usuario y AFE-Robótica, concretamente con el especialista NAVEGADOR. Por tanto, sus tareas se podrían dividir en las siguientes categorías:

1. recoger las órdenes del usuario y enviarlas a NAVEGADOR en forma de *OTs* y otros comandos de control:

   (a) Ejecución de una nueva tarea con sus argumentos y parámetros:
   ejecutar OT-<*identificador*> <*OT*> [**Hasta** *condición 1 [condición 2]* ...]

   (b) Modificación del estado de una *OT*: anular (**abortar**), suspender o reanudar.

   (a) Modificación de los parámetros de una *OT*: p.e. la prioridad, las condiciones de caducidad o de finalización.

   (b) Consulta sobre los datos de ejecución de una *OT*: p.e. tiempo de ejecución, número de intentos, etc.
   estado OT-<*identificador*>

2. recoger y mostrar las respuestas (*ROTs*) de NAVEGADOR.

3. mostrar cualquier dato de una pizarra que se considere de interés.

Por ahora la interacción entre DIÁLOGO y el usuario se realiza mediante una consola. Sin embargo, sería relativamente sencillo ampliarla mediante una interfaz gráfica con menús para las *OTs* más comunes, ventanas de texto para mostrar las *ROTs* y ventanas gráficas para visualizar y representar los datos de interés (mapas, rutas, marcas, etc.). Actualmente, cada agente muestra directamente los datos de interés que genera. Por ejemplo, los agentes del especialista PILOTO visualizan los datos en la interfaz gráfica que se utiliza para conectarse al *Nomad 200*.

5.3 Especialista NAVEGADOR

El especialista Navegador es el responsable de planificar y ejecutar los desplazamientos del robot en su entorno.

5.3.1 Entradas y salidas

Órdenes de tarea

Las *OTs* que recibe NAVEGADOR tienen la siguiente estructura:

ejecutar OT-<*identificador*> <*tarea*> [**Hasta** *condición 1 [condición 2]...*]

Las tareas que NAVEGADOR puede realizar son:

- **Localizar** {**Robot** | **Posición** <x> <y>}. Para determinar en qué lugar del mapa topológico se encuentra el robot según la posición interna almacenada. También se puede especificar una posición en un sistema de referencia absoluto asociado al entorno.
- **Planificar Ruta** <*destino*> [<*origen*>]. Para encontrar el camino más corto entre dos lugares. Ambos, destino y origen, se pueden indicar bien como un **Lugar** <*nombre*> bien como una **Posición** <x> <y>. Si no se especifica el origen de la ruta se sobreentiende que es el lugar que ocupa el robot en ese momento.
- **Ejecutar Ruta**. Su objetivo es seguir un camino previamente calculado y almacenado en la pizarra de NAVEGADOR.
- **Ir A** {**Lugar** <*nombre*> | **Posición** <x> <y>}. Es la tarea más útil y completa porque con ella podemos conseguir que el robot se desplace a cualquier lugar de su entorno.
- **Pose Robot** <x> <y> <θ>. Con ella se pueden modificar los valores de posición y orientación odométricos que el robot calcula internamente.
- **Mapa** <*nombre*>. Ejecuta todas las operaciones necesarias para cargar un nuevo mapa del entorno.

Excepto **Ir A**, ninguna tarea necesita definir condiciones de caducidad o de finalización. La condición de finalización de **Ir A** es que el robot llegue al lugar indicado

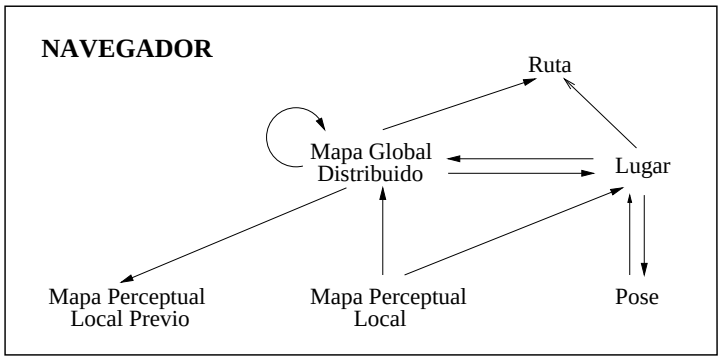

Figura 5.2: Estructura de datos en la pizarra del especialista NAVEGADOR.

como destino. Las condiciones de caducidad son el número de intentos y el tiempo máximo para conseguirlo.

Datos de entrada y de salida

En el prototipo implementado no se establecen datos de entrada ni de salida, salvo las consultas que pueda realizar el usuario a NAVEGADOR a través de DIÁLOGO. De haber implementado el especialista MAESTRO sí los habría. Se considera como entrada cualquier información previa sobre el entorno, bien en forma de mapas, bien como anotaciones (p.e., lugares donde se encuentran los recargadores de baterías, nombres asociados a algunos lugares o regiones del entorno, lugares peligrosos o no aconsejables para el robot, etc.).

El principal dato de salida es el mapa global distribuido, modificado por el robot, al menos parcialmente, de acuerdo con las modificaciones detectadas en el entorno, y que sirve para intercambiar o compartir con el usuario u otros robots. La información relativa a las rutas también se puede considerar como datos de salida. Esta información hace referencia a si existe alguna ruta entre dos lugares que cumpla unos determinados criterios, las características de la ruta (distancia, tiempo y/o coste estimado), si se puede integrar con alguna otra ruta previa, etc.

5.3.2 Pizarra

En la figura 5.2 se pueden ver los datos que se almacenan en la pizarra del especialista NAVEGADOR y sus relaciones:

- MAPA GLOBAL DISTRIBUIDO (MGD) es el mapa global del entorno.

- RUTA es la última ruta planificada o especificada por el usuario. Podría haber más de una.
- LUGAR es la región que ocupaba el robot la última vez que fue localizado de manera global (típicamente, antes de planificar una ruta).
- MAPA PERCEPTUAL LOCAL PREVIO (MPLP) es la recopilación de los datos que se conocen sobre una determinada región. Es la información que utiliza o necesita el agente PILOTO cuando el robot recorre dicha región.
- MAPA PERCEPTUAL LOCAL (MPL) es una copia de la información que el agente PILOTO recoge de una región mientras la recorre. Es un dato *externo* de la pizarra que sólo se actualiza cuando un agente accede a él. Se puede utilizar para mantener actualizado el MGD.
- POSE es la posición y orientación del robot que calcula y mantiene el agente PILOTO en su pizarra. Es un dato *externo* que se actualiza sólo cuando se accede a él.

Mapa global del entorno

El mapa global distribuido (MGD) se confecciona a partir del conocimiento previo proporcionado al sistema (p.e. un plano del entorno) y de la información que va extrayendo PILOTO de los datos sensoriales, es decir, el MAPA PERCEPTUAL LOCAL (MPL) y la posición y orientación del robot (POSE). En el primero se incluyen la posición de los obstáculos (MAPA DE OBSTÁCULOS) y datos sobre los objetos particulares (MARCAS) detectados en una cierta región del entorno. La POSE es imprescindible para poder relacionar los distintos mapas locales entre sí y obtener un único mapa global.

El MGD está formado por un mapa topológico de todo el entorno, junto con el mapa de obstáculos y las marcas de cada una de las regiones en las que se divide dicho entorno. El mapa topológico se representa mediante un grafo donde los nodos son las regiones y los arcos son las conexiones entre ellas. En un entorno típico de oficinas se consideran tres tipos de regiones: corredores, habitaciones e intersecciones entre corredores, y dos tipos de conexiones: espacio libre y puertas.

Sin embargo, con esta filosofía un corredor se representaría como un único nodo y, por lo tanto, no se puede diferenciar entre rutas que recorran parcial o totalmente un corredor. Para evitar esta ambigüedad se divide cada región en espacios más

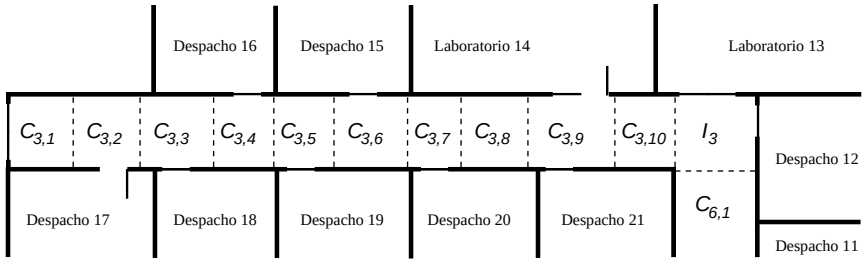

Figura 5.3: División de un corredor en lugares según las marcas detectadas (principalmente puertas).

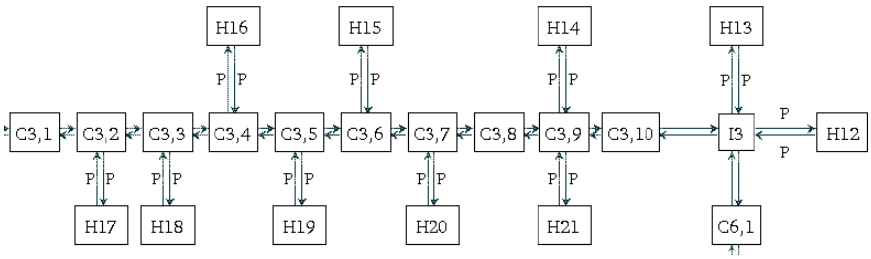

Figura 5.4: Mapa topológico que corresponde al entorno de la figura 5.3. Los nodos se clasifican en corredores (C), intersecciones entre corredores (I), habitaciones (H) y salas (S). Las conexiones que representan espacio libre no tienen etiqueta y las puertas se marcan como "P".

reducidos, denominados lugares, es decir, se sustituyen los nodos con muchas conexiones por un conjunto de nodos con una menor conectividad, típicamente hasta un máximo de cuatro. De esta manera, es posible representar rutas parciales dentro de una región (p.e., un corredor).

El criterio para dividir las regiones en lugares es similar al de (Nourbakhsh, 1998): dos posiciones pertenecen a lugares diferentes (iguales) cuando desde ellas se detectan marcas diferentes (iguales). En la figura 5.3 se muestra un ejemplo de cómo se divide un corredor según las puertas que contiene. Si existe mucha distancia entre dos marcas las posiciones intermedias pueden constituir un lugar que se diferencia de sus vecinos porque en él no se detecta ninguna puerta. El mismo criterio se aplica a habitaciones e intersecciones entre corredores, excepto cuando son de dimensiones reducidas. En la figura 5.4 se puede ver el resultado de aplicar estos criterios al entorno de la figura 5.3. En la figura 5.5 se muestra el mapa topológico generado para la sede del Departamento de Electrónica y Computación de la universidad de Santiago de Compostela.

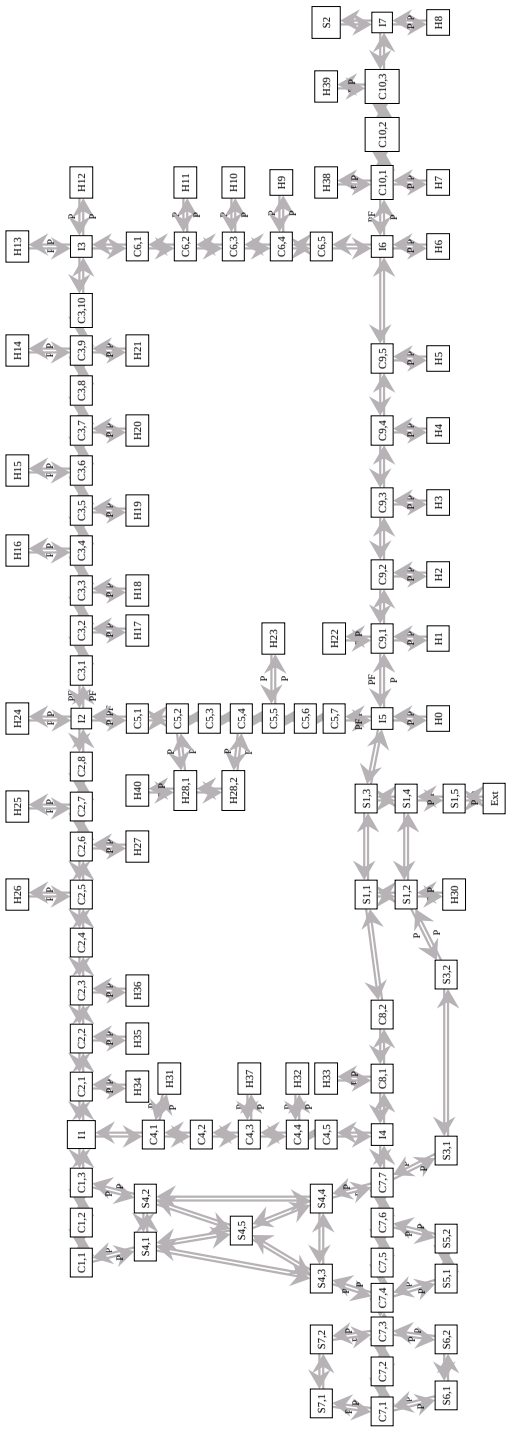

Figura 5.5: Mapa topológico de la sede del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela.

Para finalizar, comentar que el mapa topológico se amplía con información métrica para cada conexión que representa espacio libre. Con estos datos se consigue que la operación de algunos comportamientos (p.e., seguir contorno) sea más robusta y se puedan aplicar en casos donde la información topológica es insuficiente. La información métrica sirve para dar una idea, aunque sea aproximada, de la distancia que separa cada par de nodos unidos mediante espacio libre. Una forma sencilla de estimarla consiste en sustituir cada nodo del grafo por un punto bidimensional (p.e., el centro del lugar asociado a dicho nodo) y después calcular las distancias euclídeas entre cada par de puntos conectados.

5.3.3 Agentes

Se han implementado tres de los agentes necesarios en el especialista NAVEGADOR. En la tabla 5.1 se resume la función y los datos de entrada y de salida de cada uno. El cuarto agente, CONFECCIONADOR_MAPA, no se implementa porque se supone que el mapa global, confeccionado *a priori* por el usuario, no necesita ser actualizado.

El agente PLANIFICADOR_RUTAS utiliza el mapa global del entorno para trazar una ruta que pueda seguir el robot para llegar al destino elegido desde la posición que ocupa el robot o desde otra cualquiera. La ruta es una mera secuencia de lugares del entorno (nodos en el grafo del mapa topológico) que el módulo de control de NAVEGADOR se encarga de traducir en tareas que sus agentes puedan ejecutar para que el robot siga dicha ruta.

El agente PILOTO es responsable de mover el robot según las órdenes que recibe del módulo de control de NAVEGADOR. También se encarga de procesar los datos sensoriales para extraer información a partir de la cual se pueda localizar el robot.

El agente LOCALIZADOR determina o verifica en qué lugar del mapa global se encuentra el robot según la posición del robot calculada y las marcas del entorno percibidas por PILOTO. Esta tarea es crítica, puesto que si la localización del robot no es correcta el mapa del entorno se torna inútil.

5.3.4 Síntesis de planes y control

Prácticamente existe un plan por cada tipo de *OT* que puede reconocer y ejecutar NAVEGADOR, aunque algunas *OTs* se ejecutan en base a otras más simples.

Agente	Entradas	Salidas	Función
PILOTO	MPLP, POSE	MPL POSE	realizar los desplazamientos locales y procesar los datos sensoriales
LOCALIZADOR	MGD, MPL, POSE	LUGAR	identificar el lugar que ocupa el robot en el mapa global del entorno
PLANIFICADOR RUTAS	MGD, LUGAR	RUTA	calcular un camino entre dos puntos del entorno según el mapa global

Tabla 5.1: Función de los agentes del especialista NAVEGADOR y los datos de su pizarra que necesita cada uno como entrada y como salida.

Localizar Posición

En general consiste en determinar a qué lugar corresponde una posición especificada en un sistema de referencia absoluto asociado al entorno. Consiste simplemente en recoger los datos de la posición en la propia OT y en enviar una orden al agente LOCALIZADOR. Este mismo plan sirve para determinar la localización actual del robot, con la diferencia de que la primera orden es actualizar la POSE del robot almacenada en la pizarra de NAVEGADOR y el hecho de que el agente LOCALIZADOR utiliza la pizarra para obtener sus datos de entrada y escribir su resultado, datos POSE y LUGAR respectivamente.

Planificar Ruta

Una ruta consta de un origen y un destino que se pueden especificar directamente como un lugar o indirectamente a través de un punto en el sistema de coordenadas absoluto asociado al entorno. Como la planificación de rutas se hace a nivel topológico, sólo se puede utilizar información cualitativa de posición, es decir, los lugares en los que se ha divido el entorno. Por lo tanto, los primeros pasos del plan que planifica una ruta sirven para calcular los lugares tanto de origen como de destino.

Si no se especifica el origen se considera que la ruta parte de la posición actual del robot, por lo tanto, el primer paso es activar el plan **Localizar Posición Robot** explicado previamente. Si alguno de los extremos del plan se especifica de forma indirecta se activa el plan **Localizar Posición**. Como se puede comprobar en AFE-Robótica, un plan puede estar formado por otros más simples, haciendo el agente de control más modular. Conocidos los lugares de origen y de destino de la ruta, sólo resta ordenar al agente PLANIFICADOR_RUTAS que determine el camino más

corto entre ambos.

Para simplificar la implementación del control de NAVEGADOR, toda la información referente a una ruta (puntos y/o lugares de origen y de destino, camino trazado, etc.) se almacena en la pizarra, de modo que es accesible y manipulable para todos sus agentes. Esta técnica facilita el procesamiento de múltiples rutas simultáneamente y evita interferencias indeseables.

Ejecutar Ruta

Este plan se limita a ejecutar una ruta que ya ha sido previamente trazada. Una ruta consiste en un subconjunto ordenado y continuo de nodos del grafo que constituye el mapa topológico del entorno. Su traducción a *OTs* ejecutables por los agentes de NAVEGADOR se realiza en el control de NAVEGADOR y se basa en la idea de agrupar nodos consecutivos de la ruta en subrutas. En general, cada subruta equivale a una o varias *OTs.* Una vez enviadas, el control de NAVEGADOR espera la respuesta de cada una para buscar la siguiente subruta, traducirla y ejecutarla. Como se puede ver, los procesos de traducción y ejecución de una ruta se entrelazan y se condicionan mutuamente.

La división de la ruta en subrutas depende de la funcionalidad de los agentes de NAVEGADOR, mientras que la traducción de cada subruta a *OTs* se realiza conociendo la sintaxis de las órdenes de tarea para cada agente. A grandes rasgos, el algoritmo utilizado es el siguiente.

Si la conexión entre el nodo actual de la ruta (en el que está el robot) y el siguiente corresponde a una puerta, entonces se envían las órdenes oportunas para cruzarla. En caso contrario significa que es espacio libre[1] y se analizan el resto de nodos de la ruta hasta encontrar una puerta, una intersección entre corredores o el final de la ruta. Todos esos nodos forman una subruta que se puede ejecutar utilizando un mismo comportamiento (típicamente, seguir un contorno o moverse manteniendo una orientación). Cuando se llega a una intersección, su cruce es algo más laborioso. Veamos las *OTs* que se generan en cada caso.

Cruzar una puerta. En primer lugar, se ejecuta el plan de **Localizar Robot** para comprobar si realmente el robot está en el lugar en el que se supone (nodo actual). A continuación hay que localizar la puerta a cruzar. Lo más habitual es que el robot haya tenido que seguir un contorno hasta detectar la puerta que

[1] Por ahora no se establecen diferencias según el robot esté en una habitación o en un corredor.

se quiere cruzar. Si no es el caso, lo que se hace es ordenar al agente PILOTO que ejecute un barrido completo (giro de la torreta) detectando todas las puertas próximas y después se ordena escoger la más cercana.[2] Seleccionada la puerta se envía a PILOTO la orden para cruzarla. Para mayor seguridad, después de cruzar la puerta también se puede comprobar si el lugar es el previsto o el robot se ha equivocado de puerta.

Seguir un contorno. El primer paso consiste en calcular la orientación del corredor y ordenar al robot que gire para alinearse con él. Si se acaba de cruzar una puerta llega con girar un cuarto de vuelta hacia el lado apropiado. Después se estudia toda la subruta para establecer el lado del contorno a seguir (o el ángulo) y las condiciones de finalización: distancia a recorrer y/o número de puertas a detectar, bien a la derecha, bien a la izquierda. También se podrían utilizar intersecciones entre corredores si el agente PILOTO es capaz de detectarlas. En general, cuantas más condiciones de finalización se definan más robusta es la ejecución del comportamiento.

La condición de distancia se calcula sumando los pesos de las conexiones que conforman la subruta. Para fijar una condición basada en puertas es imprescindible que el último nodo de la subruta contenga al menos una puerta. Si la ruta continúa por una de ellas (y, por tanto, dicha puerta será cruzada después) entonces se elige su lado. Si hay dos se elige el lado donde haya más puertas en la subruta, o el lado derecho en caso de empate. Por último, se cuenta el número se puertas que la subruta posee en dicho lado.

Cruzar una intersección. En general hay dos maneras sencillas de cruzar una intersección: siguiendo un contorno o manteniendo una dirección. La primera es más robusta, pero exige que el corredor de entrada en la intersección y el de salida sean adyacentes. En estos casos, lo más práctico es integrar el nodo de la intersección con la subruta anterior, de forma que ésta acabe *después* de la intersección. Lo normal es que el cruce de la intersección fije el contorno a seguir (es decir, el lado del corredor).

Si la ruta no cruza una intersección siguiendo un contorno entonces siempre se puede hacer en dos etapas: una para entrar en la intersección y otra para salir. En

[2]Este sistema no siempre selecciona la puerta acertada, pero es muy simple y funciona razonablemente bien en las situaciones más usuales.

```
1. Localizar Robot

2. Planificar Ruta al destino deseado

3. Ejecutar Ruta

4. Si el robot no en destino, volver a 2
```

Figura 5.6: Etapas en las que se divide la tarea **Ir A**.

ambos casos se utiliza el comportamiento de PILOTO de mover el robot manteniendo una orientación y fijando como condición de finalización la distancia a recorrer. Como se puede intuir, esta técnica no es tan robusta como la anterior. Tampoco conviene olvidar que la subruta anterior acaba *antes* de llegar a la intersección. En cualquier caso, en ambos métodos es deseable comprobar el lugar que ocupa el robot después de cruzar la intersección por si se ha cometido algún error.

Ir A destino

Este plan se ejecuta a partir de los tres anteriores (**Localizar Robot**, **Planificar Ruta** y **Ejecutar Ruta**) siguiendo la secuencia de la figura 5.6. Una vez ejecutada la ruta se comprueba si el robot ha alcanzado el destino pedido. En caso negativo se intenta de nuevo, aunque tomando como origen la posición que ocupa realmente el robot.

También se replanifica la ruta si durante la ejecución de la ruta alguna de las localizaciones del robot que se utilizan como verificación intermedia no es correcta. Ahora bien, si el agente PILOTO no es capaz de ejecutar correctamente alguna de las *OTs* que recibe (p.e., no puede seguir un contorno o cruzar una puerta) o no existe una ruta practicable, entonces el plan se aborta.

Reactividad

La reactividad en el agente NAVEGADOR pasa, generalmente, por suspender la tarea en ejecución y sustituirla por otra que resuelva el problema que ha generado el evento detectado (localizar el robot, recargar las baterías, etc.). Una vez solucionado el problema se puede restaurar la tarea original o, lo más probable y con menos efectos colaterales, simplemente anularla.

Nivel de batería muy bajo. Este evento se genera en la pizarra PILOTO cuando el nivel de voltaje en las baterías del robot es peligrosamente bajo. Ante una situación tan extrema la primera tarea consiste en ordenar a PILOTO que pare el robot (p.e., arrimado a la pared más próxima) y, como no existe la posibilidad de reducir el consumo del robot (p.e., desconectando dispositivos no vitales para la operatividad del robot), se apaga el sistema de forma ordenada.

Nivel de batería bajo. Este evento también se genera en la pizarra de PILOTO. El plan más simple y general para atender este evento consiste en ejecutar la tarea "**Ir A** <recargador más cercano>". Para esta tarea sólo se necesita almacenar los distintos lugares en dónde se encuentran los cargadores de las baterías del robot y determinar cuál de ellos está más cerca (no en distancia euclídea sino en coste energético). Sencillas ampliaciones de este plan serían reenviar el evento al especialista DIÁLOGO para que a su vez avise al usuario o comprobar si hay suficiente energía para finalizar la tarea en curso antes de desplazar el robot para su recarga.

5.3.5 Evaluación

En esta sección se mostrará el funcionamiento del especialista NAVEGADOR, estudiando su tarea más general **(Ir A)** y utilizando el mismo ejemplo visto en el capítulo anterior, pero centrándonos principalmente en los detalles de la implementación. La validación de la tarea se deja para el capítulo siguiente, ya que se solapa en gran medida con la validación de la arquitectura.

Ejemplo Ir A

En la figura 5.7 se muestra parte del entorno del robot así como su posición en el momento en el que se le envía la *OT*: "**Ir A** *despacho 17*". La primera etapa consiste en activar el plan **Localizar Robot**: actualizar la POSE del robot almacenada en la pizarra de NAVEGADOR (**ejecutar** OT-*1* **Posición Robot**) y a continuación ordenar al agente LOCALIZADOR que determine el lugar que ocupa el robot (**ejecutar** OT-*2* **Localizar Robot**). En el ejemplo de la figura 5.7 el robot está en la posición *(1900, 1500, 2700, 2700)*[3] y en el lugar *H14*.

[3]Los dos primeros términos, que corresponden a las coordenadas x e y, están expresados en décimas de pulgada, mientras que los otros dos, ángulos de la base y de la torreta, están en décimas de grado.

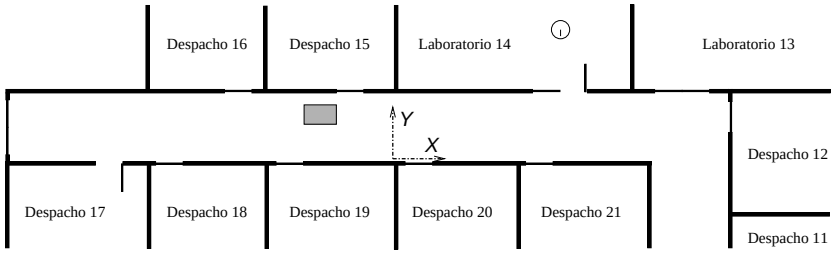

Figura 5.7: Vista parcial del Departamento de Electrónica y Computación donde se muestra el origen de coordenadas del sistema de referencia absoluto que se ha asociado a dicho entorno.

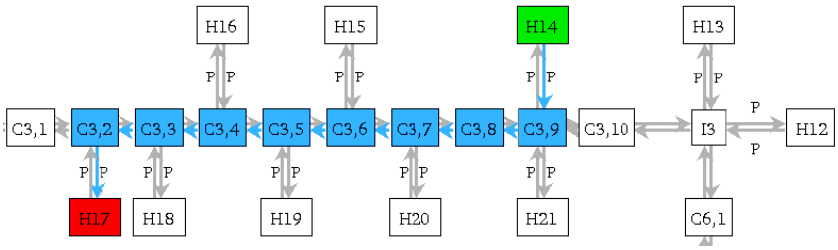

Figura 5.8: Ruta entre el laboratorio 14 y el despacho 17.

La segunda etapa de **Ir A** consiste en activar el plan **Planificar Ruta**. Como se conoce tanto la localización topológica del robot como la de su destino, este plan se reduce a ordenar al agente PLANIFICADOR_RUTA que determine el mejor camino entre ambos (**ejecutar** OT-*3* **Planificar Ruta** H14 H17[4]). La ruta encontrada se representa en la figura 5.8.

La tercera etapa, y la más compleja, consiste en ir traduciendo y ejecutando las distintas subrutas del camino encontrado (**Ejecutar Ruta**). En primer lugar, como la conexión entre el primer y el segundo nodo de la ruta corresponde a una puerta, la primera tarea consistirá en cruzarla. Al estar el robot en el nodo inicial de la ruta, no es necesario comprobar la localización del robot. Pero, como no se conoce la posición de la puerta a cruzar, la primera orden será para el agente PILOTO y consistirá en realizar un giro completo de la torreta para detectar todas las puertas próximas (**ejecutar** OT-*4* **Girar** *1,1 Derecha* **Hasta Puerta** *100*).[5] A continuación se ordena a PILOTO que cruce la puerta más cercana (**ejecutar** OT-*5* **Cruzar Puerta Cercana**).

Una vez cruzada la puerta (figura 5.9) se podría volver a localizar el robot para

[4]Como se puede observar, primero se indica el origen y después el destino.
[5]El sentido del giro de la torreta es arbitrario.

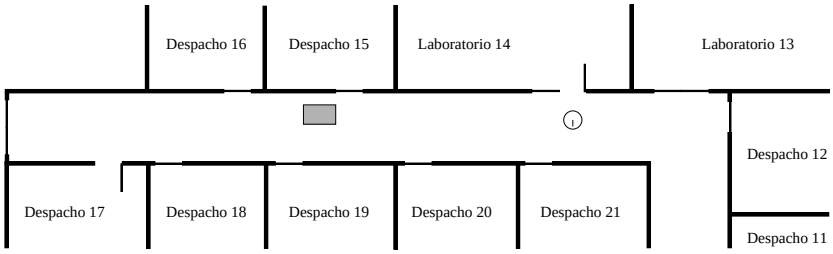

Figura 5.9: Posición del robot después de cruzar la puerta del laboratorio 14.

comprobar que está en el nodo apropiado (*C3,9*) y que se ha cruzado la puerta correcta. En cualquier caso, a partir de ese punto los siguientes nodos de la ruta hasta llegar al nodo *C3,2*, es decir, justo antes de entrar en el despacho 14 (figura 5.8), se unen mediante conexiones de espacio libre. Por lo tanto, se integran en una misma subruta que se puede seguir mediante un comportamiento de seguir contorno. Sin embargo, antes se orienta correctamente el robot, es decir, se ordena a PILOTO que lo gire un cuarto de vuelta a la derecha (**ejecutar** OT-*6* **Mover A** Ángulo *-90 -90* **Relativo**).

El primer paso para generar la orden de seguir el corredor es determinar la condición de finalización basada en la distancia recorrida. Para calcular dicha distancia únicamente se necesita sumar los pesos de todas las conexiones de espacio libre de la subruta (*1310* cm en el ejemplo de la figura 5.8). A continuación se comprueba si también se puede fijar una condición de finalización basada en la detección de puertas. En este caso, el nodo final no sólo posee una puerta sino que además forma parte de la ruta. Por lo tanto, dicha puerta determina en qué lado se deben detectar las puertas: *Izquierd*a. Sólo resta contar todas las puertas de la subruta que queden en ese lado, en este caso *4*.[6] Por último, como la subruta no cruza ninguna intersección se puede seguir el corredor por cualquiera de sus dos lados. En esas situaciones se suele escoger el lado en el que se detectan las puertas (*Izquierda*). La orden final quedaría como **ejecutar** OT-*7* **Seguir Contorno** *Izquierda* **Hasta Distancia** *1310* **Puerta** *4 Izquierda.*

Cuando PILOTO acaba de ejecutar la orden de seguir contorno se comprueba si el robot está situado en el lugar esperado (en este ejemplo sería el nodo *C3,2*). A continuación se comprueba si PILOTO ha detectado la puerta pedida o sólo se ha cubierto la distancia especificada. En el primer caso se ordena directamente a PILOTO que cruce la puerta detectada. En el segundo caso se ordena antes que gire

[6] Al contar el número de puertas no se consideran las del primer nodo de la subruta.

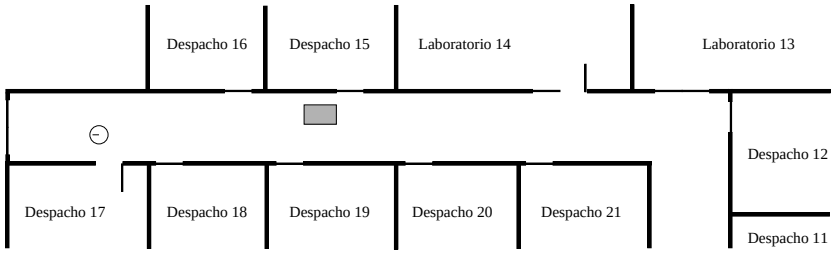

Figura 5.10: Posición prevista del robot después de ejecutar la séptima orden de tarea y seguir el contorno izquierdo del corredor.

la torreta para detectar todas las puertas próximas y después se le ordena cruzar la puerta más cercana.

Si todo sale como estaba previsto, el robot habrá alcanzado el destino final. Sin embargo, como nuestro robot se desenvuelve en un entorno dinámico y poco previsible, ésta no será la situación más usual. Por lo tanto, un buen plan de control tendrá que prever y resolver todos los problemas que pueden surgir durante su ejecución. En la ejecución de la tarea **Ir A** los problemas más comunes son tres: que no se encuentre ninguna ruta válida, que PILOTO no pueda ejecutar una tarea y que el robot no esté en el lugar previsto.

Los errores en la localización del robot son bastante habituales debido a las limitaciones sensoriales del *Nomad 200* y a lo difícil que es mantener la posición del robot correctamente actualizada. Como ya se ha visto, este tipo de errores se descubre rápidamente al localizar el robot, generalmente antes o después de cruzar una puerta o una intersección. La solución más sencilla a este problema consiste en hacer un nuevo intento de llegar al destino y replanificar una nueva ruta tomando como origen la localización real del robot.

Una situación bastante común es que el robot cruce la puerta equivocada, bien porque la puerta correcta no se haya detectado, bien porque se ha seleccionado incorrectamente. En la figura 5.11 se muestra un ejemplo de dicha situación. Si todas las tareas se ejecutan correctamente, la nueva ruta se traduce en: cruzar la puerta cercana, girar un cuarto de vuelta a la izquierda, seguir contorno hasta detectar la primera puerta a la izquierda y cruzar la puerta detectada.

Otra situación bastante común es que PILOTO no detecte la puerta prevista y continúe siguiendo el contorno hasta recorrer la distancia especificada en la *OT*. Como dicha medida es aproximada, lo normal es que el robot avance más de lo necesario y sea necesario replanificar la ruta al destino (figura 5.12). Primero se

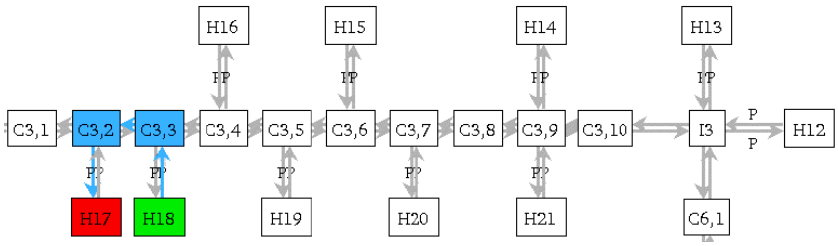

Figura 5.11: Ruta desde el despacho 18 al 17.

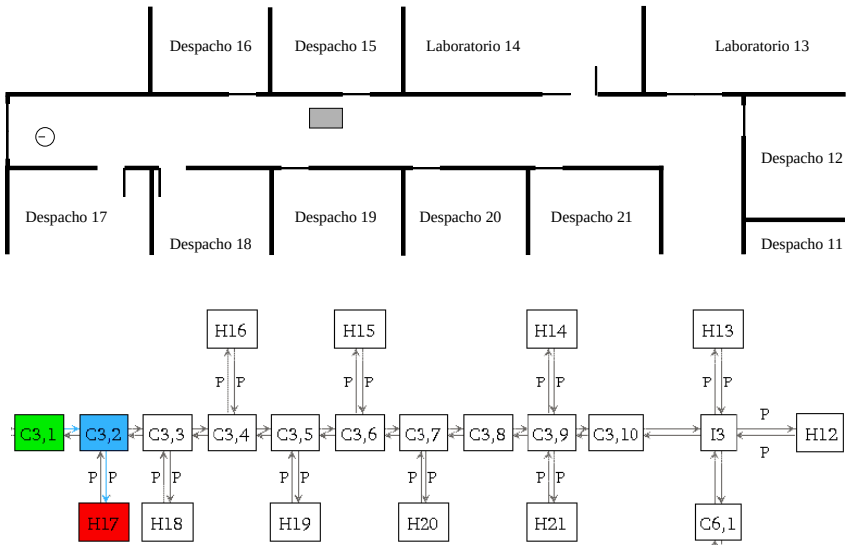

Figura 5.12: Ruta desde el inicio del corredor hasta el despacho 17.

debe comprobar si la orientación del robot coincide con el sentido en el que se va a seguir el corredor. Como en este caso no coinciden, se ordena a PILOTO que gire el robot media vuelta. Después se le ordena seguir contorno por la derecha hasta la primera puerta a la derecha y, si todo sale bien, cruzar la puerta detectada.

PILOTO no siempre es capaz de ejecutar correctamente todas las tareas que se le encomiendan porque opera en un entorno muy dinámico e impredecible. Las situaciones más comunes son no poder cruzar una puerta o no poder alinear la base y/o la torreta con un contorno especificado. Una solución sería intentarlo de nuevo esperando tener más "suerte". Aunque lo más sencillo es abortar la tarea.

Sin embargo, existen situaciones más difíciles de identificar, por ejemplo, un corredor bloqueado. Como PILOTO sigue un contorno y no el corredor, al llegar al obstáculo simplemente lo considera como parte del contorno, lo bordea y acaba siguiendo el lado contrario del corredor (figura 5.13). La anomalía se detecta rápi-

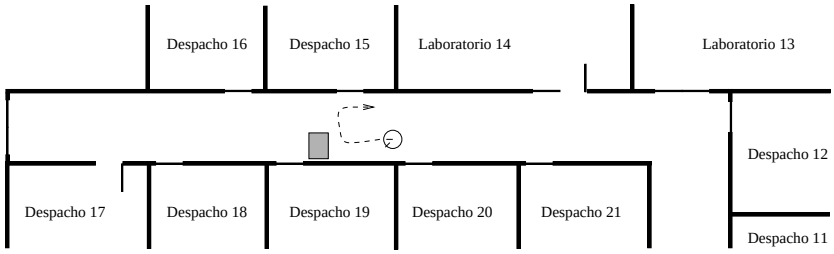

Figura 5.13: Ejemplo de un corredor bloqueado parcialmente y trayectoria del robot cuando sigue el contorno izquierdo.

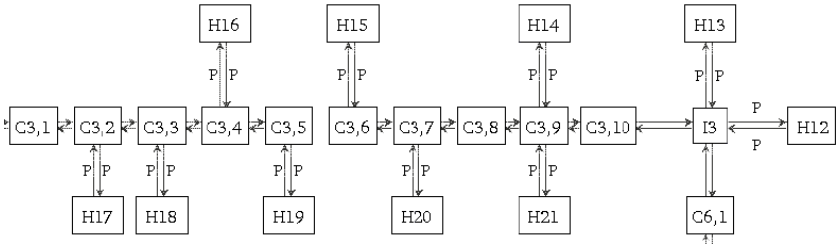

Figura 5.14: Mapa topológico donde se representa un corredor bloqueado (falta de conexión entre los nodos *C3,5* y *C3,6*).

damente al localizar el robot después de ejecutar la tarea, pero es más difícil deducir la causa y el alcance del problema: bloqueo total, parcial o temporal (p.e. por un obstáculo móvil).

En el primer caso habrá que determinar entre qué dos lugares está bloqueado el corredor para anular la conexión adecuada en el mapa topológico (figura 5.14). Si el bloqueo no es completo probablemente el robot pueda sortear el obstáculo siguiendo el corredor por el lado opuesto al ya intentado. Si es así, habrá que memorizarlo para seleccionar el lado correcto en el futuro. Si el bloqueo lo ha producido un objeto en movimiento se debería repetir el intento.

Como se ha podido comprobar con estos pocos ejemplos, la ejecución de una ruta no es un proceso fijo y predeterminado sino que es dinámico y variable, ya que depende del resultado de la ejecución de las órdenes que el controlador de NAVEGADOR va enviando a sus agentes durante la ejecución de cada subruta. En general, cuántos más problemas pueda anticipar y detectar el plan asociado a una tarea y cuánto mejor sepa resolverlos, más inteligente será la operación del especialista NAVEGADOR y, por tanto, del robot.

5.3.6 Discusión

Las tareas implementadas son suficientes para que un robot móvil pueda desplazarse por su entorno, pero, por supuesto, no se trata de un conjunto cerrado y se pueden añadir nuevas tareas o modificar las ya existentes. Por ejemplo, se pueden implementar tareas del tipo **Ir A Casa** o **Ir A Recargador** simplemente señalando o reconociendo dichos lugares sobre el mapa. Otra mejora sería indicar varios objetivos intermedios en un desplazamiento en vez de un único destino. Una forma sencilla de implementarlo sería procesar los destinos uno a uno, es decir, cada vez que se alcanza un objetivo se ejecuta el siguiente y así sucesivamente, hasta completar la lista. Para definir rutas cíclicas sólo se necesita especificar que al finalizar la lista se vuelva al punto de partida. De igual forma, las rutas cíclicas se podrían repetir hasta completar el número de veces exigido.

Otro punto que debe ser mejorado es la resolución de aquellas situaciones donde el agente PILOTO no es capaz de completar una tarea (por ejemplo, seguir contorno o cruzar una puerta). Hacerlo de manera general exigiría entender todos y cada uno de los tipos de errores que se pueden presentar, por qué se producen, bajo qué situaciones y cuáles serían las respuestas más adecuadas para cada uno. Mientras tanto, se pueden apuntar soluciones a problemas particulares. Por ejemplo, para evitar y/o detectar el bloqueo de un corredor se podría implementar en PILOTO una tarea especializada en seguir corredores y en detectar cuando se llega al final de un pasillo.

Para que la operación de NAVEGADOR sea más robusta se podría activar un evento **robot perdido** cuando no fuese posible localizar el robot. Una solución podría ser ordenar al agente PILOTO que explore el entorno inmediato y tratar de reconocerlo. En un corredor bastaría con seguirlo hasta obtener suficiente información para identificarlo y localizar el robot (p.e., ancho del corredor, marcas detectadas y distancias entre ellas, etc.). Como los lugares cerrados (p.e., una habitación) son más difíciles de diferenciar entre sí, lo más sencillo sería encontrar y cruzar una puerta con la esperanza de llegar a un corredor.

5.4 Agente LOCALIZADOR

El agente LOCALIZADOR determina o verifica a qué lugar del entorno (LUGAR) corresponde una posición y orientación del robot (POSE) y/o un conjunto de objetos del entorno (MARCAS). Si las marcas y la POSE son las que detecta el agente PILOTO

entonces el lugar calculado es el que ocupa el robot. La POSE no necesita ser muy precisa, basta una mera aproximación, pero debe ser fiable. Tampoco es necesario detectar todas las marcas de una región del entorno, pero sí un número significativo de ellas y medir sus distancias relativas.

5.4.1 Entradas y salidas

Órdenes de tarea

- **Localizar** [**Robot** | **Posición** $<x>$ $<y>$]

Datos de entrada

- MAPA GLOBAL DISTRIBUIDO (MGD)
- MAPA PERCEPTUAL LOCAL (MPL) (sobre todo, las MARCAS detectadas)
- POSE

Datos de salida

- LUGAR

5.4.2 Síntesis del agente

Para simplificar la implementación del agente LOCALIZADOR hemos desarrollado un algoritmo que se basa exclusivamente en la posición del robot. El método consiste en representar cada lugar del entorno por un polígono (generalmente un rectángulo) y en calcular a cuál de ellos pertenece la posición dada.

5.4.3 Evaluación

Al basar la localización del robot únicamente en la posición, su funcionamiento depende de la fiabilidad con la que PILOTO calcula y mantiene la POSE del robot. De hecho, el método es muy frágil, puesto que en algunas situaciones un pequeño error en la posición del robot (p.e., en la frontera entre dos lugares vecinos) provoca una confusión que, según los casos, puede ser muy seria y difícil de corregir.

5.4.4 Soluciones existentes

La localización del robot a nivel global se denomina "Problema de la Primera Localización" (Cox, 1991) o "Problema del Robot Perdido" (Leonard y Durrant-Whyte, 1991) y consiste en determinar la posición más probable del robot a partir de los datos sensoriales y del mapa interno del entorno y sin ninguna información previa o indicio sobre la posición del robot. No hay que confundir este problema con el reposicionamiento, dónde se conoce la posición inicial del robot (generalmente por odometría) y el problema consiste en mantener acotado el error en su estimación.

En los mapas geométricos se determina la posición más probable, correlacionando el mapa del entorno local del robot que éste va construyendo durante sus desplazamientos con el mapa global del entorno previamente aprendido o almacenado. Los más comunes son los métodos de tipo probabilístico (Burgard y col., 1996; Thrun, 1997), aunque también existen métodos basados en la teoría de la evidencia (Duckett y Nehmzow, 1998) o en procesos de Markov (Fox y col., 1998; Burgard y col., 1998).

Los métodos topológicos se basan en comparar las marcas detectadas en el entorno con las previamente almacenadas. Generalmente se centran en las características geométricas de las marcas, lo que simplifica la comparación. Para hacer la detección de las marcas más simple, rápida y robusta la mayoría de los autores utilizan características fácilmente identificables (p.e., color, gradiente de luz, textura, geometría, etc.).

Cuando las marcas son indistinguibles entre sí se recurre a comparar las relaciones que existen entre las marcas (por ejemplo, la posición relativa). De este modo, las técnicas de localización más utilizadas se basan en determinar la transformación del mapa local al mapa global que mantenga el mayor número de relaciones entre las marcas detectadas (Castellanos y Tardós, 1999; Wijk y Christensen, 2000). La principal ventaja de estas técnicas es que son independientes de la posición inicial del robot. Por otra parte, el principal problema radica en que el número de posibles transformaciones crece exponencialmente con el número de marcas locales detectadas.

Para evitar este y otros inconvenientes, se han introduciendo técnicas basadas en la lógica borrosa que son capaces, por un lado, de representar y procesar la incertidumbre y, por otro, de ajustar sucesivamente la posición del robot según se van detectando nuevas marcas locales (Saffiotti y Wesley, 1996; Saffiotti, 1996; Gasós y Martín, 1996).

5.4.5 Discusión

El método implementado se basa exclusivamente en la posición del robot, por lo que no constituye realmente un sistema de localización global. Se trata más bien de una primera aproximación que resuelve el problema más acuciante de calcular en qué lugar se encuentra el robot a la espera de desarrollar algoritmos que se basen principalmente en las marcas detectadas.

A pesar de todo, el sistema implementado soluciona nuestras necesidades más inmediatas, es fácil de implementar, eficiente y se puede adaptar a cada entorno. El método se puede hacer más robusto si en vez de usar sólo la última posición también se consideran los datos más recientes de la trayectoria del robot. De este modo se pueden identificar y evitar los fallos más comunes y catastróficos, por ejemplo, el de intentar salir de una habitación sin cruzar antes una puerta.

5.5 Agente Planificador_Rutas

El agente Planificador_Rutas utiliza el mapa global del entorno (MGD) para trazar una ruta que pueda seguir el robot para llegar al destino desde el origen especificado. La ruta calculada (figura 5.15) es una mera secuencia de lugares del entorno (es decir, de nodos) que minimiza una función de coste previamente definida y que puede aglutinar diferentes variables (tiempo de ejecución, distancia recorrida, energía consumida, velocidad media, etc.).

De una ruta se pueden extraer directamente todos los parámetros de interés para la ejecución de un desplazamiento: distancia total que debe recorrer el robot, el número de puertas e intersecciones que debe cruzar, las puertas y corredores que deja a cada lado, etc.

5.5.1 Entradas y salidas

Órdenes de tarea

- **Planificar Ruta Lugar** <*destino*> **[Lugar** <*origen*>]. Si no se especifica el origen, se supone que es el Lugar almacenado en la pizarra de Navegador.

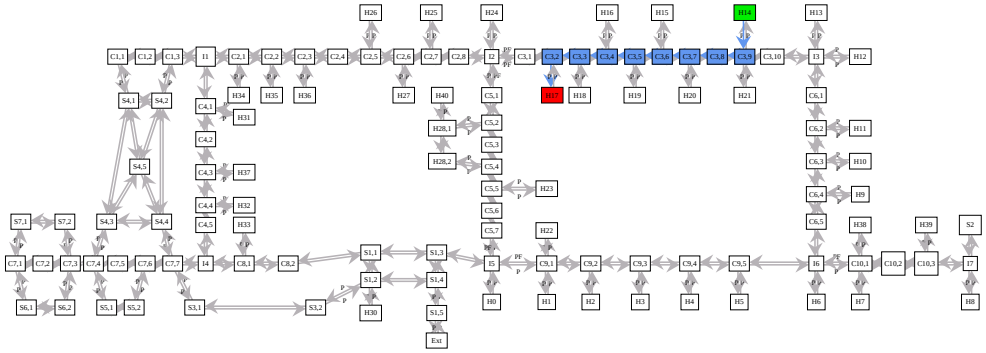

Figura 5.15: Ejemplo de una ruta en un mapa topológico.

Datos de entrada

- Mapa Global Distribuido (MGD) (sobre todo mapa topológico)
- Lugar actual del robot (si no se especifica el origen de la ruta)

Datos de salida

- Ruta

5.5.2 Síntesis del agente

Para determinar la mejor ruta entre dos nodos del mapa topológico es imprescindible establecer una función de coste asociada a las conexiones entre los diferentes nodos. En nuestro caso hemos usado el tiempo de ejecución. Si la conexión entre dos nodos representa espacio libre entonces el tiempo se considera proporcional a la distancia que separa ambos nodos,[7] ya que suponemos que la velocidad del robot en esos casos es aproximadamente constante. Si la conexión es de tipo puerta entonces su peso asociado es una constante (concretamente, 1000) indicando que el tiempo para cruzarla es casi siempre el mismo y relativamente elevado.

Con este esquema es muy sencillo adaptar el mapa topológico a las circunstancias reales de un entorno cambiante. Así, por ejemplo, si una puerta está cerrada (o es muy probable que lo esté) se puede subir su coste hasta un valor prácticamente

[7] La distancia que existe entre dos nodos vecinos es aquella que separa sus puntos centrales.

infinito, lo que representa una ruta impracticable. Lo mismo se puede hacer si el paso entre dos nodos (p.e., de un corredor) aparece cortado. Por otra parte, el peso de cada conexión podría tener en cuenta otros factores: el estado del suelo, la densidad de obstáculos, etc. También se podrían considerar aspectos temporales de dichos factores (p.e., a la salida de una clase aumenta la densidad de obstáculos).

Para crear y manipular los grafos hemos utilizado una librería de *software* libre para investigación denominada LEDA (*Library of Efficient Data types and Algorithms*) (Mehlhorn y col., 1999) en su versión 4.0. Esta librería define los tipos y algoritmos más comunes utilizados en teoría de grafos y está pensada para que su uso sea sencillo e intuitivo. Además, incluye una sencilla interfaz gráfica que facilita el proceso de creación, modificación y visualización de los grafos. De los diferentes algoritmos que se incluyen en LEDA para encontrar la mejor ruta entre dos nodos hemos seleccionado el de Dijkstra.

5.5.3 Evaluación

La eficiencia del agente PLANIFICADOR_RUTAS depende principalmente de la función de coste definida. En el caso de las puertas el coste depende del tiempo que tarda el robot en cruzar una puerta y también de la seguridad y fiabilidad con que lo hace. El resultado es que se eligen preferentemente las rutas con menos puertas. En la figura 5.16 se puede observar cómo afecta el peso asociado a las puertas en el cálculo de una ruta.

En el caso de una conexión que representa espacio libre el coste es proporcional a la distancia que separa los lugares asociados a ambos nodos de la conexión. En la práctica, como el objetivo es discriminar entre rutas claramente diferentes en longitud, más que el valor exacto interesa mantener los valores relativos entre las distintas distancias. De todas formas, en algunos casos una pequeña alteración en el origen o el destino de la ruta puede provocar un cambio drástico en la solución final (figura 5.17).

5.5.4 Soluciones existentes

La mayoría de los planificadores de rutas en mapas topológicos utilizan algoritmos basados en grafos similares al que hemos implementado (p.e. (Andersson y col., 1999)). Sin embargo, algunos trabajos intentan ir más allá. Así, en (Koenig y col., 1996) se describe un planificador de rutas basado en el algoritmo de búsqueda A^*,

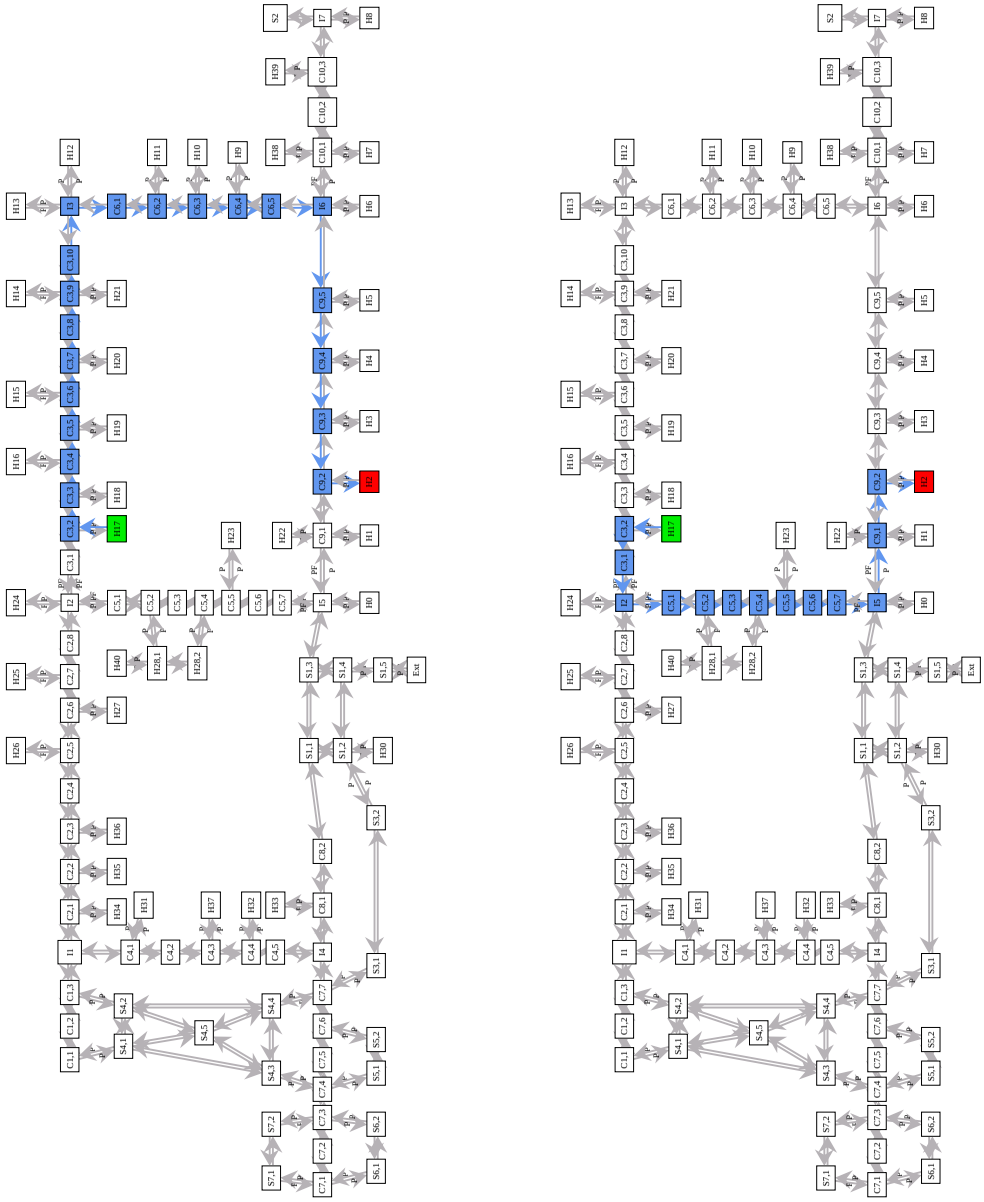

Figura 5.16: Cálculo de una ruta utilizando diferentes costes en las conexiones que representan puertas: a) 1000 y b) 500.

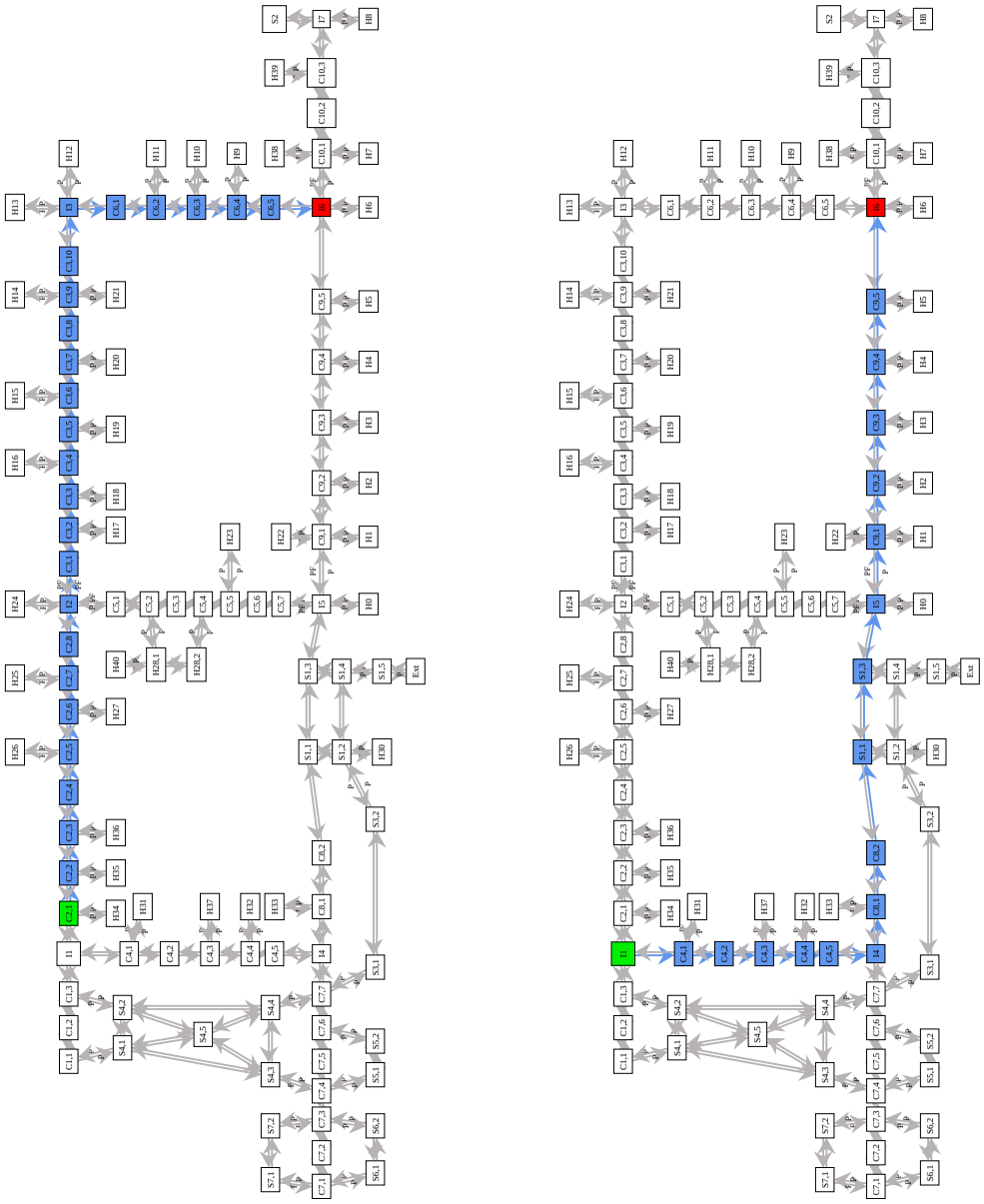

Figura 5.17: Alteración de una ruta al cambiar ligeramente su origen.

pero extendido para tener en cuenta información probabilista y cuyo objetivo es encontrar el camino con menor tiempo de ejecución esperado.

Para estimar este tiempo simulan cada una de las rutas posibles considerando la probabilidad de que ocurran ciertas incidencias: encontrar un corredor bloqueado o una puerta cerrada, tomar un corredor equivocado en una intersección, etc. En estos casos, también consideran el tiempo necesario para ejecutar el plan de contingencia oportuno, dando lugar a un proceso recursivo. Para obtener un buen rendimiento han implementado diversas estrategias de meta-control heurísticas para limitar el número de planes a examinar en cada situación pero, aún así, el tiempo de ejecución crece exponencialmente con el número de puertas en el entorno.

5.5.5 Discusión

Una limitación importante del planificador implementado es que no tiene en cuenta las probabilidades de que las puertas y los corredores puedan estar total o parcialmente bloqueados. Sin embargo, como el robot debe operar en un entorno imprevisible, la capacidad de predecir con fiabilidad estas probabilidades no es muy alta. Por lo tanto, creemos que es mejor prestar más atención en reflejar fielmente en el mapa topológico los cambios que se detectan en el entorno, que tratar de predecir el comportamiento futuro de dicho entorno.

5.6 Especialista PILOTO

El especialista PILOTO es responsable de mover el robot en su entorno local según las órdenes que recibe del módulo de control de NAVEGADOR y de procesar los datos sensoriales para extraer y contrastar la información de interés en el entorno.

5.6.1 Entradas y salidas

Órdenes de tarea

Las *OTs* que recibe PILOTO tienen la siguiente estructura:

ejecutar OT-<*identificador*> <*tarea*> [**Hasta** *condición1 [condición2]* ...]

Las tareas que PILOTO es capaz de ejecutar son:

- **Mover A Posición** <*x*> <*y*>. La posición está expresada en el sistema

odométrico de coordenadas del robot.

- **Mover A Ángulo** <*base*> <*torreta*> [**Relativo**]. Gira la base y la torreta hasta los ángulos indicados en el sistema odométrico del robot (entre -180 y 180 grados). **Relativo** indica que son incrementos.

- **Girar** <*vueltas*> <*lado*>. Sirve para girar la torreta hacia *Derecha* o *Izquierda* durante un determinado número (fracción) de vueltas.[8]

- **Cruzar Puerta** [**Pose** <x> <y> <θ> | **Id** <*identificador*> | **Cercana** | **Última**]. Se utiliza para que el robot cruce una puerta. Si no se especifica ninguna se entiende que es la última detectada.

- **Mover A Recto**. Simplemente para mover el robot en línea recta.

- **Mover Dirección** <*ángulo*>. Para mover el robot siguiendo la orientación marcada por *ángulo,* expresada en grados (entre -180 y 180).

- **Seguir Contorno** <*lado*>. Con ella el robot sigue el contorno de su *Derecha* o *Izquierda*, p.e. una pared.

- **Parar Robot**. Para el robot inmediatamente.

Todas las *OTs* pueden fijar una o varias condiciones de finalización o de caducidad. Las primeras *OTs* ya incluyen las condiciones de finalización: una puerta, una posición, unos ángulos o un número de vueltas. Por el contrario, las *OTs* **Mover A Recto**, **Mover Dirección** y **Seguir Contorno** son tareas continuas que necesitan condiciones para finalizar. Si se fija más de una condición de finalización la tarea acaba en cuanto se cumpla una de ellas. Se han definido las siguientes:

- **Distancia** <*centímetros*>, se ejecuta la tarea hasta que el robot recorre la distancia indicada (centímetros).

- **Puerta** <*número*> <*lado*>, se ejecuta la tarea hasta detectar el número de puertas indicado en el lado apropiado del robot: *Derecha*, *Izquierda*, *Atrás Derecha*, *Atrás Izquierda* o *Cualquiera.*

- **Puerta Id** <*identificador*>, la tarea se realiza hasta que se detecta la puerta indicada.

[8] **Girar** 1,1 <*lado*> **Hasta Puerta** *1000 Cualquiera* es equivalente a hacer un barrido completo para detectar todas las puertas cercanas.

Las condiciones de caducidad siempre están activadas para evitar que una tarea se ejecute de forma indefinida. Se han definido las siguientes:

- **Recorrido** <*centímetros*>. Máxima distancia que puede recorrer el robot. Por defecto, 30 metros.
- **Iteraciones** <*numero*>. Máximo número de iteraciones del ciclo de control, es decir, de datos básicos procesados. Por defecto, 3000.
- **Tiempo** <*segundos*>. Tiempo máximo para ejecutar la tarea. Por defecto, 15 minutos.
- **Intentos** <*número*>. Máximo número de intentos en una tarea con fases. Por defecto, 4.

Datos de entrada

- Mapa Perceptual Local Previo (MPLP)
- Pose

Datos de salida

- Mapa Perceptual Local (MPL)
- Pose

5.6.2 Pizarra

La pizarra del especialista Piloto está formada por la información local y reciente de los sensores del robot y por los datos que generan los agentes de Piloto. En la figura 5.1 se pueden observar las dependencias que existen entre ellos.

- datos sensoriales:
  - *Básicos:* Odometría, Ultrasonidos, Infrarrojos, Táctiles y Velocidades,
  - Láser,
  - Voltajes de las baterías tanto de la CPU como de los motores.

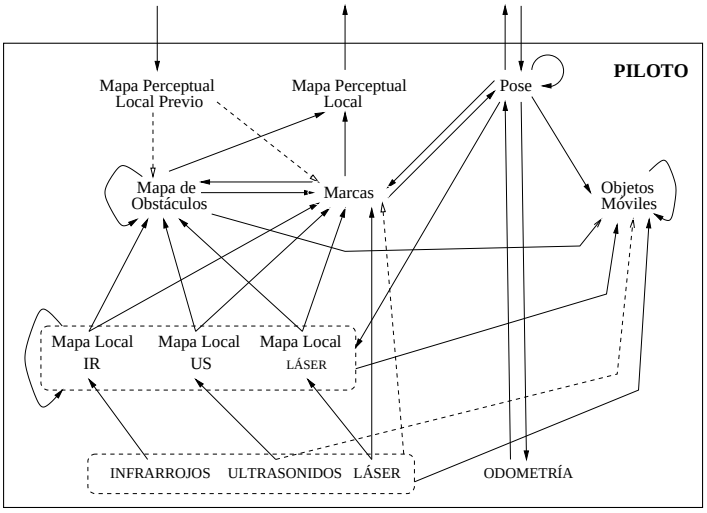

Figura 5.18: Estructura de los datos en la pizarra de PILOTO junto con los datos que se envían y reciben del especialista NAVEGADOR.

- MAPA PERCEPTUAL LOCAL (MPL), el mapa percibido de la región que el robot acaba de transitar. Incluye MAPA DE OBSTÁCULOS y MARCAS.
- POSE, posición del robot calculada según la posición odométrica integrada por el robot, las MARCAS actuales y las del MPLP.
- MAPA PERCEPTUAL LOCAL PREVIO (MPLP), el mapa de la región que el robot se dispone a cruzar con toda la información conocida por NAVEGADOR. Incluye MAPA DE OBSTÁCULOS y MARCAS.
- Los comandos generados por cada uno de los agentes.

Cada sensor posee su propia frecuencia de adquisición. Los datos de VOLTAJES se toman cada 2 minutos, puesto que son medidas que no cambian bruscamente. La información del sensor láser se adquiere dos veces por segundo, tiempo suficiente para su procesamiento y el aconsejable para evitar colapsar la percepción con información repetitiva. Los datos sensoriales *básicos* se recogen a una frecuencia de 5 Hz, también un compromiso entre el coste computacional y la capacidad para mantener una descripción actualizada del entorno: a una velocidad típica (entre 20 y 40 cm/s) el robot se desplaza entre 4 y 8 cm entre dos adquisiciones sucesivas.

Eventos

- CONTACTO (o choque),

- OBSTÁCULO MUY PRÓXIMO,
- NIVEL DE BATERÍAS BAJO *(externo)*,
- NIVEL DE BATERÍAS MUY BAJO *(externo)*.

5.6.3 Agentes

Existen cinco tipos de agentes en el especialista PILOTO: sensores, efectores, reactivos, voluntarios y perceptuales. Los AGENTES SENSORES se encargan de volcar en la pizarra la información que proporcionan los distintos sensores del robot. Los AGENTES EFECTORES integran los comandos generados por los distintos agentes activos (puede haber más de uno) y envían el comando final a cada actuador del robot. En el *Nomad 200* existen tres grados de libertad, pero dos son dependientes entre sí. En total existen dos agentes efectores:

- El agente EFECTOR BASE controla la velocidad de avance y de giro de la base.
- El agente EFECTOR TORRETA controla el giro de la torreta.

Los agentes *voluntarios* ejecutan las tareas ordenadas a PILOTO, principalmente desplazamientos. Para moverse en el interior de edificios se requiere un mínimo de dos comportamientos[9] mutuamente excluyentes entre sí:

- El agente MOVER_A mueve el robot hacia un punto cercano utilizando la información odométrica. También hace que el robot gire, avance recto o se desplace manteniendo una determinada orientación.
- El agente SEGUIR_CONTORNO sigue un contorno a una distancia de referencia.

Los agentes *reactivos* mantienen la integridad física tanto del robot como del entorno. Se necesitan dos como mínimo:

- El agente ELUDIR_CONTACTO para eliminar un contacto una vez que éste ya se ha producido.

[9]Es necesario un tercer comportamiento para que el robot cruce angosturas (p.e., puertas), pero, en la práctica, se puede implementar como tres tareas MOVER_A encadenadas y regulando apropiadamente los comportamientos reactivos.

- El agente MANTENER_DISTANCIA para mantener una distancia de seguridad mínima entre el robot y los diferentes objetos que lo rodean.

Los agentes *perceptuales* procesan e integran todos los datos que generan los sensores a lo largo del tiempo y según se va moviendo el robot, tratando en todo momento de mantener la coherencia entre la información detectada y la ya almacenada. Hay implementados tres de estos agentes:

- El agente CREADOR_MAPA_LOCAL confecciona y mantiene un mapa a corto plazo con los obstáculos que rodean al robot.
- El agente DETECTOR_MARCAS percibe los elementos de interés en el entorno y que son fundamentales para reposicionar el robot.
- El agente POSICIONADOR_ROBOT actualiza constantemente la posición real del robot a partir de la posición odométrica calculada por el propio robot, las marcas detectadas y las previamente almacenadas.

En AFE-Robótica también estaban definidos los agentes DETECTOR_MÓVILES, para detectar los objetos en movimiento cerca del robot, y EVITAR_MÓVILES, para evitarlos. Ambos agentes se han implementado (Mucientes y col., 2001a,b, 2002a) pero finalmente no se han integrado porque con los sensores del *Nomad 200* sólo son efectivos en espacios relativamente amplios (vestíbulos y salas). Para solucionar esta limitación se estudia incorporar un sensor láser más potente.

5.6.4 Síntesis de planes y control

Plan de control básico

Las *OTs* más sencillas del especialista PILOTO se ejecutan directamente (p.e. **Nueva Pose, Parar Robot**, etc.), el resto se ejecutan siguiendo un plan, o *ciclo de control*, similar al comentado en el capítulo anterior (figura 5.19) y que, con ligeras modificaciones, es válido para todas.

El primer paso del plan de control (nodo *inicio*) consiste en inicializar todas las variables internas y en establecer los esquemas de reactividad asociados a la tarea. El principal de ellos es fijar la distancia de seguridad entre el robot y el resto de objetos del entorno y que es distinta según se quiera cruzar una puerta, seguir un contorno o que no haya ninguna tarea asignada. A continuación, las primeras *OTs*

Agente	Entradas	Salidas	Función
SENSORES	*sensores robot*	datos sensores	almacenar en pizarra
EFECTORES	comandos	*efectores robot*	enviar comandos finales
ELUDIR CONTACTO	ODOMETRÍA, IR, US, TÁCTILES	comando	mover el robot para eliminar los contactos
MANTENER DISTANCIA	VELOCIDAD, US ODOMETRÍA, IR	comando	mantener el robot lejos de los obstáculos
MOVER_A	ODOMETRÍA, VELOCIDAD, MAPA OBSTÁCULOS	comando	mover robot un cierto ángulo hacia un punto cercano o según una dirección
SEGUIR CONTORNO	ODOMETRÍA, VELOCIDAD, US	comando	mover el robot siguiendo un contorno
CREADOR MAPA LOCAL	ODOMETRÍA, IR, US, LÁSER, MAPA OBSTÁCULOS	comando MAPA DE OBSTÁCULOS	crear un mapa del entorno local del robot
DETECTOR MARCAS	ODOMETRÍA, US, LÁSER, MARCAS, MAPA OBSTÁCULOS	comando MARCAS	detectar las marcas de interés en el entorno
POSICIONADOR ROBOT	ODOMETRÍA, POSE MARCAS, MPLP	comando POSE	recalcular posición del robot según las marcas

Tabla 5.2: Función de los agentes del especialista PILOTO y los datos de la pizarra que necesita cada uno.

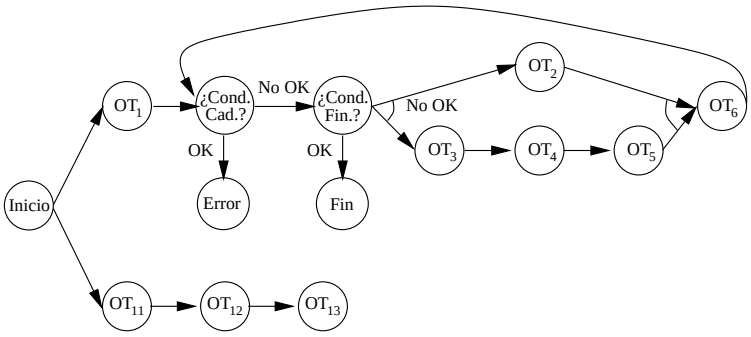

Figura 5.19: Grafo con el plan básico que utiliza PILOTO para ejecutar sus *OTs*.

(OT_1 y OT_{11}) configuran los agentes SENSORES y fijan la frecuencia de adquisición de los datos sensoriales. A partir de ese momento, cada vez que un agente SENSOR almacena un dato, el control de PILOTO recibe la correspondiente notificación y, según el tipo de dato, sigue una parte diferente del ciclo de control (figura 5.19).

Cada vez que llegan nuevos datos básicos, primero se comprueban las condiciones de caducidad (parte superior de la figura 5.19). Si se cumple alguna se aborta la ejecución de la tarea y se manda una ROT al control de NAVEGADOR, indicando la causa del error (es decir, la condición de caducidad). En caso contrario se comprueban las condiciones de finalización que se incluían en la OT. Si se cumple alguna significa que la tarea ha finalizado con éxito y así se hace saber al control de NAVEGADOR indicando también qué condiciones se han cumplido.

Para procesar los datos básicos se envían dos OTs simultáneamente: una al agente CREADOR_MAPA_LOCAL para que actualice el mapa local (OT_2) y otra al agente DETECTOR_MARCAS para que detecte nuevas marcas sobre los datos básicos (OT_3). Cuando finaliza esta última se envía al agente POSICIONADOR_ROBOT la orden de que actualice la POSE del robot (OT_4) y, cuando ésta acaba, se ordena al agente apropiado que ejecute el comportamiento voluntario necesario según la tarea en ejecución (OT_5). Así, si se desea seguir contorno se envía la OT al agente SEGUIR_CONTORNO, mientras que si se quiere girar o mover el robot la OT se envía al agente MOVER_A. Si no hay ninguna tarea activa no se ejecuta ningún comportamiento voluntario.

En cuanto finalizan tanto OT_2 como OT_5 el control ordena a los agentes EFECTORES que seleccionen y envíen al robot el comando a ejecutar en cada grupo de actuadores (OT_6). El ciclo se repite continuamente hasta que se verifique una de las condiciones asociadas al plan. En realidad, no es imprescindible esperar a que se actualice el mapa local (OT_2), ya que su comando de movimiento no es vital para el robot.

El cálculo del comando que se debe enviar al robot después de cada ciclo de control es una tarea del agente de control, aunque, como suele ser un proceso mecánico, éste la delega en los agentes EFECTORES. En su momento estudiaremos el proceso con detalle, aquí sólo comentaremos que se basa en seleccionar el comando de mayor prioridad y que la prioridad depende del agente que lo ha generado y también de las circunstancias del entorno y del sistema.

El tratamiento de los datos del láser es ligeramente diferente (parte inferior de la figura 5.19). En primer lugar, se indica al agente DETECTOR_MARCAS que procese

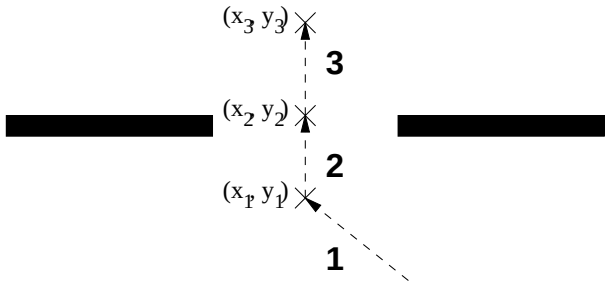

Figura 5.20: Fases para cruzar una puerta: (1) acercarse, (2) cruzar el umbral y (3) separarse.

los datos para percibir las marcas y mantener la torreta alineada con un contorno del entorno. Finalizada esta primera orden se indica al agente POSICIONADOR_ROBOT que calcule el error odométrico comparando las marcas detectadas con las que ya se conocen del entorno. Si el error es excesivo el control ordena corregir la odometría del robot. No se envía ningún comando al robot porque ya se integran en la parte del ciclo de control de los datos básicos.

Planes de control con fases

Una particularidad importante de los planes que utiliza el especialista PILOTO es que se pueden definir distintas *fases* durante su ejecución. Las fases son imprescindibles para realizar una tarea que consta de varias etapas que se deben ejecutar de manera secuencial y con diferentes comportamientos o parámetros (p.e., cruzar una puerta o seguir un contorno). Para pasar de una fase a otra se definen condiciones específicas para finalizar las fases. También se incluyen condiciones de caducidad que, en caso de cumplirse, abortan la ejecución de toda la tarea.

Cruzar una puerta. El *Nomad 200* no puede cruzar una puerta utilizando exclusivamente información sensorial porque con los sensores que posee no puede detectarla mientras la cruza; los US poseen una elevada incertidumbre angular, mientras que el láser tiene una abertura reducida y un umbral mínimo de detección muy alto (unos 50 cm). Para resolver este problema hemos optado por controlar el robot en base a la información odométrica, fiable en pequeñas distancias, y por dividir la tarea en tres fases: acercamiento, traspaso del umbral de la puerta y alejamiento (figura 5.20). La primera y tercera fases utilizan el modo estándar de evitar los objetos cercanos, mientras que la segunda fase utiliza un modo especial que sólo

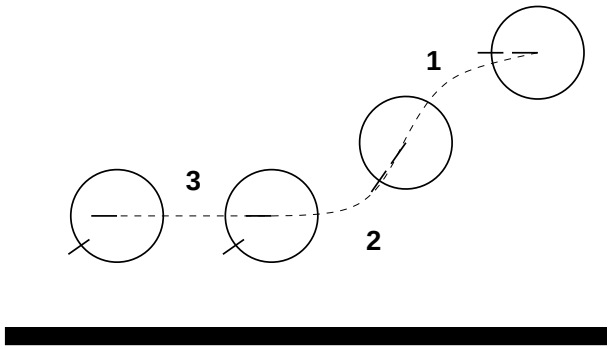

Figura 5.21: Fases para seguir un contorno: (1) buscar y alinear la base al contorno, (2) orientar la torreta al contorno y (3) seguir el contorno. El círculo representa el robot, la línea interior la orientación de la base y la línea externa la orientación de la torreta.

evita el objeto más cercano. En la segunda fase también se disminuye la distancia de seguridad para que el robot pueda pasar por el hueco de la puerta.

Todas las fases se ejecutan con el comportamiento voluntario **Mover A Posición**, donde la posición objetivo se fija en función de la posición y orientación de la puerta. La aproximación tiene como destino el umbral de la puerta (p.e., un metro antes y en la perpendicular de la puerta, posición (x_1, y_1) en la figura 5.20). Para pasar de la segunda fase a la tercera, el robot debe alcanzar el centro de la puerta (posición (x_2, y_2) en la figura 5.20) y cruzar el umbral (es decir, cambiar de lado respecto a la puerta). El objetivo de la tercera fase se sitúa después de la puerta (posición (x_3, y_3) en la figura 5.20). Si durante la segunda fase el robot se aleja excesivamente de la puerta se vuelve a la primera fase y se incrementa en uno el número de intentos.

Seguir un contorno. Como el comportamiento de seguir contorno evita tropezar con el contorno que está siguiendo, el primer paso de la tarea consiste en bajar la distancia de seguridad para evitar indeseables interferencias entre el comportamiento reactivo de evitar obstáculos. La ejecución de tarea se realiza en tres fases (figura 5.21): alineamiento de la base al contorno a seguir (derecha o izquierda), alineamiento de la torreta al contorno en dónde se deben detectar las marcas (puede coincidir o no con el contorno que sigue el robot) y seguir contorno. En todas las fases se ejecuta el mismo ciclo de control, aunque con *OTs* ligeramente diferentes.

En la primera fase, como comportamiento voluntario se ordena al agente SE-

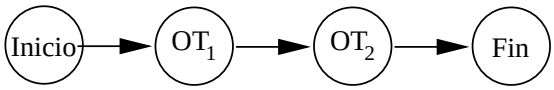

Figura 5.22: Secuencia de acciones con las que PILOTO responde a un evento.

GUIR_CONTORNO que detecte y se alinee con el contorno pedido. Se pasa a la siguiente fase cuando se detecta un contorno durante varias iteraciones consecutivas (más de cuatro). Si no es así y se ha recorrido una distancia excesiva (más de 1,5 m) o se han ejecutado demasiadas iteraciones del ciclo de control (más de 80) entonces se aborta la tarea.

La segunda fase sólo se aplica cuando se necesitan detectar marcas con los datos del láser y consiste en ordenar al agente DETECTOR_MARCAS que alinee apropiadamente la torreta. Si el robot recorre demasiada distancia (más de 2,5 m) o pasan demasiadas iteraciones antes de conseguirlo (más de 110), se aborta la tarea. Para ganar tiempo la torreta se puede empezar a alinear en la primera fase pero, como es previsible que haya cierta dependencia con el contorno sobre el que se alinea el robot, se mantiene más tiempo.

Reactividad

Cuando el agente de control recibe uno de los eventos generados en la pizarra responde con una secuencia de acciones que, aunque dependen del tipo de evento, siguen un esquema muy similar al mostrado en la figura 5.22: enviar una OT a un agente de comportamiento reactivo (OT_1) y ordenar a los agentes EFECTORES (OT_2) que envíen el comando generado al robot.

Evento CONTACTO. Si hay más de un contacto y la máxima distancia angular entre dos de ellos es menor que 180^o entonces el robot no puede zafarse. En ese caso, el agente de control aborta inmediatamente la tarea en ejecución e indica el problema al control de NAVEGADOR. En el resto de casos se considera que el robot puede escapar y se activa el agente reactivo ELUDIR_CONTACTO para calcular el movimiento que separe el robot del obstáculo con el que ha colisionado.

Evento OBSTÁCULO MUY PRÓXIMO. El control de PILOTO modula la respuesta a este evento en función de la tarea que está ejecutando. Si el robot está en la fase de cruzar el umbral de una puerta ordena al agente MANTENER_DISTANCIA

que atienda únicamente el obstáculo más cercano al robot, de lo contrario le ordena que considere todo su entorno próximo. Otro ejemplo es cuando se ignora el evento porque la posición objetivo que debe alcanzar el robot está más cerca que cualquier obstáculo.

5.6.5 Evaluación

En esta sección veremos cómo PILOTO ejecuta algunas de sus tareas más importantes.

Ejemplo girar torreta

De la orden de tarea se recoge hacia qué lado se gira la torreta, cuántas vueltas y fracciones se dan, cuántas puertas se tienen que detectar e, incluso, a qué velocidad se gira la torreta. El plan para ejecutar esta tarea no modifica los esquemas de reactividad que existen por defecto. La primera OT se envía a los agentes SENSORES para fijar la frecuencia de adquisición de los datos básicos y del láser.

Los datos básicos sirven para conocer la posición de la torreta y así chequear si se ha girado el número de vueltas requerido. Si no es así, se ordena al agente MOVER_A que siga girando la torreta, normalmente a velocidad constante. Por otra parte, cada vez que se reciben nuevos datos del láser, simplemente se ordena al agente DETECTOR_MARCAS que los procese para detectar posibles puertas. El agente de control va contabilizando las veces que se detecta una nueva puerta. Cuando se detecten las especificadas el plan finaliza.

Ejemplo cruzar puerta

En las figuras 5.23 y 5.24 se muestra cómo el robot cruza las puertas. De todos modos, a pesar de su gran eficacia la tarea de cruzar puertas no está exenta de errores. Los más típicos son: dar la vuelta cuando se está a punto de cruzar la puerta (efecto "rebote") o chocar contra uno de los marcos.

El problema del *rebote* es el más común y suele estar provocado por un error en el cálculo de la posición y orientación de la puerta. Si se percibe más cerca del robot de lo que realmente está, entonces PILOTO cree que pasa por su centro antes de tiempo y cambia a la última fase, donde al evitar obstáculos de forma más agresiva es muy improbable que la cruce. Si, por el contrario, la puerta se percibe más lejos

Figura 5.23: Cruce de una puerta: a) datos del láser y b) datos de US.

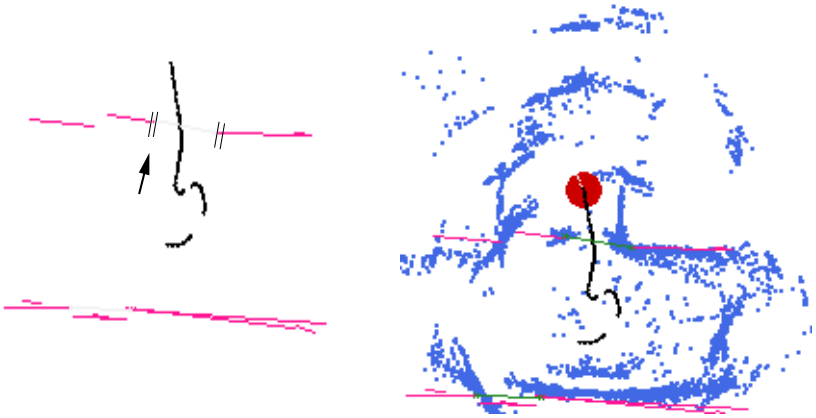

Figura 5.24: Cruce de una puerta con un marco estrecho: a) datos del láser y b) datos de US.

de lo que está en realidad, entonces es muy posible que el robot no se acerque lo suficiente para cambiar a la segunda fase. En ambos casos, lo más probable es que posteriores intentos tengan el mismo resultado.

Otra causa de rebote es que el robot crea no encontrar un hueco suficientemente grande para pasar cuando está en el umbral, incluso girando la torreta para reducir los errores propios de los US. Lo usual es que el robot acabe girando, se aleje de la puerta y lo vuelva a intentar. El robot también puede "rebotar" si no se acerca al umbral en una dirección casi perpendicular.

Los choques prácticamente sólo ocurren cuando los marcos son demasiado finos (p.e., cuando en una puerta doble sólo una hoja está abierta) y los US no los detectan correctamente. Muchos de los errores en la detección con los US se deben a que las puertas suelen ser lisas y de madera. La solución a un choque consiste en retroceder

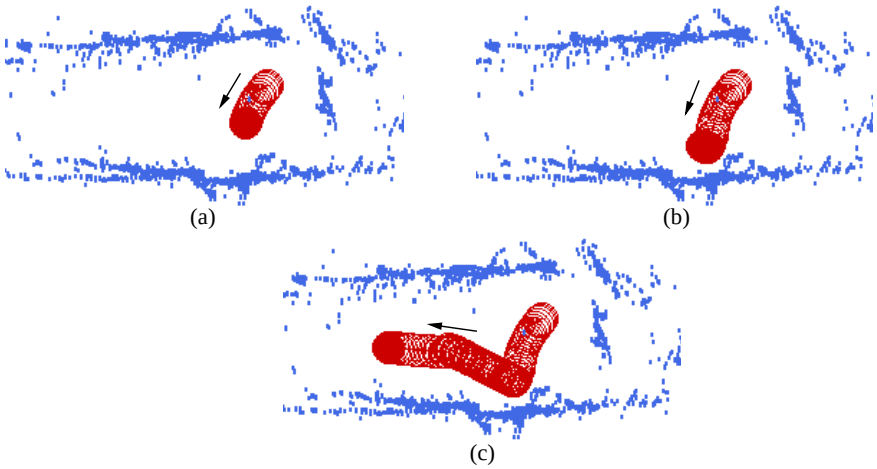

Figura 5.25: Ejecución de la tarea de seguir un contorno: a) alinear la base al contorno, b) alinear la torreta y c) seguir el contorno.

y esperar a tener mejor "suerte" en el siguiente intento.

Ejemplo seguir contorno

En la figura 5.25 se muestra un ejemplo real de cómo se sigue un contorno. El problema más usual para ejecutar esta tarea es que el robot no sea capaz de alinearse con el contorno (bien la base para seguirlo, bien la torreta para detectar marcas). Las causas pueden ser varias: no existe el contorno (lo que implica un fallo en el mapa del entorno o en la localización del robot), el robot está tan desalineado que no es posible realizar el giro necesario para alinearse, el contorno puede tener un hueco justo donde el robot intenta alinearse o, simplemente, existe algún obstáculo que impide al láser tomar datos.[10]

Sin embargo, la eficacia de la tarea de seguir un contorno radica en cómo el agente SEGUIR_CONTORNO implemente dicho comportamiento. Una situación poco afortunada se produce cuando un obstáculo bloquea parcialmente el corredor. Si el robot considera que el hueco libre del pasillo es insuficiente, entonces lo integra en el contorno que está siguiendo y gira. El resultado final es que vuelve sobre sus pasos pero siguiendo el lado contrario del corredor. Esta situación se da más a menudo cuando el robot sigue el contorno de referencia demasiado cerca y el objeto obstaculiza gran parte del corredor o el corredor es estrecho (es decir, cuando queda

[10] Como el láser del Nomad 200 funciona por triangulación, cualquier objeto que se interponga entre el haz reflejado y la cámara impedirá la detección.

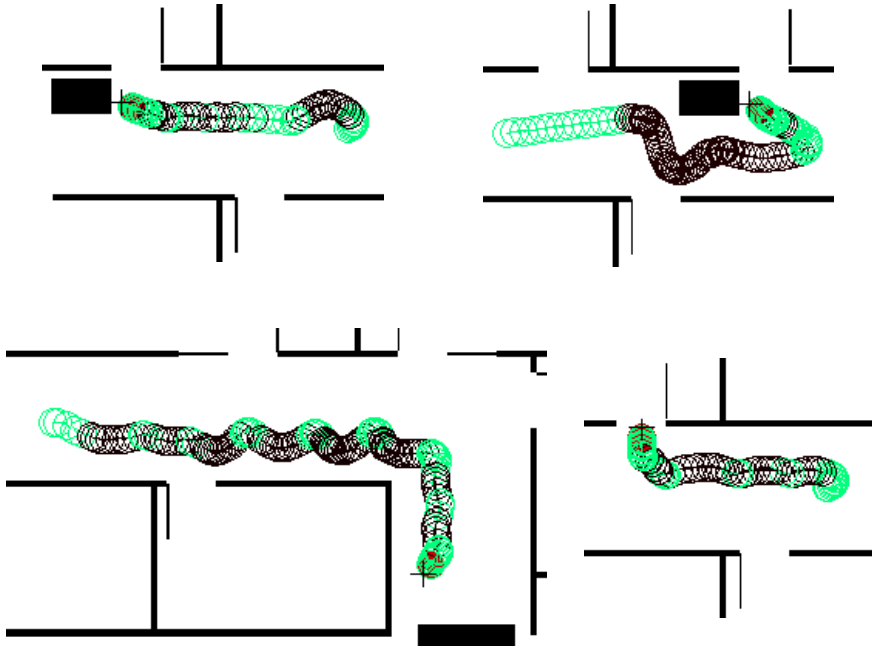

Figura 5.26: Integración de la reacción a eventos (círculos oscuros) durante la ejecución de una tarea (círculos claros). Las cruces representan el objetivo a alcanzar.

poco espacio libre).

Reactividad durante la ejecución de una tarea

Como ya se ha explicado, el agente de control de un especialista de AFE-Robótica es el encargado de integrar las reacciones ante los eventos con la ejecución del plan de control activo en ese momento. En PILOTO la integración se limita a ejecutar el plan asociado al evento (figura 5.22): activación de un comportamiento reactivo y envío del comando generado al robot.

El plan de la tarea original se sigue ejecutando en paralelo y los agentes siguen generando nuevos comandos en función de los nuevos datos recibidos. Sin embargo, como los comandos de los comportamientos reactivos tienen las prioridades más altas y como el arbitraje entre comportamientos se basa en seleccionar el comando de máxima prioridad, en la práctica, cuando en un ciclo de control se activa una reacción, ésta inhibe el resto de comportamientos. En la figura 5.26 se muestran varios ejemplos de la integración de la respuesta reactiva ante objetos cercanos y la ejecución de la tarea **Mover A Posición**.

5.6.6 Discusión

El especialista PILOTO es el responsable de que en última instancia el robot realice tareas útiles. Sin embargo, el repertorio de tareas implementadas no está cerrado y se pueden incorporar nuevas tareas o mejorar las ya existentes. Algunas de las nuevas tareas podrían ser especializaciones de las ya existentes. Por ejemplo, la tarea de seguir un corredor sería una especialización de seguir un contorno capaz de recorrer un pasillo de forma más eficiente.

Los planes de control descritos e implementados en el especialista PILOTO para ejecutar las distintas tareas están plenamente operativos y cumplen perfectamente los requisitos previstos. Sin embargo, siempre se pueden mejorar o completar. Así, el plan para cruzar una puerta sería posiblemente más robusto si se divide la segunda fase en dos: una de acercamiento al centro de la puerta y otra para rebasar el umbral, ambas manteniendo el método del mínimo para evitar obstáculos. De este modo, no sería tan importante una correcta estimación de la posición y la orientación de la puerta a cruzar. Otra posible mejora sería utilizar los datos sensoriales para detectar cuándo el robot cruza el umbral de la puerta, pudiendo pasar así a la última fase. Para esa detección se podrían usar las distancias medidas por los dos US perpendiculares a la dirección de avance del robot.

Sin duda el disponer de mejores sensores facilitaría sensiblemente la ejecución de algunas tareas. Así, un láser con mayor rango de detección que el del *Nomad 200* percibiría una porción mayor del entorno que rodea al robot y permitiría tomar mejores decisiones y con mayor antelación. Por ejemplo, si se pudiese detectar continuamente una puerta mientras el robot la cruza, entonces se podrían elegir mejor las acciones a tomar en cada momento. De la misma forma, mientras se sigue un contorno resultaría muy útil detectar lo antes posible las esquinas cerradas, así como discernir cuándo una discontinuidad en el contorno es un hueco (p.e., una puerta) o una esquina abierta. Nuevos sensores y agentes también permitirían detectar nuevas marcas y situaciones de interés en el entorno: objetos en movimiento, personas, intersecciones entre corredores, finales de los corredores, corredores bloqueados, obstáculos ocultos para los sensores actuales, etc.

5.7 Agentes SENSORES

El objetivo de los agentes SENSORES es recoger los datos de los sensores y volcarlos en la pizarra de PILOTO. Se habla de agentes SENSORES porque puede haber

uno por cada tipo de sensor y son fundamentales para hacer que la arquitectura de control sea independiente del *software* que equipa el robot.

5.7.1 Entradas y salidas

Órdenes de tarea

Como hay tres clases de datos sensoriales, existen otras tantas órdenes de tarea:

- **AdquirirDatos Básicos** <*frecuencia*>. *frecuencia* es un número real que indica la frecuencia de adquisición. El 0 significa que no se adquieren más datos.
- **AdquirirDatos Láser** <*frecuencia*>. *frecuencia* es un número real. El 0 se utiliza para no adquirir más datos.
- **AdquirirDatos Voltajes** <*segundos*>. El 0 indica, de nuevo, no tomar más datos.
- **Resetear Táctiles**. El *software* del robot no borra la activación de los sensores táctiles después de un contacto, por lo que hay que resetearlos para comprobar si se mantiene o no el contacto.

Datos de entrada

- Los datos que se recogen de los sensores del robot.

Datos de salida

Los datos sensoriales que se guardan en la pizarra de PILOTO (figura 5.18):

- TÁCTILES: detectan un choque del robot con una precisión de 18º.
- IR o INFRARROJOS: 16 sensores dispuestos cada 22,5º en un plano formando un anillo. Cada sensor mide una distancia entre 0 y 38 cm.
- US o ULTRASONIDOS: 16 sensores dispuestos cada 22,5º en un plano formando un anillo. Cada sensor mide distancias entre 15 cm y 6,5 m.

- ODOMETRÍA: la posición y los ángulos de la base y de la torreta calculados por el propio robot en base a los movimientos realizados.
- VELOCIDAD: 3 números enteros (lineal, angular de la base y de la torreta)
- LÁSER: incluye la posición odométrica del robot cuando se ha realizado la adquisición sensorial, el número de segmentos detectados y las posiciones inicial y final de cada uno de los segmentos.
- VOLTAJES: el voltaje de las baterías de la CPU y de los motores, que indica el nivel de carga. Con dichos valores se puede estimar la energía aún almacenada y, por tanto, el tiempo durante el cual el robot seguirá operativo.

5.7.2 Síntesis del agente

Los agentes SENSORES se implementan como un único proceso con tres hilos, donde cada uno se encarga de recoger los datos de cada tipo de sensor, de grabarlos en la pizarra de PILOTO y de avisar al agente de control de que se ha adquirido una nueva remesa de datos. Al acabar se vuelve a dormir y espera a que el sistema operativo vuelva a despertarlo para repetir de nuevo el proceso. En realidad es otro proceso el que controla el hardware de adquisición (típicamente microprocesadores dedicados: uno para los US, otro para los IR, etc.) y el que almacena los nuevos datos sensoriales a la espera de las peticiones de los agentes SENSORES.

La frecuencia de adquisición está limitada por restricciones de tipo hardware. Por ejemplo, entre el *disparo* de dos sensores de US del *Nomad 200* debe transcurrir un mínimo de 4 ms, con lo cual, para activar todos los US del anillo se necesita un mínimo de 64 ms. Es decir, la frecuencia máxima de adquisición de datos de US es de 15 Hz. Si se piden datos a una frecuencia mayor, no todas las medidas habrán sido actualizadas.

5.7.3 Evaluación

En la tabla 5.3 se muestran los tiempos medidos para la adquisición de los datos básicos, los otros tipos de datos se comportan de forma similar. El primer aspecto a destacar es que la frecuencia de adquisición real es muy próxima a la teórica. En segundo lugar, el tiempo para la recogida de los datos del robot es considerable y llega hasta los 100 ms. En la práctica esto reduce la frecuencia máxima a unos 8Hz.

Frecuencia	Adquisición datos			Recogida datos	
adquisición	teórico	medio	máximo	medio	máximo
1	1000	999	1633	58	691
2	500	503	1051	43	114
4	250	251	686	78	197
5	200	202	1109	82	1318
6	167	170	1078	86	243
8	125	143	780	98	732
10	100	121	842	89	367

Tabla 5.3: Tiempos (en ms) de adquisición y de recogida de los datos sensoriales básicos en función de la frecuencia.

Por último, los altos valores máximos delatan las fuertes desviaciones respecto al comportamiento medio. Existen dos causas principales para este comportamiento. La primera son los retrasos en el software de interfaz del robot que se vuelven más importantes cuantas más peticiones se realicen (alta frecuencia de adquisición). La segunda es que el sistema operativo del *Nomad 200* no es de tiempo real y sus operaciones no tienen un tiempo de ejecución máximo asegurado.

5.7.4 Soluciones existentes

En otros robots el proceso de adquisición de los datos sensoriales es diferente. Por ejemplo, en los B21 (B21, 1995) cada vez que un sensor adquiere un nuevo dato, el software del robot lo envía directamente al programa de control. De todos modos, adoptar este estilo en nuestra arquitectura es muy sencillo: cada vez que se adquiere un dato, el agente SENSOR correspondiente lo recoge y lo inserta en la pizarra.

5.7.5 Discusión

La frecuencia de adquisición de los datos sensoriales depende del sistema operativo, el encargado de *despertar* a los distintos agentes SENSORES. El problema viene de que el sistema operativo que incorpora el *Nomad 200* (*Linux*) no tiene características propiamente de tiempo real, por lo que no está garantizado que los agentes SENSORES se activen exactamente a la frecuencia pedida. El tiempo entre activación y activación depende de la carga de la CPU, del uso de la memoria y de otros factores difíciles de evaluar. Por otra parte, también hemos podido comprobar

que no se puede despreciar el tiempo que necesita el proceso que actúa de interfaz con el robot para recoger y enviar los datos. Este tiempo ronda los 50 ms, pero en ocasiones supera los 100 ms.

5.8 Agentes EFECTORES

La función de los agentes EFECTORES consiste en enviar a cada efector del robot el comando final que ha calculado el módulo de control del especialista PILOTO. Sin embargo, lo usual es que el control de PILOTO "delegue" en cada agente EFECTOR la integración de los comandos generados por los distintos agentes activos en cada ciclo de control.

Por último, señalar que la arquitectura no utiliza tantos agentes EFECTORES como actuadores posee el robot, ya que algunos actuadores se deben controlar conjuntamente. Esto es lo que sucede con el *Nomad 200*, donde sus tres actuadores se controlan con dos agentes EFECTORES: uno controla la base (con dos actuadores, uno para desplazar y otro para girar) y otro la torreta (con un sólo actuador para girarla).

5.8.1 Entradas y salidas

Órdenes de tarea

- **SeleccionarComandos**. Calcula el comando final y lo envía al robot.
- **PararRobot**. Para inmediatamente el robot.
- **AlinearÁngulos**. Alinea la base y la torreta con el valor de referencia e inicializa el sistema odométrico del robot.
- **Nueva Pose Robot** *<x> <y> <ángulo base> <ángulo torreta>*. Sirve para modificar la posición odométrica del robot.

Datos de entrada

- los comandos generados por todos los agentes en el último ciclo de control,
- el comando final de cada efector generado en el ciclo de control anterior,
- la nueva POSE generada.

Datos de salida

- Los comandos que se envían al robot.

5.8.2 Síntesis del agente

Los actuadores del *Nomad 200* se pueden controlar en velocidad, en desplazamiento o en aceleración. En AFE-Robótica el control se realiza en velocidades lineales y angulares de la base y angulares de la torreta, aunque se podría utilizar un modo diferente para cada actuador. El cálculo del comando final consiste en seleccionar de entre todos los comandos generados[11] en cada ciclo de control aquél que tenga mayor *prioridad* y que tenga una *urgencia* no nula.

La *prioridad* de cada comando posee una componente fija que indica la importancia relativa entre los distintos agentes y otra dinámica que calcula el agente de control según las condiciones del sistema. El orden de prioridad de los agentes es, en orden decreciente:

1. CREADOR_MAPA_LOCAL,
2. SEGUIR_CONTORNO,
3. MOVER_A,
4. DETECTOR_MARCAS,
5. MANTENER_DISTANCIA,
6. ELUDIR_CONTACTO.

La *urgencia* indica la importancia que asigna cada agente al comando que ha generado, obviamente, desde su perspectiva local. Por ejemplo, la urgencia del comando que genera MANTENER_DISTANCIA depende de la distancia del obstáculo más cercano. Este parámetro se puede utilizar para ponderar cada comando en el resultado final.

Por último, queda establecer qué comandos se deben integrar en cada momento, es decir, cuáles pertenecen al último ciclo de control, ya que los comandos no se generan a frecuencias fijas. La razón es que los datos se adquieren a distintas frecuencias, el tiempo entre dos adquisiciones consecutivas del mismo dato no siempre

[11] En teoría, pueden existir tantos comandos como agentes posee el especialista.

es el mismo, y a que cada tipo de dato se procesa con agentes diferentes. Además, no todos los agentes se activan en todos los ciclos y los agentes no siempre tardan el mismo tiempo en procesar sus datos.

Para resolver esta dificultad hemos asignado a cada comando (incluido el comando final) el tiempo de adquisición de los últimos datos sensoriales utilizados para su cálculo. De esta manera es muy sencillo saber qué comandos han sido generados a partir de una determinada adquisición sensorial y, en su caso, comprobar cuándo un comando ya está desfasado. Como el tiempo de adquisición es único para todo el sistema, se puede utilizar como referencia para establecer a qué ciclo de control pertenece cada comando almacenado en la pizarra. Un comando queda desfasado por dos motivos: cuando aparece otro comando del mismo agente basado en datos sensoriales más recientes o cuando tarda demasiado tiempo en calcularse.

5.8.3 Evaluación

El correcto funcionamiento de estos agentes se pone de manifiesto continuamente durante la ejecución de todas las tareas del especialista PILOTO. Los casos más interesantes son aquellos en donde se integran comandos de agentes que procesan datos sensoriales básicos con comandos de agentes que utilizan datos del láser (con distinta frecuencia de adquisición). También en aquellos casos en donde los agentes consumen demasiado tiempo en procesar los datos sensoriales para generar su comando de movimiento. Sin embargo, el robot funciona apropiadamente en todas estas situaciones.

5.8.4 Soluciones existentes

El funcionamiento de los agentes EFECTORES cuando AFE-Robótica controla un robot móvil como el *Nomad 200* es muy simple, ya que el control final de cada actuador corre a cargo de hardware específicamente dedicado (típicamente, microcontroladores empotrados). Esta situación es la más habitual en la mayoría de los robots comerciales actuales. Sin embargo, en robots más simples que carezcan del hardware oportuno, los agentes EFECTORES tendrían que encargarse del control a bajo nivel de cada actuador.

Al describir las arquitecturas de control reactivas ya se ha comentado que la integración de los comportamientos que se activan simultáneamente no es un problema sencillo y todavía no está satisfactoriamente resuelto. Entre las distintas

técnicas destacaríamos las basadas en lógica borrosa (Saffiotti y col., 1993, 1995; García-Alegre y col., 1995b,a; Konolige y Myers, 1998; Driankov y Saffiotti, 2001) porque producen transiciones menos bruscas entre comportamientos y, por lo tanto, un movimiento final del robot más suave y continuo.

5.8.5 Discusión

En esta primera implementación de AFE-Robótica nos hemos decantado por una técnica de arbitraje entre comportamientos de tipo selección por varias razones. En primer lugar, es más fácil y sencilla de implementar. En segundo lugar, es muy robusta y siempre proporciona comandos válidos y correctos en cualquier situación, por compleja que ésta sea. Por último, facilita la integración de comandos que se generan de forma asíncrona.

Sin embargo, la técnica de selección no está exenta de inconvenientes. En primer lugar, se producen cambios bruscos entre comportamientos al seleccionar comandos de diferentes agentes de un ciclo a otro. De todos modos, en la práctica los cambios no suelen ser demasiado bruscos, ya que las condiciones que obligan a ejecutar un comportamiento se mantienen de un ciclo a otro. En segundo lugar, no existe una verdadera *integración* de todos los comandos generados en un ciclo de control, por lo que se pierde cierta visión global del comportamiento del sistema. Este problema tampoco es demasiado grave, ya que dicha integración es competencia del agente de control de PILOTO y tiene lugar durante la ejecución de los planes de control.

5.9 Agente ELUDIR_CONTACTO

El agente ELUDIR_CONTACTO evita un choque una vez que éste ya se ha producido.

5.9.1 Entradas y salidas

Órdenes de tarea

- **EludirContacto**

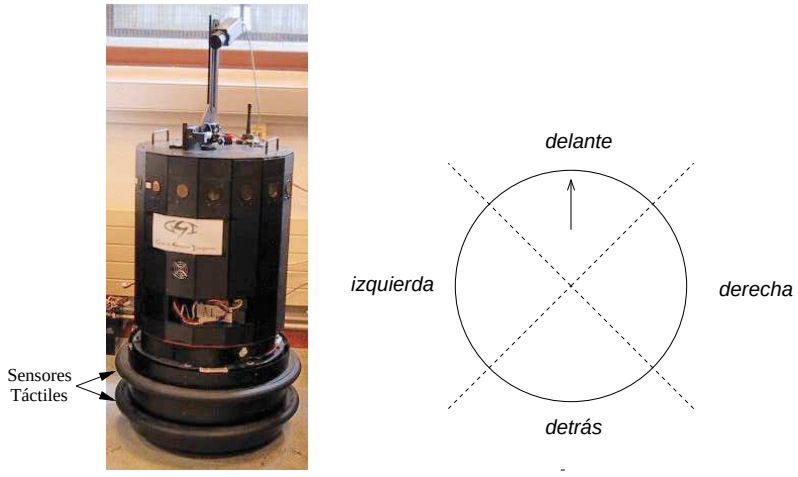

Figura 5.27: Los dos anillos de sensores táctiles en el robot móvil *Nomad 200* y las cuatro regiones en las que se dividen según la orientación de la base (flecha).

Datos de entrada

- Táctiles
- IR
- US
- Odometría

Datos de salida

- comando de movimiento

5.9.2 Síntesis del agente

El mecanismo más sencillo para que el robot eluda un contacto es realizar el movimiento contrario al que lo ha producido. Sin embargo, este mecanismo sólo funciona cuando ha sido el propio robot el responsable del choque y, en ocasiones, no siempre produce el efecto deseado. Además siempre existe el riesgo de que el robot reincida.

En el *Nomad 200* los sensores táctiles se montan sobre la base (figura 5.27), que permanece fija durante los giros (sólo rotan las ruedas). Por lo tanto, cuando se

región	velocidad lineal	velocidad angular
opuestas	nula	nula
lateral	nula	±media
delante	retroceder	mínima
detrás	avanzar	mínima

Tabla 5.4: Acciones que toma el agente ELUDIR_CONTACTO según dónde se produzca el contacto.

produce el contacto los giros de la base son seguros. Aprovechando esta característica hemos implementado el comportamiento de eludir contacto. Después de un choque, primero se gira el robot hasta que el punto de contacto esté, o bien delante, o bien detrás del robot. A continuación hacer retroceder o avanzar el robot según el caso. Para establecer los conceptos delante, detrás, derecha e izquierda se han divido los 20 sensores táctiles del robot en cuatro regiones iguales (figura 5.27).

En la tabla 5.4 se resumen las acciones que toma el robot según la región del contacto. Si hay más de un contacto en regiones contrapuestas (izquierda y derecha, delante y detrás) significa que el robot está bloqueado y que no se puede mover de forma segura. Si el contacto es lateral se gira hacia el mismo lado o hacia el contrario, según el contacto sea en la semicircunferencia frontal o en la trasera. Si el contacto es delante (detrás) además de hacer retroceder (avanzar) el robot se gira hacia la izquierda para evitar caer en bucles infinitos.

Comprobación de espacio libre

Por último, antes de avanzar (retroceder) el robot comprueba si hay suficiente espacio libre para realizar dicho movimiento. Para eso se calcula la distancia mínima de los tres sensores frontales (traseros), y si es menor que un umbral (unos 8 cm) se anula el desplazamiento dejando solamente los giros. Si no hay un giro definido se ordena girar hacia el lado donde se detecte más espacio libre.

5.9.3 Evaluación

En la figura 5.28 se muestra un ejemplo de cómo funciona el algoritmo descrito.

Figura 5.28: Funcionamiento del comportamiento ELUDIR_CONTACTO: (izquierda) detección del choque, (centro) giro y (derecha) retroceder.

5.9.4 Soluciones existentes

Casi todos los robots móviles incluyen algún mecanismo que se activa después de una colisión y cuya misión es apartar el robot. Sin embargo, no existe mucha información sobre este tipo de comportamiento. De todos modos, la implementación depende del tipo de sensores de contacto que posee el robot y de cómo están dispuestos. Por ejemplo, los robots más pequeños suelen disponer de dos sensores tipo *bigote* en su parte frontal y el comportamiento implementado suele consistir en hacer retroceder el robot y girarlo durante un tiempo o una distancia predeterminados. El sentido del giro depende de cuál es el sensor activado: derecha o izquierda.

5.9.5 Discusión

La implementación realizada de este comportamiento es muy simple y totalmente reactiva. Sin embargo, podría ser interesante incluir información de estado interno para mantener el comportamiento unos instantes aunque los sensores ya no detecten el contacto. De esta manera se conseguiría alejar el robot la distancia suficiente del lugar problemático y evitar así una posible reincidencia.

Por otra parte, también se podría utilizar el resto de la información sensorial para decidir en qué dirección se debería mover el robot, y no únicamente para comprobar si hay espacio libre en la dirección elegida. Con más información sobre el entorno también sería más fácil salir de las situaciones complicadas.

5.10 Agente MANTENER_DISTANCIA

El agente MANTENER_DISTANCIA trata de mantener una distancia de seguridad mínima entre el robot y los diferentes objetos que lo rodean.

	Índice con distancia mínima					
	0	1;15	2;14	3;13	4;12	5;11
Velocidad angular (°/s)	29	±20	±13	±8	±4	0
Velocidad lineal (cm/s)	∓2	3,8	5,1	6,1	7,1	7,1
Urgencia	1,0	0,9	0,8	0,6	0,5	0,3

Tabla 5.5: Acciones evasivas según el método del mínimo. Los sensores numerados (índice) del 6 al 10 se ignoran.

5.10.1 Entradas y salidas

Órdenes de tarea

- **MantenerDistancia** <*distancia*> [**AlinearTorreta**]. *distancia* es la distancia de seguridad y **AlinearTorreta** indica si la torreta se alinea con la base.
- **MantenerMínimo** <*distancia*>. *distancia* es la distancia de seguridad.

Datos de entrada

- ODOMETRÍA
- VELOCIDAD
- IR
- US

Datos de salida

- comando de movimiento

5.10.2 Síntesis del agente

Método del mínimo

Es la forma más simple de mantener el robot alejado de los objetos de su entorno y consiste en alejarse del objeto más próximo. Para ello se puede considerar que el par de sensores US/IR con igual numeración (índice) que mide la distancia menor es

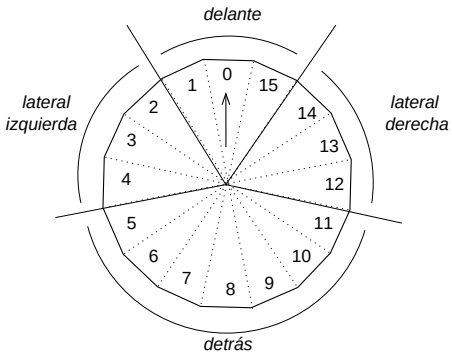

Figura 5.29: Regiones en las que se divide el entorno del robot en el método por zonas para mantenerse alejado de los obstáculos que lo rodean.

aquel que apunta al objeto más cercano al robot.[12] La acción evasiva (las velocidades lineal y angular de la base del robot) depende del índice en el que se ha detectado el objeto más cercano. Los valores se resumen en la tabla 5.5.[13]

Una vez calculada la acción evasiva se aplican una serie de filtros según la situación y, por último, se comprueba el espacio libre (apartado 5.9.2):

1. reducir a la mitad la velocidad angular si la distancia mínima es menor que 2/3 de la distancia de seguridad (girar despacio cuando haya un obstáculo cerca).

2. reducir a la mitad la velocidad lineal cuando es positiva, el índice es frontal y la distancia mínima es menor que 1/3 de la distancia de seguridad (avanzar despacio cuando existe un objeto muy cercano enfrente del robot).

3. aumentar la velocidad lineal un 50% cuando es negativa y la distancia mínima es menor que 1/3 de la distancia de seguridad (apartarse lo antes posible del objeto cuando está muy cerca y por detrás del robot).

Método por zonas

Esta técnica se basa en dividir el entorno del robot en cuatro zonas (figura 5.29), determinar el objeto más cercano en cada una de ellas y en calcular la acción evasiva en función de todos ellos. Para simplificar la casuística, las distancias se discretizan

[12]Sin embargo, no conviene olvidar que los sensores son imperfectos y que esta afirmación no siempre se cumple en todas las situaciones.

[13]La tabla de acciones se ha confeccionado manualmente, pero sería muy sencillo generarla mediante técnicas de aprendizaje (p.e., por refuerzo).

número de zonas libres	situación	clase	velocidad lineal	velocidad angular
4	libre	-	ninguna	ninguna
3	obstáculo	frontal	avanzar/retroceder	a espacio libre
		trasero	avanzar	nula
		lateral	avanzar	a espacio libre
2	pasillo	esquina	avanzar	a espacio libre
		"alineado"	avanzar	nula
		"cruzado"	nula	a espacio libre
1	encerrona	frontal	avanzar	ninguna
		trasero	ninguna	a espacio libre
		lateral	ninguna	a espacio libre
0	atrapado	-	nula	nula

Tabla 5.6: Acciones evasivas según el método por zonas.

en tres valores (lejos, cerca y muy cerca) lo que da lugar a un total de 3^4 estados diferentes. La discretización se realiza siguiendo la ecuación:

$$distancia = \begin{cases} lejos & si > distancia\,seguridad \\ muy\,cerca & si < \frac{2}{3}\,distancia\,seguridad \\ cerca & resto\,de\,casos \end{cases}$$

Para determinar la acción evasiva se utilizan los datos de la tabla 5.6 (generados de la misma manera que en la tabla 5.5). En dicha tabla la expresión *a espacio libre* significa que el robot gira hacia el lado donde detecta más espacio libre. Por otra parte, en un pasillo *alineado* los obstáculos están a la derecha y a la izquierda, mientras que en uno *cruzado* están delante y detrás. La clase de encerrona indica cuál es la única región del entorno que no tiene obstáculos (espacio libre). Por último, en el caso de un único obstáculo frontal el robot avanza o retrocede según la distancia sea *cerca* o *muy cerca*, respectivamente.

Una vez calculada la acción, si existe algún objeto muy cercano se reduce la velocidad lineal para que el robot se mueva despacio y evitar riesgos innecesarios. También se aumentan las velocidades angulares para que el robot salga lo antes posible de la situación comprometida. Por último, antes de acabar se comprueba si hay espacio libre suficiente (apartado 5.9.2).

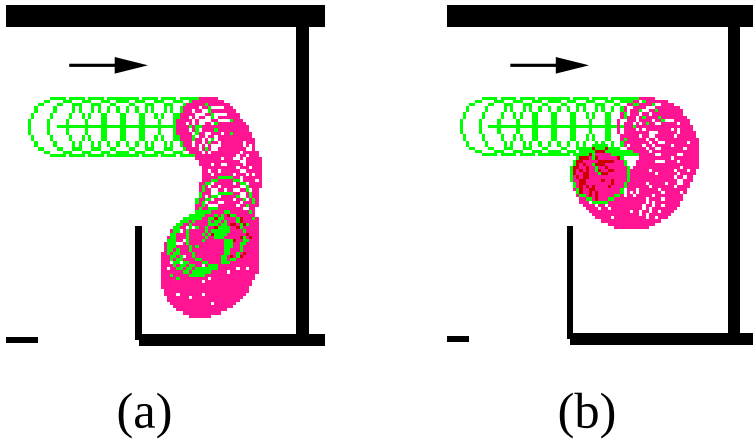

Figura 5.30: Diferencias de comportamiento del agente MANTENER_DISTANCIA según el tipo de algoritmo utilizado: (a) mínimo, (b) por zonas.

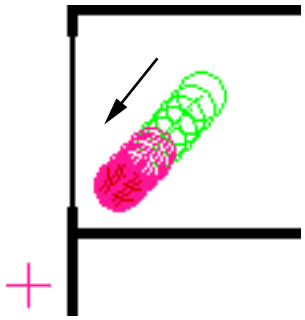

Figura 5.31: Ejemplo de cómo el agente MANTENER_DISTANCIA puede quedarse atrapado cuando utiliza el método del mínimo.

5.10.3 Evaluación

En la figura 5.30 se muestra un ejemplo donde se ven claramente las diferencias entre las dos técnicas utilizadas para ejecutar el comportamiento MANTENER_DISTANCIA. Como se puede observar, la técnica por zonas tiene la ventaja de evitar que el robot quede atrapado en lugares conflictivos, ya que para elegir cada acción considera todo el entorno que rodea al robot. Por el contrario, el método del mínimo sólo se basa en el sensor que mide la distancia más pequeña, por lo que es mucho más susceptible de caer en mínimos locales (figura 5.31) e, incluso, de chocar contra algún objeto.

Sin embargo, la técnica del mínimo permite que el robot sea capaz de cruzar una puerta sin tropezar con los marcos y sin necesidad de saber exactamente ni su posición ni su orientación. La razón es que con este método el robot se limita a

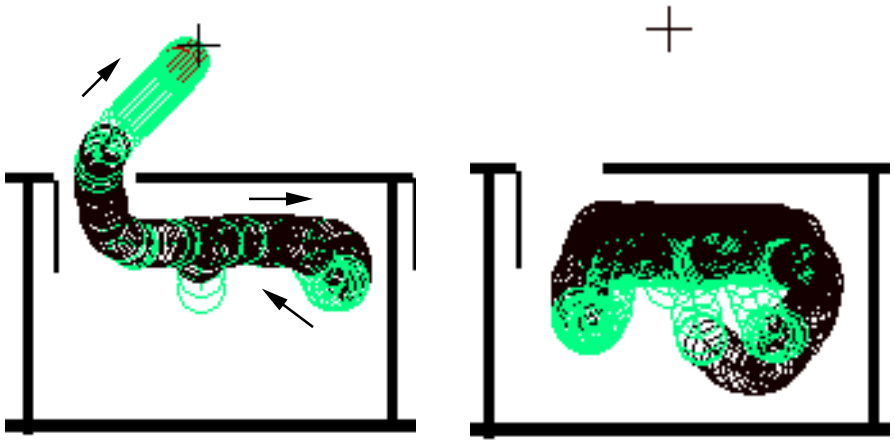

Figura 5.32: Comportamiento del robot ante los huecos según se utilice el método del mínimo (izquierda) o por zonas (derecha).

buscar el camino más corto por dónde pueda pasar, pero lo hace desde un punto de vista totalmente localista, es decir, fijándose únicamente en el sensor que indica la distancia más corta. En realidad con este método el robot muestra una gran tendencia a *colarse* por los huecos que encuentra a su paso, aunque no cumplan las condiciones de la distancia de seguridad (figura 5.32).

Las fuertes limitaciones del método del mínimo restringen su uso a tareas muy específicas (p.e., cruzar el umbral de una puerta). En los demás casos se utiliza el método por zonas. Para comprobar su robustez se encerró el robot en un espacio cerrado reducido, haciéndolo avanzar cuando no se activase el comportamiento de mantener distancia (figura 5.33). Después de cinco experimentos y 150 metros no hubo ningún contacto y la distancia media al obstáculo más próximo fue de 27 $\pm$10 cm, la mínima 12 cm y la máxima 69 cm.

5.10.4 Soluciones existentes

El comportamiento de evitar obstáculos es fundamental cuando el robot opera en entornos dinámicos, poco predecibles y muchas veces desconocidos. Existen muchas y variadas implementaciones de este comportamiento pero, por falta de espacio, nos limitaremos a comentar sólo algunas de las más importantes. Algunos autores implementan un esquema similar al nuestro y codifican en unas pocas variables las medidas que obtiene el robot de su entorno más inmediato. La principal diferencia es que utilizan lógica borrosa (Saffiotti y col., 1995; Konolige y Myers, 1998; Chatterjee

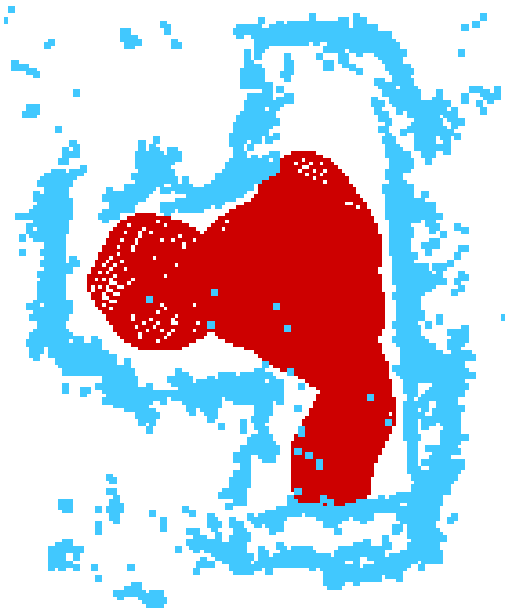

Figura 5.33: Funcionamiento del comportamiento *mantener distancia* en un entorno reducido. Los puntos claros representan las medidas de ultrasonidos y los oscuros las distintas posiciones del robot.

y Matsuno, 2001), tanto para calcular los valores de dichas variables como para determinar la acción que debe ejecutar el robot.

Uno de los métodos más populares consiste en representar los obstáculos próximos detectados por el robot mediante campos de potencial (Khatib, 1986; Latombe, 1991) o esquemas (Arkin, 1989a, 1990a) que producen fuerzas repulsivas imaginarias que tienden a separar el robot de los mismos. Su implementación es relativamente sencilla y facilita la integración de distintos comportamientos. El principal inconveniente es que surgen mínimos locales difíciles de eliminar.

Otra técnica muy común consiste en calcular en cada momento y a partir de los datos sensoriales las direcciones hacia las que se puede mover el robot de forma segura. En realidad, es más sencillo determinar aquellas hacia donde no se puede desplazar y calcular a qué distancia se encuentra el obstáculo más próximo. A partir de esta información y conociendo en qué dirección se desea mover el robot, es fácil calcular la orientación segura más próxima. Algunos autores se basan en una representación polar de una rejilla de incertidumbre local (*Vector Field Histogram*) (Borenstein y Koren, 1991b; Kortenkamp y col., 1998) mientras que otros utilizan lógica borrosa (Yen y Pfuger, 1995).

Otro mecanismo para evitar obstáculos es el de la *ventana dinámica* (Simmons,

1996; Fox y col., 1997; Thrun y col., 1998a), que consiste en seleccionar el próximo comando de control a partir del espacio de velocidades (de dos dimensiones: velocidad lineal y de rotación) formado por todas las velocidades alcanzables por el robot entre dos activaciones consecutivas. En la ventana dinámica se pueden representar todas las limitaciones del robot: hardware (aceleraciones), de seguridad (objetos próximos), etc. Con esa información el objetivo es seguir lo mejor posible la orientación deseada por el robot, dejando un margen de seguridad apropiado en cada momento y procurando maximizar la velocidad de translación.

5.10.5 Discusión

Como hemos visto, generalmente se integran los comportamientos de evitar obstáculos y mover el robot hacia una posición porque de esta forma son más fáciles de coordinar y se pueden tomar decisiones desde una perspectiva global (p.e., seleccionar el lado por el que se debe bordear cada obstáculo). En AFE-Robótica mantenemos ambos comportamientos independientes porque los coordina el control de PILOTO. De todos modos, si fuese necesario el control podría enviar indicaciones (p.e. datos sobre la tarea que está ejecutando el robot) al agente MANTENER_DISTANCIA para ayudarle en sus decisiones.

Por último, se puede mejorar este agente con sensores virtuales en el mapa de obstáculos, de manera que aunque no se detecte un objeto en un momento dado (p.e., una esquina) el agente pueda *recordar* su existencia y evitarlo.

5.11 Agente MOVER_A

El agente MOVER_A se encarga de mover el robot hacia una posición de destino y también hace que gire, avance (en línea recta o siguiendo una curva) o se desplace manteniendo una determinada orientación.

5.11.1 Entradas y salidas

Órdenes de tarea

1. **MoverA** <*x*> <*y*> [**Precisión** <*valor*>]. El destino se expresa en el sistema odométrico del robot y precisión especifica su radio.

2. **GirarA** <*ángulo_base*> <*ángulo_torreta*> [**Precisión** <*valor*>]. Los ángulos están expresados en el sistema odométrico y precisión es el margen para alcanzarlos.

3. **Avanzar** [<*velocidad*>] [**Girar** <*velocidad_angular*>]. Mover el robot a la *velocidad* indicada. Con **Girar** también se gira la base.

4. **MoverDirec** <*ángulo*>. Mover el robot manteniendo la orientación dada.

Excepto **GirarA**, en todas las *OTs* se pueden especificar valores diferentes a los establecidos por defecto para la velocidad a la que se desplaza el robot (**Velocidad** <*valor*>) y para la velocidad mínima (**VelocidadMínima** <*valor*>). Si no se especifica nada la torreta no se mueve, con **IdemTorreta** gira igual que la base y con **AlinearTorreta** sigue a la base tratando de reducir el ángulo que las separa, es decir, se alinea con la base.

Datos de entrada

- Odometría
- Velocidad
- Mapa de Obstáculos

Datos de salida

- comando de movimiento

5.11.2 Síntesis del agente

La implementación de este agente se basa en un conjunto muy simple de reglas. Se distinguen dos tipos de comportamientos: los ligados a un objetivo (**GirarA** y **MoverA**) y los continuos en el tiempo (**Avanzar**). En los primeros el agente genera un comando de movimiento para que el robot reduzca la distancia al objetivo planteado. En los segundos siempre se generan los mismos comandos. El comportamiento **MoverDirec** se puede considerar mixto porque combina un avance continuo y un giro de la base para mantener la orientación especificada (objetivo). A continuación describiremos la implementación de cada comportamiento del agente Mover_A.

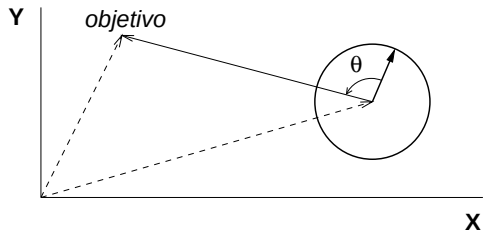

Figura 5.34: Ángulo θ para orientar el robot hacia la posición objetivo.

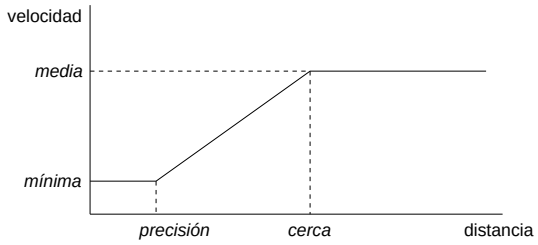

Figura 5.35: Dependencia de la velocidad de avance del agente MOVER_A con la distancia al objetivo.

Mover hacia una posición odométrica

MoverA <*x*> <*y*> [**Precisión** <*valor*>] [**IdemTorreta** | **AlinearTorreta**]]

1. Calcular el ángulo de giro para que el robot apunte hacia la posición objetivo (figura 5.34). La velocidad de giro es proporcional a dicho ángulo.
2. Calcular la velocidad de avance en función de la distancia al objetivo (figura 5.35). Se considera que *cerca* es 5 veces la *precisión*.
3. Reducir la velocidad de avance si el giro es pronunciado:
4. La torreta no se mueve, gira igual que la base (**IdemTorreta**) o se alinea con ella (**AlinearTorreta**).

Avanzar

Avanzar [<*velocidad*>] [**Girar** <*angular*> [**IdemTorreta** | **AlinearTorreta**]]

1. Desplazar el robot a la velocidad dada o, por defecto, a 15 cm/s.
2. Opcionalmente, girar también la base a la velocidad indicada.

3. La torreta no se mueve, gira igual que la base (**IdemTorreta**) o se alinea con la base (**AlinearTorreta**).

Girar base y torreta

GirarA <*ángulo_base*> <*ángulo_torreta*> [**Precisión** <*valor*>]

1. La base y la torreta giran de forma independiente entre sí. La velocidad de giro de cada una es proporcional a la diferencia entre el ángulo deseado y el que tenga en ese momento (siempre que la diferencia sea superior a la precisión).[14]

Mover manteniendo una orientación

MoverDirec <*ángulo*> [**Velocidad** <*valor*>] [**IdemTorreta**|**AlinearTorreta**]

1. Desplazar el robot a la velocidad media dada o al valor por defecto.
2. Girar la base para mantener la orientación especificada.
3. La torreta no se mueve, gira igual que la base (**IdemTorreta**) o se alinea con ella (**AlinearTorreta**).

5.11.3 Evaluación

En las figuras 5.36, 5.37 y 5.38 se muestra la trayectoria que sigue el robot con los principales comportamientos que implementa el agente MOVER_A.

5.11.4 Soluciones existentes

Los comportamientos que implementa el agente MOVER_A son, en general, muy sencillos. Aún así, existen autores que utilizan técnicas más refinadas y complejas (p.e. lógica borrosa (García-Alegre y col., 1995b; Saffiotti, 1995), método de la ventana dinámica (Fox y col., 1997), aprendizaje por refuerzo (del R. Millán, 1995; Mahadevan y Connell, 1992), etc.), capaces de generar movimientos más suaves y continuos en el tiempo.

[14] Se limitan los valores máximos y mínimos de los giros a 20º/s y 4º/s, respectivamente.

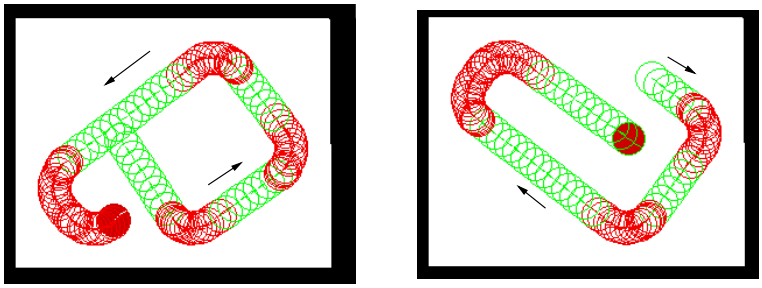

Figura 5.36: Comportamientos *avanzar* del agente MOVER_A (círculos claros) y *evitar obstáculos* del agente MANTENER_DISTANCIA (círculos oscuros).

Figura 5.37: Comportamientos *moverse siguiendo la orientación 0º* (agente MOVER_A) y *evitar obstáculos* (MANTENER_DISTANCIA). Los puntos son medidas de US y los círculos las posiciones del robot.

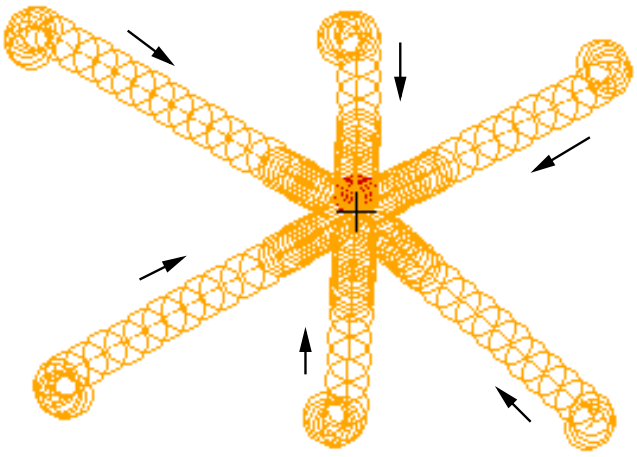

Figura 5.38: Comportamiento *mover hacia una posición odométrica* del agente MOVER_A desde distintas posiciones y orientaciones iniciales. La cruz indica la posición objetivo.

Otros autores implementan el comportamiento *mover hacia una posición odométrica* utilizando la información del entorno almacenada en un mapa y en base a pequeños desplazamientos. Si el mapa es global, este comportamiento es capaz de encontrar y seguir caminos o rutas entre cualesquiera dos puntos del entorno. Una de las combinaciones más usuales es utilizar mapas basados en rejillas de ocupación junto con campos de potencial para establecer el ángulo de giro y la velocidad hacia el objetivo y evitando los obstáculo marcados en el mapa.

Entre los autores que utilizan esta técnica destacamos Latombe (que se puede considerar como el pionero), Arkin (Arkin, 1989a), otro de los pioneros en utilizar esta técnica, Zelek (Zelek, 1996), el primero en utilizar funciones armónicas para evitar los mínimos locales, los programadores del robot *Rhino* (Thrun, 1997; Thrun y col., 1998a), Murphy(Murphy y col., 1999), con su planificador *Trulla*, etc.

5.11.5 Discusión

La gran ventaja de las técnicas empleadas para implementar todos los comportamientos del agente MOVER_A es su sencillez y robustez. Su principal limitación es que caen frecuentemente en mínimos locales, ya que sólo utilizan la información sensorial más reciente y no tienen en cuenta aspectos más globales como la tarea que está ejecutando el robot. Para evitar este problema se podría utilizar la información que se almacena en el mapa de obstáculos.

5.12 Agente SEGUIR_CONTORNO

El agente SEGUIR_CONTORNO genera los comandos de movimientos necesarios para que el robot siga un contorno lateral (bien a la derecha, bien a la izquierda) a una cierta distancia de referencia. Este comportamiento no conlleva ninguna condición explícita para su finalización y se ejecuta de forma indefinida.

5.12.1 Entradas y salidas

Órdenes de tarea

- **SeguirContorno {Derecha | Izquierda}**

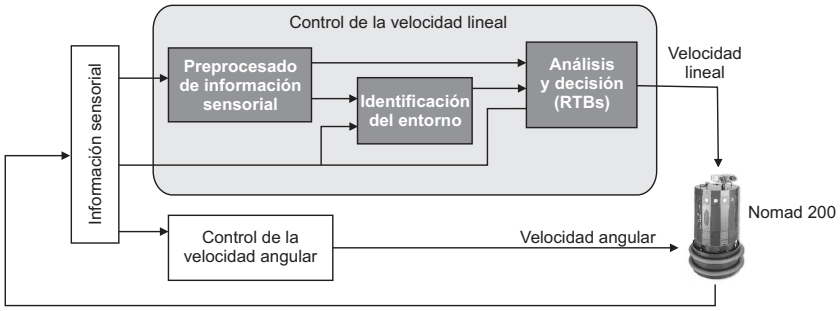

Figura 5.39: Esquema del agente SEGUIR CONTORNO (Mucientes y col., 2002b).

Datos de entrada

- ODOMETRÍA
- VELOCIDAD
- US

Datos de salida

- comando de movimiento

5.12.2 Síntesis del agente

El comportamiento de seguir contorno es básico en cualquier robot que pretenda realizar tareas de navegación. Debido a su trascendencia hemos dedicado un notable esfuerzo a resolver adecuadamente dicho comportamiento, dado que en nuestra opinión las soluciones hasta ahora existentes adolecen de severas limitaciones. Hemos implementado diferentes versiones aunque, para abreviar la exposición, sólo se describe la que se ha integrado en la arquitectura. Además, por constituir parte del trabajo realizado en el marco de otra tesis doctoral, actualmente en ejecución por otro miembro del grupo, nos limitaremos a realizar una descripción superficial que, en todo caso, nos permita conocer los aspectos mínimos de la síntesis este agente.

Un buen comportamiento de seguir contorno se caracteriza por: mantener una distancia adecuada con el contorno a seguir; moverse en cada momento a la máxima velocidad que permita el entorno; y evitar movimientos bruscos. Para conseguir estos tres objetivos se ha dividido el agente SEGUIR_CONTORNO en dos bloques independientes (figura 5.39): uno controla la velocidad lineal y otro la angular.

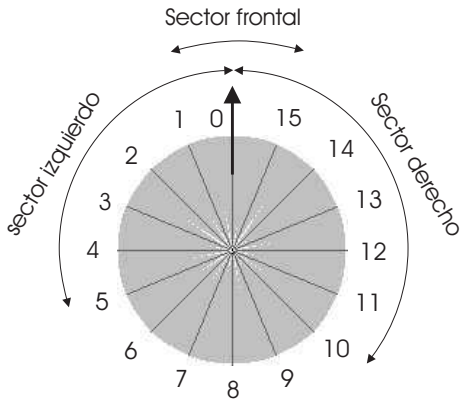

Figura 5.40: Disposición de los sensores de ultrasonidos en el agente SEGUIR_CONTORNO.

Control de la velocidad angular

El control de la velocidad angular se realiza en dos etapas (Iglesias y col., 1997). La primera procesa la información de los US frontales y laterales (derecha o izquierda) del robot (figura 5.40) para obtener un patrón de 8 bits donde cada bit indica si existe un segmento de pared próximo al robot en una orientación.

La segunda etapa consiste en asociar una velocidad angular para cada uno de los 64 estados posibles. Para generar esta tabla se ha entrenado una red perceptrón multicapa con el algoritmo de retropropagación y un pequeño conjunto de ejemplos a partir del cual la red ha sido capaz de generalizar y generar una acción adecuada para cada estado.

Usando técnicas más complejas (por ejemplo, aprendizaje por refuerzo (Iglesias y col., 1998a, 2002)) se puede generar una tabla más completa que incluye tanto la velocidad angular como la lineal y con la cual este módulo puede implementar por sí sólo todo el comportamiento. La principal desventaja de este mecanismo es que el robot no es capaz de aprovechar situaciones del entorno favorables para aumentar su velocidad.

Control de la velocidad lineal

Como se puede ver en la figura 5.39 el control de la velocidad lineal se realiza en tres etapas. En la primera, se preprocesa la información de los US de una manera similar al realizado en el control de velocidad angular, pero con la diferencia de que el estado generado es un patrón formado por 9 valores entre 1 y 15 con más

información que uno binario. El cálculo del patrón lo realizan 9 redes de Kohonen unidimensionales de 15 neuronas cada una (Iglesias y col., 1998b). La segunda etapa consiste en determinar la situación en la que se encuentra el robot a partir de la información de los sensores de ultrasonidos y de la odometría. Sólo se distinguen tres situaciones diferentes: esquina abierta, pared recta y esquina cerrada.

La tercera y última etapa consiste en aplicar la información almacenada en una base de conocimiento para determinar la velocidad lineal que mejor se adapte a la situación del robot y frenar o acelerar el robot para conseguirla. Esta base de conocimiento se codifica según un modelo de Reglas Temporales Borrosas (RTBs) (Bugarín y col., 1999), capaz de contemplar información que evoluciona con el tiempo. Otra ventaja de la capacidad de razonamiento temporal de las RTBs es que permite filtrar el ruido sensorial.

En la base de conocimiento existen reglas específicas para cada situación (un total de 108 para pared recta, 140 para esquina abierta y 66 para esquina cerrada), con distintas variables antecedentes y diferentes universos de discurso para cada una. Estas variables se calculan directamente a partir de la información sensorial que proporciona el robot: distancia frontal, derecha e izquierda, velocidad lineal del robot, orientación, tendencias de las distancias, tendencia de la orientación, etc. Más detalles sobre el control de la velocidad lineal se pueden encontrar en (Mucientes y col., 2002b,a).

5.12.3 Evaluación

En las figuras 5.41 y 5.42 se muestran dos ejemplos representativos de los resultados (Mucientes y col., 2002b) obtenidos con el agente SEGUIR_CONTORNO y sus datos más importantes se resumen en la tabla 5.7. La velocidad media en el primer ejemplo es menor que en el segundo porque posee menos tramos rectos. Sin embargo, las velocidades máximas y mínimas son prácticamente idénticas en ambos casos. La velocidad mínima es tan baja porque a veces el robot se frena para facilitar los giros. Otro punto importante es que la distancia al contorno de referencia (el derecho) es prácticamente constante.

En la figura 5.43 se puede observar cómo se adapta la velocidad lineal al entorno. Así, en paredes rectas suficientemente largas (S_1, S_2, S_3, S_5 y S_6) el robot alcanza y mantiene su velocidad máxima (entre 30 cm/s y 40 cm/s de media). Por otra parte, en las esquinas cerradas el robot reduce la velocidad lineal tanto más cuanto más cerrada sea la esquina.

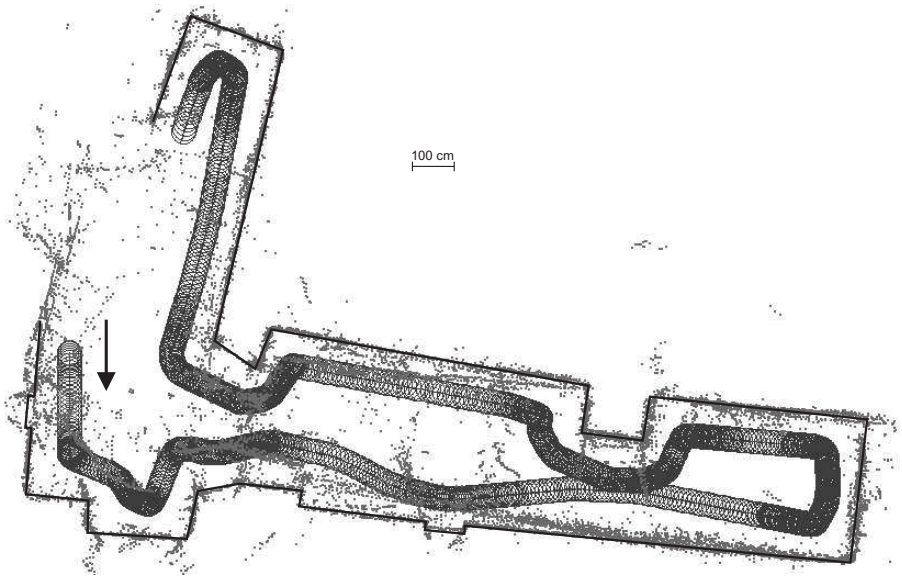

Figura 5.41: Ejemplo 1 del agente SEGUIR_CONTORNO.

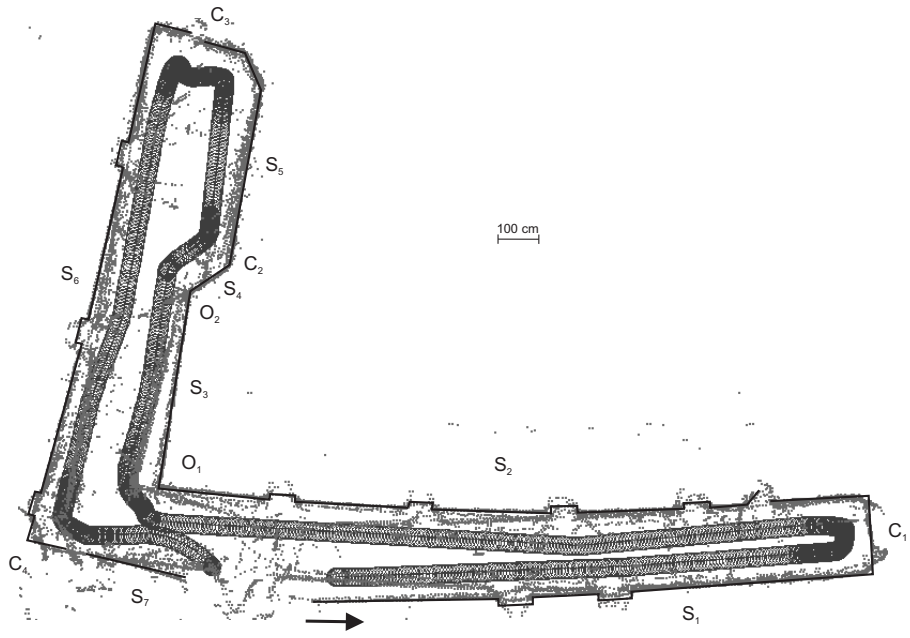

Figura 5.42: Ejemplo 2 del agente SEGUIR_CONTORNO.

	Ejemplo 1	Ejemplo 2
Distancia derecha media (cm)	47	49
Distancia izquierda media (cm)	131	127
Velocidad media (cm/s)	17	25
Velocidad máxima (cm/s)	41	44
Velocidad mínima (cm/s)	1	0
Tiempo	3' 51"	3' 55"

Tabla 5.7: Datos de los dos ejemplos de seguir contorno mostrados.

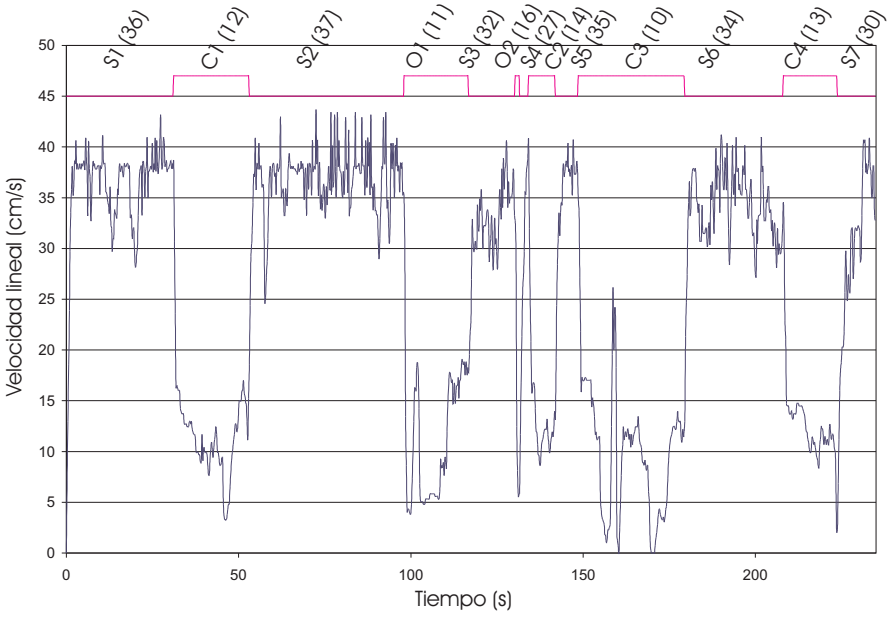

Figura 5.43: Variación de la velocidad lineal a lo largo de la ruta del ejemplo 2.

5.12.4 Soluciones existentes

El comportamiento de seguir un contorno es muy utilizado para mover el robot en entornos desconocidos. Entre todas las técnicas destacamos las basadas en redes neuronales (Meng y Kak, 1993; Pimentel y col., 1993; Iglesias y col., 1997), lógica borrosa (García-Alegre y col., 1995b; Saffiotti y col., 1995; Castellano y col., 1997; García-Cerezo y col., 1997), algoritmos genéticos (Husbands y col., 1995), aprendizaje por refuerzo (Iglesias y col., 1998a, 2002) y esquemas (Arkin, 1989a).

Sin embargo, muy pocas implementaciones tratan de aprovechar al máximo las capacidades del robot y lo hacen avanzar a la mayor velocidad posible según cada situación. Otra gran diferencia entre método implementado y las propuestas anteriores es que utiliza toda la información sensorial previa y no sólo la adquirida en un momento dado, de modo que puede anticipar mejor las situaciones futuras para el robot.

Otros autores (Braunstingl y col., 1995) han propuesto sistemas cuyas entradas reflejan variaciones respecto al tiempo tanto de la distancia como de la orientación al contorno de referencia. Sin embargo, y a diferencia de las RTBs, esas derivadas sólo reflejan variaciones entre dos instantes consecutivos, por lo que no son capaces de manejar la evolución de las variables sobre todo el intervalo de interés.

5.12.5 Discusión

Como se puede observar en los ejemplos presentados, la velocidad del robot se adapta en cada momento al entorno, reduciendo en gran medida el tiempo necesario para realizar un recorrido. Todo ello sin perder en ningún momento el contacto con el contorno. Como aspectos negativos, señalar que el identificador de situación confunde con frecuencia las puertas con esquinas abiertas, perjudicando la eficiencia del comportamiento en pasillos con muchas puertas, y que el control de velocidad angular no es capaz de mantener el robot adecuadamente alineado con una pared recta. Además de intentar mejorar este aspecto, también estamos estudiando que la distancia a la que se sigue el contorno pueda ser modificada.

5.13 Agente Creador_Mapa_Local

El agente Creador_Mapa_Local confecciona y mantiene un mapa métrico con los obstáculos que rodean el robot (Mapa de Obstáculos) a partir de la

información sensorial y de la posición odométrica. El mapa generado cubre una región del entorno y es muy útil para detectar marcas y objetos móviles.

5.13.1 Entradas y salidas

Órdenes de tarea

- **ActualizarMapa**. Actualiza el mapa con las nuevas medidas sensoriales.
- **SalvarMapa**. Graba los datos del mapa en un fichero en formato PGM.
- **BorrarMapa**. Reinicializa el mapa (p.e., al entrar en una nueva región).
- **CargarMapa**. Carga los datos iniciales del mapa (p.e., al entrar en una región de la que ya existe un mapa previo).

Datos de entrada

- Odometría
- IR
- US
- Láser
- Mapa de Obstáculos

Datos de salida

- Mapa de Obstáculos
- comando de movimiento

5.13.2 Síntesis del agente

Para representar el Mapa de Obstáculos se utiliza un mapa basado en una rejilla de incertidumbre con celdas cuadradas de 4 cm de lado, donde cada una almacena el nivel de certidumbre de que el espacio que representa esté libre u ocupado. Estos mapas son muy populares en los robots que operan en tiempo real

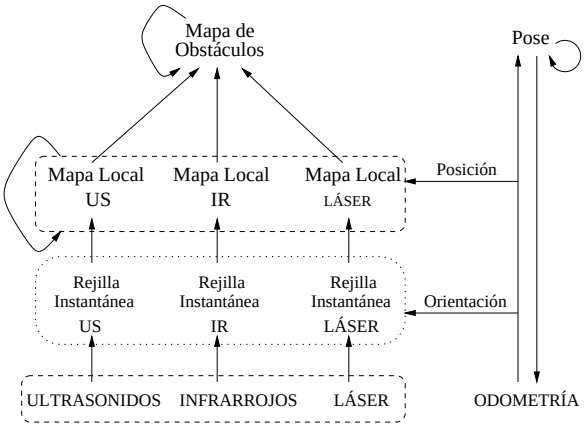

Figura 5.44: Confección del mapa de obstáculos.

porque son sencillos, requieren relativamente pocos recursos computacionales y pueden ser interpretados directamente por los seres humanos. Entre sus limitaciones podemos destacar el alto coste de almacenamiento y la dificultad para detectar en ellos características de alto nivel en el entorno (p.e., marcas).

La construcción de un mapa pasa por crear un mapa por cada tipo de sensor (IR, US y Láser) y después fundirlos todos en uno (figura 5.44). Por dos motivos: primero, cada tipo de sensor posee sus propias características y debe ser modelado de forma diferente; segundo, los sensores suelen estar dispuestos sobre el robot a distintas alturas[15] y tienen puntos de vista sobre el entorno que no siempre se pueden superponer directamente. Sin embargo, como los sensores de IR y del láser en el *Nomad 200* tienen un *área de percepción*[16] muy reducida, hemos optado por utilizar únicamente los US.

La confección de una rejilla de incertidumbre consta de dos pasos (figura 5.45). El primero consiste en traducir la distancia que proporciona cada sensor junto con su orientación en el sistema odométrico del robot (figura 5.44) en una matriz bidimensional que indica las celdas dónde es más probable que exista espacio libre u ocupado. Esta traducción depende del *modelo de sensor* elegido y el resultado es una *rejilla instantánea* centrada en el robot. De los distintos modelos probados para los sensores de US (Rodríguez y col., 2000) el más simple es, paradógicamente, el que mejor funciona (figura 5.46).

El segundo paso consiste en acumular los datos de la rejilla instantánea en la

[15]En el Nomad 200 los IR están a 36 cm, los US a 78 cm y el láser a 91 cm.

[16]El láser sólo detecta obstáculos entre 50 y 250 cm y en un ángulo de 30º, mientras que los IR tienen un ángulo de detección muy pequeño y un alcance máximo de 40 cm.

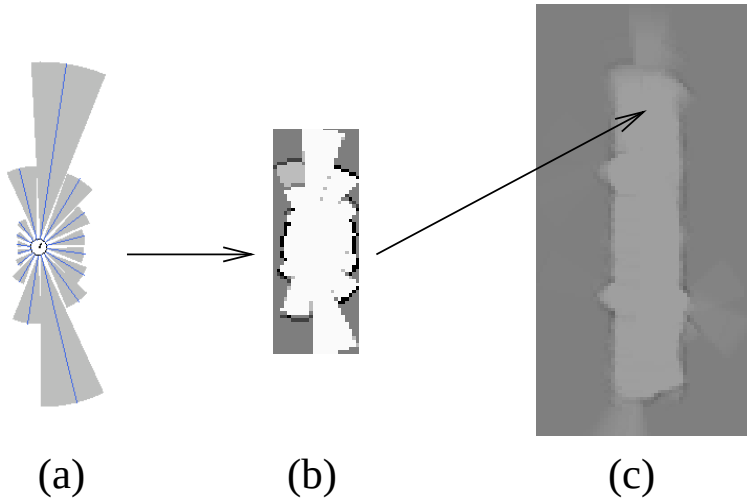

Figura 5.45: Creación del mapa local: (a) medidas sensoriales, (b) rejilla instantánea (representación en niveles de gris a efectos de visualización) y (c) rejilla de incertidumbre.

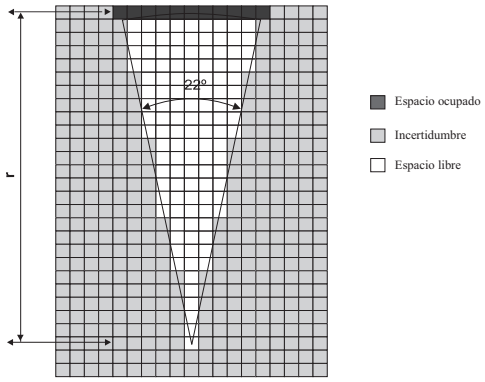

Figura 5.46: Modelo de incertidumbre del sensor de US.

rejilla de incertidumbre o mapa local. En este punto se utiliza la posición odométrica del robot para localizar cada rejilla instantánea en el mapa local (figura 5.45). Como ya se ha utilizado la orientación odométrica del robot para calcular la rejilla instantánea, no es necesario rotarla (proceso muy costoso y fuente importante de imprecisiones).

Para acumular la información en cada celda de la rejilla hay distintos métodos, pero el que mejor funciona es el de la suma (Rodríguez y col., 2000). Por último, comentar que se ha limitado el valor de ocupación de cada celda a un máximo de 127 y a un mínimo de -127. El valor inicial de todas las celdas es 0, es decir, el nivel máximo de incertidumbre.

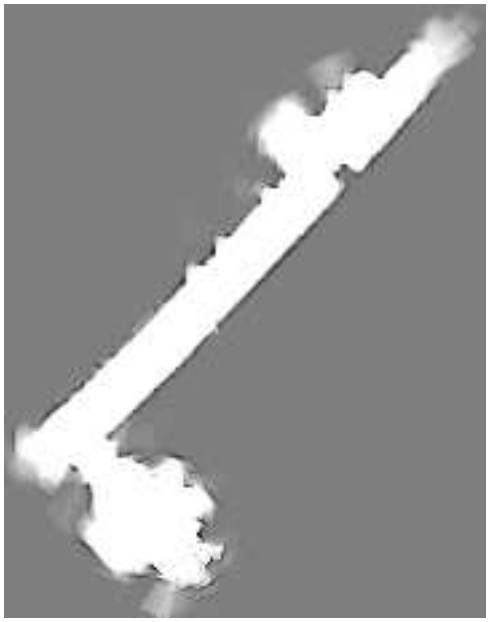

Figura 5.47: Ejemplo 1 de rejilla de incertidumbre.

5.13.3 Evaluación

En las gráficas 5.47 y 5.48 se pueden ver algunos ejemplos que muestran claramente la calidad de los mapas que se obtienen con los datos de US, por medio del método de la suma y mientras el robot recorre un corredor. Para ilustrar cómo se va construyendo un mapa, en la figura 5.49 se muestra una secuencia de mapas parciales generados cada 100 adquisiciones de US.

5.13.4 Soluciones existentes

Existen muchos métodos para confeccionar mapas métricos basados en una rejilla de incertidumbre. Entre ellos los que se basan en el teorema de Bayes (Moravec, 1988; Elfes, 1989; Cho, 1990; Konolige, 1996; Thrun, 1998), la lógica borrosa (Gambino y col., 1996; Oriolo y Ulivi, 1998), la teoría de la evidencia de Dempster-Shafer (Pagac y col., 1998; Duckett y Nehmzow, 1998) y los métodos de conteo (Borenstein y Koren, 1991a). Las principales diferencias con el propuesto es que utilizan distintos modelos de sensor y métodos de acumulación (Rodríguez y col., 2000). Exceptuando el método de conteo, todos son más difíciles de computar y producen peores resultados (mapas con menor definición y precisión).

Por su parte, en (Borenstein y Koren, 1991a) se utiliza un modelo de sensor en

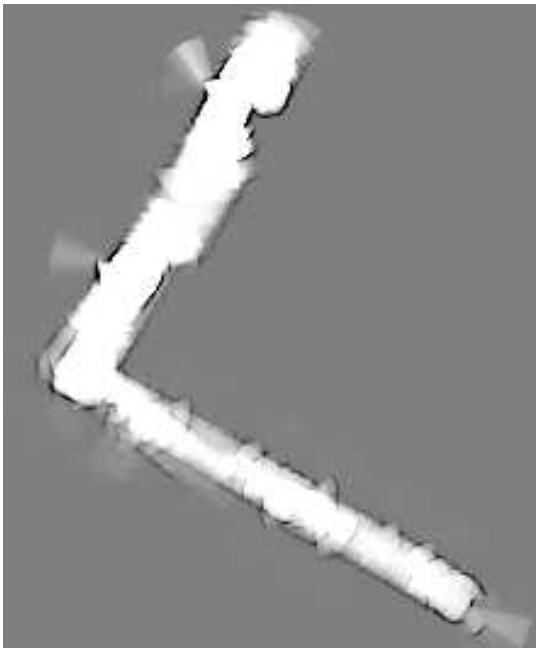

Figura 5.48: Ejemplo 2 de rejilla de incertidumbre.

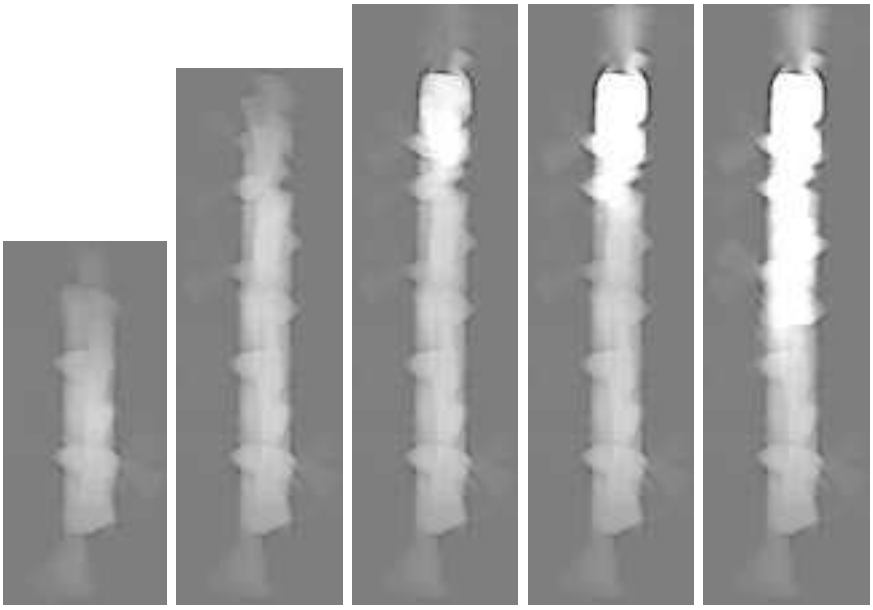

Figura 5.49: Secuencia que ilustra la construcción de una rejilla de incertidumbre. Cada imagen se crea tras 100 nuevas adquisiciones sensoriales.

un sola dimensión, lo que incide muy negativamente en la detección de espacio libre. El coste computacional de utilizar modelos en dos dimensiones se compensa por la mejor calidad de los mapas confeccionados. En (Rodríguez y col., 2000) se estudian en más detalle las técnicas para construir rejillas de incertidumbre, los mapas que se obtienen con cada una de ellas y las comparaciones estadísticas realizadas.

Por último, comentar que algunos autores crean una rejilla de incertidumbre a partir de los datos de otros sensores, principalmente láser y de visión (Dillmann y col., 1995; Bolles y col., 1995). Ambos tipos de sensores tienen limitaciones de las que carecen los US (p.e., no detectan las superficies acristaladas) pero poseen una mejor precisión y resolución angular. Por lo tanto, la fusión de los mapas generados con estos sensores y con los creados a partir de US producen mapas mucho más precisos y robustos (Fabrizi y col., 2001).

5.13.5 Discusión

El mapa de obstáculos creado es fiable y rápido pero no está exento de errores. Hay dos fuentes básicas de errores: los propios datos sensoriales y la estimación de la posición odométrica. En el primer tipo se incluyen todos los problemas que generan los sensores de US: falsos ecos, múltiples ecos, reflexiones especulares, baja resolución angular, variaciones con la humedad y la temperatura, etc. Estos errores son sistemáticos y afectan al modelado del sensor y se producen preferentemente en ciertas configuraciones especiales del entorno (esquinas, puertas, paredes rectas, etc.). Estos errores se reducirían considerablemente si se funde la información de diferentes tipos de sensores.

El segundo tipo de errores se debe a la incorrecta acumulación de las medidas sensoriales en el mapa y su principal causa es una posición del robot poco precisa. Este fenómeno se aprecia claramente en la figura 5.48 y es que cuando el robot recorre el corredor por segunda vez las paredes no coinciden exactamente con las ya detectadas. Los errores de la posición también se aprecian en el pequeño arqueo de las paredes en los mapas. El error en la odometría es acumulativo, porque se basa exclusivamente en información propioceptiva (principalmente, el giro de las ruedas) y para mantenerlo acotado se debe corregir la posición a partir del mapa del entorno y de las marcas y mapas percibidos.

Otra causa de errores, aunque mucho menos importante, se debe a que en la acumulación de la información sensorial en los mapas se presupone erróneamente que las adquisiciones sensoriales son independientes entre sí. Para tener en cuenta este

efecto habría que utilizar información sobre el tipo de sensor durante la acumulación. P.e., considerando las reflexiones especulares cuando se detecta una pared en un cierto ángulo o la aparición de falsos ecos cerca de una esquina cerrada.

5.14 Agente DETECTOR_MARCAS

El agente DETECTOR_MARCAS detecta a través de la información sensorial aquellos objetos que se consideran más relevantes en el entorno (MARCAS) y que son fundamentales para la localización y el reposicionamiento. Para mejorar la detección también genera comandos, principalmente para girar la torreta que es en dónde se montan los principales sensores del *Nomad 200*.

5.14.1 Entradas y salidas

Órdenes de tarea

- **DetectarMarcas** <*lado*>. Detectar marcas a partir de los datos sensoriales y alinear la torreta con el contorno del *lado* especificado (*Derecha*, *Izquierda*, *AtrásDerecha* o *AtrásIzquierda*). Si no se indica el lado o es *Cualquiera* no se gira la torreta.

- **AlinearTorreta** <*lado*>. Alinear la torreta con el contorno del *lado* indicado.

- **BorrarDatos**. Reinicializar todos los datos que se usan en la detección.

Datos de entrada

- ODOMETRÍA
- US
- LÁSER
- MAPA DE OBSTÁCULOS
- MARCAS

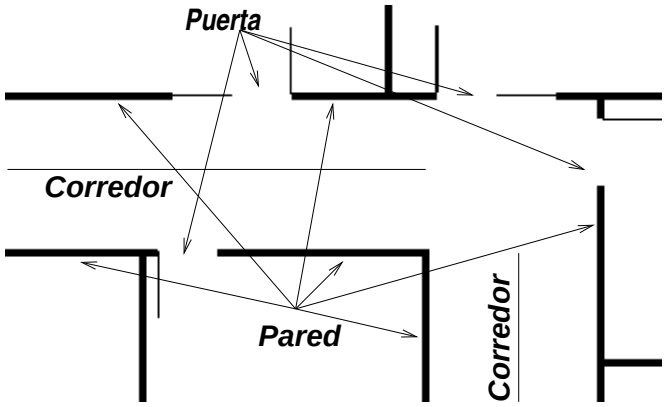

Figura 5.50: Ejemplos de marcas tipo pared, puerta y corredor.

Datos de salida

- MARCAS
- comando para alinear torreta con el contorno de interés

5.14.2 Síntesis del agente

Marcas

Una marca es un objeto característico del entorno que cumple varios requisitos: es fácilmente reconocible desde distintas posiciones, las características que se usan para su detección no cambian excesivamente con el tiempo, permanece relativamente inmóvil y es suficientemente abundante. Si las marcas se disponen en el entorno especialmente para cubrir las necesidades del robot se consideran *artificiales*, mientras que si su función principal es otra, ajena al robot, son *naturales*. Como es obvio, es deseable que el robot se pueda desenvolver en entornos no modificados, por lo que siempre se prefieren las marcas *naturales*. Para navegar en el interior de edificios hemos seleccionado tres tipos de marcas naturales (figura 5.50):

pared, cualquier segmento recto suficientemente largo. Se representa mediante un segmento orientado (es decir, una flecha).

puerta, los bordes de dos paredes, generalmente alineados y separados una cierta distancia. Se representa con un segmento que une ambos bordes.

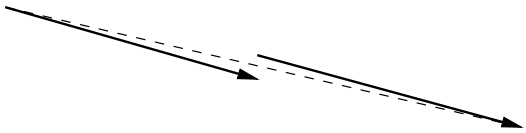

Figura 5.51: Fusión de dos segmentos colineales y muy próximos entre sí.

corredor, dos paredes suficientemente largas y paralelas entre sí. Se representa mediante tres segmentos: las dos paredes y el eje central.

Detección de marcas con el láser

El *Nomad 200* posee un sensor láser que funciona por triangulación (el *Sensus 500* (Nom, 1995)) y que puede operar en dos modos: detectar puntos o detectar segmentos. Los datos que proporciona el sensor son las coordenadas de los puntos o de los segmentos en un sistema de referencia centrado en el robot y con la misma orientación que el láser.

El algoritmo para detectar marcas a partir de los datos del láser es el siguiente. En primer lugar, se ordenan los puntos inicio y fin de cada segmento detectado para fijar una orientación. El criterio utilizado es que al ir del principio del segmento al final el *espacio libre queda a la derecha.* Después se trasladan los segmentos del sistema de referencia centrado en el robot al sistema de referencia global que proporciona el sistema odométrico del robot. Ésta es la única forma de manipular segmentos procedentes de distintas adquisiciones pero requiere una odometría muy fiable.

Las *paredes* del entorno se detectan integrando los segmentos de cada nueva adquisición del láser con los ya detectados. Para fundir dos segmentos en uno (figura 5.51) deben ser paralelos (menos de 6º de diferencia) y estar próximos entre sí (a menos de 5 cm). Este proceso se aplica de forma recursiva formando segmentos (paredes) cada vez más largos. Para evitar que los errores en la odometría influyan negativamente en la detección se eliminan las paredes que no hayan sido percibidas recientemente, p.e. en los últimos 20 segundos.

Un *borde* es un "salto" entre dos segmentos consecutivos en una misma adquisición del láser (figura 5.52). Sin embargo, se pueden detectar de una forma más simple y robusta simplemente considerando que los extremos de las paredes resultado de fundir todos los segmentos detectados son los bordes del entorno. Dos bordes muy próximos entre sí formarían una esquina.

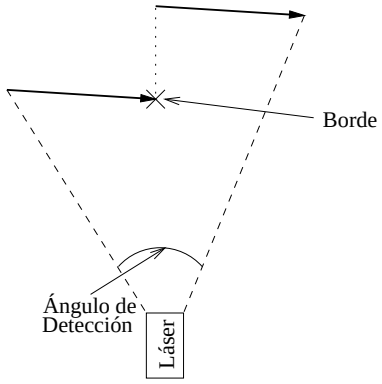

Figura 5.52: Condiciones para que se detecte un borde.

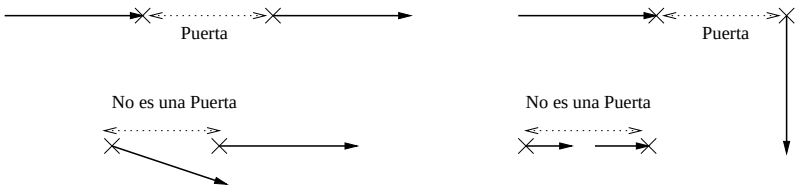

Figura 5.53: Condiciones para que se detecte o no una puerta.

La detección de una *puerta* no se puede realizar directamente a partir de los datos de una única adquisición del láser debido a las limitaciones del *Sensus 500*. Por lo tanto, es necesario recurrir a métodos indirectos. Hemos implementado una técnica muy simple pero bastante efectiva que consiste en asociar una puerta con un par de bordes (es decir, extremos de una pared), separados entre 60 y 100 cm, siempre y cuando ninguno de los segmentos asociados a dichos bordes "ocupe" el espacio intermedio, es decir, el teórico umbral de la puerta (figura 5.53).

La detección de un corredor más sencilla es por medio de sus dos paredes. Sin embargo, debido a la reducida área de percepción del láser del *Nomad 200*, sólo hay dos posibilidades para detectar las dos paredes de un corredor: girar la torreta constantemente o desplazarse varias veces por el corredor detectando una pared diferente en cada recorrido.

La primera estrategia no es eficiente, porque incluso girando la torreta a la máxima velocidad se necesitan casi 6 segundos para dar toda una vuelta. Por lo tanto, para detectar de forma continua las dos paredes de un corredor se necesitaría que el robot avanzase muy despacio. La segunda estrategia tampoco es demasiado viable debido a los errores que comete la odometría y exigiría que el robot recorriese un mínimo de dos veces cada corredor.

Control de la torreta

Como el área de percepción del láser del *Nomad 200* es muy reducida, se debe prestar especial atención en orientarlo adecuadamente hacia el contorno en dónde se quiere detectar marcas. Cuanto más perpendicular está el láser al contorno más se reduce el área de percepción mientras que si está casi paralelo más lejos y con más dificultad se detecta el contorno. Experimentalmente se ha encontrado que el mejor ángulo es de 45º, aunque el margen es algo mayor (entre 25º y 70º). Para controlar la torreta hemos implementado un algoritmo muy simple que se puede resumir en los siguientes pasos:

1. Determinar el segmento del láser más largo entre las últimas adquisiciones. Consideramos que éste marca la orientación predominante del contorno.

2. Girar la torreta hasta conseguir el ángulo óptimo. Para evitar cabeceos innecesarios sólo se consideran desviaciones superiores a un cierto umbral.

3. Si el ángulo entre la torreta y la base no es el correcto es porque la torreta no está bien alineada con el contorno y se deben descartar las medidas sensoriales.

Detección de marcas con los US

Al contrario que el láser del Nomad, los US cubren todo el espacio que rodea al robot hasta una distancia aceptable (unos 6,5 m). Sin embargo, aunque su precisión es relativamente buena (un 5% de la distancia medida) su resolución angular es pésima y las medidas se distorsionan por múltiples causas (efecto especular, múltiples ecos, *crosstalking*, etc.). A pesar de estos problemas, los US se pueden usar para detectar las paredes del entorno mientras el robot las sigue. Además, como los US apuntan en todas direcciones, es más fácil detectar conjuntamente las dos paredes de un corredor, simplificando así los algoritmos y reduciendo los errores. El proceso de detección consta de varias etapas.

En primer lugar, se filtran los datos de US de forma que sólo quede una medida a cada lado del robot (derecha e izquierda). Cuando el robot está en un corredor o ha detectado un corredor o una pared, entonces se filtran los datos de manera que sólo queden los US que estén más próximos a la perpendicular del corredor o de la pared. El objetivo es escoger en cada momento los sensores que probablemente apuntan perpendicularmente a una pared. A continuación se traduce cada dato del sistema

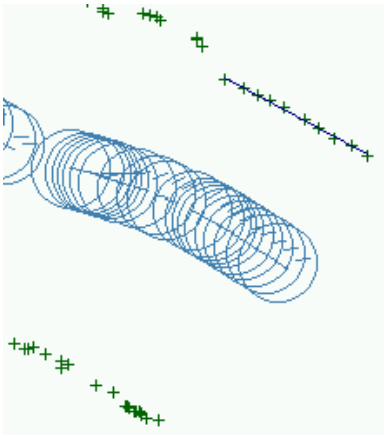

Figura 5.54: Detección de una pared con los US mientras el robot la sigue.

de referencia en coordenadas polares, centrado en el robot, al sistema odométrico en coordenadas cartesianas.

Cada punto se almacena en el *buffer* circular que le corresponde (derecha e izquierda). Para evitar redundancias se descartan aquellos puntos que estén demasiado próximos a los últimos almacenados (a menos de 2,5 cm). Por otra parte, si la distancia es mayor que otro umbral (50 cm) se considera que se ha perdido la pared y se borra el *buffer* para empezar de nuevo.

Cuando un *buffer* está completo (10 valores), con cada nuevo punto se realiza un ajuste por mínimos cuadrados de su contenido. Si el error es pequeño significa que están alineados y que pueden formar parte de una pared. Cuando se detecta una pared se amplía el tamaño del *buffer* para que pueda contener todos los puntos que puedan formar parte de ella, haciendo que su detección sea más fiable y robusta. Cuando se pierde una pared, el *buffer* circular vuelve a su tamaño normal.

Existen dos vías para detectar un corredor (figura 5.55). La primera consiste en detectar dos paredes simultáneamente, una por la derecha y otra por la izquierda, con una longitud mínima (más de 20 cm), paralelas (con un margen de 3º) y no demasiado alejadas entre sí (a menos de 4 m). La segunda consiste en detectar una pared también con una longitud mínima (20 cm), paralela a un corredor ya detectado (margen de 2º) y muy próxima a una de sus paredes (menos de 5 cm). En el primer caso se crea un nuevo corredor, mientras que en el segundo se amplía el ya existente.

Cada vez que se crea o amplía un corredor se trata de integrar con los ya existentes. Se funden dos corredores cuando son paralelos (margen de 3º), están próximos

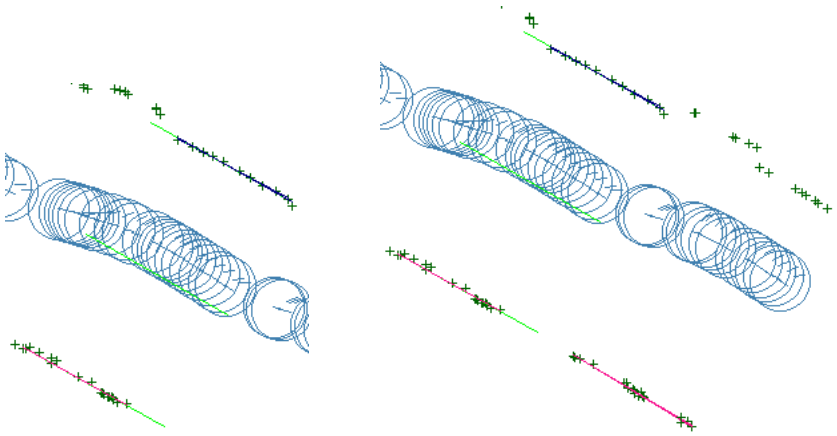

Figura 5.55: Condiciones para que se detecte un corredor: dos paredes paralelas o una pared a continuación de un corredor.

Figura 5.56: Detección de paredes y puertas moviendo el robot en línea recta y girando la torreta a velocidad constante.

(a menos de 1,2 m) y tienen el mismo ancho (margen de 25 cm). El resultado tiene la orientación del corredor detectado en último lugar y su longitud es la máxima distancia entre los extremos de los dos corredores fusionados. Si un corredor nuevo y otro ya detectado sólo difieren en el ángulo, entonces lo más probable es que sean el mismo y que el agente POSICIONADOR_ROBOT deba recalibrar la odometría del robot.

5.14.3 Evaluación

A continuación se muestran algunos de los resultados más representativos en la detección de las marcas a partir de los datos del sensor láser del *Nomad 200*. En la figura 5.56 se muestran las marcas detectadas con el láser en un pasillo cuando la torreta gira a velocidad constante (-15 º/s), mientras el robot avanza despacio (unos 15 cm/s) y en línea recta. Como se puede apreciar, sólo se detectan tramos incompletos de las paredes, haciendo poco fiable la detección de puertas.

En la figura 5.57 el robot recorre el mismo pasillo que en la figura 5.56, pero

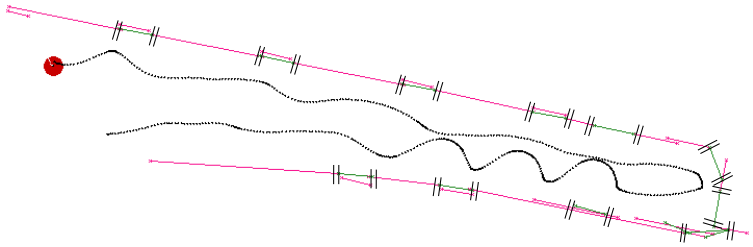

Figura 5.57: Detección de paredes y puertas con los datos del láser, alineando la torreta con el contorno que sigue el robot (el de la derecha).

siguiendo el contorno de su derecha (no demasiado bien en algunos tramos) y alineando la torreta con dicho contorno. Como se puede observar, los resultados son mucho mejores mientras se permite que el robot alcance mayores velocidades medias. La principal dificultad del proceso de detección es que los errores odométricos distorsionan la integración de los datos de diferentes adquisiciones. Un ejemplo de este problema se observa en la figura 5.57, donde no parece que las paredes del corredor sean paralelas.

Para evaluar la capacidad de detección de las puertas del agente DETECTOR_MARCAS hemos hecho que el robot siguiese las paredes de un corredor con 11 puertas hasta completar varias vueltas (4 por la izquierda y 5 por la derecha). De las 99 puertas el agente ha detectado 90 (es decir, una fiabilidad del 90%) y 14 falsos positivos.

Por último, en las figuras 5.58 y 5.59 se muestran las paredes y corredores detectados con los sensores de ultrasonidos mientras el robot recorre un pasillo. El viaje de ida se muestra en la primera figura y en la segunda se muestra la ida y la vuelta. En esta última figura se aprecia fácilmente el error acumulado en la odometría a pesar de lo reducido del recorrido.

5.14.4 Soluciones existentes

No todos los autores consideran necesario utilizar marcas para la tarea de navegación, aunque nosotros consideramos que su uso simplifica la localización, el reposicionamiento y la confección de un mapa global del entorno. También consideramos que las marcas naturales son preferibles a las artificiales porque no se necesita adaptar el entorno para que el robot pueda operar.

A partir de los sensores de US, Konolige (Konolige y Myers, 1998; Konolige,

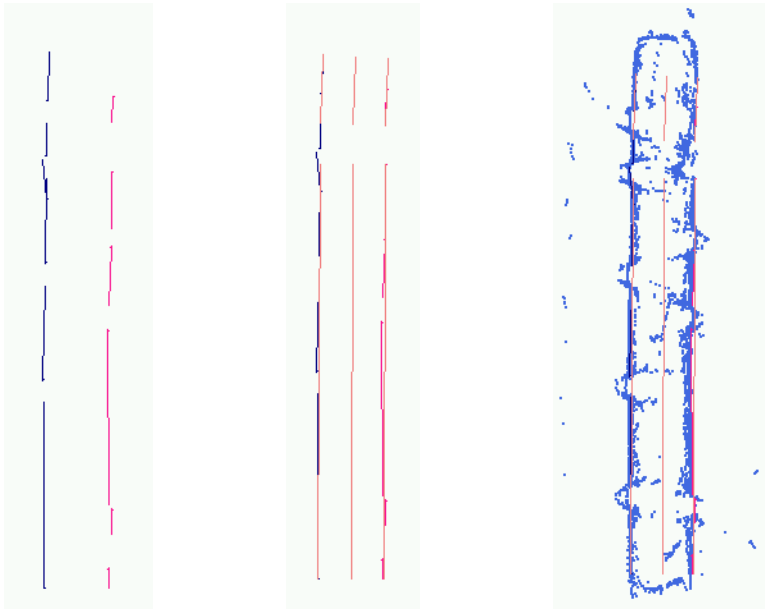

Figura 5.58: Detección de paredes (izquierda) y corredores (centro) a través de los datos de ultrasonidos (derecha).

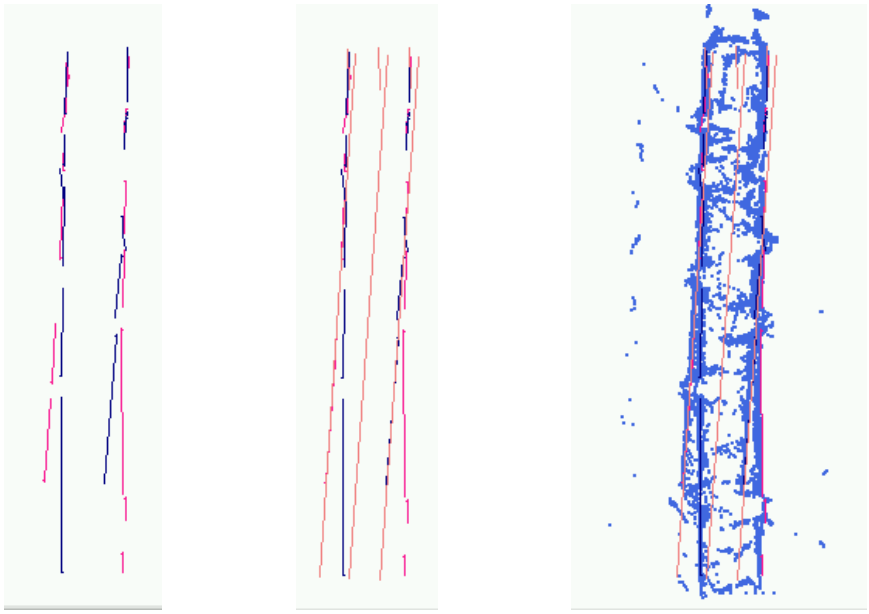

Figura 5.59: Detección de paredes (izquierda) y corredores (centro) a través de los datos de ultrasonidos (derecha). Los mismos datos que en la figura 5.58, pero haciendo el recorrido de ida y vuelta.

1999) es capaz de detectar paredes, corredores, puertas y aberturas (*junctions*). En realidad, nuestros algoritmos para detectar paredes y corredores son muy similares a los suyos. Otros autores (Wijk y Christensen, 2000), mediante técnicas de triangulación entre medidas de US consecutivas, incluso detectan de forma robusta marcas relevantes a nivel local (básicamente esquinas), muy comunes en un gran número de entornos de interiores, puesto que están llenos de muebles, rincones, etc.

Los sensores láser son muy utilizados en la actualidad para la construcción de mapas y para el reposicionamiento del robot debido a su gran precisión, resolución angular, largo alcance y amplio ángulo de percepción (características que no posee el láser de nuestro *Nomad 200*). Algunos de los trabajos más relevantes son los de (Andersson y col., 1999), (Muller y Rodriguez, 1995), (von Puttkamer, 1995), (Reina y Gonzalez, 2000), (Fox y col., 1998) y (Castellanos y Tardós, 1999).

5.14.5 Discusión

Como se ha podido ver, los métodos propuestos son muy simples y robustos, pero tienen ciertas limitaciones. En primer lugar, la reducida área de percepción hace que la detección de marcas con datos del láser sea muy sensible a los errores de la odometría. Errores de unos pocos centímetros se traducen en un fuerte descenso en el rendimiento. Para complicar aún más las cosas, a veces se produce un desfase entre la posición odométrica asociada a la adquisición de los datos del láser y la real, que se hace más apreciable en los giros bruscos de la torreta.

Las limitaciones del láser del *Nomad 200* hace que sea imposible detectar, por ejemplo, una puerta o ambas paredes de un corredor con los datos de una sola adquisición y que las paredes se detecten en tramos muy pequeños. Por ese motivo es imprescindible focalizar la percepción en un único contorno sobre el que se debe mantener alineado el láser en todo momento.

El algoritmo para alinear la torreta funciona bastante bien en los casos en dónde la base tiene aproximadamente la misma orientación que el contorno a detectar (es decir, se supone que el robot sigue dicho contorno u otro paralelo). Si esto no es así, el rendimiento del sistema de alineamiento baja de forma acusada. Lo cual es lógico, porque si la orientación del robot es muy diferente a la del contorno a detectar, entonces pueden surgir serias dudas para identificar el contorno que interesa en los distintos segmentos que percibe el láser.

Por último, la detección mediante sensores de US es muy precisa, teniendo en

cuenta los errores propios de este tipo de sensores. Sin ninguna duda, este éxito se debe a la simplificación que supone el hecho de que el robot esté siguiendo la pared que interesa detectar. Como era de esperar, los ruidos, falsos ecos, objetos en movimiento y otros problemas similares, afectan negativamente a la detección, sobre todo en el ajuste por mínimos cuadrados. Sin embargo, el gran número de medidas y el desplazamiento paralelo a la pared reducen el efecto negativo.

Los errores en la odometría también afectan, pero menos que en la detección con el láser, ya que aunque individualmente cada sensor de US es menos preciso y fiable que el láser, el anillo en su conjunto percibe un porción mayor de las cercanías del robot. Otra dificultad aún no resuelta en la detección de corredores es que un pequeño error en el cálculo de la orientación puede ocasionar un error considerable en la estimación de la posición de sus extremos. Por ejemplo, un error de 1^o en 20 metros se traduce en un desplazamiento de 35 cm.

5.15 Agente Posicionador_Robot

El agente Posicionador_Robot actualiza constantemente la posición y orientación (Pose) real del robot a partir de la posición odométrica calculada por el propio robot, las marcas detectadas y las previamente almacenadas. Determinar con precisión la posición del robot respecto a su entorno es imprescindible para confeccionar un mapa correcto y coherente y para poder usarlo.

El agente Posicionador_Robot parte del supuesto de que la posición odométrica que ha calculado el robot es básicamente correcta, aunque imprecisa y sujeta a múltiples errores, muy difíciles o imposibles de eliminar. Por lo tanto, su objetivo se centra en refinar la Pose calculada de manera que sea coherente con las marcas detectadas en el entorno. Sin embargo, dado que en Piloto sólo se maneja información local, si el error en la posición es excesivo entonces es imposible reposicionar el robot y la localización pasa a ser una responsabilidad del especialista superior, que dispone de toda la información sobre el entorno.

5.15.1 Entradas y salidas

Órdenes de tarea

- **RecalibrarOdometría**, detectar y corregir errores excesivos en la Pose del robot.

Datos de entrada

- ODOMETRÍA
- POSE
- MARCAS del MPL, es decir, las que se han detectado recientemente
- MARCAS del MPLP y que se consideran como las *marcas del entorno*

Datos de salida

- POSE
- comando de movimiento

5.15.2 Síntesis del agente

El sistema odométrico de un robot móvil se encarga de calcular y actualizar una estimación sobre la POSE del robot en un sistema de referencia absoluto. El cálculo se basa en integrar el giro que realiza cada una de las ruedas, de manera que conociendo la estructura del robot (diámetro y posición de cada rueda) se puede deducir con una precisión aceptable el movimiento del robot. Sin embargo, por muy precisas que sean las medidas y los cálculos, los errores cometidos no se pueden corregir y se van acumulando. Los errores también tienen causas externas: el derrapaje de una rueda (se desplaza sin girar), pequeños desniveles (una rueda gira más de lo que se desplaza en el plano), etc.

El resultado final es que la odometría se va haciendo progresivamente menos fiable y, por lo tanto, es necesario diseñar un mecanismo que se pueda utilizar en cualquier momento mientras el robot se está moviendo, y que reduzca o elimine el error cometido. La mejor forma de hacerlo consiste en relacionar la posición odométrica con el entorno en el que se desenvuelve el robot y, más concretamente, con las marcas detectadas. La influencia es mutua, ya que la odometría también sirve para relacionar los datos sensoriales que adquiere el robot en distintas posiciones y permite detectar las marcas del entorno.

En la práctica, el reposicionamiento del robot consiste en recalibrar de forma autónoma y continua la posición odométrica a partir de la información que extrae el robot mientras se mueve. El proceso se divide en las siguientes etapas:

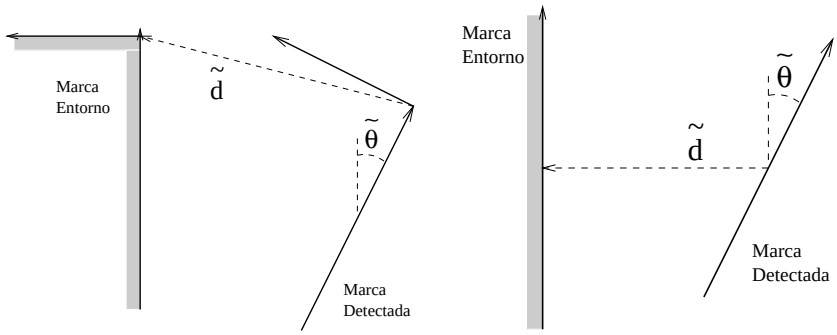

Figura 5.60: Cálculo del error en la posición y orientación del robot según el tipo de marca: a) esquina y b) pared.

1. Posicionamiento del robot comparando las últimas marcas detectadas con las marcas del entorno (MPLP) más próximas.

2. Cálculo del error en la Pose del robot.

3. Si el error sobrepasa los umbrales establecidos:

   (a) Recalibración de la odometría.

   (b) Reposicionamiento de las marcas detectadas en la última adquisición.

Si el error en la Pose sobrepasa los umbrales permitidos, lo más probable es que el robot esté perdido y sea necesario empezar a construir un nuevo mapa mientras se trata de localizar el robot de manera absoluta.

Una dificultad añadida del *Nomad 200* es que posee 4 grados de libertad (x, y, θ_{base}, $\theta_{torreta}$) y no se puede medir de forma directa la diferencia entre el ángulo entre de la base y el de la torreta. Como en la base sólo se montan los sensores táctiles, no es posible recalibrar el ángulo de la base y se hace necesario desarrollar mecanismos específicos para eliminar los errores en su estimación. Curiosamente, no nos consta que ningún usuario del *Nomad 200* haya mencionado este grave problema.

Recalibración de la odometría

El primer paso para recalibrar la odometría es posicionar el robot y para ello se comparan las marcas detectadas con las marcas del entorno. Como las marcas de cada tipo son indistinguibles, se utiliza un criterio de proximidad y se consideran iguales aquellas marcas que están más próximas entre sí.

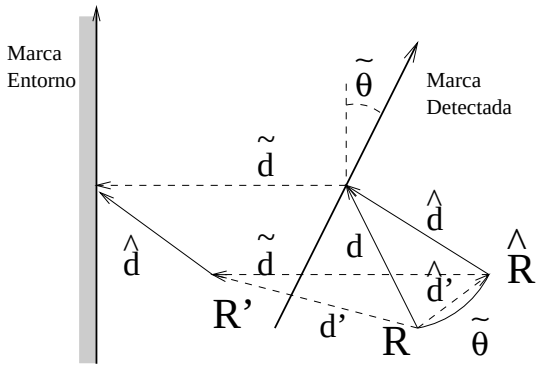

Figura 5.61: Cálculo del vector de desplazamiento para recalibrar la odometría.

La calidad en el posicionamiento del robot depende del número y tipo de marcas detectadas y "mapeadas", es decir, asociadas con una marca del entorno. Así, con una marca de tipo esquina o puerta se puede determinar tanto la orientación como la posición del robot (figura 5.60.a), es decir, tres grados de libertad. Sin embargo, con una marca de tipo pared o corredor sólo se puede calcular la orientación y la posición del robot en la dirección perpendicular a la marca (figura 5.60.b), es decir, dos grados de libertad.

El método de recalibración descrito es genérico, pero se han adoptado dos simplificaciones. La primera es posicionar el robot utilizando únicamente la marca reciente más fiable. La segunda es limitarse a las paredes y corredores del entorno, lo que permite simplificar la construcción y actualización del mapa, manteniendo el robot correctamente posicionado.

A partir de las marcas mapeadas se puede extraer el vector y el ángulo de recalibración (d' y $\widetilde{\theta}$, respectivamente). El ángulo de recalibración ($\widetilde{\theta}$) y la distancia ($\widetilde{d}$) se calculan directamente al mapear una marca (figura 5.60). Para calcular el vector de recalibración d' se debe tener en cuenta que el robot debe ver la marca detectada desde el mismo punto de vista (d) antes y después de la recalibración de la odometría. Podemos deducir d' de la siguiente forma:

$$\begin{aligned} d' &= \widetilde{d} + \widehat{d'} \\ &= \widetilde{d} + d - \widehat{d} \\ &= \widetilde{d} + d - dM_{\widetilde{\theta}} \end{aligned}$$

en donde el significado de las variables intermedias se refleja en la figura 5.61.

Una vez calculado el error en la odometría (d' y $\widetilde{\theta}$) se comprueba si es necesario

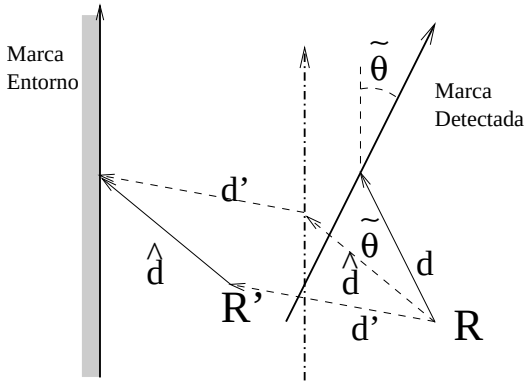

Figura 5.62: Reposicionamiento de una marca después de recalibrar la posición odométrica un ángulo $\widetilde{\theta}$ y un vector d'.

proceder a su recalibración o no. Por un lado, si el error es muy pequeño ($\widetilde{\theta} < 3^o$ y $d' < 10$ cm) la recalibración podría resultar contraproducente sobre todo si el posicionamiento del robot respecto a las marcas no es fiable. Por el contrario, si es demasiado alto ($\widetilde{\theta} > 45^o$ o $d' > 2$ m) lo más probable es que el robot se encuentre perdido y que se necesite localizar el robot a partir de una posición inicial desconocida. La recalibración de la odometría consiste en cambiar el actual sistema de referencia (posición de origen R y orientación θ) por otro nuevo (R', θ'):

$$\begin{aligned} R' &= R + d' \\ \theta' &= \theta + \widetilde{\theta} \end{aligned}$$

Para hacer efectiva la recalibración se calcula la POSE del robot en el nuevo sistema de referencia R' y se envía al robot para que actualice la odometría.

Además de reajustar la posición odométrica, es necesario actualizar la posición de las marcas detectadas (figura 5.62). Teóricamente se deberían mover todas las marcas detectadas desde la última recalibración pero, como resulta muy difícil establecer el deterioro de la odometría durante ese intervalo, se ha optado por reposicionar tan sólo las más recientes.

Recalibración del ángulo entre la base y la torreta

La recalibración de la odometría se basa en la detección de marcas en el entorno. Sin embargo, como en la base sólo hay sensores táctiles, no se puede aplicar para corregir la estimación del ángulo de la base. Por lo tanto, es necesario desarrollar un método específico para detectar y corregir estos errores.

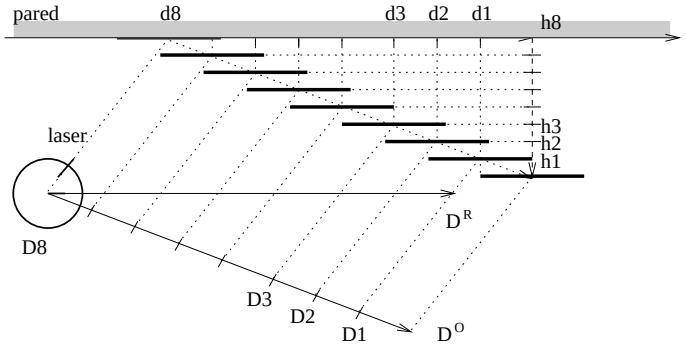

Figura 5.63: Adquisiciones del láser sobre una pared recta larga cuando el ángulo de la base según la odometría no es correcto.

Una forma sencilla y robusta de detectarlos es orientar el láser hacia una pared recta suficientemente larga y mover el robot a lo largo de la misma. Si el ángulo de la base odométrico es correcto, las sucesivas medidas del láser (segmentos) se solapan, puesto que son colineales. Si el ángulo es erróneo, los segmentos no se solapan y forman una estructura característica fácilmente detectable, que consiste en un conjunto de segmentos paralelos separados entre sí una distancia proporcional al desplazamiento del robot entre cada adquisición (figura 5.63).

Por lo tanto, se puede asegurar que existe un error en la estimación del ángulo de la base cuando el cociente entre la distancia perpendicular entre los segmentos de dos adquisiciones sucesivas del láser (h_i) y la distancia recorrida por el robot entre dichas adquisiciones (D_i) es constante, es decir

$$\frac{h_1}{D_1} = \frac{h_2}{D_2} = \frac{h_3}{D_3} = \cdots = \frac{h_n}{D_n} = constante \tag{5.1}$$

Para calcular el error en la estimación odométrica del ángulo de la base (α) vamos a estudiar el ejemplo de la figura 5.64. El error (α) es la diferencia entre el ángulo de la base odométrico (θ^O) y el ángulo de la base real (θ^R):

$$\alpha = \theta^O - \theta^R$$

Para calcular α empezamos calculando el error cometido al estimar la posición del robot, es decir, la diferencia entre la distancia real que se ha desplazado el robot (D^R) y la medida según la odometría (D^O):

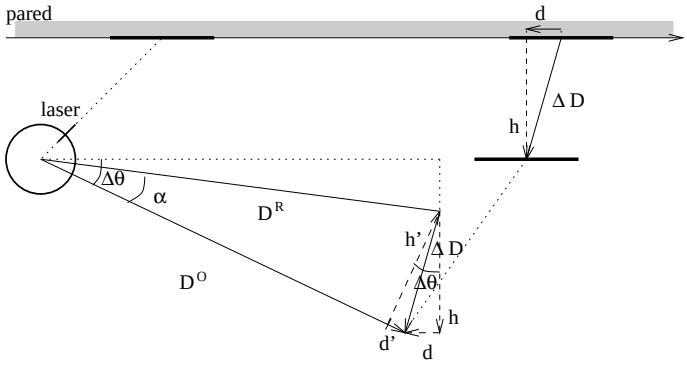

Figura 5.64: Relaciones geométricas entre dos adquisiciones láser de una pared recta cuando el ángulo de la base odométrico está mal calculado.

$$\Delta D \;=\; D^O \;-\; D^R$$

ΔD se puede expresar tanto en el sistema de referencia odométrico (^{O}) como en un sistema de referencia basado en la propia pared detectada (^{P}):

$$\begin{aligned} \Delta D &= (h,\, d)^P \\ &= (h',\, d')^O \end{aligned}$$

En realidad, ambos vectores son el mismo pero girados un ángulo $\Delta\theta$, es decir, la diferencia entre el ángulo de la pared (θ^P) y el ángulo odométrico θ^O:

$$\Delta\theta \;=\; \theta^P \;-\; \theta^O$$

Por lo tanto,

$$\begin{aligned} \begin{pmatrix} h^O \\ d^O \end{pmatrix} &= Giro_{\Delta\theta} \begin{pmatrix} h^P \\ d^P \end{pmatrix} \\ &= \begin{pmatrix} \cos\Delta\theta & \sin\Delta\theta \\ -\sin\Delta\theta & \cos\Delta\theta \end{pmatrix} \begin{pmatrix} h^P \\ d^P \end{pmatrix} \end{aligned}$$

Despejando cada componente del vector, nos queda

$$h^O = d^P \sin\Delta\theta + h^P \cos\Delta\theta \quad (5.2)$$

$$d^O = d^P \cos\Delta\theta - h^P \sin\Delta\theta \quad (5.3)$$

De la figura 5.64 se pueden extraer las relaciones geométricas que ligan el error en la estimación del ángulo de la base (α) y el desplazamiento del robot (D) con las distancias perpendicular y paralela entre dos segmentos que pertenecen a la misma pared expresadas en el sistema de referencia de la odometría del robot (h^O y d^O):

$$D^R \sin\alpha = h^O \quad (5.4)$$

$$D^R \cos\alpha = D^O - d^O \quad (5.5)$$

Si el desplazamiento del robot es relativamente pequeño, se puede asumir que el medido por el sistema odométrico es aproximadamente igual al real:

$$D^R \simeq D^O \quad (5.6)$$

Las últimas cinco ecuaciones se pueden reducir a tan sólo dos:

$$\sin\alpha = \frac{d^P \sin\Delta\theta + h^P \cos\Delta\theta}{D^O} \quad (5.7)$$

$$\cos\alpha = 1 - \frac{d^P \cos\Delta\theta - h^P \sin\Delta\theta}{D^O} \quad (5.8)$$

Despejando sale la expresión para calcular α:[17]

$$\sin(\alpha+\Delta\theta) = \sin\Delta\theta + \frac{h^P}{D^O} \quad (5.9)$$

Para implementar este método se almacenan las últimas medidas del láser (p.e., 6). Con cada nueva adquisición se fija un segmento de referencia (p.e., el último de más de 20 cm) y se seleccionan aquellos segmentos suficientemente largos (20 cm)

[17] α es positivo si los segmentos se acercan al robot y negativo en caso contrario.

que sean paralelos y que estén próximos (unos 20 cm) al de referencia. Para cada segmento seleccionado se mide el desplazamiento realizado por el robot (D^O) y la distancia perpendicular (h^P). Si D^O es significativo (mayor de 2,5 cm) se almacena el cociente $\frac{h^P}{D^O}$en una lista. Existe un error en el ángulo odométrico de la base si hay más de tres cocientes iguales en la lista (con un margen del 10%). Para calcularlo se utiliza la ecuación 5.9 utilizando como h^Py D^O los calculados en primer lugar, es decir, h_1y D_1. Por último, si el α calculado es mayor de 90 grados se utiliza el ángulo complementario.

5.15.3 Evaluación

Recalibración con datos del láser

Para testear el algoritmo de recalibración se recogieron datos con el láser de una sección de la sede del Departamento de Electrónica y Computación. El robot recorrió el rectángulo en el sentido contrario a las agujas del reloj, a una velocidad constante (15 cm/s) haciendo girar la torreta a una velocidad fija (-15 º/s) y girando la base de forma manual. Los resultados obtenidos con y sin recalibración de la odometría se pueden ver en la figura 5.65.[18]

El sistema es tan robusto y consistente que los resultados siguen siendo muy buenos recalibrando sólo el ángulo (figura 5.66) o utilizando corredores un metro más largos que los reales (figura 5.67). Aunque en este último caso se observa claramente el salto que se produce justo a la entrada de cada nuevo corredor en cuanto se detecta una pared fiable para ajustar la odometría.

Recalibración con datos de los US

Para testear el algoritmo de la recalibración de la odometría utilizando datos de US se ha realizado un experimento similar al anterior, pero haciendo que la base y la torreta girasen solidariamente. El experimento se ha hecho dos veces: primero a 15 cm/s y después a 25 cm/s para ver cómo afecta la velocidad. Los resultados se pueden ver en las figuras 5.68 y 5.69, respectivamente.

La recalibración con los datos de US se limita a un máximo de 5º en ángulo y de 20 cm en distancia para disminuir el efecto de los posibles errores en el posicionamiento

[18]Dos tramos de una pared del pasillo inferior son de cristal y no pueden ser detectados por el láser.

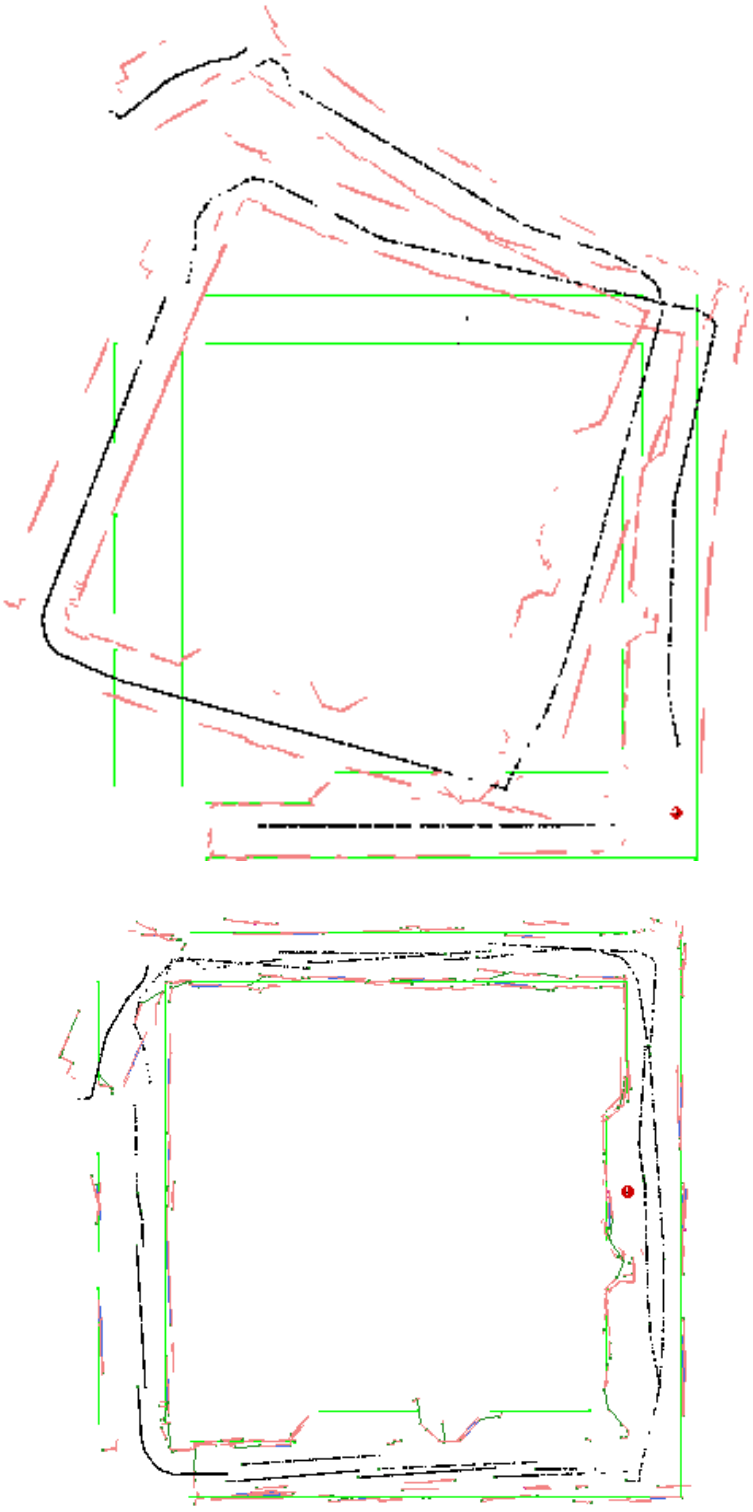

Figura 5.65: Detección de paredes con datos del láser sin (parte superior) y con recalibración de la odometría y girando la torreta a velocidad constante. Las rectas muestran las paredes reales del entorno.

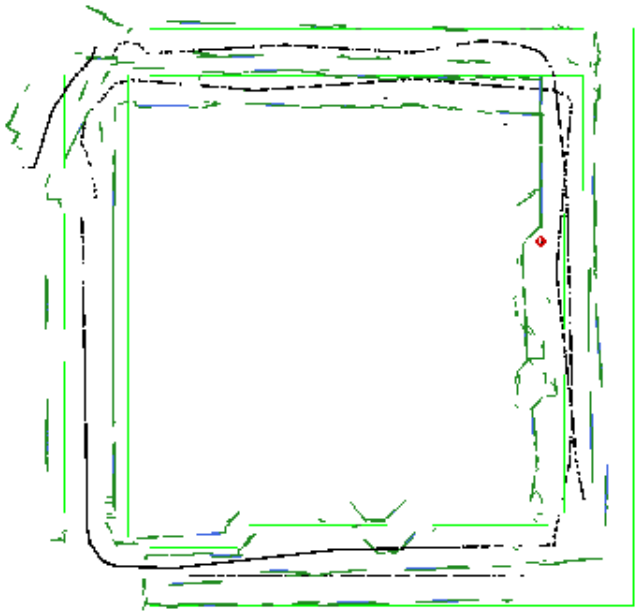

Figura 5.66: Resultado de recalibrar sólo el ángulo de la odometría.

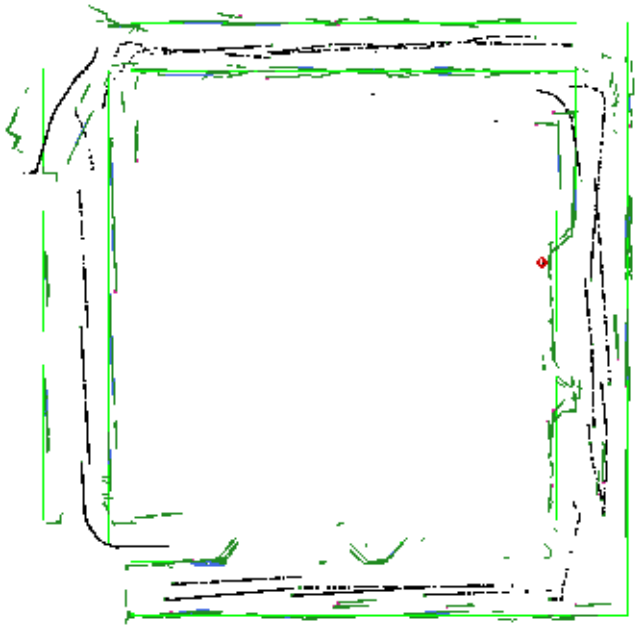

Figura 5.67: Resultado de utilizar corredores un metro más largos que los reales para la recalibración de la odometría.

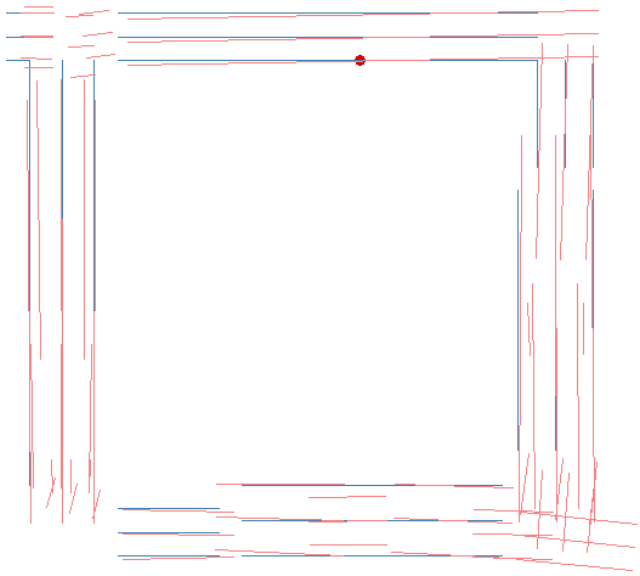

Figura 5.68: Recalibración de la odometría usando datos de ultrasonido y moviendo el robot a unos 15 cm/s.

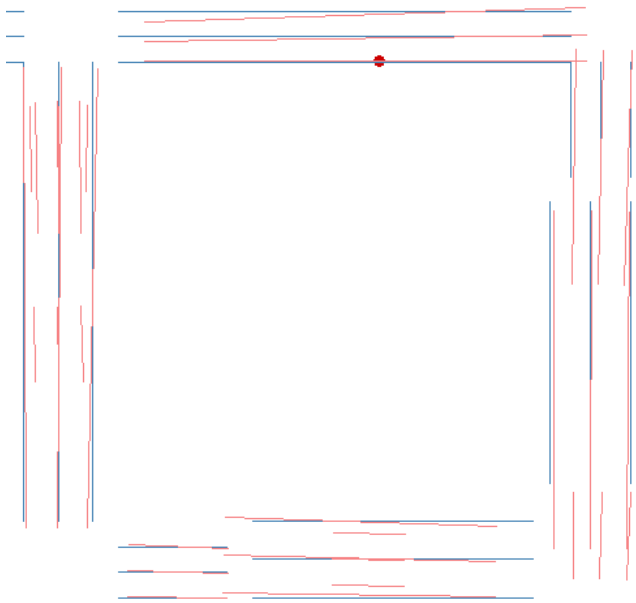

Figura 5.69: Resultado de la recalibración de la odometría utilizando únicamente datos de ultrasonidos, moviendo el robot a unos 25 cm/s.

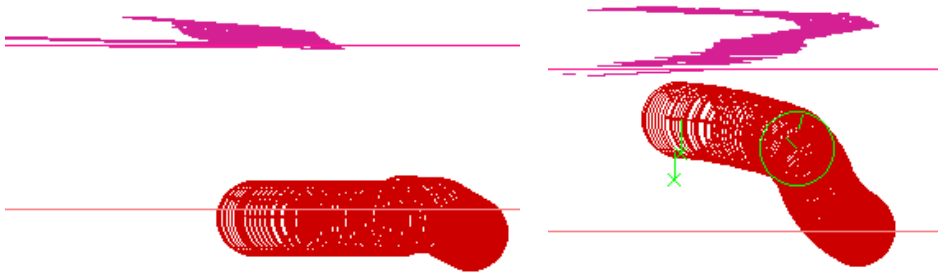

Figura 5.70: Recalibración del ángulo de la base partiendo de un error de 30º: a) movimiento rectilíneo y b) siguiendo el contorno de la derecha.

del robot y en el mapeado de las marcas (aspectos muy problemáticos cuando se utilizan los datos del láser). Además, si el robot sigue detectando la misma pared o corredor entonces las recalibraciones se realizan en cascada (una en cada nueva adquisición sensorial) hasta localizar correctamente al robot. El principal riesgo de esta recalibración progresiva es que puede perturbar la detección de marcas con el sensor láser.

Para comprobar el método de recalibración basado en US se ha realizado un experimento que consiste en hacer que el robot recorra un corredor rectangular conocido siguiendo una pared, primero en un sentido y después en el contrario. La precisión de la recalibración se calcula midiendo el error en la posición de algunos puntos del entorno (p.e., esquinas) a partir de los datos del láser. Después de recorrer 12 vueltas y 500 metros, el máximo error cometido fue de 73 cm en el eje longitudinal del corredor y 30 cm en el perpendicular.

Recalibración del ángulo entre la base y la torreta

Para comprobar el funcionamiento del algoritmo se ha situado el robot en un pasillo con el láser apuntando a una pared y un ángulo de la base incorrecto. Una vez que el robot se empieza a desplazar y a tomar medidas del láser, el sistema detecta el desajuste y lo corrige en menos de una decena de iteraciones (figura 5.70). Si la percepción del láser no es la apropiada, el mecanismo deja de funcionar. Por ejemplo, si no existen medidas (p.e., en espacios amplios o cuándo un objeto obstaculiza la visión de la cámara), si se detectan sólo segmentos pequeños (p.e., en entornos muy irregulares o con objetos en movimiento) o si el robot no se desplaza lo suficiente (p.e., sólo gira).

5.15.4 Soluciones existentes

Existen diferentes métodos para reposicionar el robot según el tipo de navegación utilizada: geométrica o topológica. Los primeros comparan las medidas sensoriales con el mapa métrico de la región en la que se encuentra el robot, mientras que en los segundos sólo se comparan las marcas detectadas con las marcas almacenadas del entorno. La principal ventaja de los métodos topológicos es que los cálculos se simplifican significativamente, al operar sobre un conjunto de datos más reducido y porque el problema de la recalibración se puede plantear y resolver de forma analítica. Su principal limitación es la fuerte dependencia con el sistema de detección de marcas.

En la navegación topológica se suelen utilizar los mismos tipos de marcas (paredes, corredores y puertas). Las diferencias surgen en el modo de representar y calcular los errores odométricos. Se aplican técnicas de lógica borrosa (Saffiotti y Wesley, 1996; Saffiotti, 1996; Gasós y Rosetti, 1999; Demirli y Türkçen, 2000), de la teoría de la evidencia (Duckett y Nehmzow, 1998), sistemas de votaciones (Reina y Gonzalez, 2000) o métodos heurísticos (Dedeoglu y Sukhatme, 2000).

Una técnica muy robusta es la de (Wijk y Christensen, 2000), basada en la detección de esquinas a partir de los datos de US y que permite no sólo la recalibración de la odometría del robot, sino que también permite su localización de forma absoluta, es decir, sin tener conocimiento de la posición inicial del robot. Por otra parte, algunos autores (Bauer, 1995) incluso han estudiado el problema de la localización activa, es decir, el compromiso entre el coste de realizar un determinado movimiento o desplazamiento para mejorar la detección de una marca y el beneficio que se obtiene.

Por último, mencionar que también existen sistemas de reposicionamiento que se basan en asociar o identificar cada lugar del entorno con la información sensorial que recibe el robot en ese punto. La asociación se realiza a partir de una red neuronal artificial (Lemon y Nehmzow, 1998; Nehmzow, 2000; Nehmzow y Owen, 2000) o mediante Procesos de Decisión de Markov Parcialmente Observables (Simmons y Koenig, 1995). En general, estos métodos son más bien cualitativos y su precisión no es muy elevada. Los principales problemas de estos métodos son el *perceptual aliasing*, es decir, lugares distintos que no pueden ser diferenciados por las medidas sensoriales y que las implementaciones actuales aún no parecen muy escalables. En (Gutmann y col., 1998) y (Duckett y Nehmzow, 2000) se comparan algunas de las técnicas de localización topológica más importantes.

5.15.5 Discusión

La recalibración de la odometría en los corredores es más robusta con los datos de US que con los del láser, porque los primeros detectan las dos paredes del corredor mientras que el láser sólo detecta una. Por lo tanto, es más difícil confundirse y el posicionamiento es más preciso, incluso aunque un sensor de US tenga mucha menos precisión que el láser. De esta manera, el sistema es más fiable en los pasillos, es decir, donde más se necesita, puesto que son las regiones más transitadas por el robot.

Por el contrario, en regiones con menos espacio libre (p.e., laboratorios y despachos) suele ser más fiable la recalibración con el láser. Principalmente porque el láser puede detectar las marcas de una pequeña región con mucha precisión y con sólo desplazarse y realizar barridos con la torreta en unos pocos puntos. Sin embargo, en estos entornos el principal problema es que no haya marcas (p.e., en entornos muy densos y cambiantes) o que no sean reconocibles (p.e., por estar ocultas detrás de otros objetos).

Por último, conviene recordar, y tener siempre presente, que ningún método de reposicionamiento y recalibración es infalible, porque se basa en información local, es decir, parcial del entorno. Cuando faltan datos o éstos no son coherentes, el problema de la localización del robot se debe resolver en instancias superiores y utilizando toda la información sobre el entorno disponible.[19]

[19]En nuestra arquitectura ésta es una tarea del agente Localizador dentro del especialista Navegador.

Capítulo 6

Validación experimental

6.1 Validación

Al igual que en cualquier sistema software, la validación de una arquitectura de control es fundamental en el ciclo de desarrollo. En realidad, en la bibliografía de la ingeniería del software se distingue entre *validación* y *verificación* (VyV) aunque, en general, no queda clara la diferencia entre ambos términos. Según (Vermesan, 1998) la verificación es el proceso que consiste en comprobar que el sistema software ha sido construido de acuerdo con sus especificaciones. Por otra parte, la validación consiste en comprobar que el sistema cumple con los requerimientos impuestos por el usuario.

La verificación sólo tiene sentido cuando se realiza la transición *entre* dos niveles consecutivos en la etapa de desarrollo del sistema software, según éste se va concretando y particularizando (figura 6.1.(a)). Por otro lado, la validación se lleva a cabo *dentro* de cada nivel o etapa y su objetivo consiste en comprobar que dicho nivel cumple adecuadamente todas las funciones para las que ha sido diseñado y posee todas las propiedades previstas (figura 6.1.(b)).

A diferencia de la mayoría de las aplicaciones software, una arquitectura de control para robótica móvil es muy difícil de validar. La razón principal es que, como se ha dicho con insistencia, tanto el robot como el entorno en el que éste se desenvuelve son sistemas muy complejos, dinámicos y, en general, no totalmente predecibles. La falta de un modelo teórico completo dificulta la utilización de herramientas y métodos analíticos en el estudio de las propiedades de estos sistemas. Una de las principales consecuencias es que no es posible establecer *a priori* todas las propie-

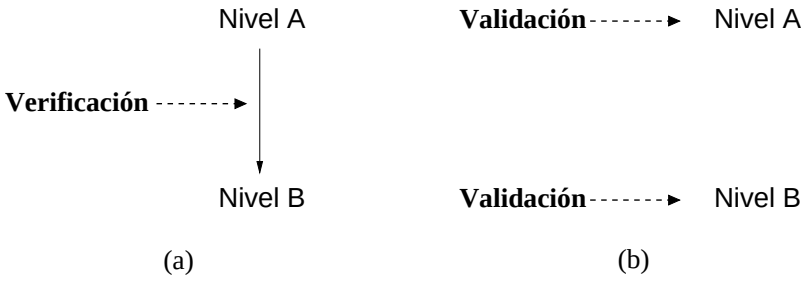

Figura 6.1: Ilustración de los conceptos de validación y verificación (Lama, 2000).

dades que debe cumplir el sistema. Como mucho se pueden indicar una serie de propiedades y requisitos generales que debe cumplir una arquitectura de control para robótica y con los cuales se espera que ésta pueda realizar correctamente su función. Obviamente, es posible verificar que en cada una de las sucesivas etapas del desarrollo se mantienen todas y cada una de las propiedades y funcionalidades del sistema. Sin embargo, si no es posible demostrar que el planteamiento inicial es correcto la verificación no aportará nada nuevo.

La única forma práctica de validar una arquitectura de control y demostrar que es viable consiste en desarrollar un prototipo y comprobar empíricamente si cumple todas las propiedades que habían sido previstas inicialmente. Sin embargo, la validación no se puede considerar como una prueba final y definitiva del modelo de arquitectura propuesta, ya que, no lo olvidemos, se basa en una implementación particular sobre una plataforma específica, ejecutando una tarea determinada en un entorno relativamente limitado. Sin embargo, muchas veces es el único modo en el que se puede demostrar que una arquitectura es en principio válida. Cuantas más implementaciones existan y cuanto más diferentes sean los dominios, los robots móviles y las tareas sobre las que se apliquen, mejor y más completa resulta la validación. La validación experimental es el método más ampliamente utilizado en robótica.

Sin embargo, algunos aspectos concretos de una arquitectura de control pueden ser validados y verificados mediante métodos analíticos más formales. La sistematización del proceso de verificación permite que el análisis pueda ser accesible a no especialistas y pueda ser integrado en un entorno de programación completo. Las arquitecturas de control pueden facilitar el modelado de dos maneras (Coste-Manière y Simmons, 2000): si su comportamiento puede ser modelado formalmente o si proporcionan lenguajes que puedan ser restringidos de manera que la verificación formal pueda ser tratable. Un ejemplo serían las arquitecturas que representan

los comportamientos mediante autómatas de estados finitos (Brooks, 1986).

En (Medeiros y col., 1996) se indica que una forma de extraer o deducir las propiedades que posee el conocimiento de control en su sistema basado en *Procedural Reasoning System* (PRS) es representarlo mediante redes de Petri coloreadas. En una red de Petri se puede demostrar que el sistema no posee bucles infinitos, situaciones de bloqueo, etc. También se pueden deducir otras propiedades de interés, como la alcanzabilidad, tiempo de ejecución, etc., que, salvo ciertos matices, pueden ser aplicables al sistema original, es decir, a la arquitectura de control. Sin embargo, debido a las limitaciones que impone el formalismo de las redes de Petri coloreadas, esta traducción sólo ha sido posible en sistemas PRS con una sintaxis restringida.

Las arquitecturas también pueden incluir mecanismos que faciliten el proceso de validación y depuración. Uno de ellos es registrar los eventos significativos del sistema sin que ello afecte a sus prestaciones de funcionamiento en tiempo real. Por ejemplo, la arquitectura TCA (Simmons, 1994) permite la habilitación *on-line* del registro de todo el tráfico de mensajes. Otro aspecto importante es que exista la posibilidad de validar o testear de forma independiente cada uno de los componentes antes de que el sistema se haya completado (Coste-Manière y Simmons, 2000). De todos modos, como la respuesta de un componente depende del resto del sistema, lo usual es reemplazar los componentes que no están siendo testeados, bien por otros módulos más simples pero con la misma funcionalidad, bien con un simulador.

Bajo esta perspectiva, y dado que la arquitectura AFE-Robótica es altamente modular, cada uno de sus componentes ha sido testeado de forma exhaustiva e independiente de la arquitectura después de su implementación. Los datos más relevantes de las validaciones realizadas ya se han incluido al describir y comentar cada uno de los agentes implementados. Por este motivo, en el presente capítulo nos vamos a centrar principalmente en estudiar el comportamiento global e integrado de toda la arquitectura, más que en una o varias de sus componentes por separado.

6.1.1 Alcance de la validación experimental

En una validación se estudia no sólo el modelo de arquitectura sino también su aplicación a una tarea concreta y la posterior implementación. Este hecho delimita el alcance de la validación experimental y añade una mayor dificultad a la hora de extraer resultados y conclusiones, ya que es necesario establecer si las propiedades y los defectos observados se deben a la propia implementación o son parte consustancial de la arquitectura. Además, el rendimiento final del sistema depende también

de otras variables: el tipo de robot móvil (es decir, sus capacidades y limitaciones sensoriales y motoras), el entorno, las distintas decisiones y compromisos mantenidos durante la implementación de la tarea, etc. Por lo tanto, la evaluación se debe plantear en tres niveles diferentes, pero muy estrechamente relacionados entre sí.

En primer lugar, el estudio de la propia **arquitectura de control** y, más concretamente, las facilidades que aporta para el diseño, la implementación y la ejecución de la tarea encomendada (en esta memoria, la tarea de navegación) en el ámbito elegido (entornos típicos de oficinas) y en una plataforma particular (un robot móvil *Nomad 200*). En este apartado también se deben incluir las restricciones o limitaciones que impone la arquitectura en todo el proceso.

Por ejemplo, se debe comprobar si es posible ejecutar tareas en tiempo real, si las comunicaciones dentro de la arquitectura son suficientemente flexibles y rápidas, si la jerarquización de tareas que impone AFE-Robótica es ventajosa, si las restricciones que impone dicha jerarquización son aceptables y no demasiado traumáticas, si los mecanismos de reactividad y los intercambios de datos entre los especialistas funcionan correctamente, etc.

En segundo lugar, la influencia de la **tarea seleccionada** (en nuestro caso, la navegación) en la instanciación del modelo de arquitectura propuesto y en el posterior rendimiento del sistema. Al analizar los resultados se deben tener en cuenta las ventajas y desventajas del tipo de navegación elegido y cómo ha podido influir (positiva o negativamente) la división de tareas realizada en las propiedades del sistema final obtenido (robustez, fiabilidad, eficiencia, modularidad, etc.).

En nuestro caso particular se ha seleccionado una navegación topológica basada en marcas pero con información métrica adicional y privilegiando los comportamientos basados en los datos sensoriales respecto a aquellos que se basan en la odometría del robot. También se ha hecho especial hincapié en utilizar marcas naturales del entorno para fijar la posición del robot e indicar las condiciones de finalización de los comportamientos. Por último, se ha supuesto que se conoce *a priori* el mapa topológico del entorno y cómo se traduce la posición del robot a un nodo de dicho mapa.

En tercer lugar, el análisis de la **implementación** concreta de cada una de las tareas y funciones junto con el conocimiento necesario para que todo el sistema sea operativo: el tipo de datos y de eventos a utilizar, la definición de los diferentes planes de control, los algoritmos de percepción, los distintos comportamientos, etc. En el nivel de implementación es donde se hacen más patentes tanto las simplifica-

ciones como las restricciones ligadas a la plataforma física y al medio en donde se desenvuelve el robot.

6.2 Experimentos

La validación de la arquitectura AFE-Robótica se ha realizado sobre la ejecución de una tarea compleja, concretamente, la planificación y ejecución de una ruta a un destino, es decir, "**Ir A** *destino*". Esta es la tarea más general en un sistema de navegación y prácticamente incluye a todas las demás: localización, planificación de rutas, ejecución de rutas y resolución de contingencias. Por tanto, pone en juego la máxima capacidad y funcionalidad del sistema y conlleva el mayor grado de exigencia tanto del especialista NAVEGADOR como, en definitiva, de toda la arquitectura.

Los primeros experimentos de validación se han realizado sobre trayectos cortos, es decir, aquellos que prácticamente se reducen a cruzar una o dos puertas y a seguir un par de corredores. Aprovecharemos una de estas pruebas para explicar en detalle el funcionamiento del especialista NAVEGADOR y de su agente/especialista PILOTO. También nos servirá para mostrar cómo se activan los eventos en AFE-Robótica y cómo se generan las respuestas reactivas correspondientes. A continuación, se muestra y analiza la eficiencia global del sistema sobre trayectos más largos y se comentan las contingencias más comunes en la ejecución de esta tarea y cómo se resuelven.

6.2.1 Ejecución de una tarea en NAVEGADOR

En esta sección vamos a mostrar sólo un ejemplo representativo del funcionamiento de AFE-Robótica cuando ejecuta un desplazamiento corto. En la figura 6.2 se muestra una vista parcial de la sede del Departamento de Electrónica y Computación de la Universidad de Santiago de Compostela, en donde se observa la posición inicial del robot (el laboratorio de robótica o habitación 14). El usuario envía la orden "**Ir A** *Habitación 17*" (OT_1) al especialista NAVEGADOR a través del especialista DIÁLOGO, que, en la presente implementación de AFE-Robótica, además de encargarse de las funciones de interfaz con el usuario también es el especialista superior de NAVEGADOR.

A esta tarea el agente de control de NAVEGADOR le asoció el plan P_1 para ejecutar órdenes del tipo **Ir A**. Como ya se ha comentado en el capítulo anterior,

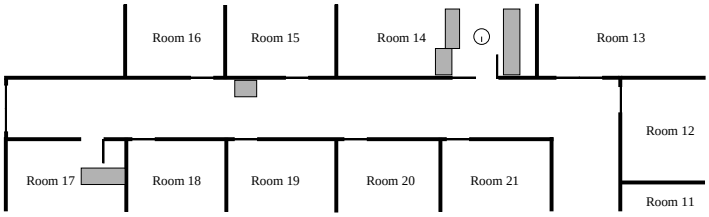

Figura 6.2: Vista parcial de la sede del Departamento de Electrónica y Computación. El círculo representa el robot mientras que los obstáculos se muestran en gris.

1. `Localizar robot`
2. `Planificar ruta al` *`destino`* `deseado`
3. `Traducir y ejecutar cada sección de la ruta`
4. `Si robot no está en` *`destino`* `, volver a 2`

Figura 6.3: Etapas en las que se divide la tarea **Ir A** *destino*.

este plan puede ser dividido en varias etapas, que resumimos de nuevo en la figura 6.3. La primera consiste en localizar el robot, para lo cual el agente de control de NAVEGADOR necesita ejecutar dos tareas. En la primera se ordena al agente PILOTO (OT_1^1 en la figura 6.4) que almacene en la pizarra de NAVEGADOR los detalles relevantes del entorno inmediato del robot (MAPA PERCEPTUAL LOCAL), es decir, marcas, obstáculos y posición del robot. A continuación, el control ordena al agente LOCALIZADOR (OT_1^2) que en base a dicha información determine y almacene en su pizarra el LUGAR del entorno donde es más probable que se encuentre el robot (en este caso la habitación 14).

Sabiendo dónde estaba el robot y cuál era el destino pedido por el usuario, el control de NAVEGADOR tenía todos los datos importantes de la ruta y ordenó al agente PLANIFICADOR_RUTAS (OT_1^3) que calculase el mejor camino para el desplazamiento requerido, utilizando para ello el mapa sobre el entorno (MAPA DISTRIBUIDO GLOBAL) que se encuentra en la pizarra de NAVEGADOR. Como dicho mapa es de tipo topológico (un grafo), la ruta resultante es un conjunto de nodos ordenados que minimiza un criterio (básicamente la distancia). La ruta calculada se almacenó en la pizarra de NAVEGADOR junto con su origen (nodo *H14*) y su destino

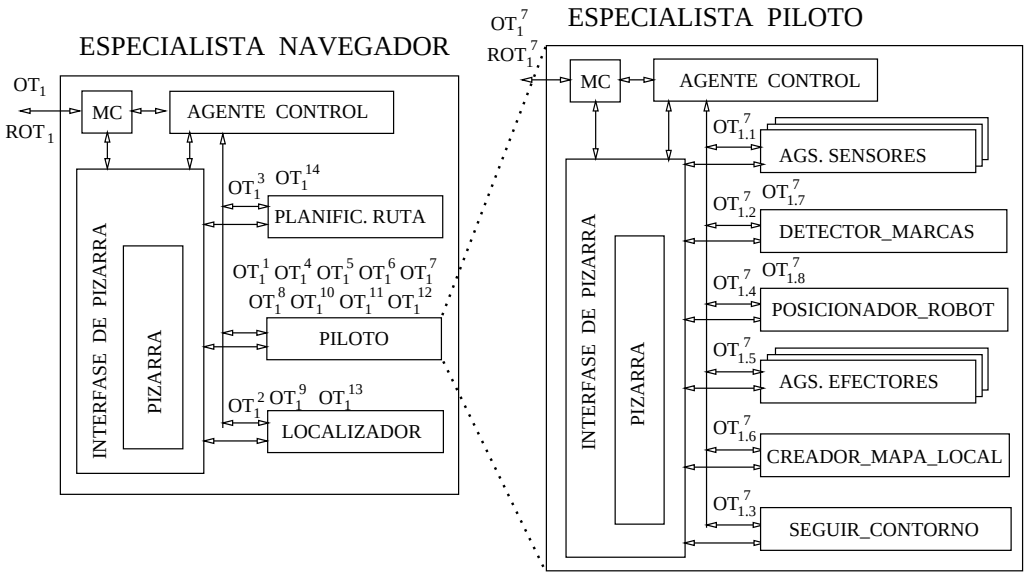

Figura 6.4: Órdenes de Tarea en la ejecución de una tarea de desplazamiento en AFE-Robótica.

(nodo *H17*). En la figura 6.5 podemos ver cuál fue esa ruta.

Después de recibir la respuesta ROT_1^3 el agente de control pasó a la tercera fase del plan P_1, la más compleja y delicada de todas. Como ya se ha explicado al describir el especialista NAVEGADOR, el agente de control "traduce" la primera sección a una o varias órdenes de tarea que sus agentes puedan ejecutar. Si la ejecución es adecuada la siguiente sección es traducida y el proceso se repite hasta que la ruta se completa.

En este ejemplo, el primer tramo consistía en cruzar la puerta de acceso a la habitación 14. Cuando la ruta pasa por una puerta el agente de control de NAVEGADOR primero comprueba la localización del robot y, si ésta es la esperada, entonces se plantea cruzarla. Sin embargo, como en este caso la puerta coincidía con el inicio de la ruta el robot, éste ya está localizado y se ha obviado esta parte.

En general, para cruzar una puerta el control de NAVEGADOR envía una primera orden (OT_1^4) a su agente PILOTO para que gire la torreta del robot una vuelta completa, detectando todas las puertas próximas. El objetivo es determinar de forma precisa la posición y orientación de todas las puertas cercanas al robot. La segunda tarea (OT_1^5) también se envía a PILOTO y en ella se le pide seleccionar y cruzar la puerta más cercana o, de una forma más explícita, la puerta más cercana en una determinada dirección (en este caso aquella que lleve al corredor *C3*, es decir,

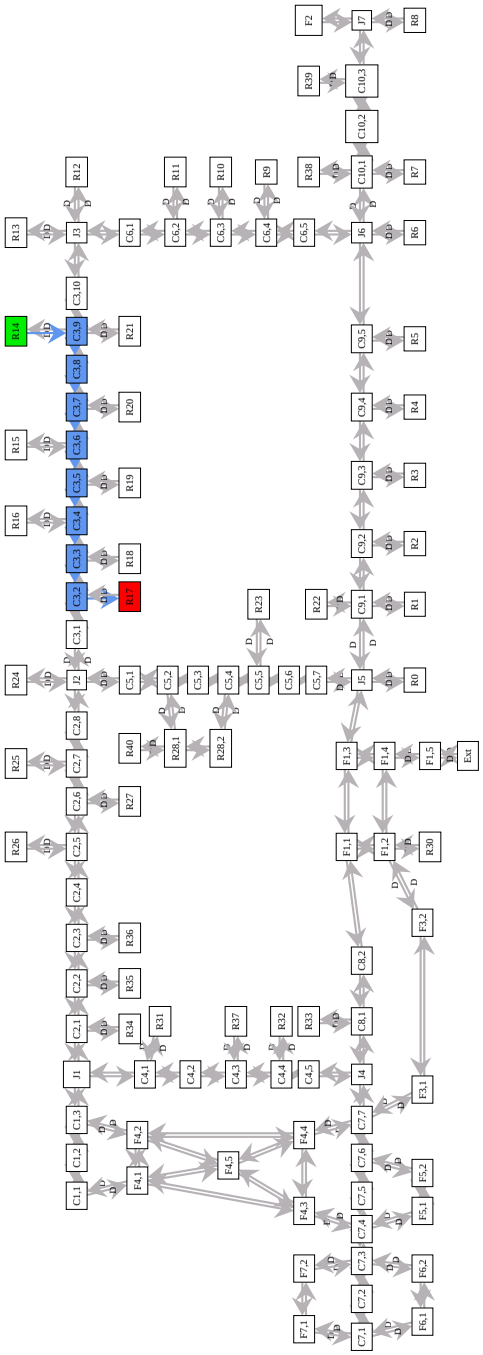

Figura 6.5: Ruta encontrada entre los nodos *H14* y *H17* marcada en el mapa topológico de la sede del Departamento de Electrónica y Computación.

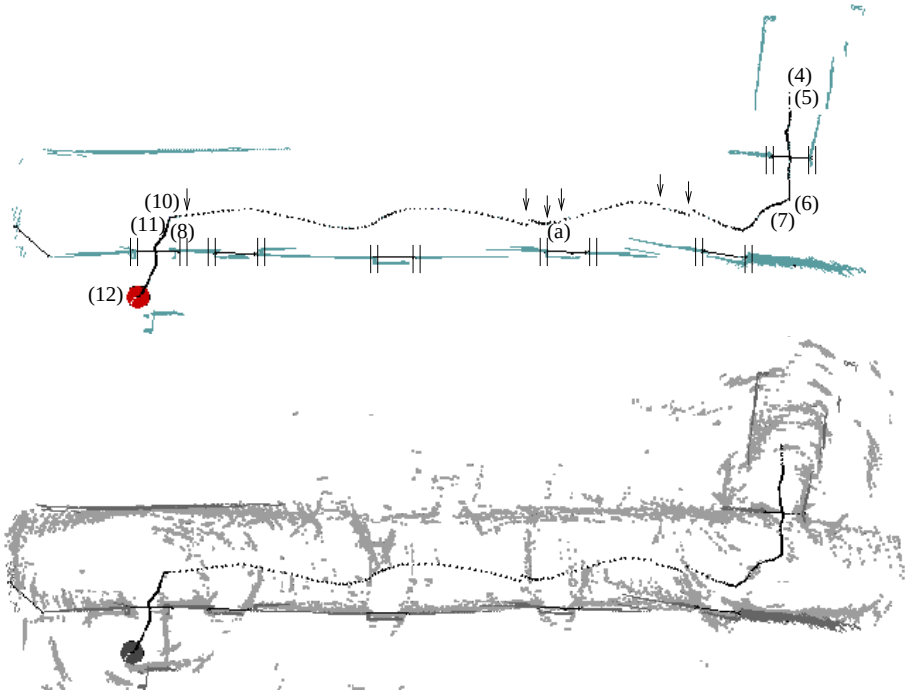

Figura 6.6: Trayectoria del robot al ejecutar la orden "**Ir A** *Habitación 17*". En la parte superior sólo se muestran los datos del láser y las puertas detectadas. En la parte inferior también los datos de los ultrasonidos. Las flechas muestran las recalibraciones del sistema odométrico y los números las órdenes de tarea en la ejecución de la ruta.

-90 grados). Para mayor seguridad el control podría comprobar si la posición del robot después de cruzar la puerta es la prevista. En este ejemplo no se ha hecho así porque según el mapa topológico no había posibilidad de confusión ni de error.

En el siguiente tramo el control de NAVEGADOR agrupó todos los nodos de la ruta que discurrían por el corredor *C3* e hizo que el robot los recorriese siguiendo un contorno. Como al final de este segundo tramo el robot tenía que cruzar una puerta (la de acceso a la habitación 17) se eligió detectar las puertas y seguir el contorno por el lado izquierdo. Por último, como condición de finalización se incluyó, además de la distancia que suman todas las conexiones del tramo, detectar la cuarta puerta a la izquierda. Sin embargo, antes de enviar esta orden se pidió al agente PILOTO que alinease la base del robot con el corredor que después se iba a seguir y la torreta con el lado en el que se iban a detectar las puertas. Esto supuso para el robot girar a la derecha 90 grados la base y 45 grados la torreta (OT_1^6). Una vez alineado correctamente el robot, se envió a PILOTO la orden para seguir el pasillo (OT_1^7).

TO_1^4	**Girar Torreta** *1.1 Derecha*	
TO_1^5	**Cruzar Puerta** *Cercana* **Ángulo** *-90*	
TO_1^6	**Mover A Ángulo** *90 45 Derecha*	
TO_1^7	**Seguir Contorno** *Izquierda* **Hasta**	*Puerta 4 Izquierda*
		O *Distancia 1310*
TO_1^8	**Pedir Mapa_Perceptual_Local**	
TO_1^9	**Localizar Robot** *C3,2*	
TO_1^{10}	**Girar Torreta** *1.1 Derecha*	
TO_1^{11}	**Cruzar Puerta** *Cercana* **Ángulo** *-90*	

Figura 6.7: Secuencia de tareas generadas por NAVEGADOR para ejecutar la ruta mostrada en la figura 6.6.

En la figura 6.6 se muestra la trayectoria que siguió el robot para ejecutar la tarea OT_1. La primera puerta del corredor después de salir de la habitación 14 se detectó mientras el robot se estaba alineando con el pasillo y antes de que PILOTO ejecutase la orden OT_1^7. Como se puede comprobar, en este trayecto el robot detectó sin ningún error las cuatro puertas con las que se cruzó, por lo que la tarea OT_1^7 finalizó correctamente. El último tramo de la ruta era cruzar la puerta de acceso a la habitación 17 y se siguieron los pasos ya comentados. Primero, el control de NAVEGADOR comprobó si el robot estaba frente a la habitación 17 (nodo *C3,2*), es decir, pidió a PILOTO (OT_1^8) que almacenase el MAPA PERCEPTUAL LOCAL en la pizarra de NAVEGADOR, ordenó a LOCALIZADOR (OT_1^9) que localizase el robot y después comprobó si coincidía con el lugar previsto.

Como el robot estaba perfectamente situado, el control se dispuso a cruzar la puerta. En primer lugar, obligó a PILOTO a que confirmara la posición de la puerta haciendo que el robot girase de nuevo la torreta una vuelta completa, detectando todas las puertas cercanas (OT_1^{10}). Este paso se puede obviar cuando PILOTO ha detectado todas las puertas en la última orden ejecutada, como así ha ocurrido en este ejemplo. A continuación, el control de NAVEGADOR ordenó de nuevo a PILOTO (OT_1^{11}) que seleccionase y cruzase la puerta más cercana en la orientación apropiada (-90 grados). En la figura 6.7 se han resumido las órdenes de tarea que el control de NAVEGADOR envió para ejecutar la ruta calculada (figura 6.5) mientras que en la figura 6.4 se muestra a qué agente se dirigió cada una.

Finalizada la ejecución de la ruta el control de NAVEGADOR pasó a la cuarta fase del plan P_1 (figura 6.3), es decir, confirmar si el robot ha llegado al destino. Como

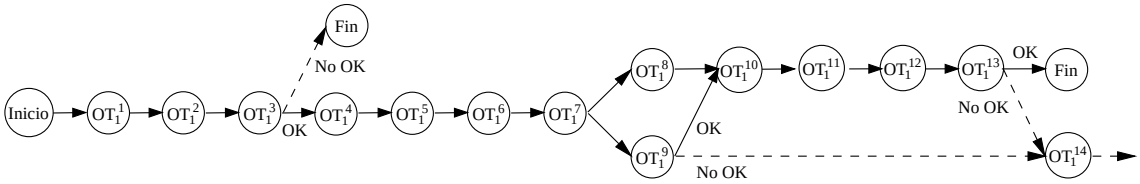

Figura 6.8: Instanciación del plan asociado en NAVEGADOR a "**Ir A** *destino*" para la secuencia de tareas mostrada en la figura 6.7.

siempre, el primer paso fue ordenarle a PILOTO (OT_1^{12}) que actualizase en la pizarra de NAVEGADOR el MAPA PERCEPTUAL LOCAL y después pedirle a LOCALIZADOR (OT_1^{13}) que determinase el lugar dónde se encontraba el robot. Como el robot llegó al destino ordenado por el usuario (H17), el plan P_1 finalizó y NAVEGADOR envió a su especialista superior (DIÁLOGO en esta implementación de AFE-Robótica) una respuesta, indicando que la ejecución de la tarea fue correcta (ROT_1).

El control de NAVEGADOR ejecuta los planes dinámicamente y si una OT no se ejecuta correctamente, el agente de control tendría que redirigir la situación. Por ejemplo, si cuando se chequea la posición del robot ésta no coincide con la prevista, entonces la ruta en progreso se hubiese abortado, replanificándose un nueva ruta desde el LUGAR que realmente ocupara el robot (OT_1^{14}) y repitiéndose de nuevo el proceso de ejecución de una ruta, pero sobre la nueva ruta calculada. Esta posibilidad ya se contempla en el plan P_1 y se muestra con líneas punteadas en la figura 6.8 sobre la instanciación del plan finalmente ejecutada. De igual modo, si en algún momento se concluye que el desplazamiento requerido no puede ser ejecutado (por ejemplo, no existe una ruta factible hacia el destino o el número de intentos ya es demasiado alto) entonces el agente de control aborta la ejecución del plan y envía el motivo de la cancelación al especialista superior.

6.2.2 Ejecución de una tarea en PILOTO

Una vez estudiado el comportamiento general del especialista NAVEGADOR, vamos a centrarnos en el funcionamiento de su agente PILOTO, el más relevante de todos, debido a las restricciones y limitaciones que impone el hecho de ser el encargado de interaccionar con el entorno a través de la plataforma física del robot. Concretamente, nos vamos a centrar en la orden OT_1^7 "**Seguir Contorno** *Izquierda* **Hasta** *Puerta 4 Izquierda* **O** *Distancia 1310*" del ejemplo de la sección anterior.

En el capítulo anterior ya se ha comentado que los requisitos y restricciones en

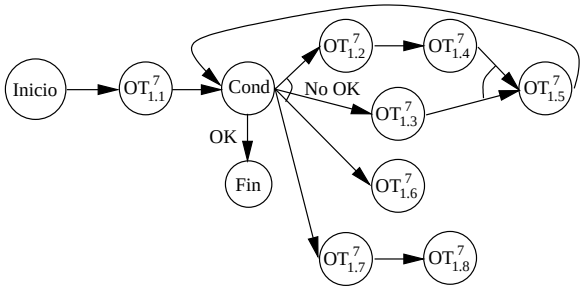

Figura 6.9: Plan P_1^7 asociado a OT_1^7 en PILOTO.

PILOTO son muy fuertes y no cambian excesivamente entre una tarea y otra, por lo que todas ellas se ejecutan siguiendo un plan muy similar (denominado ciclo básico de control), pero activando un comportamiento voluntario diferente[1] y distintos esquemas de reactividad y condiciones de finalización. El plan P_1^7, que el control de PILOTO asoció a la orden OT_1^7, se puede ver en la figura 6.9, mientras que en la figura 6.4 se muestra a qué agente se dirige cada tarea de dicho plan.

La primera tarea de P_1^7 consiste en configurar los sensores a través de los Agentes Sensores ($OT_{1.1}^7$) y los esquemas de reactividad (p.e., ajustar la distancia de seguridad). A partir de dicho momento, cada vez que se reciben nuevo datos sensoriales se ejecutan una serie de tareas cíclicas hasta que se cumplan las condiciones para finalizar o abortar la tarea. Las primeras están impuestas en la propia OT recibida, mientras que las segundas suelen ser implícitas y generales, es decir, independientes de la tarea que se ejecuta (p.e., el máximo tiempo de ejecución o el máximo recorrido). Cuando una de las condiciones se cumple el plan finaliza y el especialista PILOTO avisa a NAVEGADOR (ROT_1^7) que la orden ha sido adecuadamente ejecutada o no.

Cada vez que se reciben nuevos datos sensoriales se ejecutan en paralelo el agente perceptual DETECTOR_MARCAS ($OT_{1.2}^7$) y el comportamiento voluntario que debe guiar los movimientos del robot, en nuestro caso SEGUIR_CONTORNO ($OT_{1.3}^7$). Cuando el agente perceptual finaliza su ejecución, se activa POSICIONADOR_ROBOT ($OT_{1.4}^7$). La última tarea del ciclo consiste en integrar los comandos generados y en ordenar a los Agentes Efectores Base y Torreta que envíen los resultados al robot para su ejecución ($OT_{1.5}^7$), es decir, los comandos finales calculados para cada efector.

Además del ciclo que acabamos de comentar, existen otras dos ramas en el plan

[1] Recordemos que en PILOTO los comportamientos voluntarios son excluyentes entre sí y sólo se puede ejecutar uno en cada momento.

P_1^7 (figura 6.9). Por un lado, cada vez que se reciben datos sensoriales básicos, además de las acciones antes descritas, el control de PILOTO también ordena al agente CREADOR_MAPA_LOCAL ($OT_{1.6}^7$) que actualice el mapa de obstáculos. La segunda rama se ejecuta cuando se reciben nuevos datos del sensor láser, y es equivalente a la ya descrita con los datos básicos: primero se activa el agente DETECTOR_MARCAS ($OT_{1.7}^7$) y a continuación el agente POSICIONADOR_ROBOT ($OT_{1.8}^7$).

Sin embargo, la gran diferencia con las acciones cíclicas antes descritas es que el control de PILOTO no espera a que finalicen estas tareas para seleccionar los comandos que se van a enviar al robot ($OT_{1.5}^7$). En el primer caso, porque la actualización del mapa consume muchos recursos computacionales y la espera ralentiza excesivamente el tiempo de ejecución de todo el ciclo. En el segundo caso, porque los datos del láser se reciben con menos frecuencia que los datos básicos. De cualquier modo, si alguno de estos agentes genera algún comando de movimiento, el control de PILOTO lo tendrá en cuenta en la ejecución del siguiente ciclo asociado a los datos básicos, ya que para seleccionar los comandos se revisan los últimos comandos generados por todos los agentes.

El agente DETECTOR_MARCAS se encarga de detectar las marcas de interés en el entorno. Actualmente utiliza los datos del láser para detectar paredes y puertas y los datos de ultrasonidos para detectar paredes y corredores.[2] Además, y dentro de la filosofía de la percepción activa, también calcula los comandos necesarios para mover el robot y mejorar la percepción. En la práctica, los movimientos se concentran en mantener la torreta correctamente orientada (formando unos 45º) hacia el contorno en donde se deben detectar las puertas. Esta orientación es fundamental, debido a las fuertes limitaciones del láser del *Nomad 200*.

El agente POSICIONADOR_ROBOT se encarga de calcular la posición y orientación del robot (POSE) y determinar el error que comete su sistema odométrico. La posición y el ángulo correctos se determinan comparando las marcas detectadas (principalmente corredores) con las marcas del entorno previamente almacenadas, bien detectadas en una fase previa de exploración, bien proporcionadas por el usuario. Cuando el error es excesivo (flechas en la figura 6.6) se calcula la recalibración del sistema odométrico y también se reposicionan las últimas marcas detectadas.

A partir de una misma adquisición sensorial se pueden generar varios comandos dentro de la ejecución de un mismo plan. Al describir el especialista PILOTO ya se ha

[2]Es muy difícil detectar puertas sólo con los sensores de ultrasonidos y, por ahora, los algoritmos no son muy fiables.

explicado que se integran utilizando una técnica de arbitraje basada en seleccionar el comando con mayor prioridad, es decir, el generado por el agente más importante. Existen técnicas de arbitraje entre comportamientos más complejas, que producen un movimiento final del robot más suave (p.e., las basadas en lógica borrosa (Saffiotti y col., 1995; Saffiotti, 1997b)), pero el método elegido cubre sobradamente las necesidades planteadas, tal como se ha comprobado en los experimentos realizados.

Por último, comentar que se han detectado ciertos problemas en el comportamiento SEGUIR_CONTORNO cuando el robot debe seguir una pared recta, y que se traducen en una trayectoria excesivamente curvilínea (puede verse un ejemplo en la figura 6.6). Sin embargo, cuando se activa el comportamiento por separado y la torreta se mueve alineada con la base, no se observa este fenómeno. La explicación de este comportamiento es que el desalineamiento entre la torreta y la base hace que no se cumpla la hipótesis de que uno de los sensores de US está apuntando en la misma dirección que la base y provoca confusiones y errores en el módulo de control de la velocidad angular (ver capítulo anterior). El problema se agrava cuando hay puertas en el contorno seguido, debido al ruido que provocan en las mediciones de los US, y cuando se activan otros comportamientos (p.e., MANTENER_DISTANCIA).

6.2.3 Respuestas reactivas

Mientras un especialista responde a una orden de tarea pueden ocurrir eventos no esperados ante los cuales debe reaccionar. En AFE-Robótica las pizarras son almacenes activos, capaces de generar eventos asociados a ciertos cambios en sus datos. Una de tales situaciones tienen lugar cuando un objeto está demasiado próximo al robot (p.e., punto (a) en la figura 6.6). La pizarra de PILOTO detecta el problema porque chequea todos los datos sensoriales cada vez que se almacenan y, si alguna de las distancias medidas es demasiado pequeña, genera el evento correspondiente (E_a en la figura 6.10). Como este evento OBSTÁCULO MUY PRÓXIMO está etiquetado como interno, la pizarra lo remite a su agente de control (*reactividad interna*).

La respuesta del agente de control de PILOTO depende del estado actual del especialista y de la información almacenada en su pizarra. Sin embargo, lo más habitual es activar un plan de contingencia (P_a) para responder a dicho evento y que consiste en dos tareas. En la primera se activa el comportamiento reactivo MANTENER_DISTANCIA (OT_a^1) para que calcule cómo mover el robot para mantener una distancia mínima con todos los objetos que le rodean. En la segunda se activan los Agentes Efectores Base y Torreta (OT_a^2) para conseguir que el robot ejecute la

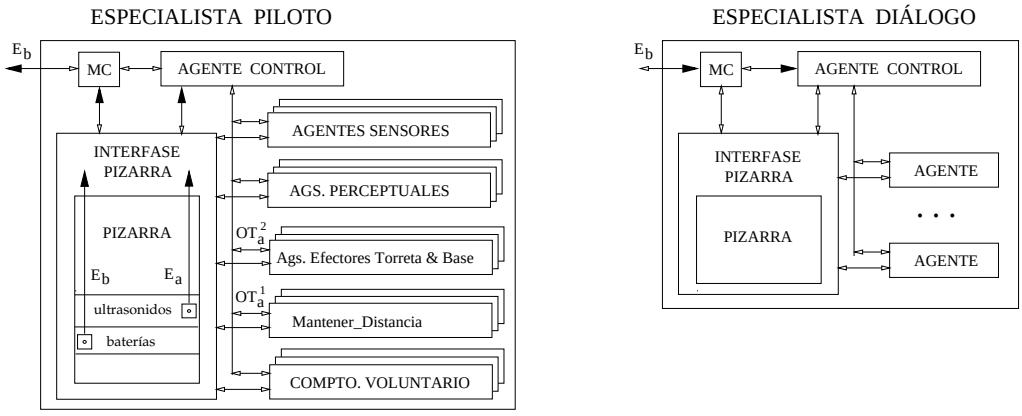

Figura 6.10: Ejemplos de la activación de la reactividad interna y externa en AFE-Robótica.

maniobra calculada. P_a es compatible con casi todos los posibles planes activos en PILOTO, por lo que se puede ejecutar de forma independiente y en paralelo.

En la figura 6.11 se muestra una ampliación del punto etiquetado como *(a)* en la figura 6.6 y en ella se pueden observar las sucesivas respuestas reactivas del especialista PILOTO al evento OBSTÁCULO MUY PRÓXIMO. En este caso el robot se ha acercado demasiado a la pared mientras sigue el contorno de la izquierda, confundido por los falsos ecos del sonar que genera la puerta. El evento se ha activado en cada adquisición de datos sensoriales hasta que el robot ha logrado separarse lo suficiente de la pared.

El evento OBSTÁCULO MUY PRÓXIMO también se puede activar cuando se aumenta la distancia de seguridad en la ejecución de una determinada tarea, por ejemplo, para detectar las puertas más cercanas. La distancia de seguridad se amplía debido a las limitaciones del sensor láser por triangulación del *Nomad 200*, ya que se persigue que el robot esté suficientemente lejos (unos 50 cm) de todos los obstáculos que le rodean, de forma que el láser pueda operar con las máximas garantías y detectar los marcos de las puertas.

Cuando el robot está en un espacio excesivamente reducido el evento OBSTÁCULO MUY PRÓXIMO también se activa constantemente. Normalmente, el especialista PILOTO trata de evitar esas zonas pero, en ocasiones, entra en ellas sin pretenderlo, por ejemplo, al cruzar una puerta (figura 6.12). Las sucesivas respuestas reactivas de PILOTO consisten en desplazar continuamente el robot tratando inútilmente de aumentar la distancia con los obstáculos que le rodean.

Como ya se ha comentado en el capítulo anterior, las dependencias entre los

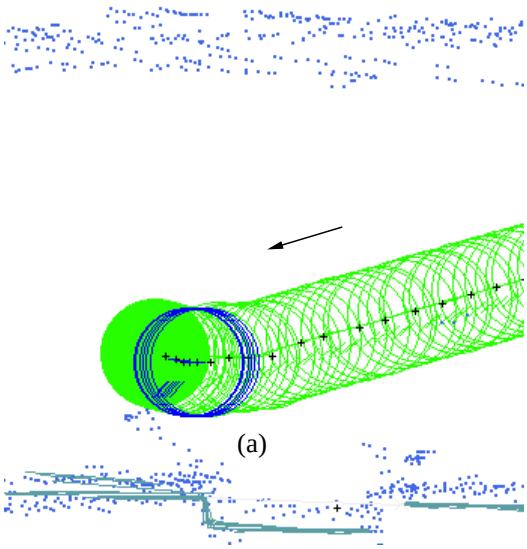

Figura 6.11: Ampliación de la región (a) de la figura 6.6 dónde se ilustra claramente la integración de las respuestas reactivas (círculos oscuros) con la ejecución de un plan (círculos claros). Los puntos son los datos de ultrasonidos y las líneas los datos del láser.

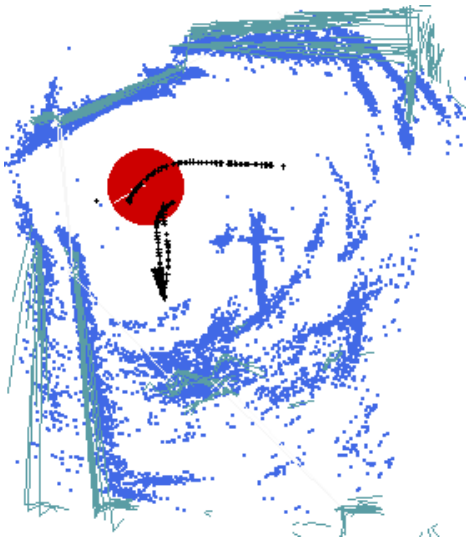

Figura 6.12: Activación continua del evento Obstáculo Muy Próximo buscando espacio libre cuando el robot está en un espacio muy reducido. Los segmentos muestran las medidas del láser, los puntos las medidas de los ultrasonidos y las cruces la trayectoria del robot.

datos en un robot móvil son múltiples y complejas, por lo que a veces las respuestas reactivas no se adaptan a la jerarquía de tareas ni a la abstracción de datos. En estos casos un evento que se produce en un especialista tiene una respuesta en otro especialista diferente (*reactividad externa*). Un ejemplo es la presentación de un aviso al usuario cuando el nivel de voltaje es bajo, refiriéndonos a las baterías. El evento lo genera la pizarra de PILOTO, pero al estar etiquetado como externo lo envía a su módulo de comunicaciones (MC en la figura 6.10) para que lo remita al especialista asociado al evento, en este caso DIÁLOGO. El MC de PILOTO determina la dirección del MC de DIÁLOGO y le envía un mensaje KQML con los datos del evento. Cuando el MC de DIÁLOGO recibe el mensaje y reconoce que es un evento lo envía directamente a su agente control (figura 6.10) para que ejecute la respuesta adecuada igual que si fuese un evento interno.

6.2.4 Ejecución de trayectos largos

Después de mostrar el funcionamiento de la arquitectura cuando el robot debe ejecutar pequeñas rutas vamos a estudiar cómo se ejecutan los desplazamientos largos (dentro de las limitaciones de espacio que tenemos). De las pruebas realizadas hemos seleccionado una que ilustra los resultados generales que se han obtenido y donde se observan los problemas y dificultades más comunes que aparecen al ejecutar trayectos largos y cómo se resuelven.

El trayecto seleccionado es un recorrido circular que empieza y acaba en la habitación 14 (nodo *H14*) y que tiene como destinos intermedios la habitación 8 (nodo *H8*) y el aula (nodo *S3,1*), es decir, tres rutas que se traducen en las órdenes de tarea al especialista NAVEGADOR: **"Ir A** *Lugar H8*", "**Ir A** *Lugar S3,1*" e **"Ir A** *Lugar H14*".[3] La trayectoria final que ha seguido el robot se puede ver en la figura 6.13 mientras que en la figura 6.14 se observa, únicamente a modo ilustrativo, un mapa métrico global del entorno, confeccionado a partir de los sucesivos mapas métricos locales (MAPA DE OBSTÁCULOS) creados utilizando las técnicas descritas en el capítulo anterior.

El robot ha recorrido más de 210 metros en un trayecto que, teóricamente y a partir de las distancias métricas almacenadas en el mapa topológico del entorno, debería ser de unos 150 m. En el trayecto el robot ha cruzado sin ningún incidente

[3] Como ya se ha mencionado en el capítulo anterior, sería relativamente sencillo crear una nueva orden de tarea más compleja que **Ir A** y capaz de seguir una ruta (cíclica o no) y con un número arbitrario de destinos intermedios.

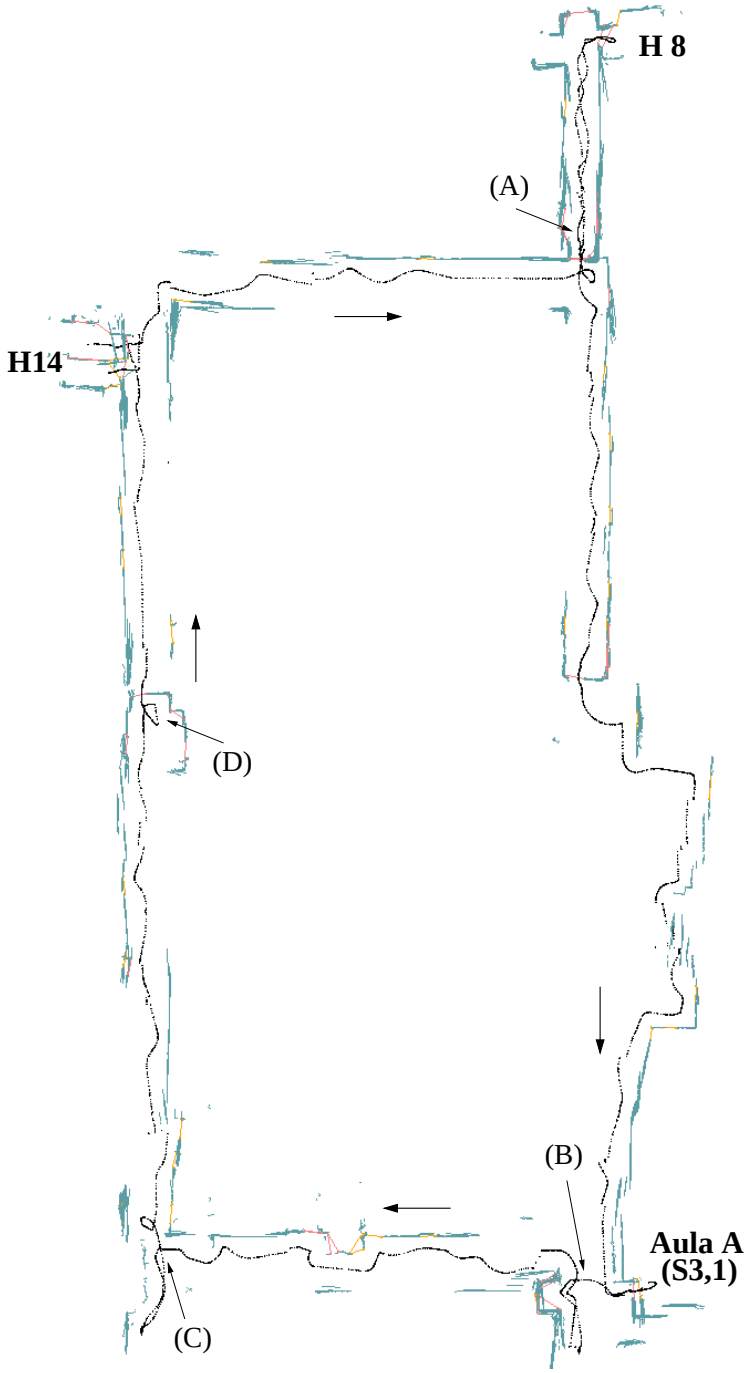

Figura 6.13: Datos del sensor láser y la trayectoria seguida por el robot en un trayecto circular con principio y fin en la habitación *14* y pasando por la habitación *8* y el aula A (*S3,1*).

Figura 6.14: Mapa métrico global construido a partir de los diferentes mapas de obstáculos locales creados durante el trayecto de la figura 6.13. El alcance efectivo de los US es de tres metros.

un total de 10 puertas (4 de ellas en los dos sentidos) de los dos tipos posibles (simples y dobles). Para mantener acotado el error en la odometría del robot se han realizado un total de 49 recalibraciones con una media de 21 cm de desplazamiento y 3,3º de ángulo. El desplazamiento total provocado por todas las recalibraciones ha sido de 10,4 m. El error de la posición odométrica corregida es aceptable, ya que no suele sobrepasar el metro. Como se puede observar en la figura 6.13, la mayor discrepancia se produce entre el principio y el final de la trayectoria, pero en gran parte se debe a la incorrecta posición inicial fijada para el robot. También influye que la corrección de la odometría sólo se realiza en los corredores y únicamente en su eje transversal.

El robot ha necesitado algo más de 50 minutos para realizar todo el trayecto, es decir, a una velocidad media de 6,9 cm/s. Si no consideramos los momentos en donde la velocidad es negativa o nula (giros de la torreta, alineamiento del robot con los corredores, falta de espacio libre cuando se cruza una puerta, etc.) entonces la velocidad media sube a más de 11 cm/s. De todos modos, es necesario resaltar que los distintos comportamientos y tareas no han sido optimizadas para aumentar la velocidad.

Por ejemplo, la velocidad máxima para cruzar una puerta se ha limitado a 8 cm/s y en los tramos críticos a tan sólo 3 cm/s. Durante las alineaciones y las detecciones de las puertas la torreta y la base también podrían girar más rápido. Por último, en el comportamiento de seguir un contorno sólo se ha optimizado la velocidad de avance pero no la de giro, que es fundamental en las esquinas. Por otro lado, en algunos casos la ejecución de las tareas es demasiado conservadora y enlentece su ejecución. Por ejemplo, cuando ya se ha detectado la puerta que después se quiere cruzar se podría evitar el giro de la torreta para fijar mejor su posición.

El trayecto descrito se ha repetido varias veces y los datos obtenidos son similares a los ya comentados (tabla 6.1). La principal diferencia es que en cada ejecución han surgido diferentes contingencias y errores, por lo que la trayectoria del robot es diferente y existe una cierta variabilidad en los principales parámetros de cada desplazamiento.

6.2.5 Contingencias en la ejecución de un plan

Durante la ejecución de una ruta pueden surgir diversos contratiempos y dificultades, ante los cuáles el sistema debe responder adecuadamente. En la figura 6.15 se muestran dos rutas cortas en donde se pueden ver dos ejemplos del contratiempo

ejecución	1	2	3	4	media
distancia total (m)	213	201	213	207	209
tiempo total (s)	3088	2942	3724	3253	3251
velocidad media trayecto (cm/s)	6,9	6,8	5,8	6,4	6,4
media velocidades positivas (cm/s)	11,2	10,7	10,0	10,0	10,6
número de rutas intentadas	26	23	23	21	23
número de puertas cruzadas	10	9	11	11	10
número de recalibraciones	49	46	63	58	54

Tabla 6.1: Medidas realizadas sobre distintas ejecuciones del trayecto *H14-H8-S3,1-H14* (los datos de la columna 1 corresponden a la trayectoria mostrada en la figura 6.13).

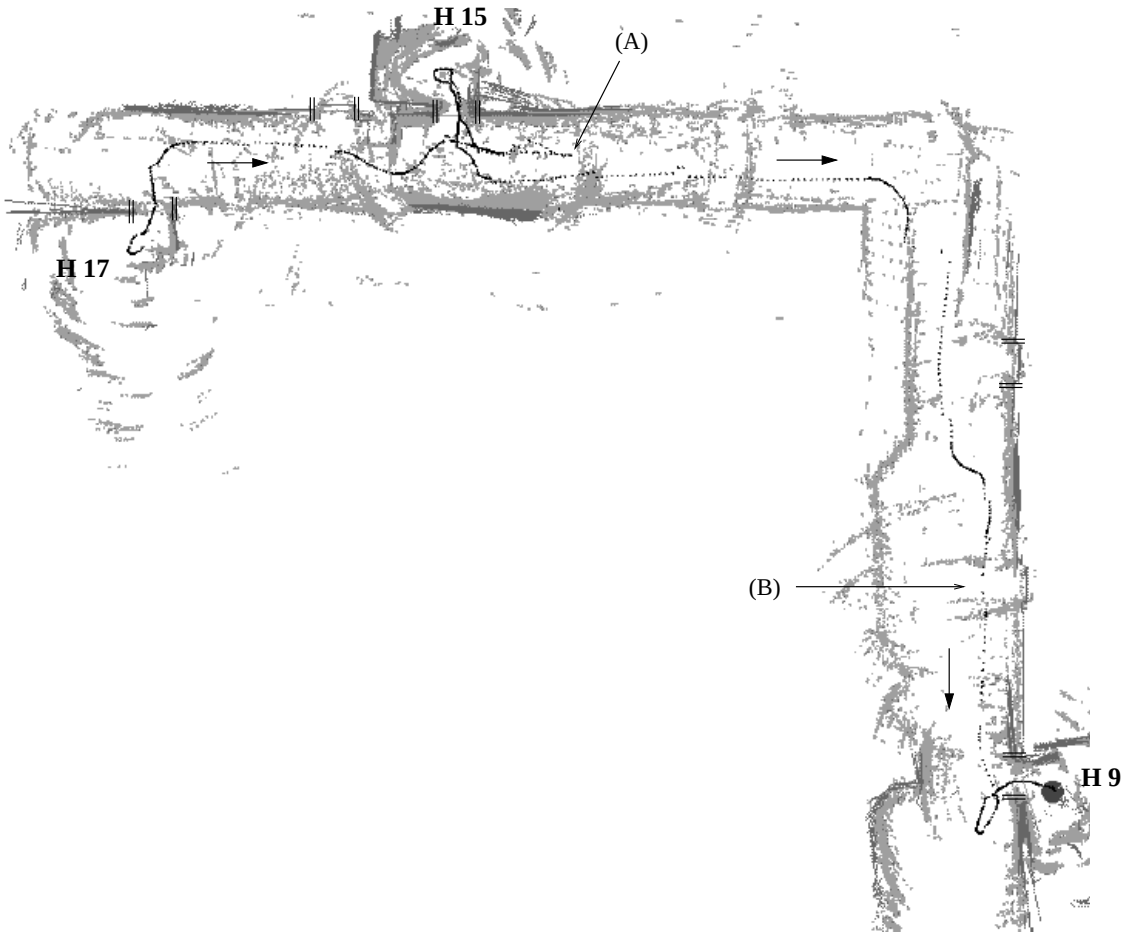

Figura 6.15: La trayectoria del robot primero desde la habitación *17* a la habitación *15* y después desde ésta última a la *9*.

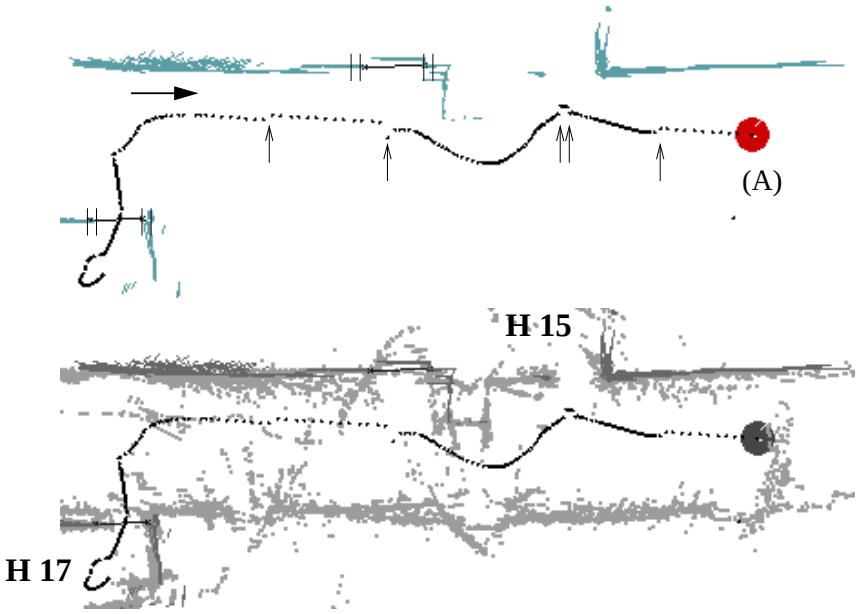

Figura 6.16: Primera parte del trayecto entre la habitación *17* y la *15* mostrado en la figura 6.15; arriba se muestran sólo los datos del láser y las puertas detectadas, y abajo también los datos de los ultrasonidos. Las flechas verticales indican las recalibraciones odométricas.

más habitual: la no detección de una puerta mientras se sigue un contorno de un pasillo. En la figura 6.16 se muestra la primera parte del recorrido seguido entre la habitación 17 y la 15 (región A en la figura 6.15). En este caso, PILOTO no detectó la puerta de acceso a la habitación 15, en parte porque el robot antes tuvo que bordear un obstáculo muy próximo a ella. El control de NAVEGADOR detectó el problema, primero porque PILOTO detectó sólo una de las dos puertas previstas, y segundo porque LOCALIZADOR localizó el robot en un lugar diferente al previsto (nodo *C3,7*). La solución consistió en intentarlo de nuevo desde el lugar que ocupaba el robot. Como podemos ver en la vista general mostrada en la figura 6.15, el nuevo intento sí que tuvo éxito

Otra muestra del mismo problema se observa en el tramo etiquetado como B en la figura 6.15, en el que PILOTO sólo detectó dos de las tres puertas previstas. Sin embargo, a diferencia del caso anterior, el lugar final del robot sí es el apropiado (*C6,4*) y la última puerta detectada está a la distancia correcta respecto al principio del corredor. Por lo tanto, el control de NAVEGADOR decide seguir con el plan y cruzar la última puerta detectada. Estos dos ejemplos sirven para demostrar que en la navegación basada en marcas, tan fundamental es detectar las marcas del entorno

(en nuestro caso sobre todo las puertas) como saber identificar y manejar aquellas situaciones en donde no se detectan todas (lo que en la práctica es lo más habitual).

Una contingencia de otra naturaleza es que PILOTO no ejecute una orden de tarea tal y como NAVEGADOR había previsto, en parte por las deficiencias del propio agente, en parte porque el robot opera en un entorno cambiante y complejo. NAVEGADOR debe tener siempre presente esta posibilidad a la hora de traducir una ruta en órdenes de tarea. Por ejemplo, finalizando cada tramo de una ruta al pasar por una intersección de corredores y haciendo comprobaciones del avance del robot. En general, cuanto más largo sea el trayecto más posibilidades existen de que aparezcan estas alteraciones. En el trayecto mostrado en la figura 6.13 se observan dos de estos casos (puntos B y C).

El caso B es algo complejo de explicar. Después de alinear el robot con el corredor, el control de NAVEGADOR le ordenó a PILOTO que siguiese el contorno de la izquierda con la idea de cruzar la intersección y tomar el corredor hacia arriba. Sin embargo, la orden no se pudo cumplir, porque el robot recorrió demasiado espacio sin detectar el contorno pedido.[4] El control de NAVEGADOR calculó una nueva ruta pero, debido a un pequeño error en la odometría, LOCALIZADOR creyó que el robot ya estaba en la intersección y el control ordenó directamente a PILOTO que siguiese el contorno de la derecha. Como se puede ver en la figura 6.13, como resultado de esta orden el robot tomó el corredor de la izquierda. El fallo se hubiera evitado si antes el control de NAVEGADOR hubiera ordenado seguir la dirección vertical (90°).

En el caso C el control de NAVEGADOR ordenó a PILOTO que siguiese el contorno de la derecha con la idea de que el robot tomara el corredor derecho después de la intersección. Sin embargo, unos obstáculos en el medio de la intersección hacen que la entrada de dicho corredor sea tan reducida que el comportamiento SEGUIR_CONTORNO la haya considerado igual al hueco de una puerta. Como se puede ver en la figura 6.13 esta confusión provocó que el robot no tomara el corredor correcto. En ambos casos, una vez que PILOTO finalizó la ejecución de la tarea el control de NAVEGADOR comprobó la posición del robot y detectó que no coincidía con la prevista. Como antes, la solución consistió en trazar una nueva ruta desde la posición real del robot.

Otro tipo de contingencia durante la ejecución de una ruta es que una orden enviada al agente PILOTO no haya podido ser ejecutada correctamente. El caso más

[4]Después de este ejemplo se ha subido la distancia máxima para detectar el contorno antes de que PILOTO aborte la tarea.

típico y problemático es la tarea de seguir un contorno que falla por dos motivos: no se ha encontrado el contorno o la torreta no se puede alinear con el contorno. El primer problema (punto B de la figura 6.13) se ha resuelto haciendo que el robot realice un giro más abierto (de tres metros de radio) para encontrar el contorno y permitir un tiempo y un desplazamiento mayores antes de abortar la tarea. De esta forma, el robot casi siempre encuentra un contorno, aunque otra cosa es que sea el correcto. El segundo problema no tiene una fácil solución, ya que se da en aquellas regiones en donde el láser no puede detectar obstáculos (puertas abiertas, ocultación del haz láser, etc.). En todo caso, si no se puede ejecutar la tarea de seguir un contorno se intenta ejecutar de nuevo la ruta con la esperanza de que sea más fácil desde la nueva posición.

Un ejemplo más grave de una tarea que no se puede ejecutar se produce cuando PILOTO no ha detectado la puerta que se le ha ordenado cruzar. El motivo principal de la no detección es la miopía del sensor láser (no detecta obstáculos a menos de medio metro) y su corto alcance (en teoría hasta 2,5 metros, en la práctica un metro menos). Por lo tanto, si el robot está situado muy cerca o muy lejos de la puerta no podrá detectarla. Para resolver esta contingencia se han tomado dos medidas según se conozca o no la orientación de la puerta.[5]

En el primer caso, se ordena a PILOTO que mueva el robot una pequeña distancia (40 cm) siguiendo la orientación de la puerta deseada y después se intenta ejecutar la ruta otra vez. La idea es que si el robot está muy lejos de la puerta se acabará acercando lo suficiente para detectarla (puntos A y D en la figura 6.13). Por otra parte, si el robot está muy cerca, lo más probable es que se aleje o, al menos, cambie de posición. En ambos casos el primer tramo de la nueva ruta, un nuevo intento de detectar la puerta deseada, probablemente tendrá más éxito. Si no se conoce la orientación de la puerta, simplemente se hace girar la torreta en sentido contrario al habitual, con la esperanza de detectarla mejor. Quizás una opción mejor sería mover el robot en línea recta una pequeña distancia.

Oportunismo en la ejecución de tareas

Sin embargo, no todas las contingencias son necesariamente negativas. Por ejemplo, el robot puede cruzar una puerta de la ruta de forma involuntaria. Existen al menos tres casos en donde se puede dar esta situación. En primer lugar, puede

[5] Cuando una puerta da a un corredor se puede deducir su orientación de entrada y de salida a partir de la dirección del corredor.

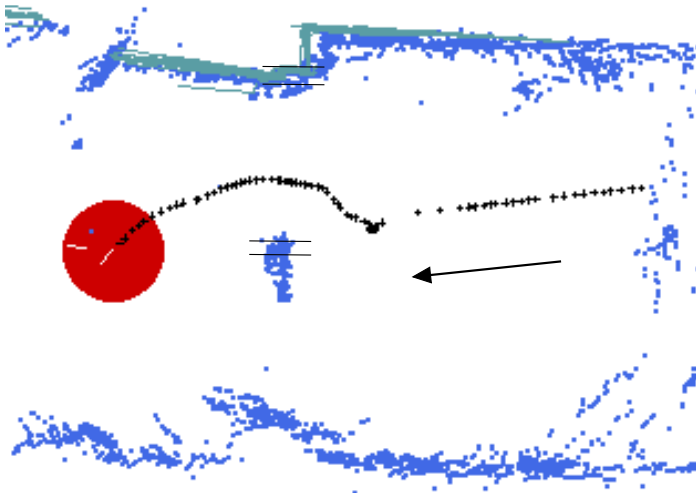

Figura 6.17: Ejemplo de cómo el robot puede cruzar una puerta (no detectada) mientras PILOTO ejecuta la tarea de seguir contorno.

ocurrir que una de las puertas dobles que se encuentran en algunos de los pasillos de nuestro entorno esté totalmente abierta, de tal modo que, al no haber un obstáculo que se lo impida (la propia puerta), PILOTO continúa ejecutando la tarea de seguir el pasillo (realmente, uno de sus lados) y la cruza.

El segundo caso ocurre cuando el robot llega a una puerta doble siguiendo el lado por donde está abierta.[6] Por ejemplo, en la figura 6.17 el robot sigue el lado derecho del corredor, mientras que la hoja cerrada de la puerta doble está a su izquierda. El robot es capaz de cruzar la puerta porque el comportamiento SEGUIR_CONTORNO sigue el contorno a muy poca distancia y porque PILOTO ejecuta esta tarea con una distancia de seguridad muy baja (18 cm en vez de la normal de 38). En el capítulo anterior ya se explicó que esta reducción se aplica porque siendo SEGUIR_CONTORNO muy fiable, por lo que no choca con los objetos del entorno, se minimizan las interferencias no deseables con el comportamiento MANTENER_DISTANCIA. Por último, la dificultad para detectar las puertas de madera con los US también contribuye (como se puede ver en la figura 6.17 sólo unos pocos datos delatan el borde de la hoja cerrada de la puerta).

El tercer caso es cuando el robot trata de acercarse a una puerta después de no haberla detectado girando la torreta. Para que esto ocurra el robot debe estar muy cerca de la puerta y, sobre todo, casi orientado hacia su centro. Aún así, este caso es muy infrecuente porque la distancia de seguridad es la normal. De todos

[6]Una puerta doble tiene dos hojas y normalmente sólo una de las dos está abierta.

modos, como en ciertas orientaciones los sensores de US no detectan bien los marcos de madera de las puertas, puede ocurrir que el evento OBSTÁCULO MUY PRÓXIMO no se active hasta que el robot esté casi en el umbral de la puerta y después puede ser demasiado tarde, ya que en esa situación es muy probable que el robot acabe de cruzarla por falta de espacio para girar y retroceder.

En cualquier caso, lo importante es que el control de NAVEGADOR adapta rápidamente la ejecución de la ruta a la nueva situación creada en cuanto la detecta. Actualmente el mecanismo implementado es muy sencillo y se basa en la estrategia general para ejecutar una ruta. Como ya se ha mencionado anteriormente, si la localización del robot antes de cruzar una puerta no es la prevista, el control de NAVEGADOR procede a replanificarla de nuevo. En estos casos, como ya se ha cruzado la puerta, el control decide seguir la anterior ruta o trazar otra nueva, pero a partir del lugar o nodo en el que se ha localizado el robot. Por lo tanto, el control de NAVEGADOR no se empeña en alcanzar los objetivos intermedios que se ha marcado sino en alcanzar el objetivo final, es decir, es *oportunista* en la ejecución de las tareas encomendadas.

6.3 Características de AFE-Robótica

En esta sección analizaremos las características más importantes de AFE-Robótica. En primer lugar, aquellas que previamente hemos señalado como las más interesantes en la evaluación de una arquitectura de control para robótica móvil, es decir, la robustez, la reactividad, la integración de comportamientos, la flexibilidad y la escalabilidad. Pero también otras que, por su relevancia, se entienden como implícitas en cualquier arquitectura de control para robots, principalmente la ejecución de tareas en tiempo real y los mecanismos de monitorización del funcionamiento interno de la arquitectura. Por último, también se analizarán las posibilidades de AFE-Robótica respecto a su ejecución distribuida, a su portabilidad y a la reutilización de sus componentes.

6.3.1 Criterios de comparación

Robustez

Se puede hablar de robustez tanto a nivel de arquitectura como a nivel de operación. Respecto a la primera recordar lo ya comentado al comparar AFE-Robótica

con las otras arquitecturas de control para robótica móvil. A nivel de especialista la robustez de AFE es pobre, ya que si deja de operar el agente de control, la pizarra o un módulo de comunicaciones de datos o de control, entonces falla todo el especialista. Ahora bien, para el conjunto de la arquitectura cada especialista no deja de ser más que un simple agente, por lo que cuando deja de operar, la arquitectura sigue siendo operativa y sólo se pierde la funcionalidad asociada a dicho agente. Aunque, obviamente, no todos los agentes/especialistas tienen la misma importancia. Cuando hablemos de la monitorización en AFE-Robótica veremos cómo se pueden implementar mecanismos para detectar cuando un agente deja de funcionar y cómo soslayar esa dificultad.

Respecto a la robustez de operación, además de la arquitectura también influyen decisivamente el diseño del sistema, la implementación, las capacidades y limitaciones motrices y sensoriales del robot, las características y la complejidad del entorno y la dificultad de las tareas a ejecutar. La mejor demostración de que la operación del prototipo implementado de AFE-Robótica es robusta, son los propios resultados experimentales obtenidos en la sección anterior. Así, podemos decir que el sistema ha logrado adaptarse perfectamente a un gran número de dificultades, errores e imprecisiones propias e intrínsecas a la operación en un entorno real complejo, dinámico y que no puede ser completamente modelado.

Por ejemplo, el robot ha cruzado más de un centenar de puertas dobles y sencillas, tanto correcta como incorrectamente detectadas, y sólo en muy contadas ocasiones ha chocado con uno de sus marcos. De hecho, la razón última de estos errores se puede atribuir a una incorrecta detección de los propios marcos de madera por los sensores de US. Otro ejemplo notable es cómo se mantiene acotado el error acumulado en la posición odométrica del robot en valores razonables, incluso en trayectos de centenares de metros, y a pesar de que los datos que ha proporcionado el usuario sobre las marcas del entorno son únicamente topológica y cualitativamente correctos.

Por otra parte, el sistema también es muy robusto respecto a deficiencias perceptuales graves, tales como la no percepción de todas las marcas del entorno (puertas, paredes y corredores) o su detección incorrecta. Estos errores simplemente degradan el rendimiento del sistema. El sistema también es robusto respecto a errores en los comportamientos de bajo nivel. Por ejemplo, que el agente SEGUIR_CONTORNO no detecte el contorno a seguir, no doble en una esquina, no siga una pared en línea recta o cruce involuntariamente una puerta, no impide realizar el desplazamiento encomendado, sólo incrementa el tiempo de ejecución.

Por último, la existencia de objetos en movimiento alrededor del robot tampoco suele confundir al sistema, aunque sí dificulta la detección de las marcas del entorno y, por tanto, el reposicionamiento del robot respecto a su mapa interno del mundo y a la finalización de los desplazamientos. De todos modos, los objetos en movimiento aumentan los errores en los sensores de US, lo que puede incidir negativamente en algunos comportamientos, sobre todo porque el robot presupone que el entorno no es hostil.

De todos modos, conviene señalar que algunos aspectos de la implementación actual de AFE-Robótica pueden afectar negativamente a la robustez del sistema. En primer lugar, no existe duplicidad en la ejecución de tareas. Por ejemplo, para reposicionar el robot sólo se tienen en cuenta los datos y las marcas detectadas con los sensores de US, mientras que para detectar las puertas sólo se utilizan los datos del láser. De igual modo, los comportamientos voluntarios y reactivos sólo están implementados en un único agente. Aunque un comportamiento puede ser suplido por otro (p.e., SEGUIR_CONTORNO en ocasiones se puede sustituir por MOVER_DIRECCIÓN), en la mayoría de los casos la pérdida es irreparable.

En segundo lugar, y quizás es el problema más importante, la localización global del robot es bastante frágil y poco consistente, ya que se basa exclusivamente en la posición odométrica corregida por PILOTO. De este modo, si el error en la odometría sobrepasa los límites de un reposicionamiento local, entonces el robot deja de estar localizable y los mapas internos pierden su utilidad. De igual forma, un error de algunos centímetros puede suponer un cambio ficticio en la localización del robot sobre el mapa topológico que puede provocar un trastorno considerable y, a veces, la no terminación de la ruta.

En tercer lugar, habría que mejorar la capacidad de adaptación de los agentes de control para enfrentarse a aquellas situaciones en donde uno de sus agentes deja de estar operativo o cuando una respuesta a un evento no produce los resultados deseados. En la misma línea, habría que diseñar mecanismos para minimizar los problemas que puedan surgir en la capacidad sensorial y motora del robot, aunque la falta de redundancia puede ser una seria limitación.

En cuarto lugar, también sería necesario actualizar la información del mapa topológico cuando se modifica el entorno (p.e., un corredor está bloqueado) o cuando una de las puertas que se debe cruzar está cerrada. Con la representación utilizada estos cambios son muy fáciles de realizar, lo realmente difícil es detectar con fiabilidad estas situaciones con la limitada capacidad sensorial del *Nomad 200*.

Reactividad

Una de las principales aportaciones de AFE es que existe reactividad o, más concretamente, respuestas reactivas en todos los niveles de la arquitectura y sin ningún tipo de restricciones o limitaciones arquitectónicas. De todos modos, donde la reactividad es muy importante es en aquellos agentes-especialistas encargados de controlar a bajo nivel los efectores del robot, por ejemplo, en PILOTO. En la sección anterior ya se han comentado sobre algunos ejemplos representativos cuáles son los mecanismos para activar los eventos y generar las respuestas reactivas apropiadas.

Sin embargo, la implementación actual de la reactividad en AFE-Robótica es muy sensible a la activación continua de eventos en PILOTO, ya que, al considerar el agente de control las respuestas reactivas más prioritarias que las acciones del ciclo de control, se puede llegar a bloquear la ejecución normal de las tareas encomendadas. Existen dos modos de evitar este problema. Uno de ellos sería conseguir que el agente de control revise y actualice las prioridades tanto de las acciones del ciclo de control como de los eventos. El segundo sería más sencillo y consistiría en añadir una cierta capacidad de adaptación para bloquear la activación continua del evento en cuestión, bien directamente en la generación del evento en la pizarra, bien de forma indirecta en el propio plan de respuesta. El objetivo sería subir progresivamente el umbral de activación de un evento que se activa de forma repetitiva y, pasado un tiempo sin ninguna activación, bajarlo paulatinamente a sus valores normales.

La reactividad es cada vez un problema menos crítico. En primer lugar, porque se ha generalizado en las arquitecturas de control delegar el control a bajo nivel del robot en uno o varios módulos basados en comportamientos, con las conocidas ventajas que ello conlleva. El segundo motivo es más prosaico y se debe a que cada vez los computadores son más rápidos y, por tanto, necesitan menos tiempo para realizar los cálculos y permiten una mayor frecuencia en la adquisición de los datos sensoriales. De esta forma, por un lado se reduce la latencia en la detección de los eventos y, por otro, se reduce el tiempo de respuesta.

Por último, resaltar que una acertada reactividad que permita, entre otras muchas cosas, mantener la integridad del robot y la de su entorno, no sólo depende de los mecanismos que proporcione la arquitectura de control sino que, antes de nada, depende de la adecuada capacidad de percepción sensorial adaptada tanto al entorno como al robot y a la tarea a ejecutar. Obviamente, si los sensores del robot no son capaces de detectar un determinado evento o situación crítica, entonces es imposible que la arquitectura pueda reaccionar adecuadamente ante ella. Esto es

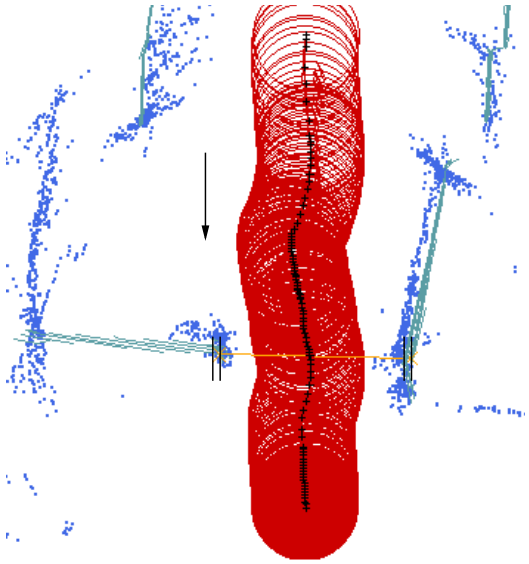

Figura 6.18: Ejemplo de cómo el especialista PILOTO es capaz de lograr que el robot cruce una puerta.

lo que nos ocurre con ciertos objetos tales como sillas, mesas bajas, papeleras, etc., puesto que los sensores del *Nomad 200* fallan estrepitosamente en su detección y provocan no pocos problemas.

Integración de comportamientos

Una de las principales características de AFE-Robótica es que permite la integración de comportamientos reactivos, deliberativos y planificados en todos los niveles de la arquitectura, concretamente en los agentes de control de cada uno de sus especialistas, sin ningún tipo de restricción, como sí ocurre en otras arquitecturas.

La correcta integración de comportamientos siempre es importante, pero toma especial relevancia en los especialistas inferiores de la arquitectura como PILOTO y, sobre todo, en la ejecución de ciertas tareas específicas tales como cruzar una puerta. Este tipo de situaciones ponen a prueba no sólo la reactividad de la arquitectura, ya que el robot se tiene que mover muy próximo a los marcos de la puerta y se activan eventos constantemente, sino también la capacidad para integrar la respuestas reactivas con la ejecución del plan para realizar la tarea en cuestión.

Aunque en esta memoria mostramos sólo unos pocos ejemplos, en prácticamente todas las pruebas realizadas el robot ha logrado cruzar con éxito todas las puertas que ha intentado cruzar. El robot sólo choca contra un marco cuando es muy fino

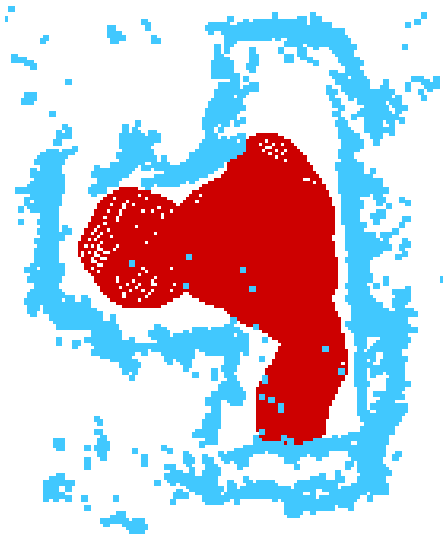

Figura 6.19: Integración en el especialista PILOTO del comportamiento voluntario AVANZAR y el reactivo MANTENER_DISTANCIA cuando un obstáculo está muy próximo al robot. Los puntos claros son medidas de ultrasonidos y los oscuros las posiciones del robot.

(p.e., puertas dobles) y está orientado de tal manera que los sensores de ultrasonidos no llegan a detectarlo. En la figura 6.18 se muestra cómo el robot ha cruzado la primera puerta del trayecto mostrado en la figura 6.6. En la figura 6.19 se muestra otro ejemplo en donde se comprueba el resultado de integrar en el especialista PILOTO el comportamiento voluntario AVANZAR y el reactivo de MANTENER_DISTANCIA. Como se puede apreciar, el robot no ha chocado con ningún objeto de su entorno.

Flexibilidad

La flexibilidad de AFE-Robótica se manifiesta en las facilidades que proporciona la arquitectura para modificar los planes, las tareas, los agentes y los datos. En cuanto a los planes, como se representan mediante grafos su modificación es muy fácil e intuitiva. Actualmente los planes están directamente codificados en los agentes de control (realmente en el módulo de ejecución de los planes) pero sería muy sencillo diseñar herramientas para editar y manipular directamente los grafos, incluso durante la ejecución de la arquitectura.

Una característica muy importante en el diseño de AFE-Robótica es que los especialistas son independientes entre sí, por tanto, cada uno de ellos se puede ejecutar, comprobar y desarrollar por separado e, incluso, por diferentes programadores. Sólo

se necesita definir y respetar una interfaz de programación que se compone de la sintaxis de las órdenes de tarea que se pueden ejecutar, las respuestas a dichas órdenes y los datos de entrada y de salida. El que todos los mensajes, tanto de datos como de control, estén codificados en ASCII, facilita en gran medida la interacción entre los distintos componentes.

La relativa independencia entre los distintos agentes/especialistas facilita que un agente se pueda modificar o sustituir por otro mejor de forma transparente al resto del sistema, dotando de gran flexibilidad a la arquitectura. Además, también permite que las modificaciones en la interfaz de programación (órdenes de tarea) de un agente sólo afecten a su agente de control y no tiene porqué alterar al resto de la arquitectura. De igual forma, la modificación de un dato de la pizarra sólo afecta a aquellos agentes que lo utilicen. Como el intercambio de datos entre los agentes y la pizarra se realiza mediante mensajes en ASCII en realidad no es necesario realizar ningún cambio para acceder a los datos de la pizarra, ya que se mantiene la interfaz básica de las comunicaciones.

Esta flexibilidad permite que para probar un especialista se pueda sustituir un agente por otro más simple (p.e., NAVEGADOR y PILOTO). O también que se ejecute un especialista con sólo parte de sus agentes para probarlos conjuntamente y detectar posibles interacciones no esperadas, problemas de sincronización, uso de recursos computacionales, etc. Por otra parte, todos los agentes de AFE-Robótica pueden ser ejecutados de forma autónoma con una simple interfaz que supla la pizarra y el control de su especialista. La interfaz se encarga de recoger los datos sensoriales, enviar los comandos generados a los efectores y crear las órdenes de tarea. Las órdenes se generan automáticamente (p.e., para un comportamiento) o las envía el usuario (p.e., agente PLANIFICADOR_RUTAS).

Escalabilidad

La clave de la buena escalabilidad de AFE-Robótica radica en un principio fundamental de su diseño que consiste en la encapsulación de las tareas más estrechamente relacionadas entre sí, junto con el control y los datos necesarios para su ejecución, dentro de lo que hemos denominado especialistas. Como ya se ha comentado, esta característica fomenta la independencia entre los distintos agentes/especialistas y facilita la introducción de nuevos agentes cuando se necesitan nuevas tareas, se quieren añadir nuevas funcionalidades o aprovechar nuevas capacidades del sistema.

De esta forma, resulta muy sencillo incorporar a la arquitectura los nuevos sen-

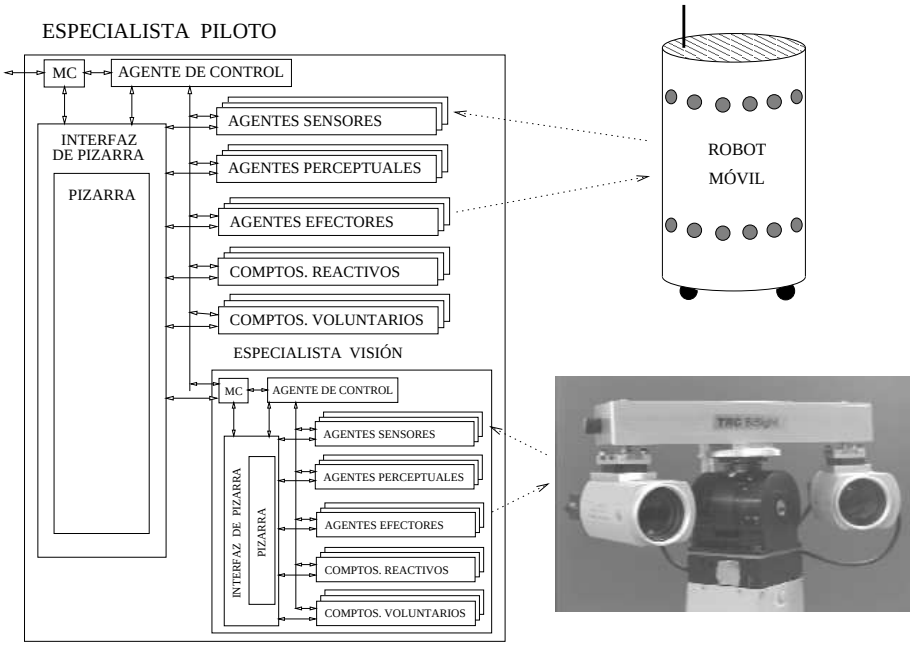

Figura 6.20: Posible modificación de AFE-Robótica con la incorporación al robot móvil de una cabeza robótica con cámaras de vídeo.

sores que se puedan añadir al robot (p.e., un láser mejor, unas cámaras digitales de vídeo o un dispositivo GPS). Basta asignar el nuevo sensor a un especialista (p.e., PILOTO), incluir los datos sensoriales que produce en su pizarra y añadir el agente sensor para manipularlo y los agentes perceptuales necesarios para extraer la información de interés (p.e., marcas del entorno). Por último, la flexibilidad de AFE-Robótica facilita la modificación de los planes que gestionan cómo y cuándo se deben activar los nuevos agentes así como la definición de los nuevos eventos en la pizarra. Se seguiría una estrategia similar en el caso de añadir nuevos efectores (p.e., una cámara con varios grados de libertad) o, simplemente, nuevos comportamientos (p.e., explorar, detectar y evitar objetos en movimiento, etc.) y funciones (p.e., confeccionar el MAPA GLOBAL DISTRIBUIDO).

Sin embargo, la gran aportación de la arquitectura AFE es que permite implementar un agente que resulta ser demasiado complejo como otro especialista con su propio agente de control, su pizarra y sus agentes y, por lo tanto, replicar la arquitectura matriz. Esta situación se daría cuando se instale en el robot un dispositivo complejo de controlar, que dispone de sus propios sensores y efectores independientes. Por ejemplo, un brazo articulado o una cabeza robótica con cámaras digitales de vídeo con varios grados de libertad. El agente-especialista encargado de controlar

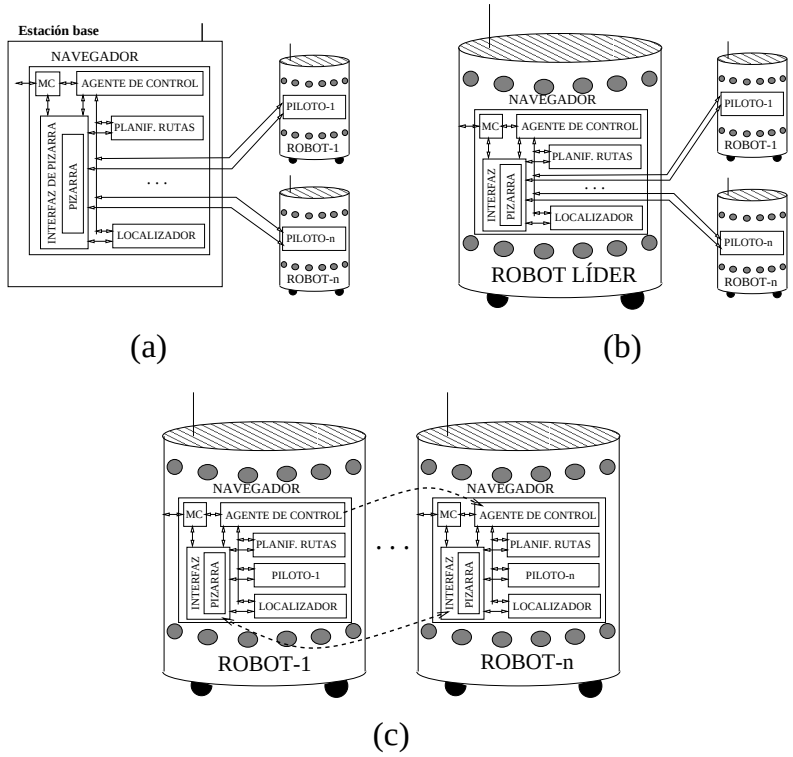

Figura 6.21: Uso de AFE-Robótica para controlar varios robots móviles no autónomos: a) un sólo especialista NAVEGADOR en una estación base, b) en uno de los robots (el líder), y c) varios especialistas NAVEGADOR integrados.

dicho dispositivo debe incluir los agentes sensores, los efectores y los perceptuales y, por supuesto, también los comportamientos (voluntarios y reactivos) necesarios para su correcta operación (figura 6.20).

La escalabilidad se puede entender de un modo más abstracto y, por ejemplo, considerar que un grupo de robots móviles se puede gobernar como si constituyesen una única plataforma compuesta de varios dispositivos físicamente independientes entre si.[7] AFE-Robótica se puede adaptar muy fácilmente a esta filosofía. Por ejemplo, implementando un único especialista NAVEGADOR que contenga tantas instanciaciones del agente PILOTO como robots móviles existan en el equipo. Como cada agente PILOTO puede ser particularizado según las características del robot que debe controlar, los integrantes del equipo no tendrían porqué ser homogéneos.

Como veremos al hablar de la ejecución distribuida de AFE-Robótica, es con-

[7]No hay que confundir este concepto con las agrupaciones "sociales" de robots móviles donde cada componente es totalmente autónomo.

veniente que cada PILOTO se ejecute internamente en el robot que debe gobernar. El especialista NAVEGADOR se podría ejecutar, bien en una estación base (figura 6.21.(a)), bien en uno de los robots, p.e., el líder (figura 6.21.(b)). Incluso se podría replicar en cada robot (figura 6.21.(c)) y mantener todas las instanciaciones "sincronizadas" (datos de la pizarra y del agente de control) para que cualquiera pudiese tomar el relevo en el caso de que el NAVEGADOR del líder dejase de funcionar. Otra opción más compleja sería coordinar todos los especialistas NAVEGADOR replicados y crear un especialista virtual. Para obtener una mayor autonomía de cada robot otra posibilidad sería considerar un único especialista MAESTRO e instanciar un agente NAVEGADOR en cada robot.

6.3.2 Otras propiedades

Ejecución en tiempo real

El especialista PILOTO es el responsable de interaccionar con el entorno a través del robot. Esto, entre otras cosas, significa que debe procesar apropiadamente la información sensorial y procesarla a tiempo, es decir, en tiempo real. Las restricciones de este especialista son tan o más severas que las de cualquier otro agente-especialista de AFE-Robótica y por eso lo hemos utilizado para estudiar el rendimiento de la arquitectura. Si el procesamiento de la información es apropiado o no, se refleja directamente en la capacidad real del sistema para ejecutar las tareas encomendadas y en la precisión y eficacia con que las realiza. Por lo tanto, en esta sección nos centramos exclusivamente en la ejecución en tiempo real y, más concretamente, en el tiempo de ejecución del ciclo de control básico del especialista PILOTO. Este tiempo limitará la capacidad de reacción del especialista y reflejará la posibilidad de operar en tiempo real, es decir, de procesar los datos *a tiempo*.

La ejecución del ciclo de control depende, por un lado, de la frecuencia de adquisición de los datos sensoriales y, por otro lado, del tiempo necesario para la ejecución de cada orden de tarea presente en el ciclo. Para conocer cuándo un agente finaliza la OT que ha recibido, se comprueba en qué momento ha grabado en la pizarra el comando que ha generado. Se considera que empieza un nuevo ciclo de control cuando se almacena en la pizarra el comando final que se va a enviar al robot. La figura 6.22 ilustra una pequeña muestra de los datos obtenidos durante la ejecución de una tarea de seguir contorno. Se puede observar que existen fuertes desviaciones sobre el comportamiento periódico previsto que se traducen en retrasos en la ejecu-

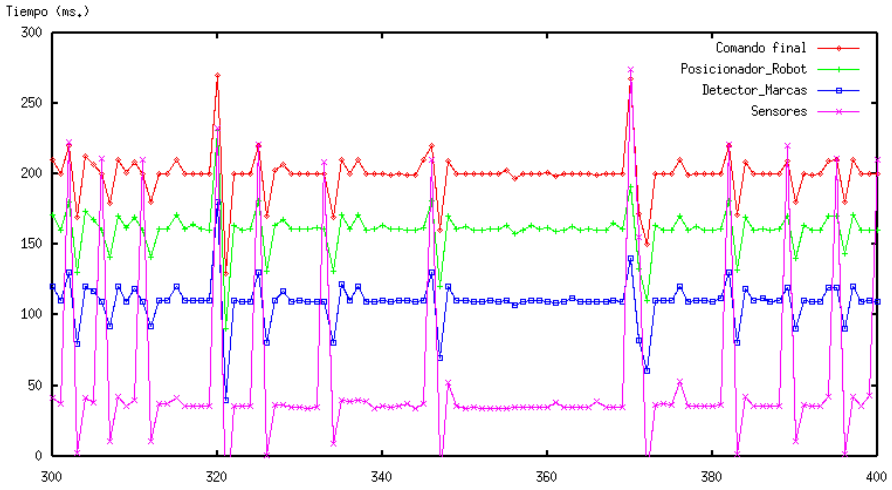

Figura 6.22: Ejemplo del tiempo de ejecución de las principales *OTs* del ciclo de control del especialista PILOTO con una frecuencia de adquisición de los datos sensoriales de 5 Hz y permitiendo ejecutar dos ciclos de control concurrentemente.

ción de algunos ciclos de control. En ocasiones estos retrasos pueden ser superiores a un segundo pero, en general, no llegan a comprometer la seguridad ni la fiabilidad en la operación del robot.

Existen varias causas para la aparición de estos retrasos. En primer lugar, el *Nomad 200* utiliza un sistema operativo que no es de tiempo real y no puede garantizar un tiempo de ejecución máximo para cada tarea (p.e., el envío de un mensaje). En segundo lugar, el tiempo empleado para recoger los datos sensoriales del robot tarda entre 40 y 90 ms de media, aunque puede llegar a bastantes centenas de milisegundos. Sin embargo, como la interfaz con el robot es software propietario, ésta no puede ser modificada.[8] En tercer lugar, el intercambio de información a través de mensajes KQML también enlentece ligeramente el sistema. La implementación de la pizarra de PILOTO como una memoria compartida ha mitigado en gran medida su impacto negativo, ya que sólo se utilizan mensajes para el intercambio de información de control.

Las medidas realizadas muestran que un mensaje KQML tarda una media de 20 ms en llegar a su destino, aunque sea el mismo ordenador en donde se ha generado, independientemente de los recursos computacionales. Como un ciclo de control sobre datos sensoriales básicos posee un mínimo de tres pasos secuenciales (figura 6.9) se

[8] Este es un motivo más para elegir siempre que sea posible software libre.

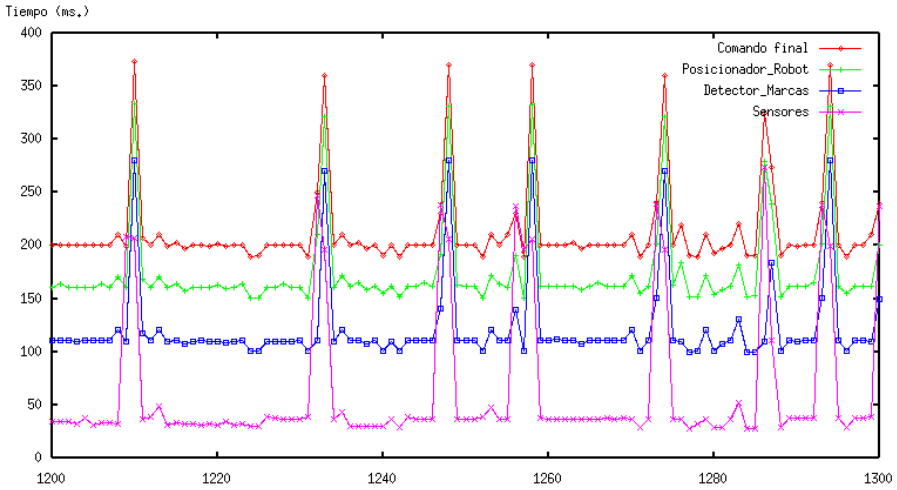

Figura 6.23: Ejemplo del tiempo de ejecución de las *OTs* del ciclo de control de PILOTO cuando se espera a procesar todos los datos del ciclo anterior.

necesitan un total de 6 mensajes de control entre órdenes de tarea y respuestas. Por lo tanto, el mínimo tiempo necesario para procesar una adquisición sensorial es de unos 120 ms (en realidad 140 ms porque el agente Efector es el más lento que los demás). De todos modos, lo grave es que si uno de los mensajes necesita más tiempo que la media entonces todo el ciclo se retrasa, lo que provoca los *picos* de la figura 6.22.

En principio, antes de que el ciclo de control pueda procesar una nueva adquisición sensorial, tiene que haber finalizado completamente el procesamiento de la anterior. Esta restricción significa que un pequeño retraso puntual en un ciclo de control hace que se dejen de procesar los datos del siguiente ciclo y provoca un retraso mucho mayor (figura 6.23), independientemente de que existan suficientes recursos computacionales para procesar todos los datos. Así, se puede apreciar que en la figura 6.23 el tiempo de ejecución de una *OT* entre ciclo y ciclo no suele ser menor que un cierto nivel de referencia, lo que sí ocurre en la figura 6.22. Del mismo modo, se puede comprobar que procesar dos adquisiciones sensoriales en un mismo ciclo de control permite amortiguar los retrasos y evitar su acumulación.

Para minimizar este problema se permite la ejecución simultánea de dos ciclos de control y procesar así dos adquisiciones sensoriales de forma concurrente. La integración de comandos de PILOTO resuelve los conflictos que surgen cuando en un mismo ciclo de control se procesan datos que pertenecen a distintas adquisiciones

sensoriales. Para evitar el colapso del sistema se ha implementado un mecanismo de salvaguarda que consiste en no procesar más datos (no activar más ciclos de control) cuando ya se están procesando dos adquisiciones. También se podría aplicar a cada agente para evitar enviarle más *OT*s mientras no haya procesado los datos del penúltimo (o antepenúltimo) ciclo de control. Si esta situación se repite muy a menudo, entonces es que existe un problema de fondo en el agente o en el sistema.

Monitorización

Por monitorización entendemos los mecanismos que proporciona una arquitectura de control para seguir su funcionamiento y registrar las incidencias y datos relevantes. Esta capacidad es muy importante tanto en la implementación y posterior depuración de la arquitectura como para explicar su comportamiento y entender por qué ha tomado cada decisión. En AFE-Robótica se han especificado distintos tipos de monitorización.

En el más primitivo se imprimen una serie de mensajes con los datos de interés de cada agente y de la propia arquitectura. Los mensajes se agrupan en varios niveles o categorías (todos, experto, usuario, normal, mínimo y ninguno) y se puede especificar el nivel de mensajes que se desea recibir. También existe la posibilidad de ejecutar cada agente en un terminal de texto y observar directamente los datos más relevantes de su ejecución (p.e., activación de un evento, marcas detectadas, recalibraciones, rutas, etc.).

Un segundo tipo de monitorización consiste en "dialogar" directamente con los componentes del sistema mediante mensajes KQML. Así, por ejemplo, se puede interrogar directamente a cualquier agente (sorteando la jerarquía de especialistas mediante *OTs* específicas) sobre su estado, pedir información sobre las tareas en ejecución, modificar el estado de una tarea (abortar o reanudar), etc. También se pueden consultar o modificar los datos que contienen las pizarras de los distintos especialistas.

El tercer tipo de monitorización consiste en representar en una interfaz gráfica los datos más relevantes de AFE-Robótica para la ejecución de una tarea. Para simplificar la implementación de AFE-Robótica, actualmente se utiliza la interfaz gráfica del *Nomad 200* y se "permite" que ciertos agentes (principalmente los de percepción) puedan representar en ella los datos que han generado: marcas del entorno, marcas detectadas, datos sensoriales, recalibraciones, etc. Otros agentes utilizan su propia interfaz para mostrar gráficamente sus resultados (p.e., el agente

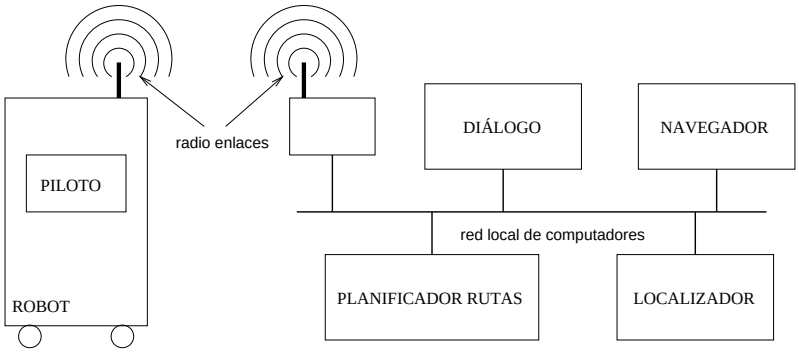

Figura 6.24: Posible distribución de los especialistas y agentes de AFE-Robótica en una red de computadores.

PLANIFICADOR_RUTA). En un futuro todas estas interfaces se integrarán en una sola dependiente del especialista DIÁLOGO.

El último tipo de monitorización está orientado principalmente al análisis en diferido de los resultados y su objetivo es almacenar toda la información necesaria para reconstruir fielmente la operación de la arquitectura y detectar posibles errores de funcionamiento. Además de los datos de interés, se registran todos los mensajes de control recibidos y enviados por cada agente, excepto los de PILOTO, porque no aportan información útil y el volumen de datos sería excesivo. Entre los datos almacenados están las medidas sensoriales, las procesadas por cada agente perceptual, las marcas detectadas, las recalibraciones realizadas, los comandos generados y los enviados al robot, el resultado final de cada tarea, las localizaciones realizadas, las rutas calculadas, etc. Cada dato está identificado por una etiqueta temporal.

Los mecanismos de monitorización también se pueden utilizar internamente para comprobar el correcto funcionamiento de los distintos componentes de la arquitectura. Por ejemplo, en la fase de inicialización cada agente comprueba su conexión con la pizarra de su especialista y cada agente de control chequea todos sus agentes. También se podrían utilizar para detectar cuando un agente deja de estar operativo. Si su agente de control no pudiera reinicializarlo, entonces tendría que eliminarlo y ejecutarlo de nuevo.

Ejecución distribuida

El *Nomad 200* se puede conectar a una red de computadores mediante radio enlaces (figura 6.24), por lo que hemos estudiado la posibilidad de ejecutar la arquitectura de forma distribuida. El criterio elegido para establecer la distribución es

que las funciones vitales y más importantes deben ser ejecutados en los computadores del robot, para asegurar que en caso de problemas (p.e., falta de cobertura del radio enlace) se pueda garantizar la integridad del robot y la suficiente capacidad de reacción.

Por todos estos motivos el especialista PILOTO debe ser ejecutado en el robot, lo mismo que todos los comportamientos reactivos. En cuanto a los agentes perceptuales, hay que considerar que se activan prácticamente en todos los ciclos de control, es decir, con cada nueva adquisición sensorial, por lo que sería conveniente ejecutarlos en el mismo computador que PILOTO, máxime cuando la pizarra se implementa en una memoria compartida para una ejecución más eficiente. Algo similar ocurre con los comportamientos voluntarios.

Siguiendo el criterio establecido sería deseable que NAVEGADOR también se ejecutase en el robot para asegurar una completa autonomía en su funcionamiento y evitar que en caso de pérdida de cobertura del radio enlace se produzca un "descabezamiento" de la arquitectura. El resto de sus agentes sí se podrían ejecutar remotamente en la red, o bien en el mismo robot, pero en versiones simplificadas que utilicen pocos recursos computacionales.[9]

Portabilidad

La portabilidad de una aplicación software es la posibilidad o facilidad para su ejecución en distintas plataformas de computación. Respecto a AFE-Robótica, cabe destacar que ha sido desarrollada con un lenguaje orientado a objetos estándar (C++), bajo un sistema operativo abierto (GNU/Linux) y utilizando herramientas de software libre (editores, compiladores, depuradores, etc.) totalmente validados en un amplio abanico de plataformas y con resultados claramente contrastados que aseguran su portabilidad. Las librerías externas utilizadas (LEDA y la implementación del lenguaje de comunicaciones KQML) tienen las mismas propiedades, aunque no han sido tan ampliamente validadas.

Sin embargo, en una arquitectura de control para robots móviles también se puede hablar, forzando la acepción inicial del término, de una portabilidad hardware, es decir, de la capacidad o facilidad de la arquitectura para controlar distintas plataformas físicas o robots móviles. Una propiedad importante de AFE-Robótica es que sólo los agentes de bajo nivel dependen directamente de las características físicas del

[9] En la implementación actual, AFE-Robótica se puede ejecutar completamente en el computador del *Nomad 200*.

robot, mientras que el resto de agentes operan sobre datos y ejecutan tareas cada vez más abstractas y generales, según se sube en la jerarquía de especialistas.

Para estudiar las modificaciones que implicaría en AFE-Robótica un cambio de robot, nos vamos a centrar en aquellos robots móviles con los que hemos tenido oportunidad de trabajar y sobre los cuales poseemos cierta experiencia. En primer lugar, como todos los robots de la misma familia que el *Nomad* son compatibles entre sí, no es de esperar ningún cambio significativo en la arquitectura. El cambio a un robot móvil cilíndrico (p.e., el *B21* o el *B14*) tampoco implica muchas modificaciones, ya que la disposición de los sensores y el tipo de movimientos serían similares a los del *Nomad 200*. Sólo se deberían adaptar los agentes Sensores y Efectores a la nueva interfaz del robot. Sin embargo, como los *Bxx* no pueden girar la torreta independiente de la base es aconsejable utilizar un sensor láser con un gran ángulo de abertura (p.e., el SICK PLS), con la consiguiente modificación de los algoritmos de percepción. No obstante, es de esperar que este cambio produzca una notable mejoría dadas las pobres prestaciones, ya comentadas, del láser del *Nomad 200*.

Los robots móviles rectangulares (p.e., *Pioneer*) dan más problemas, no tanto por la diferente disposición de los sensores sino que su forma obliga a modificar todos los comportamientos reactivos y voluntarios implementados. La razón es que, aunque sean robots holonómicos y puedan girar sobre un punto, su proyección sobre el suelo cambia mientras giran. Otros robots móviles (p.e., *Robuter*) tienen la dificultad añadida de operar bajo sistemas operativos propietarios (p.e., *Albatros*) que no suelen ser compatibles con GNU/Linux, lo que impone cambios importantes en todo el software.

Reutilización de componentes

Una de las características que más se han potenciado en el diseño de AFE-Robótica es la reutilización de componentes y la definición de clases genéricas que luego pudiesen ser particularizadas para cada caso. Así, por ejemplo, todos los especialistas poseen la misma estructura: una pizarra, un agente de control, módulos de comunicaciones y un conjunto de agentes. A su vez, cada agente es una réplica de un especialista, excepto los agentes terminales que sólo disponen de módulos de comunicaciones y de control. También los módulos de comunicaciones, tanto de datos como de control, son genéricos y se pueden utilizar para cualquier agente/especialista. Lo mismo ocurre con el agente de control y sus componentes (módulo de comunicaciones de agentes, gestores de eventos y de tareas, ejecutor de

planes). Por otra parte, la estructura de las pizarras, los datos y los eventos también son las mismas para todos los especialistas, salvo que se aplican y manejan datos diferentes. De igual forma, aunque la estructura básica de las tareas y de los planes es común, su contenido cambia según el especialista.

6.4 Discusión

Con los resultados obtenidos y comentados en las secciones anteriores, disponemos de suficientes argumentos para hacer una valoración global de la arquitectura propuesta y de la implementación realizada. También se puede comprobar hasta qué punto se han alcanzado los objetivos propuestos inicialmente y cuáles han sido las limitaciones más importantes que se han encontrado.

Si analizamos únicamente las prestaciones obtenidas en los experimentos de navegación que hemos realizado, éstas son algo inferiores a las recogidas en algunos trabajos publicados recientemente (Konolige y Myers, 1998; Koenig y Simmons, 1998; Andersson y col., 1999; Thrun y col., 1999; Wijk y Christensen, 2000). Esta diferencia se debe, fundamentalmente, a la menor capacidad sensorial y, por tanto, de percepción, disponible en el *Nomad 200*. Por otra parte, el diseño y la síntesis de AFE-Robótica no ha ido dirigido a obtener un perfecto y completo sistema de navegación sino a desarrollar una nueva arquitectura de control para robots móviles capaz de operar en entornos complejos, dinámicos e imposibles de modelar completamente y que, al mismo tiempo, proporcionase mecanismos para facilitar su ampliación modular, característica fundamental para desarrollar sistemas de control cada vez más autónomos, inteligentes y complejos.

De los resultados obtenidos se puede deducir que la teórica rigidez en la especificación y ejecución de las tareas en la arquitectura AFE no ha sido tan grave como se había supuesto en un principio y no ha limitado en ningún momento el diseño del sistema implementado. En gran parte debido a los mecanismos que incorpora la propia arquitectura y que permiten que los datos y los eventos se puedan intercambiar libremente y sin restricciones entre todos los especialistas.

Respecto a la implementación de AFE-Robótica, es oportuno comentar algunos aspectos. En primer lugar, la utilización de mensajes en KQML como protocolo de comunicaciones ha aportado una gran flexibilidad, sencillez y portabilidad, pero ha introducido ciertas limitaciones en el tamaño de los mensajes, ha reducido la eficiencia de las comunicaciones y ha limitado la velocidad de ejecución en tiempo

real. En segundo lugar, se ha implementado una versión simplificada de los agentes de control que ha servido para ejecutar las tareas encomendadas, pero que, posiblemente, sea insuficiente para realizar otras tareas más abstractas y generales, por ejemplo, la planificación de misiones.

Obviamente, el rendimiento final del sistema depende en gran medida de las implementaciones de los agentes. Así, como ya se ha discutido después de validar cada agente, algunos de ellos son más bien soluciones de compromiso que no llegan a cumplir perfectamente todos los requisitos iniciales. Por ejemplo, la detección de las puertas no es siempre muy fiable, lo que obliga al sistema a verificar constantemente la posición del robot y de las puertas para establecer la correcta finalización de las tareas en ejecución. Tampoco facilita los desplazamientos en aquellos espacios donde una correcta y fiable detección de las puertas es fundamental (p.e., intersecciones de corredores). Al mismo tiempo, también se detectan un gran número de falsos positivos y no siempre se percibe cuando una puerta está cerrada, lo que tiende a engañar al sistema.

Por otra parte, la localización global aún es demasiado frágil y poco consistente, mientras que la utilización del mapa del entorno se basa más en las distancias métricas que en las características topológicas. Además, debido a la poca fiabilidad de la percepción, tampoco se han implementado mecanismos para actualizar el mapa del entorno según las modificaciones detectadas (p.e., una puerta cerrada o un corredor bloqueado), por lo que el sistema aún no es capaz de resolver estas contingencias.

Para finalizar, destacar que el funcionamiento final del sistema está condicionado por las limitaciones de la plataforma física elegida (un *Nomad 200*), es decir, de los sensores (láser, US e IR) y de los efectores. Estas limitaciones han sido ampliamente comentadas en el primer capítulo, por lo que ahora sólo destacaremos dos puntos. Primero, el hecho de que un mejor sensor láser permitiría una detección más fiable de un número mayor de marcas (p.e., intersecciones entre corredores), lo que redundaría en un sistema más robusto e inteligente, ya que podría utilizar más condiciones para verificar el correcto funcionamiento de cada tarea. Segundo, que el movimiento independiente de la torreta respecto a la base nos ha permitido orientar el sensor láser hacia las paredes mientras el robot las sigue, pero al tiempo nos ha planteado problemas muy serios para eliminar los errores de la odometría en ambos ejes, sobre todo de la base.

A pesar de todo, el sistema implementado es suficientemente robusto para desplazarse de forma autónoma por el interior de un edificio de oficinas, propósito que

nos habíamos fijado para el prototipo desarrollado. A partir de esta funcionalidad, básica para un robot móvil autónomo (RMA), el sistema se puede adaptar con muy poco esfuerzo a muchas otras tareas, siempre con las limitaciones que impone la plataforma física actual. Entre otras podemos destacar las tareas de reparto (p.e., simplemente añadiendo una bandeja al robot ya podría repartir el correo, libros, etc.), la vigilancia de la planta de un edificio o las visitas guiadas por el interior de un edificio (Burgard y col., 1998; Thrun y col., 1999). Por otra parte, añadiendo el hardware necesario (p.e., un brazo articulado, cámaras, micrófonos, etc.), el RMA podría desplazarse de un modo aún más autónomo (p.e., abriendo las puertas cerradas (Nagatani y Yuta, 1998)), así como realizar tareas más complejas,como son: buscar y recolectar objetos para depositarlos en un lugar predeterminado (Connell, 1990; Firby y col., 1998), comunicarse con los seres humanos que cohabitan en su entorno (Burgard y col., 1998), detectar y seguir personas (Konolige y col., 1997), etc.

El prototipo implementado se puede mejorar en varias direcciones. Respecto a los comportamientos implementados, SEGUIR_CONTORNO se podría hacer más rápido y robusto o, mejor aún, diseñar un nuevo comportamiento especializado en seguir corredores. Algunas tareas de bajo nivel también deberían mejorarse, principalmente el tratamiento de los choques (p.e., hacer que el comportamiento reactivo siga actuando algo después de desaparecer el contacto) y el cruce de una puerta (p.e., dividir el proceso en cuatro fases y referir su finalización al aumento de la distancia a ambos lados del robot, es decir, a traspasar el espacio angosto que caracteriza a una puerta). Por último, se deberían definir tareas específicas que permitiesen al robot desplazarse en espacios reducidos o poco estructurados, como habitaciones, aulas y laboratorios. Por ejemplo, utilizando el comportamiento MOVER_A en conjunción con el mapa local de obstáculos o con marcas locales detectadas con los sensores de US (Wijk y Christensen, 2000).

Otro punto crucial es la detección más fiable y robusta, gracias a nuevos sensores, de marcas que permitan una mejor recalibración odométrica y que proporcionen más y mejor información topológica. De esta manera se podrían utilizar más condiciones de finalización de tipo topológico (p.e., fin de corredor e intersección de corredores) y mejorar la ejecución de ciertas tareas (p.e., cruzar puerta) y los desplazamientos, tanto en espacios reducidos como en lugares amplios o con pocas referencias (p.e, salas e intersecciones entre corredores).

Una mejor percepción de las marcas del entorno permitiría localizar el robot a nivel global utilizando información topológica en vez de la posición odométrica.

También posibilitaría confeccionar un mapa topológico del entorno y mantenerlo constantemente actualizado, tarea ingente, aunque imprescindible en un entorno dinámico. Para ello, además, sería imprescindible incorporar estrategias de exploración, tanto a nivel local como global o topológico, y definir y responder a nuevos eventos (p.e., ROBOT PERDIDO).

6.4.1 Consideraciones finales

Es difícil establecer comparaciones entre arquitecturas aunque se estén aplicando a las mismas tareas y, por tanto, también lo es el valorar las distintas mejoras introducidas por cada autor y determinar hasta qué punto son interesantes y prácticas. No parece haber otra solución que proponer algún grupo de *benchmarks* o pruebas y establecer una métrica con la cual comparar cuantitativamente distintos sistemas.

La idea de las pruebas no es nueva. Pensemos, por ejemplo, en las competiciones periódicas donde concurren distintos equipos, tales como los concursos que se organizan de forma paralela a las conferencias anuales de la AAAI o la *RoboCup* (Rob, 1997), donde varios equipos de robots móviles divididos en tres categorías compiten en un juego parecido al fútbol.

De todos modos, estas pruebas se centran en elegir un ganador y no se ha definido una métrica para evaluar el rendimiento de los distintos sistemas. Lo ideal sería diseñar una serie de pruebas que cada investigador pudiese realizar con unos medios relativamente modestos y que sirviese para validar su sistema particular y comparar los resultados obtenidos con los de otros autores. Éste es el método de trabajo comúnmente utilizado y aceptado en el resto de campos científicos y con el cual se abriría la puerta a la confirmación de experimentos por parte de equipos independientes (Nehmzow, 2000).

Sin embargo, esta filosofía plantea retos muy importantes y que en la práctica pueden resultar muy difíciles de superar. En primer lugar, existe una falta de homogeneidad y de estandarización en las plataformas hardware y software que se utilizan en el campo de la robótica. En segundo lugar, cuando se trabaja con robots móviles un poco grandes no siempre es viable modificar las características de los entornos con los que se puede experimentar para replicar unos valores estándar predefinidos. Por lo tanto, la gran cantidad de parámetros de un entorno (p.e., materiales, formas, texturas, colores, etc.), dificultan enormemente la extrapolación de los resultados obtenidos.

En tercer lugar, está el problema de la selección de las tareas que deben formar parte de los *benchmark*s. Como ya se ha comentado al hablar sobre los aspectos funcionales de un robot móvil autónomo, falta un estudio en profundidad sobre este tema que sirva para identificar, definir y clasificar las funcionalidades y capacidades que debe poseer un robot móvil. Una clasificación de este tipo tendría que estar consensuada entre todos los investigadores del campo y obtener un respaldo suficientemente amplio antes de ser adoptada. Aunque, sin lugar a dudas, la navegación estaría incluida en esa categoría.

En ausencia de líneas bien trazadas que permitan una evaluación comparativa rigurosa entre cada nueva arquitectura propuesta y sus predecesoras más valiosas, hemos intentado ser suficientemente exhaustivos en los estudios y pruebas realizadas tras la síntesis de AFE-Robótica, operando en condiciones y entornos reales. Creemos que los resultados obtenidos, y parcialmente mostrados en esta memoria, demuestran las aportaciones de nuestra propuesta y el interés por seguir trabajando en la línea de investigación abierta.

Conclusiones y principales aportaciones

El diseño de una arquitectura de control para robótica móvil plantea muchos e importantes retos, ya que los robots operan e interaccionan con entornos complejos, dinámicos e impredecibles a través de sensores y actuadores imperfectos y, al mismo tiempo, deben planificar la consecución de múltiples objetivos, a veces contradictorios entre si, ser autónomos y operar en tiempo real. Por todo ello, el dominio de la robótica móvil es ideal para estudiar, diseñar y sintetizar sistemas inteligentes autónomos capaces de ejecutar tareas complejas en tiempo real en entornos no estructurados.

El objetivo del presente trabajo ha sido desarrollar una arquitectura de control para robots móviles autónomos. Se han analizado los diversos paradigmas de control existentes en robótica móvil y sus principales realizaciones. A falta de estudios exhaustivos sobre las propiedades que han de destacarse en una arquitectura de control para un robot móvil autónomo (RMA), se han reseñado las que consideramos más importantes, analizando bajo esta perspectiva las prestaciones y limitaciones que presentan dichos paradigmas.

Nuestra arquitectura trata de dar respuesta a algunas de ellas. Concretamente, se ha prestado especial atención a la robustez, reactividad e integración de comportamientos, así como a la flexibilidad para su fácil adaptación a nuevos requerimientos del sistema y a la facilidad para la incorporación de nuevas funcionalidades, cualidades estas últimas imprescindibles para el desarrollo modular de sistemas de control cada vez más inteligentes y complejos.

La arquitectura propuesta enfatiza la encapsulación de datos, tareas y control en módulos independientes, especializados en funciones específicas del sistema, denominados especialistas. Para lograr una implementación más regular, los especialistas se implementan mediante una arquitectura basada en pizarra. Sin embargo, cada

agente de un especialista se puede implementar, a su vez, como un especialista, por lo que la arquitectura se va replicando en distintos niveles, razón por la cual la hemos denominado Arquitectura Fractal de Especialistas (AFE). Las características de AFE permiten que pueda ser utilizada en distintos ámbitos. Su particularización a la robótica móvil se ha denominado AFE-Robótica.

Para validar el modelo de arquitectura, y probar así su utilidad y viabilidad, el único método práctico es la validación experimental. Como una arquitectura es un mero cascarón vacío, se ha hecho necesario realizar una implementación "completa" de la misma. Esta tarea resulta enormemente compleja, ya que además de especificar e implementar las comunicaciones, el intercambio de datos o el flujo de control, también se deben incluir todos los componentes que permiten que un robot móvil pueda funcionar en un entorno real, es decir, desde los algoritmos de percepción hasta los comportamientos de bajo nivel del robot, pasando por todas las funciones de control. En este punto se debe descender hasta aspectos de bajo nivel e, incluso, resolver las singularidades o peculiaridades propias del entorno o de la plataforma física utilizada.

La síntesis de la arquitectura propuesta se ha realizado sobre una función fundamental para cualquier dispositivo móvil autónomo, la navegación, en un entorno fácilmente asequible para la experimentación, el interior de un edificio de oficinas, y con un robot móvil comercial sencillo y completo, un *Nomad 200*. El prototipo de AFE-Robótica desarrollado es un compromiso entre la implementación factible en un plazo de tiempo razonable y con recursos limitados, la validación de la arquitectura propuesta y la capacidad de realizar tareas útiles, suficientemente complejas y usuales en un robot móvil.

Se ha elegido la tarea de navegación porque es fundamental para la operatividad y autonomía de un robot móvil y porque aún no es una tarea bien resuelta en entornos no estructurados, dinámicos y no totalmente modelables. Se ha desarrollado una navegación de tipo topológico basada en las características reseñables (marcas) del entorno. Para simplificar la implementación se ha partido de un mapa *a priori* del entorno. También se ha optado por no implementar los especialistas de más alto nivel de abstracción de la jerarquía de especialistas. Por último, la implementación de los distintos agentes ha sido el resultado de un compromiso entre simplicidad, robustez, eficiencia y fiabilidad, aunque también han influido decisivamente las limitaciones de los sensores y actuadores del robot móvil utilizado.

El sistema ha sido validado experimentalmente en la sede del Departamento de

Electrónica y Computación de la Universidad de Santiago de Compostela mediante la ejecución de múltiples desplazamientos bajo condiciones diversas en el entorno. Los resultados obtenidos han sido satisfactorios desde distintos puntos de vista. Se ha demostrado que el sistema es muy robusto, tanto a nivel de arquitectura como en la ejecución de las tareas (es decir, en sus desplazamientos); Que es flexible y facilita las modificaciones, tanto del tipo de datos y su representación, como de los agentes, las tareas, etc.; Que es capaz de integrar adecuadamente los comportamientos reactivos, deliberativos y planificados, tanto en la realización de las tareas como en las distintas contingencias que puedan surgir durante la ejecución de las mismas; No menos importante, se ha probado que puede operar en tiempo real.

La especialización de AFE-Robótica conlleva una división jerárquica del sistema en niveles de responsabilidad, favorece la implementación de tareas complejas y simplifica tanto la implementación como la ejecución de mecanismos reactivos. También reduce las comunicaciones entre módulos y permite la ejecución distribuida de la arquitectura, lo que, a su vez, hace posible acceder a una mayor potencia de cálculo. Por último, la especialización también incrementa la escalabilidad y la flexibilidad del sistema y mejora su capacidad de adaptación a nuevos requerimientos.

Las prestaciones obtenidas no han sido tan buenas como las esperadas, fundamentalmente debido a las limitaciones de la plataforma física, y más concretamente, del sensor láser y de la independencia de giro entre la torreta y la base. En el primer caso porque la detección de las marcas del entorno (sobre todo de las puertas) es poco fiable, y en el segundo caso porque hace más difícil la corrección de los errores en la posición odométrica del robot. A pesar de ello, se han conseguido unos resultados alentadores, en parte por las bondades de la arquitectura.

Las cualidades de AFE-Robótica, principalmente su gran flexibilidad y excelente escalabilidad, y el hecho de que ya esté integrada la funcionalidad más importante para un RMA, la navegación, abren un amplio abanico de futuras líneas de investigación. A continuación destacaremos las más importantes.

En primer lugar, convendría mejorar lo que ya se ha apreciado y comentado como tal en el prototipo implementado. Entre otras cosas, definir más planes de contingencia, ejecutar más rápido el ciclo de control del especialista PILOTO, incorporar nuevos comportamientos, mejorar la localización del robot y la detección de marcas, etc.

También se pueden rentabilizar las prestaciones de AFE-Robótica en distintos aspectos. Por ejemplo, adaptando la arquitectura para controlar otros robots mó-

viles. El primer paso será un robot *Pioneer 2 AT*, recientemente adquirido por nuestro grupo de investigación. De igual forma, también se puede pensar en operar en entornos diferentes al previsto, por ejemplo en exteriores.

Otra posibilidad interesante consiste en probar y comparar diferentes técnicas para resolver la tarea de navegación, por ejemplo, mediante diferentes representaciones del entorno. La flexibilidad de AFE-Robótica facilita la realización de los cambios y permite fijar un marco común para comparar la eficacia relativa entre los distintos métodos. Esta característica es muy importante para establecer y realizar *benchmarks* o bancos de pruebas.

Otro reto es modificar la implementación de la arquitectura, de forma que sea más fácil e intuitiva de usar para personal no especializado (p.e., estudiantes). Entre otras cosas, añadiendo nuevos y más robustos interfaces y desarrollando herramientas que permitan definir y modificar gráficamente los planes de control, los eventos, etc.

También se puede aplicar la arquitectura a otras tareas añadiendo las competencias y funcionalidades necesarias (p.e., manipulación, visión artificial, etc.). A a su vez, las nuevas funciones requerirán nuevos sensores (micrófonos, cámaras de visión, etc.), actuadores (brazo, cabeza robótica, etc.) y computadores. Entre las nuevas competencias estaría la creación automática del mapa topológico del entorno y su continua actualización para capturar las modificaciones que en él se produzcan (p.e., corredores bloqueados o puertas cerradas).

Las nuevas funcionalidades permitirán mejorar la robustez del sistema al aumentar la capacidad sensorial, perceptual y de modificación del entorno que rodea el robot y, probablemente, añadir mayor redundancia en la ejecución de las tareas. Sin embargo, la necesidad de ingentes recursos computacionales pondrá a prueba la naturaleza distribuida de la arquitectura.

Por último, también nos parece interesante estudiar la cooperación entre varios robots móviles para ejecutar tareas en equipo (p.e., creación conjunta y mantenimiento de un mapa global del entorno, vigilancia de una cierta región, juegos de equipo, etc.). Sería interesante estudiar tanto colectivos compuestos por varios robots totalmente autónomos e independientes entre sí, como sistemas formados por varios robots móviles independientes a nivel físico pero integrados en una única arquitectura de control.

Como no podía ser de otro modo, son más las ideas que han surgido para continuar el trabajo que el propio trabajo hecho. A ir concretando parte de estas ideas dedicaremos nuestro esfuerzo a partir de ahora.

Glosario de abreviaturas

AFE	**A**rquitectura **F**ractal de **E**specialistas
GPS	***G****lobal* ***P****ositioning* ***S****ystem*
IR	**I**nfra**r**rojos
KQML	***K****nowledge* ***Q****uery and* ***M****anipulation* ***L****anguage*
LEDA	***L****ibrary of* ***E****fficient* ***D****ata types and* ***A****lgorithms*
MC	**M**ódulo de **C**omunicaciones
MCC	**M**ódulo de **C**omunicaciones de **C**ontrol
MCA	**M**ódulo de **C**omunicaciones de **A**gentes
MCD	**M**ódulo de **C**omunicaciones de **D**atos
MGD	**M**apa **G**lobal **D**istribuido
MPL	**M**apa **P**erceptual **L**ocal
MPLP	**M**apa **P**erceptual **L**ocal **P**revio
OT	**O**rden de **T**area
RMA	**R**obot **M**óvil **A**utónomo
ROT	**R**espuesta **O**rden de **T**area
US	**U**ltra**s**onidos

Bibliografía

Nomad 200 User's Guide. Nomadic Technologies, Mountain View CA, 1993.

RangeLAN2 User's Guide. Proxim, Mountain View CA, 1993.

Robuter Manual. Robosoft, Bidart, France, 1994.

B21 User's Guide. Real World Interface, Jaffrey NH, 1995.

The Sensus 500 User's Guide. Nomadic Technologies, Mountain View CA, 1995.

Aibo. Sony, http://www.aibo.com, 1997.

"RoboCup." http://www.robocup.org, 1997.

"Sahabot2." http://www.ifi.unizh.ch/ailab/projects/sahabot/ sahabot2/index.html, 1997.

Pioneer 2 Mobile Robot Operations Manual. ActivMedia Robotics, Peterborough, NH, 2000.

PMS Laser Manual. SICK, Germany, 2000.

Adams, B., Breazeal, C., Brooks, R. y Scassellati, B. "Humanoid Robots: A New Kind of Tool." *IEEE Intelligent Systems*, 15(4):25–31, 2000.

Agre, P. E. y Chapman, D. "What Are Plans for?" *Robotics and Autonomous Systems*, 6:17–34, 1990.

Albus, J. S. "RCS: A Reference Model Architecture for Intelligent Systems." En H. Hexmoor y D. Kortenkamp (eds.), *Lessons Learned from Implemented Software Architectures for Physical Agents*, Papers from the 1995 AAAI Symposium, págs. 1–6. AAAI Press, 1995.

Albus, J. S., Lumia, R. y McCain, H. G. "Nasa/nbs standard reference model for telerobot control system architecture (NASREM)." Inf. Téc. 1235, NASA SS-GFSC-0027, National Bureau of Standards, 1986.

Andersson, M., Orebäck, A., Lindström, M. y Christensen, H. I. "ISR: An Intelligent Service Robot." En H. I. Christensen (ed.), *Sensor based intelligent robots: international workshop*, tomo 1724 de *Lecture Notes on Computer Science*, págs. 287–310, 1999.

Angulo Usátegui, J. M., Romero Yesa, S. y Angulo Martínez, I. *Microrobotica. Tecnologia, Aplicaciones y Montaje Practico*. ITP Paraninfo, 1999.

Arkin, R. C. *Towards Cosmopolitan Robots: Intelligent Navigation of a Mobile Robot in Extended Man-made Environments*. Tesis Doctoral, U. Massachusetts. Dpt. of Computer and Information Science, 1987.

Arkin, R. C. "Motor Schema-Based Mobile Robot Navigation." *The International Journal of Robotics Research*, 8(4):92–112, 1989a.

Arkin, R. C. "Towards de Unification of Navigational Planning and Reactive Control." En *AAAI Spring Symposium on Robot Navigation*, págs. 1–5, 1989b.

Arkin, R. C. "Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation." *Robotics and Autonomous Systems*, 6:105–122, 1990a.

Arkin, R. C. "The Impact of Cybernetics on the Design of a Mobile Robot System: A Case Study." *IEEE Transactions on Systems, Man and Cybernetics*, 20(6):1245–1257, 1990b.

Arkin, R. C. *Behavior-based Robotics*. MIT Press, 1998.

Bares, J. E. y Wettergreen, D. S. "Dante II: Technical Description, Results and Lessons Learned." *The International Journal of Robotics Research*, págs. 621–649, 1999.

Bares et al, J. "Ambler, an Autonomous Rover for Planetary Exploration." *IEEE Computer*, 1989.

Barrientos, A., Peñín, L. F., Balaguer, C. y Aracil, F. *Fundamentos de robótica*. McGraw-Hill, 1997.

Bauer, R. "Active manoeuvres for supporting the localisation process of an autonomous mobile robot." *Robotics and Autonomous Systems*, 16:39–46, 1995.

Bolles, R. C., Baker, H. H. y Konolige, K. G. "Local environment modeling through integrated stereo and motion analysis." En H. Bunke, T. Kanade y H.Ñoltemeier (eds.), *Modelling and Planning for sensor based Intelligent Robot Systems*, págs. 311–324. World Scientific, 1995.

Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D. P. y Slack, M. G. "Experiences with an Architecture for Intelligent, Reactive Agents." *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):237–256, 1997.

Bonasso, R. P. y Kortenkamp, D. "Using a Layered Control Architecture to Alleviate Planning with Incomplete Information." En *Proc. AAAI 1996 Spring Symposium on Planning with Incomplete Information for Robot Problems*, 1996.

Bonasso, R. P., Kortenkamp, D., Miller, D. P. y Slack, M. G. "Experiences with an Architecture for Intelligent, Reactive Agents." En M. Wooldridge, J. P. Mueller y M. Tambe (eds.), *Intelligent Agents II: Agents Theories, Architectures and Languages*. Springer Verlag, 1995.

Borenstein, J., Everett, B. y Feng, L. *Navigating Mobile Robots: Systems and Techniques*. AK Peters, 1996.

Borenstein, J. y Koren, Y. "Histogramic in-motion mapping for mobile robot obstacle avoidance." *IEEE Transactions on Robotics and Automation*, 7(4):535–539, 1991a.

Borenstein, J. y Koren, Y. "The vector field histogram - fast obstacle avoidance for mobile robots." *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991b.

Brady, M. "Artificial Intelligence and Robotics." *Artificial Intelligence*, 26:79–121, 1986.

Braunstingl, R., Mujika, J. y Uribe, J. P. "A wall following robot with a fuzzy logic controller optimized by a genetic algorithm." En *Proceedings of the Fourth IEEE International Conference on Fuzzy Systems and the Second International Fuzzy Engineering Symposium*, tomo 5, págs. 77–82. Yokohama (Japan), 1995.

Breazeal, C. *Sociable Machines: Expressive Social Exchange Between Humans and Robots*. Tesis Doctoral, Electrical Engineering and Computer Science. MIT, 2000.

Brooks, R. A. "A Robusted Layered Control System for a Mobile Robot." *IEEE Journal Robotics and Automation*, RA-2:14–24, 1986.

Brooks, R. A. "A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network." *Neural Computation*, 1:253–262, 1989.

Brooks, R. A. "The Behavior Language: User's guide." Inf. Téc. AI Memo 1227, MIT, 1990a.

Brooks, R. A. "Elephants Don't Play Chess." *Robotics and Autonomous Systems*, 6:3–15, 1990b.

Brooks, R. A. "Integrated Systems Based on Behaviors." *SIGART Bulletin*, 2(4):46–50, 1991a.

Brooks, R. A. "Intelligence without representation." *Artificial Intelligence*, 47:139–160, 1991b.

Brooks, R. A. y Rosenberg, C. "L - A Common Lisp for embedded systems." En *proceedings of the Lisp Vendors and Users Conference*, 1995.

Bugarín, A., Cariñena, P., Félix, P. y Barro, S. *Fuzziness in Petri Nets*, tomo 22 de *Studies in Fuzziness and Soft Computing*, cap. Reasoning with Fuzzy Temporal Rules on Petri Nets, págs. 174–202. Physica-Verlag, 1999.

Burgard, W., Cremers, A., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W. y Thrun, S. "The Interactive Museum Tour-Guide Robot." En *proceedings of international conference AAAI*, 1998.

Burgard, W., Fox, D., Hennig, D. y Schmidt, T. "Position Tracking with Position Probability Grids." En *EUROBOT'96. Proc. first euromicro workshop on advanced mobile robots*, 1996.

Cao, Y. U., Fukunaga, A. S. y Kahng, A. B. "Cooperative Mobile Robotics: Antecedents and Directions." *Autonomous Robots*, 4:1–23, 1997.

Carver, N. y Lesser, V. "The Evolution of Blackboard Control Architectures." Inf. Téc. 92-71, Computer and Information Science Dpt. Univ. Massachusetts, 1992.

Castellano, G., Attolico, G. y Distante, A. "Automatic generation of fuzzy rules for reactive robot controllers." *Robotics and Autonomous Systems*, 22:133–149, 1997.

Castellanos, J. A. y Tardós, J. D. *Mobile Robot Localization and Map Building. A Multisensor Fusion Approach*. Kluwer, 1999.

Castellanos, J. A., Tardós, J. D. y Schmidt, G. "Building a global map of the environment of a mobile robot: The importance of correlations." En *Proc. of the 1997 IEEE Conference on Robotics and Automation*, 1997.

Chatila, R. "Deliberation and reactivity in autonomous mobile robots." *Robotics and Autonomous Systems*, 16:197–211, 1995.

Chatterjee, R. y Matsuno, F. "Use of single side reflex for autonomous navigation of mobile robots in unknown environments." *Robotics and Autonomous Systems*, 35:77–96, 2001.

Cho, D. W. "Certainty grid representation for robot navigation by a Bayesian method." *Robotica*, 8:159–165, 1990.

Chong, K. S. y Kleeman, L. "Sonar based map building for a mobile robot." En *Proc. of the IEEE Conference on Robotics and Automation*, 1997.

Connell, J. "A Colony Architecture for an Artificial Creature." Inf. Téc. 1151, MIT Artificial Intelligence Laboratory, 1989.

Connell, J. H. *Minimalist Mobile Robotics. A Colony-style Architecture for an Artificial Creature*. Academic Press, 1990.

Connell, J. H. "SSS: A Hybrid Architecture Applied to Robot Navigation." En *Proc. 1992 IEEE International Conference on Robotics and Automation*, 1992.

Corkill, D. "Design alternatives for parallel and distributed blackboard systems." En V. Jagannathan, R. Dodhiawala y L. Baum (eds.), *Blackboard Architectures and Applications*, tomo 3 de *Perspectives in Artificial Intelligence*, págs. 99–136. Academic Press, 1989.

Corkill, D. "Blackboard systems." *AI Expert*, 9(6):40–47, 1991.

Coste-Manière, E. y Simmons, R. "Architecture, the Backbone of Robotic Systems." En *Proceedings of the IEEE International Conference on Robotic and Automation*. San Francisco, CA, 2000.

Cox, I. J. "Blanche - an experiment in guidance and navigation of an autonomous robot vehicle." *IEEE Transactions on Robotics and Automation*, 7(2):193–204, 1991.

Craig, I. *Blackboard Systems*. Ablex Plublishing, 1995.

Dedeoglu, G., Mataric, M. J. y Sukhatme, G. S. "Landmark-based Matching Algorithm for Cooperative Mapping by Autonomous Robots." En *Proceedings of Mobile Robots XIV SPIE'99*, págs. 129–139, 1999.

Dedeoglu, G. y Sukhatme, G. S. "Landmark-based Matching Algorithm for Cooperative Mapping by Autonomous Robots." En *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems*, 2000.

del R. Millán, J. "Reinforcement learning of goal-directed obstacle-avoiding reaction strategies in an autonomous mobile robot." *Robotics and Autonomous Systems*, 15:275–299, 1995.

Demirli, K. y Türkçen, I. B. "Sonar based mobile robot localization by using fuzzy triangulation." *Robotics and Autonomous Systems*, 33:109–123, 2000.

Dillmann, R., Kaiser, M., Wallner, F. y Weckesser, P. "PRIAMOS: an advanced mobile system for service, inspection, and surveillance tasks." En H. Bunke, T. Kanade y H.Ñoltemeier (eds.), *Modelling and Planning for sensor based Intelligent Robot Systems*, págs. 362–383. World Scientific, 1995.

Doty, K. L. y Bou-Ghannam, A. "Controlling Situated Agent Behaviors with Perception and Cognition." En *AAAI Spring Symposium Series on Lessons Learned from Implemented Software Architectures for Physical Agents*, 1995.

Driankov, D. y Saffiotti, A. (eds.). *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, tomo 61 de *Studies in Fuzziness and Soft Computing*. Springer-Verlag, 2001.

Duckett, T. y Nehmzow, U. "Mobile robot self-localisation and measurement of performance in middle-scale environments." *Robotics and Autonomous Systems*, 24:57–69, 1998.

Duckett, T. y Nehmzow, U. "Performance Comparison of Landmark Recognition Systems for Navigating Mobile Robots." En *proceedings of AAAI-2000*, 2000.

Dudek, G., Jenkin, M. R. M., Milios, E. y Wilkes, D. "A Taxonomy for Multi-Agents Robotics." *Autonomous Robots*, 3:375–397, 1996.

Durfee, K. y Lesser, V. "Partial global planning: A coordination framework for distributed hypothesis formation." *IEEE Transactions on Systems, Man and Cybernetics*, 21(5):1167–1183, 1991.

Edlinger, T. y von Puttkamer, E. "Exploration of an Indoor-Environment by an Autonomous Mobile Robot." En *IROS'94*, 1994.

Elfes, A. "A distributed control architecture for an autonomous mobile robot." *Artificial Intelligence*, 1(2), 1986.

Elfes, A. "Sonar-based real-world mapping and navigation." *IEEE Journal of Robotics and Automation*, 3:249–265, 1987.

Elfes, A. "Using Occupancy Grids for Mobile Robot Perception and Navigation." *IEEE Computer*, 22(6):46–57, 1989.

Elfes, A. "A Distributed Control Architecture for an Autonomous Mobile Robot." En S. S. Iyengar y A. Elfes (eds.), *Autonomous Mobile Robots*, págs. 135–144. IEEE Press, 1991.

Engelmore, R. S. y Morgan, A. J. (eds.). *Blackboard Systems*. Addison-Wesley, 1988.

Everett, H. R. *Sensors for Mobile Robots. Theory and Application.* AK Peters, 1995.

Fabrizi, E., Oriolo, G. y Ulivi, G. "Accurate Map Building via Fusion of Laser and Ultrasonic Range Measures." En D. Driankov y A. Saffiotti (eds.), *Fuzzy Logic Techniques for Autonomous Vehicle Navigation*, págs. 257–279. Springer-Verlag, 2001.

Fayek, R. E. *A Blackboard Activity-Based Control Architecture for the Navigation of Autonomous Vehicle*. Proyecto Fin de Carrera, Ottawa-Carleton Institute for Electrical Engineering, Dpt. of Systems and Computer Engineering, Faculty of Engineering, Carleton University, Canada, 1992.

Fayek, R. E., Liscano, R. y Karam, G. M. "A System Architecture for a Mobile Robot Based on Activities and a Blackboard Control Unit." En *Proc. IEEE Int'l Conf. Robotics and Automation*, págs. 267–274. IEEE CS Press, 1993.

Ferrell, C. *Robust Agent Control of an Autonomous Robot with Many Sensors and Actuators*. Proyecto Fin de Carrera, MIT, 1993.

Finin, T., Labrou, Y. y Mayfield, J. "KQML as an agent communication agent." En J. Bradshaw (ed.), *Software agents*, cap. 14. MIT Press, 1997.

Finin, T., Weber, J., Wiederhold, G., Genesereth, M., Fritzson, R., McKay, D., McGuire, J., Pelavin, P., Shapiro, S. y Beck, C. "Specification of the KQML Agent-Comunication Language." Inf. Téc. EIT-92-04, Enterprise Integration Technologies, 1993.

Firby, R. J. "Adaptive Execution in Complex Dynamic Worlds." Inf. Téc. YALEU/CSD/RR 662, Computer Science Department, Yale University, 1989.

Firby, R. J. "The RAP Language Manual. Animate Agent Project." Inf. Téc. Working Note AAP-6 Version 1, University of Chicago, 1995.

Firby, R. J., Prokopowicz, P.Ñ. y Swain, M. J. "The Animate Agent Architecture." En D. Kortenkamp, R. P. Bonasso y R. Murphy (eds.), *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, págs. 243–275. AAAI Press/ The MIT Press, 1998.

Fox, D., Burgard, W. y Thrun, S. "The Dynamic Window Approach to Collision Avoidance." *IEEE Robotics and Automation Magazine*, págs. 23–33, 1997.

Fox, D., Burgard, W. y Thrun, S. "Active Markov localization for mobile robots." *Robotics and Autonomous Systems*, 25:195–207, 1998.

Fraga, S., Félix, P., Lama, M., Sánchez, E. y Barro, S. "A proposal for a real time signal perception specialist." En E. Alpaydin (ed.), *Proceedings of EIS-98*, tomo 3, págs. 261–267. Tenerife, Spain, 1998.

Franz, M. O. y Mallot, H. A. "Biomimetic robot navigation." *Robotics and Autonomous Systems*, 2000.

Fuller, J. L. *Robotics. Introduction, Programming, and Projects*. Prentice-Hall, 1995.

Fusiello, A. y Caprile, B. "Synthesis of indoor maps in presence of uncertainty." *Robotics and Autonomous Systems*, 22:103–114, 1997.

Gambino, F., Oriolo, G. y Ulivi, G. "A comparison of three uncertainty calculus techniques for ultrasonic map building." En *SPIE Int. Symp. Aerospace/Defense sensing contr.*, págs. 249–260. Orlando, FL., 1996.

García-Alegre, M. C., Bustos, P. y Guinea, D. "The behavioural agents approach in a hierarchical architecture for robotics." En *ECLA'94 Workshop on Advanced Automation*, 1995a.

García-Alegre, M. C., Bustos, P. y Guinea, D. "Complex behaviour generation on autonomous robots: A case study." En *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics*, págs. 1729–1734, 1995b.

García-Alegre, M. C. y Recio, F. "Basic Visual and Motor Agents for Increasingly Complex Behavior Generation on a Mobile Robot." *Autonomous Robots*, 5:19–28, 1998.

García-Cerezo, A., Mandow, A. y López-Baldán, M. J. "Fuzzy modelling operator navigation behaviours." En *Proceedings of the sixth IEEE International Conference on Fuzzy Systems (Fuzz-IEEE'97)*, págs. 1339–1345. Barcelona (Spain), 1997.

Gasós, J. y Martín, A. "Mobile Robot Localization Using Fuzzy Maps." En A. Ralescu y T. Martin (eds.), *Fuzzy Logic in Artificial Intelligence*, tomo 1188 de *Lecture Notes in Artificial Intelligence*, págs. 207–224. Springer Verlag, 1996.

Gasós, J. y Rosetti, A. "Uncertainty representation for mobile robots: Perception, modeling and navigation in unknown environments." *Fuzzy sets and systems*, 107:1–24, 1999.

Gat, E. "Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots." En *Proc. 1992 AAAI*, 1992.

Gat, E. "On the Role of Stored Internal State in the Control of Autonomous Mobile Robots." *AI Magazine*, 14(1):64–73, 1993.

Gat, E. "Three-Layer Architectures." En D. Kortenkamp, R. P. Bonasso y R. Murphy (eds.), *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, págs. 195–210. AAAI Press/ The MIT Press, 1998.

Genesereth, M. y Ketchpel, S. "Software agents." *Communications of the ACM*, 37(7):48–53, 1994.

Gilmore, J., Roth, S. y Tynor, S. "A blackboard system for distributed problem solving." En V. Jagannathan, R. Dodhiawala y L. Baum (eds.), *Blackboard Architectures and Applications*, tomo 3 de *Perspectives in Artificial Intelligence*, págs. 371–393. Academic Press, 1989.

Gómez Skarmeta, A., Martínez Barberá, H. y García López, P. "Una arquitectura de agentes difusos para robots autónomos móviles." En *Proc. ESTYLF'98*, págs. 443–448, 1998.

Gutierrez-Osuna, R. y Luo, R. C. "LOLA. Probabilistic Navigation for Topological Maps." *AI Magazine*, págs. 55–62, 1996.

Gutmann, J. S., Burgard, W., Fox, D. y Konolige, K. "An Experimental Comparison of Localization Methods." En *proceedings of International Conference on Intelligent Robot and Systems*, 1998.

Harnad, S. "The Symbol Grounding Problem." *Physica D*, 42:225–234, 1990.

Harris, K. D. y Recce, M. "Experimental modelling of time-of-flight sonar." *Robotics and Autonomous Systems*, 24:33–42, 1998.

Hartley, R. y Pipitone, F. "Experiments with the Subsumption Architecture." En *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, págs. 1652–1658. IEEE Computer Society Press, 1991.

Hayes-Roth, B. "An architecture for adaptive intelligent systems." *Artificial Intelligence*, 72:329–365, 1995.

Hayes-Roth, B., Pfleger, K., Morignot, P. y Lalanda, P. "Plans and Behavior in Intelligent Agents." Inf. Téc. KSL-95-35, Computer Science Department. Stanford University, 1995.

Hayes-Roth, B., Washington, R., Ash, D., Hewett, R., Collinot, A., Vina, A. y Seiver, A. "Guardian: A Prototype Intensive-Care Monitoring Agent." *Artificial Intelligence in Medicine*, 4:165–185, 1992.

Hoff, J. y Bekey, G. "An Architecture for Behavior Coordination Learning." En *IEEE International Conference on Neural Networks*, 1995.

Horswill, I. "Polly, A Vision-Based Artificial Agent." En *proceedings of the AAAI-93*, págs. 824–829, 1993.

Howard, A. y Kitchen, L. "Navigation using natural landmarks." *Robotics and Autonomous Systems*, 26:99–115, 1999.

Hu, H. y Brady, M. "Towards Advanced Mobile Robots for Manufacturing." En D. Kortenkamp, R. P. Bonasso y R. Murphy (eds.), *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, págs. 297–321. AAAI Press/ The MIT Press, 1998.

Husbands, P., Harvey, I. y Cliff, D. "Circle in the round: State space attractors for evolved sighted robots." *Robotics and Autonomous Systems*, 15:83–106, 1995.

Iglesias, R., Regueiro, C. V., Correa, J. y Barro, S. "Implementation of a Basic Reactive Behavior in Mobile Robotics Through Artificial Neural Networks." En J. Mira, R. Moreno-Díaz y J. Cabestany (eds.), *Biological and Artificial Computation: From Neuroscience to Technology*, tomo 1240 de *LNCS*, págs. 1364–1373. Springer Verlag, 1997.

Iglesias, R., Regueiro, C. V., Correa, J. y Barro, S. "Supervised Reinforcement Learning: Application to a Wall Following Behaviour in a Mobile Robot." En A. P. del Pobil, J. Mira y M. Ali (eds.), *Tasks and Methods in Applied Artificial Intelligence*, tomo 1416 de *LNAI*, págs. 300–309. Springer Verlag, 1998a.

Iglesias, R., Regueiro, C. V., Correa, J. y Barro, S. "Supervised Reinforcement Learning: A New Approach for Behaviour Learning in Mobile Robotics." *Robótica*, 2002. En prensa.

Iglesias, R., Regueiro, C. V., Correa, J., Sánchez, E. y Barro, S. "Improving wall following behaviour in a mobile robot using reinforcement learning." En E. Alpaydin (ed.), *Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems*, tomo 3, págs. 531–537. ICSC Academic Press, University of La Laguna, Tenerife, Spain, 1998b.

Ingrand, F. F., Chatila, R., Alami, R. y Robert, F. "PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots." En *Proceedings of the IEEE Int. Conference on Robotics and Automation*. Minneapolis, USA, 1996.

Ingrand, F. F. y Coutance, V. "Procedural Reasoning versus Blackboard Architecture for Real-Time Reasoning." En *Proceedings of the Thirteenth International Conference on Artificial Intelligence*, págs. 449–458, 1993.

Ingrand, F. F., Georgeff, M. P. y Rao, A. S. "An Architecture for Real-Time Reasoning and System Control." *IEEE Expert*, 7(6):33–44, 1992. Technical Report 92-521.

Jagannathan, V. "Realizing the concurrent blackboard model." En V. Jagannathan, R. Dodhiawala y L. Baum (eds.), *Blackboard Architectures and Applications*, tomo 3 de *Perspectives in Artificial Intelligence*, págs. 85–97. Academic Press, 1989.

Jagannathan, V., Dodhiawala, R. y Baum, L. (eds.). *Blackboard Architectures and Applications*, tomo 3 de *Perspectives in Artificial Intelligence*. Academic Press, 1989.

Jones, J. L. y Flynn, A. M. *Mobile Robots: inspiration to implementation*. AK Peters, 1993.

Joyanes, L. *Programación orientada a objetos*. McGraw-Hill, 1996.

Jung, D. y Zelinsky, A. "An architecture for distribute cooperative planning in a behaviour-based multi-robot systems." *Robotics and Autonomous Systems*, 26:149–174, 1998.

Kaelbling, L. "REX: A Symbolic Language for the Design and Parallel Implementation of Embedded Systems." En *Proceedings of the AIAA Conference on Computers in Aerospace*, págs. 255–260, 1987.

Kaelbling, L. P. y Rosenchein, S. J. "Action and Planning in Embedded Agents." *Robotics and Autonomous Systems*, 6:35–48, 1990.

Kagami, S., Nishiwaki, K., Sugihara, T., Kuffner, J. J., Inaba, M. y Inoue, H. "Design and Implementation of Software Research Plataform for Humanoid Robotics: H6." En *Proceedings of the International Conference on Robotics and Automation (ICRA'01)*, págs. 2431–2436, 2001.

Khatib, O. "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots." *The International Journal of Robotics Research*, 1986.

Knasel, T. M. "Mobile Robots. State of the art review." *Robotics*, 2, 1986.

Koenig, S., Goodwin, R. y Simmons, R. G. "Robot Navigation with Markov Models: A Framework for Path Planning and Learning with Limited Computational Resources." En L. Dorst, M. van Lambalgen y V. Voorbraak (eds.), *Reasoning with Uncertainty in Robots*, tomo 1093 de *LNAI*, págs. 322–337, 1996.

Koenig, S. y Simmons, R. G. "Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models." En D. Kortenkamp, R. P. Bonasso y R. Murphy (eds.), *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, págs. 91–122. AAAI Press/ The MIT Press, 1998.

Konolige, K. "A Refined Method for Occupancy Grid Interpretation." En L. Dorst, M. van Lambalgen y F. Voorbraak (eds.), *Reasoning with Uncertainty en Robotics*, tomo 1093 de *Lecture Notes in Artificial Intelligence*, págs. 338–352. Springer, 1996.

Konolige, K. "COLBERT: A Language for Reactive Control in Sapphira." En *German Conference on Artificial Intelligence*. Freiburg, 1997.

Konolige, K. *Saphira Manual*, 1999.

Konolige, K. y Myers, K. "The Saphira Architecture for Autonomous Mobile Robots." En D. Kortenkamp, R. P. Bonasso y R. Murphy (eds.), *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, págs. 211–242. AAAI Press/ The MIT Press, 1998.

Konolige, K., Myers, K., Ruspini, E. y Saffiotti, A. "The Saphira Architecture: A Design for Autonomy." *Journal of Experimental and Theoretical Artificial Intelligence*, 9:215–235, 1997.

Kortenkamp, D., Huber, M., Cohen, C., Raschke, U., Koss, F. y Congdon, C. "Integrating High-Speed Obstacle Avoidance, Global Path Planning, and Vision Sensing on a Mobile Robot." En D. Kortenkamp, R. P. Bonasso y R. Murphy (eds.), *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, págs. 51–71. AAAI Press/ The MIT Press, 1998.

Kuipers, B. J. y Byun, Y.-T. "A Robust, Qualitative Approach to a Spatial Learning Mobile Robot." En S. S. Iyengar y A. Elfes (eds.), *Autonomous Mobile Robots*, págs. 353–362. IEEE Press, 1991a.

Kuipers, B. J. y Byun, Y. T. "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations." *Robotics and Autonomous Systems*, 8:47–63, 1991b.

Labrou, Y. y Finin, T. "A proposal for a new KQML specification." Inf. Téc. CS-97-03, Computer Science and Electrical Dpt., University of Maryland Baltimore Country, 1997.

Labrou, Y., Finin, T. y Peng, Y. "Agent communication languages: The current Landscape." *IEEE Intelligent Systems*, 14(2):45–52, 1999.

Lama, M. *Modelo del conocimiento y arquitectura para la síntesis de un especialista terapéutico en Unidades de Cuidados Intensivos Coronarios*. Tesis Doctoral, Dpto. Electrónica y Computación, 2000.

Lama, M., Taboada, M., Barro, S., Marín, R. y Palacios, F. "Design of a therapeutic specialist for acute myocardial infarct." *Cybernetics and Systems*, 30(3):227–248, 1999.

Latombe, J. C. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

Lee, D. *The Map-Building and Exploration Strategies of a Simple Sonar-Equipped Mobile Robot*. Cambridge University Press, 1996.

Lemon, O. y Nehmzow, U. "The scientific status of mobile robotics: Multi-resolution mapbuilding as a case study." *Robotics and Autonomous Systems*, 24:5–15, 1998.

Leonard, J. J. y Durrant-Whyte, H. F. "Simultaneous map building and localization for an autonomous mobile robot." En *IEEE/RSJ Int. Workshop on Intelligent Robots and Systems*, págs. 1442–1447. Osaka, Japan, 1991.

Leonard, J. J. y Durrant-Whyte, H. F. *Directed Sonar Sensing for Mobile Robot Navigation*. 1992, Kluwer.

Lesser, V. y Corkill, D. "The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks." *AI Magazine*, 4(3):15–33, 1983.

Lewitt, T. S. y Lawton, D. T. "Qualitative Navigation for Mobile Robots." *Artificial Intelligence*, 1990.

Liscano, R., Manz, A., Stuck, E. R., Fayek, R. E. y Tigli, J. Y. "Using a Blackboard to Integrate Multiple Activities and Achieve Strategic Reasoning for Mobile-Robot Navigation." *IEEE Expert*, págs. 24–36, 1995.

Maes, P. "A Spreading Activation Network for Action Selection." En T. Kanade, F. C. A. Groen y L. O. Hertzberger (eds.), *Intelligent Autonomous Systems*, págs. 875–885. The IAS Foundation, 1989.

Maes, P. "Situated Agents Can Have Goals." *Robotics and Autonomous Systems*, págs. 49–70, 1990.

Maes, P. "Behavior-Based Artificial Intelligence." En *Proceedings of the Second Conference on Adaptive Behavior*, 1993.

Mahadevan, S. y Connell, J. "Automatic Programming of Behavior-Based Robots Using Reinforcement Learning." *Artificial Intelligence*, 55:311–365, 1992.

Malcolm, C., Smithers, T. y Hallam, J. "An Emerging Paradigm in Robot Architecture." En T. Kanade, F. Groen y L. Hertzberger (eds.), *proceedings Intelligent Autonomous Systems*, págs. 545–564, 1989.

Martin, C. E. y Firby, R. J. "An Integrated Architecture for Planning and Learning." *SIGART Bulletin*, 2(4):125–129, 1991.

Mataric, M. J. "Behavior-Based Control: Main Properties and Implications." En *Proceedings of Workshop on Intelligent Control Systems, International Conference on Robotics and Automation*, 1992a.

Mataric, M. J. "Integration of Representation into Goal-Driven Behavior-based Robots." *IEEE Transactions on Robotics and Automation*, 8(3):304–312, 1992b.

Mataric, M. J. *Interaction and Intelligent Behavior*. Tesis Doctoral, MIT Artificial Intelligent Lab, 1994.

Mayfield, J., Labrou, Y. y Finin, T. "Evaluating KQML as an Agent Communication Language." En M. Wooldridge, J. Müller y M. Tambe (eds.), *Proc. of the IJCAI'95 Workshop on Intelligent Systems (II): Agent Theories, Architectures and Languages*, tomo 1037 de *Lecture Notes in Artificial Intelligence*, págs. 347–360. Springer Verlag, 1996.

McFarland, D. y Bösser, T. *Intelligent Behavior in Animals and Robots*. MIT Press, 1993.

Medeiros, A. A. D., Chatila, R. y Fleury, S. "Specification and Validation of a Control Architecture for Autonomous Mobile Robots." En *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, págs. 87–94. Osaka, Japan, 1996.

Mehlhorn, K., Näher, S., Seel, M. y Uhrig, C. *The LEDA User Manual. Version 4.0*. Max-Planck-Institut für Informatik, Saarbrücken. Germany, 1999.

Meng, M. y Kak, A. C. "Mobile Robot Navigation Using Neural Networks and Nonmetrical Environment Models." *IEEE Control Expert*, págs. 30–39, 1993.

Merriam-Webster (ed.). *Webster's Third New International Dictionary*. Könemann, 1993.

Meystel, A. "Knowledge Based Nested Hierarchical Control." En G. Saridis (ed.), *Advances in Automation and Robotics*, tomo 2, págs. 63–152. JAI Press, 1990.

Meystel, A. (ed.). *Autonomous mobile robots: vehicles with cognitive control*. World Scientific, 1991.

Müller, J. P. *The Design of Intelligent Agents. A layered approach*. Springer-Verlag, 1996.

Moravec, H. P. "Sensor fusion in certainty grids for mobile robots." *AI Magazine*, 9(2):61–74, 1988.

Mucientes, M., Iglesias, R., Regueiro, C. V., Bugarin, A., Cariñena, P. y Barro, S. "Use of fuzzy temporal rules for avoidance of moving obstacles in mobile robotics." En G. Mayor y J. Suñer (eds.), *Proceedings Eusflat-Estylf'99*, págs. 167–170. Palma de Mallorca. Spain, 1999.

Mucientes, M., Iglesias, R., Regueiro, C. V., Bugarín, A. y Barro, S. "A Fuzzy Temporal Rule-based Approach to the Design of Behaviours in Mobile Robotics." En C. Leondes (ed.), *Fuzzy Systems, Neural Networks, and Expert Systems*, tomo 2 de *Intelligent Systems: Technology and Applications*. CRC Press, 2002a. En prensa.

Mucientes, M., Iglesias, R., Regueiro, C. V., Bugarín, A. y Barro, S. "A Fuzzy Temporal Rule-based velocity controller for mobile robotics." *Fuzzy sets and systems*, 2002b. En prensa.

Mucientes, M., Iglesias, R., Regueiro, C. V., Bugarín, A., Cariñena, P. y Barro, S. "Fuzzy Temporal Rules for Mobile Robot Guidance in Dynamic Environments." *IEEE Transactions on Systems, Man and Cybernetics*, 31(3):391–398, 2001a.

Mucientes, M., Rodríguez, M., Regueiro, C. V., Iglesias, R., Bugarín, A. y Barro, S. "Avoidance of mobile obstacles in real environments." En D. Fox y A. Saffiotti (eds.), *Reasoning with Uncertainty in Robotics. Seventeenth International Joint Conference on Artificial Intelligence*, págs. 69–76. Seattle, Washington, 2001b.

Muller, J. P. y Rodriguez, M. "Representation and planning for behaviour-based robots." En H. Bunke, T. Kanade y H.Ñoltemeier (eds.), *Modelling and Planning for sensor based Intelligent Robot Systems*, págs. 403–418. World Scientific, 1995.

Murphy, R. y Mali, A. "Lessons Learned in Integrating Sensing into Autonomous Mobile Robot Architectures." *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2):191–209, 1997.

Murphy, R. R. *Introduction to AI robotics*. MIT Press, 2000.

Murphy, R. R., Hughes, K., Marzilli, A. y Noll, E. "Integrating explicit path planning with reactive control of mobile robots using Trulla." *Robotics and Autonomous Systems*, 27:225–245, 1999.

Myers, K. L. "A Procedural Knowledge Approach to Task-level Control." En *Proc. of the Third International Conference on AI Planning Systems*, 1996.

Nagatani, K. y Yuta, S. "Autonomous Mobile Navigation incluiding Door Opening Behavior." En A. Zelinsky (ed.), *Field and Service Robotics*, págs. 195–202. Springer, 1998.

Nehmzow, U. "Animal and Robot Navigation." En L. Steels (ed.), *The Biology and Technology of Intelligent Autonomous Agents*, págs. 258–274. Springer Verlag, 1995.

Nehmzow, U. *Mobile Robotics: A Practical Introduction*. Springer, 2000.

Nehmzow, U. y Owen, C. "Robot navigation in the real world: Experiments with Manchester's FortyTwo in unmodified, large environments." *Robotics and Autonomous Systems*, 33:223–242, 2000.

Nii, H. "Blackboard systems Part Two: Blackboard application systems." *AI Magazine*, 7(3):82–106, 1986a.

Nii, H. "Blackboard systems: The Blackboard model of problem solving and the evolution of blackboard architectures." *AI Magazine*, 7(3):38–53, 1986b.

Nilsson, N. J. "A Mobile Automaton: An Application of Artificial Intelligence Techniques." En *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'69)*, 1969.

Nilsson, N. J. y col. "Shakey the Robot." Inf. Téc. 323, SRI A.I. Center, 1984.

Noreils, F. R. y Chatila, R. G. "Plan Execution Monitoring and Control Architecture for Mobile Robots." *IEEE Transactions on Robotics and Automation*, 11(2):255–266, 1995.

Nourbakhsh, I. "DERVISH. An Office-Navigating Robot." En D. Kortenkamp, R. P. Bonasso y R. Murphy (eds.), *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, págs. 73–90. AAAI Press/ The MIT Press, 1998.

Nourbakhsh, I., Powers, R. y Birchfield, S. "DERVISH. An Office-Navigation Robot." *AI Magazine*, págs. 53–60, 1995.

Oriolo, G. y Ulivi, G.and Vendittelli, M. "Real-Time Map Building and Navigation for Autonomous Robots in Unknown Environments." *IEEE Transactions on Systems, Man and Cybernetics*, 28(3):316–333, 1998.

Pagac, D., Nebot, M. y Durrant-Whyte, H. "An Evidential Approach to Map-Building for Autonomous Vehicles." *IEEE Transactions on Robotics and Automation*, 14(4), 1998.

Pang, G. K. H. y Shen, H. C. "Intelligent control of an autonomous mobile robot in a hazardous material spill accident - a blackboard approach." *Robotics and Autonomous Systems*, 6:351–365, 1990.

Payton, D. W. "Internalized Plans: A Representation for Action Resources." *Robotics and Autonomous Systems*, 6:89–103, 1990.

Payton, D. W., Rosenblatt, J. K. y Keirsey, D. M. "Plan Guided Reaction." *IEEE Transactions on Systems, Man and Cybernetics*, 20(6):1370–1382, 1990.

Pfeifer, R. "Building "Fungus Eaters": Design Principles of Autonomous Agents." En *From Animals to Animats 4*. MIT Pres, 1996.

Pfeifer, R. y Scheier, C. *Understanding Intelligence*. MIT Press, 2000.

Pfleger, K. y Hayes-Roth, B. "Plans Should Abstractly Describe Intended Behavior." En A. Meystel, J. Albus y R. Quintero (eds.), *Intelligent Systems: A Semiotic Perspective*, tomo I. Theoretical Semiotics, págs. 29–34, 1996.

Pfleger, K. y Hayes-Roth, B. "An Introduction to Blackboard-Style Systems Organization." Inf. Téc. KSL-98-03, Knowledge Systems Laboratory. Computer Science Deparment. Stanford University, 1998a.

Pfleger, K. y Hayes-Roth, B. "Using Abstract Plans to Guide Behavior." Inf. Téc. KSL-98-02, Computer Science Department. Stanford University, 1998b.

Pimentel, J. R., Gachet, D., Moreno, L. y Salichs, M. A. "On-line Performance Enhancement of a Behavioral Neural Network Controller." En J. Mira, J. Cabestany y A. Prieto (eds.), *New Trends in Neural Computation*, tomo 686 de *LNCS*, págs. 694–701, 1993.

Pirjanian, P. "An overview of system architectures for action selection in mobile robotics." Inf. téc., Laboratory of Image Analysis, Institute of Electronic Systems, Aalborg University, 1997.

Pirjanian, P. y Blasvaer, H. "AMOR - An Autonomous Mobile Robot System." Inf. téc., Laboratory of Image Analysis, Institute of Electronic Systems, Aalborg University, 1994.

Pirjanian, P. y Christensen, H. I. "Hierarchical Control for Navigation Using Heterogeneous Models." En H. Bunke, T. Kanade y H.Ñoltemeier (eds.), *Modelling and Planning for sensor based Intelligent Robot Systems*, págs. 344–361. World Scientific, 1995.

Regueiro, C. V., Rodríguez, M., Correa, J., Iglesias, R. y Barro, S. "Aplicación a la robótica móvil de una arquitectura basada en especialistas." En S. Barro, N. Brisaboa, J. Busta y F. Rivera (eds.), *Proceedings SEID'99*, págs. 117–126. Santiago de Compostela, 1999.

Regueiro, C. V., Rodríguez, M., Correa, J., Moreno, D. L., Iglesias, R. y Barro, S. "A Control Architecture for Mobile Robotics Based on Specialists." En C. Leondes (ed.), *Control and Electric Power Systems*, tomo 6 de *Intelligent Systems: Technology and Applications*. CRC Press, 2002. En prensa.

Reina, A. y Gonzalez, J. "A two-stage mobile robot localization method by overlapping segment-based maps." *Robotics and Autonomous Systems*, 31:213–225, 2000.

Rodríguez, M., Correa, J., Iglesias, R., Regueiro, C. V. y Barro, S. "Probabilistic and Count Methods in Map Building for Autonomous Mobile Robots." En J. Wyatt y J. Demiris (eds.), *Advances in Robot Learning*, tomo 1812 de *LNAI*, págs. 120–137. Springer Verlag, 2000.

Rosenblatt, J. K. *DAMN: A Distributed Architecture for Mobile Navigation*. Tesis Doctoral, Carnegie Mellon University, 1997.

Roth, B. H., Pfleger, K., Lalanda, P., Morignot, P. y Balabanovic, M. "A Domain-Specific Software Architecture for Adaptive Intelligent Systems." *IEEE Transactions on Software Engineering*, 21(4):288–301, 1995.

Saffiotti, A. "Fuzzy Logic in Autonomous Robot Navigation: a case study." Inf. Téc. TR/95-25, IRIDIA, 1995.

Saffiotti, A. "Robot navigation under approximate self-localization." En M. Jamshid, F. Pin y P. Dauchez (eds.), *Procs. of the 6th ISRAM Symposium*, págs. 589–594, 1996.

Saffiotti, A. "The uses of fuzzy logic in autonomous robot navigation." *Soft Computing*, 1(4):180–197, 1997a.

Saffiotti, A. "The Uses of Fuzzy Logic in Autonomous Robot Navigation: a catalogue raisonné." Inf. Téc. TR/97-6, IRIDIA, Université Libre de Bruxelles, 1997b.

Saffiotti, A., Konolige, K. y Ruspini, E. H. "A multivaluated logic approach to integrating planning and control." *Artificial Intelligence*, 76:481–526, 1995.

Saffiotti, A., Ruspini, E. H. y Konolige, K. "Blending Reactivity and Goal-Directedness in a Fuzzy Controller." En *Proceedings IEEE Int. Conference on Fuzzy Systems*, págs. 134–139. San Francisco, CA, 1993.

Saffiotti, A. y Wesley, L. P. "Perception-Based Self-Localization Using Fuzzy Locations." En L. Dorst, M. van Lambalgen y F. Voorbraak (eds.), *Reasoning with Uncertainty en Robotics*, tomo 1093 de *Lecture Notes in Artificial Intelligence*, págs. 368–385. Springer, 1996.

Shafer, S. A., Stentz, A. y Thorpe, C. E. "An Architecture for Sensor Fusion in a Mobile Robot." En *proc. International Conference on Robotics and Automation*, págs. 2002–2011, 1986.

Shaw, M. y Garlan, D. *Software architecture : perspectives on an emerging discipline*. Prentice Hall, 1996.

Simmons, R. G. "Coordinating Planning, Perception, and Action for Mobile Robots." *SIGART Bulletin*, 2(4):156–159, 1991.

Simmons, R. G. "Concurrent Planning and Execution for Autonomous Robots." *IEEE Control Systems*, págs. 46–50, 1992.

Simmons, R. G. "Structured Control for Autonomous Robots." *IEEE Transactions on Robotics and Automation*, 10(1):34–43, 1994.

Simmons, R. G. "The Curvature-Velocity Method for Local Obstacle Avoidance." En *Proceedings of the International Conference on Robotics and Automation (ICRA)*, págs. 3375–3382, 1996.

Simmons, R. G., Goodwin, R., Haigh, K. Z., Koenig, S. y O'Sullivan, J. "A Layered Architecture for Office Delivery Robots." En *Proceedings First International Conference on Autonomous Agents*, 1997a.

Simmons, R. G., Goodwin, R., Haigh, K. Z., Koenig, S., O'Sullivan, J. y Veloso, M. "XAVIER: Experience with a Layered Rrobot Architecture." *SIGART Bulletin*, págs. 22–33, 1997b.

Simmons, R. G. y Koenig, S. "Probabilistic robot navigation in partially observable environments." En *Proc. of the 14th International Joint Conference on Artificial Inteligence*, 1995.

Simmons, R. G. y Mitchell, T. M. "A task control architecture for mobile robots." En *AAAI Spring Symposium on Robot Navigation*, 1989.

Smieja, F. y Beyer, U. "JANUS: A Robot Manipulator System Implemented on a Blackboard Architecture." Inf. Téc. 1994/3, German National Research Centre for Computer Science (GMD), 1994.

Sonka, M., Hlavac, V. y Boyle, R. *Image Processing, Analysis and Machine Vision*. Chapman and Hall, 1993.

Stone, P. y Veloso, M. "Multiagent Systems: A Survey from a Machine Learning Perspective." *Autonomous Robots*, 8(3):345–383, 2000.

Thrun, S. "An approach to learning mobile robot navigation." *Robotics and Autonomous Systems*, 15:301–319, 1995.

Thrun, S. "Learning Maps for Indoor Mobile Robot Navigation." *Artificial Intelligence*, 1997.

Thrun, S. "Learning Metric-Topological Maps for Indoor Mobile Robot Navigation." *AI Journal*, 99(1):21–71, 1998.

Thrun, S., Bücken, A., Burgard, W., Fox, D., Fröhlinghaus, T., Hennig, D., Hofmann, T., Krell, M. y Schmidt, T. "Map Learning and High-Speed Navigation in RHINO." En D. Kortenkamp, R. P. Bonasso y R. Murphy (eds.), *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, págs. 21–52. AAAI Press/ The MIT Press, 1998a.

Thrun, S., Bennewitz, M., Burgard, W., Dellaert, F., Fox, D., Haehnel, C., Rosenbert, C., Roy, N., Schulte, J. y Schulz, D. "MINERVA: A second generation mobile tour-guide robot." En *Proceedings 1999 IEEE International Conference on Robotics and Automation*, tomo 3, págs. 1999–2005, 1999.

Thrun, S., Fox, D. y Burgard, W. "A probabilistic approach to concurrent mapping and localisation for mobile robots." *Machine Learning*, 31:29–55, 1998b.

Tigli, J. Y., Fayek, R. E., Liscano, R. y Thomas, M. C. "Methology and Computing Model for a Reactive Mobile Robot Controller." En *Proc. IEEE Int'l Conf. Systems, Man, and Cybernetics*, págs. 17–20. IEEE Press, 1993.

Ulrich, I. *The GuideCane - A Computerized Travel Aid for the Active Guidance of Blind Pedestrians*. Proyecto Fin de Carrera, Dpt. Mechanical Eng. and Applied Mechanics. Univ. Michigan, 1997.

Vermesan, A. "Foundation and Application of Expert System Verification and Validation." En J. Liebowitz (ed.), *The Handbook of Applied Expert Systems*, cap. 5, págs. 1–32. CRC Press LLC, London, 1998.

Vivancos, E., Hernández, L. y Botti, V. "Inteligencia artificial distribuida en entornos de tiempo real." *Inteligencia Artificial*, (6):24–35, 1998.

von Puttkamer, E. "Line based modelling from laser radar data." En H. Bunke, T. Kanade y H.Ñoltemeier (eds.), *Modelling and Planning for sensor based Intelligent Robot Systems*, págs. 419–430. World Scientific, 1995.

Webb, B. "Using robots to model animals: a cricket test." *Robotics and Autonomous Systems*, 1995.

Wijk, O. y Christensen, H. I. "Localization and navigation of a mobile robot using natural point landmarks extracted from sonar data." *Robotics and Autonomous Systems*, 31:31–42, 2000.

Xu, H. y van Brussel, H. "A behaviour-based blackboard architecture for reactive and efficient task execution of an autonomous robot." *Robotics and Autonomous Systems*, 22:115–132, 1997.

Yen, P. y Pfuger, N. "A Fuzzy Logic Based Extension to Payton and Rosenblatt's Comand Fusion Method for Mobile Robot Navigation." *IEEE Transactions on Systems, Man and Cybernetics*, 25(6):971–978, 1995.

Zelek, J. S. *SPOTT: A real-time, distributed and scalable architecture for autonomous mobile robot control*. Tesis Doctoral, Dpt. Electrical Engineering. McGill University, 1996.

Zelek, J. S. y Levine, M. D. "SPOTT: A Mobile Robot Control Architecture for Unkown or Partially Known Environments." En *AAAI Spring Symposium on Planning with Incomplete Information for Robot Problems*, 1996.