Sistemas Informáticos

Curso 2008/2009

Metodologías de diseño de comportamientos para AIBO ERS7

Ángel Montero Mora

Gustavo Méndez Muñoz

José Ramón Domínguez Rodríguez

Dirigido por:

Prof. María Belén Díaz Agudo

Dpto. Ingeniería del Software e Inteligencia Artificial

Facultad de Informática
Universidad Complutense de Madrid

Autorización

Por la presente, los autores de este proyecto autorizan a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Ángel Montero Mora

Gustavo Méndez Muñoz

José Ramón Domínguez Rodríguez

Palabras clave

- AIBO
- CBR
- Comportamiento
- URBI
- RoboCup.
- Telecommande.
- Metodología.
- Robótica.

Resumen

Este proyecto surgió con la idea inicial de desarrollar un sistema de Inteligencia Artificial que controle el comportamiento de un robot Sony AIBO desde un ordenador o una PDA.

A lo largo de este documento se explicarán la metodología GMG desarrollada para la implementación de comportamientos para un robot y el uso de un sistema CBR (Razonamiento Basado en Casos) autónomo para la reutilización de esos comportamientos y el diseño de nuevos comportamientos complejos a partir de otros más simples.

También queda reflejado el trabajo de investigación y documentación que se ha realizado sobre nuestro dispositivo en concreto, el AIBO ERS-7, y sobre sistemas expertos CBR.

Abstract

This project arises from the initial idea of developing an Artificial Intelligence system which controls the behavior of a Sony AIBO robot from a CPU or a PDA.

Throughout this document we will explain the GMG methodology developed for the implementation of behaviors for robots and the use of an autonomous CBR system (Case-Based Reasoning) for the reuse of those behaviors and the design of new complex behaviors from other more simple ones.

It is also contained the investigation and documentation work executed for our specific device, the AIBO ERS-7, and for CBR expert systems.

Índice

Autorización 2
Palabras clave 3
Resumen / Abstract 4
Índice 5
1. Introducción, Objetivos y Alcance del trabajo. 7
2. AIBO ERS7. 9
2.1 Presentación y consideraciones preliminares. 9
2.2 Modelo basado en sensores y actuadores. 12
2.3 Sistema operativo y Open-R.. 15
2.4 Lenguajes de programación e interfaces. 17
2.4.1 Inconvenientes de la programación con OPEN-R. 17
2.4.2 Lenguajes basados en Scripts: URBI (Universal Real-Time Behavior Interface).. 18
2.5 Interfaces de manejo para AIBO 19
2.5.1 Tekkotsu.. 19
2.5.2 Telecommande.. 22
2.5.3 Simuladores 30
2.6 Comparativa Urbi, OpenR y Tekkotsu. Ventajas e inconvenientes. 33
3. Metodología GMG de diseño de comportamientos para AIBO ER7. 35
3.1 Análisis de problemas.. 35
3.1.1 Planificación 36
3.1.2 Diseño de la máquina de estados. 37
3.2 Traducción del diseño al lenguaje elegido. 43
3.2.1 Comportamientos autónomos o guiados por equipos remotos. 44
3.2.2 Limitaciones del AIBO. 45
3.2.3 Sincronización. 46
3.3 Observación de errores y fallos y vuelta a empezar 46
3.4 Ejemplo. Persecución de una pelota. 47
4. Reutilización de comportamientos con razonamiento basado en casos 53
4.1 Razonamiento Basado en Casos 53
4.1.1 ¿Qué es CBR?.. 53
4.1.2 ¿Por qué lo proponemos a nuestra metodología? 59
4.2 Metodología GMG con reutilización de comportamientos. 60
4.3 Implementación 62
4.3.1 Módulo AIBO. 63
4.3.1.1. Percepción. 63
4.3.1.2. Actuación. 64
4.3.2 Módulo CBR. 65
4.3.2.1. Cognición. 66
4.3.2.2. Casos. 67
4.3.2.2.1. Atributos. 67
4.3.2.2.2. Solución. 68
4.3.2.2. Librería Casos. 69
4.3.2.3. Algoritmo recuperación casos. 72
4.3.3. Comunicación AIBO-CBR. 73

5. Conclusiones y trabajo futuro 76
6. Índice de Figuras 78
7. Referencias Bibliográficas 80

1. Introducción, Objetivos y Alcance del trabajo.

El término robótica se refiere a la ciencia o arte relacionada con la inteligencia artificial (para razonar) y con la ingeniería mecánica (para realizar acciones físicas sugeridas por el razonamiento). Este término fue acuñado en 1942 por el bioquímico, escritor y divulgador científico Isaac Asimov en su novela “Runaround”.

En un primer momento, el desarrollo de los robots estuvo orientado a sustituir a los humanos en los trabajos más peligrosos y repetitivos. Con el paso del tiempo, la evolución de la robótica ha ido abarcando cada vez más campos de la vida cotidiana, dejando de aplicarse únicamente en la industria. Paralelamente a la evolución robótica, la inteligencia artificial aplicada al campo de la robótica ha experimentado un gran avance en el desarrollo de sistemas informáticos que controlen estos robots y puedan realizar tareas de forma autónoma sin la necesidad de una supervisión humana.

Este proyecto surgió con la idea inicial de desarrollar un sistema de Inteligencia Artificial que controle el comportamiento de un robot Sony AIBO desde un ordenador o una PDA. Desde un primer momento los tres integrantes del grupo estuvimos muy interesados en este proyecto, ya que en él se pueden trabajar conjuntamente en las ramas de la Inteligencia Artificial y la Robótica que son dos campos por los que nos sentimos atraídos y en los que queríamos profundizar.

Trabajar con un robot Sony AIBO es un reto y una oportunidad llamativa e interesante, dado que sus múltiples características permiten un sinfín de posibilidades y posibles proyectos. Además hay a disposición de los desarrolladores multitud de documentación y herramientas ya desarrolladas; por lo que se puede implementar sistemas específicos de manera sencilla sin tener que preocuparse por los aspectos de más bajo nivel con los que funciona el robot, como pueden ser el manejo de los sensores o de las articulaciones, lo cual facilita centrarse en los aspectos más relacionados con la I.A. en sí. Toda esta cantidad de trabajo y documentación existente puede convertirse también en un hándicap a la hora de desarrollar un sistema que sea original y que pueda cumplir las expectativas que un proyecto de estas características representa.

Al empezar a trabajar con el AIBO tuvimos que realizar un trabajo de investigación para poder tener una idea clara y concisa de cómo trabaja y los límites a los que se podía llegar tanto en sus movimientos como en su programación. Esta primera etapa de recopilación de información, a la que llamamos estado del arte, nos sirvió de gran ayuda para poder definir objetivos intermedios, como pueden ser la implementación de comportamientos para el robot, y a largo plazo para definir el objetivo final del proyecto.

Una vez superada la primera etapa en la que se vieron aspectos únicamente relacionados con la robótica, empezamos a documentarnos y a investigar sobre como queríamos enfocar el desarrollo del módulo de I.A. En principió nos surgieron varias opciones, entre las que destacaba una idea por encima de las demás, un sistema CBR autónomo.

Queríamos que este sistema, apoyado por las percepciones del robot, fuese el que tomase las decisiones de qué comportamientos se iban a ejecutar en cada momento.

A partir de este momento empezamos a investigar sobre cómo se podía definir un sistema CBR: los pasos que hay que dar para empezar su implementación y los objetivos que esperábamos que se cumpliesen al final del proyecto.

Estos objetivos son definir una metodología de trabajo para implementar comportamientos autónomos para un robot en general y como ligar estos comportamientos mediante un programa CBR que los convirtiese en un sistema experto. Como en todo proyecto de I.A. es importante centrarse en un ámbito en concreto. Decidimos elegir el dominio de la simulación de fútbol debido en gran parte a la existencia de un campeonato de futbol robótico llamado Soccerbots de cierta importancia y renombre, pensando en la posible reutilización de comportamientos y rutinas. El tiempo demostró que no es oro todo lo que reluce.

A lo largo de los distintos apartados de este documento se irán explicando paso a paso todas las etapas transcurridas a lo largo del proyecto, detallando en cada una de ellas los aspectos que consideramos más importantes y los que queremos que el lector de este trabajo tenga más en cuenta. Creemos que se han alcanzado los objetivos definidos en esta introducción y esperamos que nuestro trabajo sea de utilidad para estudiantes que quieran desarrollar sistemas parecidos aunque sea en otros ámbitos y dominios.

2. AIBO ERS7.

2.1 Presentación y consideraciones preliminares.

En 1999 Sony lanzó al mercado la mascota electrónica AIBO (**A**rtificial **I**ntelligence ro**BO**t, amigo en japonés). El robot con forma de perro AIBO fue concebido inicialmente para poder interactuar con su dueño como si de una mascota real se tratase, ya era capaz de percibir los estímulos del exterior mediante los múltiples sensores de los que dispone y actuar en consecuencia, dando la sensación de que la mascota se comporta como un perro real. AIBO es capaz de comportarse como una mascota gracias a un motor software llamado AIBO MIND que provee SONY con el propio robot y que está localizado en una pequeña tarjeta de memoria tipo memory stick especial para AIBO y tiene las características necesarias para que el perro pueda reconocer a su dueño, interactúe y juegue con él e incluso le permite aprender para adaptarse a la vida con su propietario. AIBO MIND está implementado de tal manera que el robot puede evolucionar desde cachorro hasta un perro adulto.

A lo largo del tiempo, Sony ha desarrollado distintas versiones de AIBO y AIBO MIND, y en cada una de ellas se ha ido mejorando los sensores de los que dispone, su movilidad y su capacidad de conectarse a otros equipos para poder ser controlado de manera remota. En la Tabla 1 se muestran los distintos modelos de AIBO que ha fabricado SONY:

AIBO ERS-110.	
AIBO ERS-210	

AIBO ERS-311	
AIBO ERS-220	
AIBO ERS7	

Tabla 1. Evolución física de las distintas generaciones de AIBO comercializadas por SONY

Ya desde los primeros modelos comercializados, SONY desarrolló un sistema operativo propio para el robot (APERIOS) y permitió la manipulación de sensores y motores, así como la programación de comportamientos mediante el lenguaje OPEN-R, razones por las cuales el robot AIBO se convirtió rápidamente en una herramienta muy útil para los desarrolladores de sistemas de Inteligencia Artificial para la investigación y experimentación. Son varios los campos en los que se ha investigado con este pequeño robot y nos podemos encontrar desde aplicaciones realizadas por usuarios poco expertos a nivel de entretenimiento como pueden ser scripts para que el AIBO realice pequeños bailes o haga una tarea sencilla, a proyectos de investigación mucho más complejos desarrollados por grupos de investigación y que trabajan en realizar módulos de Inteligencia Artificial [1]. Una de las áreas más conocidas y que ha tenido mayor repercusión ha sido el de la creación de equipos de robots AIBO para participar en la RoboCupSoccer. La RoboCup es una iniciativa internacional de investigación y educación, cuyo principal objetivo es fomentar la Inteligencia Artificial y la Robótica y que está dividida en distintos dominios de los cuales RoboCupSoccer, RoboCupRescue y RoboCupHome son los más conocidos. AIBO, por su estructura física y fácil adaptación al dominio, ha sido muy utilizado en las competiciones de RoboCupSoccer en la denominada four-legged league, liga en la que se enfrentan dos equipos de 6 componentes del mismo tipo cada uno, por lo que podemos encontrar infinidad de información disponible sobre los distintos equipos que han participado en esta liga. La creación de sistemas que permitan al robot jugar en un equipo de fútbol es una tarea muy completa ya que se debe trabajar en los distintos aspectos que involucran al robot, desde el aspecto puramente cinemático para llegar a perfeccionar los movimientos que

deberá hacer el robot durante los partidos, hasta los más relacionados con la Inteligencia Artificial como pueden ser la decisión y reutilización de comportamientos, el trabajo en equipo y el aprendizaje.

Para este proyecto hemos utilizado la versión ERS 7M3 que Sony lanzó al mercado en el año 2003 y que fue la última versión comercializada. Las especificaciones técnicas de esta versión son las siguientes:

CPU:
Procesador RISC de 64 bits, 576 Mhz.
RAM
64 MB
Medio de Almacenamiento
Tarjeta Memory Stick para AIBO de 16 o 32 mb.
Partes Móviles:
Cabeza – 3 grados de libertad de movimiento
Boca - 1 grado de libertad de movimiento
Patas - 3 grados de libertad de movimiento x 4
Orejas - 1 grado de libertad de movimiento x 2
Cola – 2 grados de libertad de movimiento (Total 20 grados de libertad de movimiento)
Sección de entrada:
Contactos de Carga
Interruptores
Control de volumen
Interruptor de LAN Inalámbrica
Entrada de Imagen
Sensor de Imagen CMOS de 350.000-pixels
Entrada de Audio
Micrófonos Stereo
Salida de Audio
Altavoz 20.8mm, 500mW
Sensores Integrados
Sensores Infrarrojos de Distancia x 2
Sensor de Aceleración
Sensor de Vibración
Sensores de Entrada
Sensor en la cabeza
Sensor en el lomo
Sensor en la parte inferior de la boca
Sensores en las patas
Consumo de energía
Aproximadamente 7 W
Tiempo de operación
Aproximadamente 1,5 horas
Dimensiones
Aproximadamente 180 (ancho) x 278 (Alto) x 319 (Longitud) mm
Peso
1.6 kg (incluyendo batería y memory stick)
Temperatura operativa
5°C a 35°C

Humedad operativa
10% a 80% (no condensada)
Temperatura de almacenamiento
-10°C a 60°C
Humedad de almacenamiento
10% a 90% (no condensada)
Función de LAN Inalámbrica
Módulo LAN Inalámbrica WiFi certificado
Compatibilidad estándar
IEEE 802.11b/IEEE 802.11
Banda de Frecuencia
2.4 Ghz
Canales
1 – 11
Modulación
DS-SS (IEEE 802.11 compatible)
Cifrado
WEP 64 (40 bits), WEP 128 (104 bits)

Con esta versión del robot SONY facilita la versión AIBO MIND 3

2.2 Modelo basado en sensores y actuadores.

En los robots utilizados y desarrollados durante la historia, ya sean para aplicaciones industriales, para realizar investigaciones o para uso domestico, podemos hacer una distinción clara de sus componentes en tres categorías principales:

- Sensores. Los sensores son los encargados de proporcionar al robot de percepción de su entorno. Existen multitud de tipos de sensores y suelen trabajar de manera conjunta para que el robot pueda observar el mundo que lo rodea. Algunos ejemplos son sensores de proximidad, de contacto, de presión, de distancia, etc.

- Actuadores: Los actuadores son los encargados de que el robot pueda interactuar con su entorno y los objetos que se encuentran en el. Algunos ejemplos de actuadores pueden ser brazos mecánicos, pinzas, servo-motores, etc.

- Procesadores. Un robot puede tener uno o varios procesadores que son los encargados de recibir las señales provenientes de los sensores, traducirlas en algo que el sistema pueda interpretar (por ejemplo un vector de características), y decidir en cada caso la acción que deben realizar los actuadores, independientemente de si el robot se comporta de manera autónoma o es guiado de manera remota.

Teniendo en cuenta esta distinción de los componentes del robot, surgen varios paradigmas [2] para poder controlar su comportamiento teniendo en cuenta el estado en el que se encuentra el mundo que lo rodea y que es percibido mediante los sensores. Los dos paradigmas más utilizados son el jerárquico y el reactivo.

Paradigma jerárquico: este paradigma ha sido el más utilizado en robótica desde sus inicios y se basa en una estructura denominada percepción/cognición/acción.

En este proceso se realiza una percepción del mundo que rodea al robot por medio de sus sensores. Esta información es tratada en un procedimiento al que podemos denominar de cognición, el cual da lugar a una serie de directivas necesarias para realizar alguna tarea; finalmente, estas directivas se traducen en comandos de los actuadores para producir un cambio en el entrono del robot. Al producirse un cambio en el entorno del robot, es necesario un nuevo proceso de percepción, por lo que se produce un nuevo ciclo. El ciclo anteriormente descrito se tiene que repetir constantemente para que el robot pueda realizar distintas tareas, y se muestra en la siguiente figura 1:

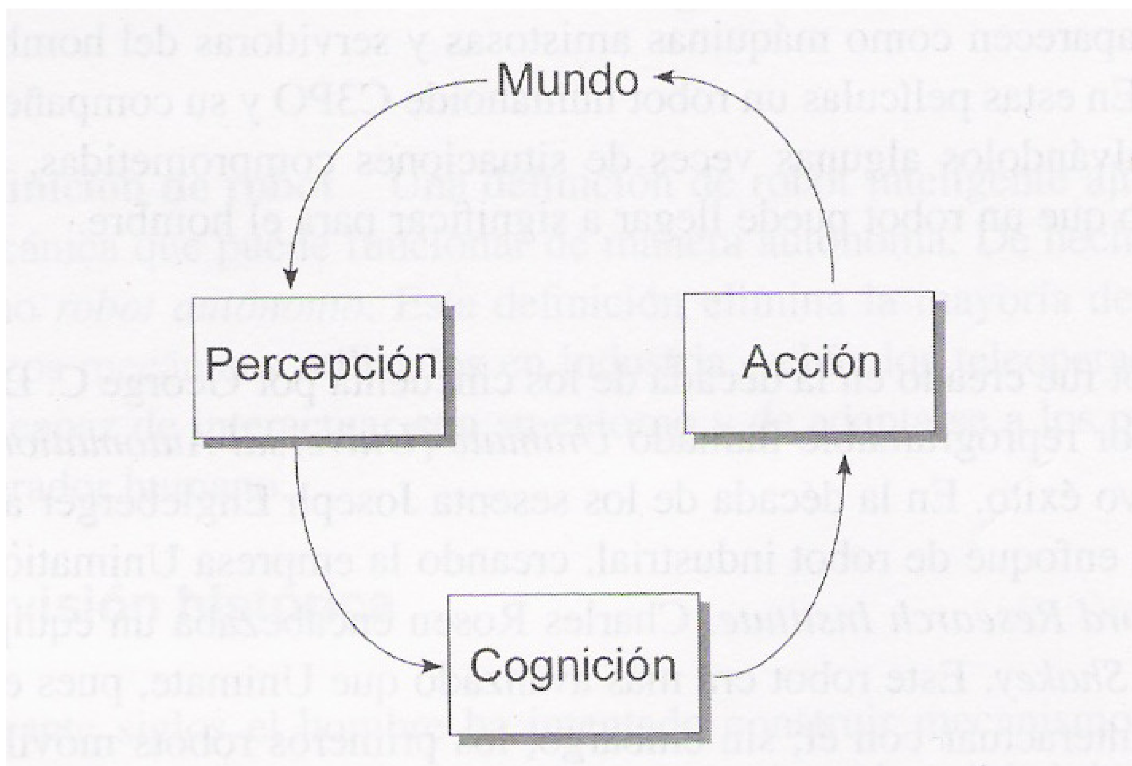

Figura 1. Ciclo del Paradigma Jerárquico

Parece evidente que dentro de este ciclo la parte más importante y la más relacionada con la Inteligencia Artificial es el proceso de cognición. Este proceso es el encargado de recibir las señales provenientes de los sensores, transformarlas en algo que el sistema pueda interpretar y después decidir las acciones que el robot debe realizar. Existen multitud de funciones que puede realizar este proceso de selección de acciones, sistemas de producción basados en reglas, redes que simulan circuitos electrónicos, algún tipo de proceso relacionado con el razonamiento, y es tarea de los desarrolladores elegir aquella que más se ajuste al robot que se esté manejando y a la tarea que se quiera realizar.

El gran problema de este paradigma es el tiempo empleado en la etapa de cognición ya que crea un cuello de botella entre las etapas de percepción y acción que es inadmisible en muchos sistemas que trabajan en tiempo real. Este problema unido a que el robot tiene que estar constantemente interpretando las señales que recibe para poder percibir

los cambios producido en su entrono ha dado lugar a la aparición de nuevos paradigmas o métodos que facilitan el manejo de robots.

Paradigma reactivo: El método jerárquico no se corresponde con el comportamiento natural con el que se comportan la mayoría de los seres vivos, por esta razón el paradigma reactivo propone un solapamiento de la percepción y la acción mediante lo que podemos llamar “conductas reflejas”. Podemos definir estas conductas como respuestas muy rápidas a los estímulos externos; un ejemplo de conducta refleja es cuando notamos que se nos está quemando una mano y la apartamos intuitivamente de la fuente de calor sin pararnos a pensar en la misma acción de apartarla. Estas conductas pueden ser implementadas mediante funciones que recibe como entrada datos de los sensores y producen como salida datos para los actuadores mediante algún tipo de función que tiene la característica principal que no consumen prácticamente nada de tiempo. El objetivo principal es dar una respuesta rápida a los datos de entrada. Otra de las propiedades de estas conductas es que se pueden ejecutar varias de manera paralela e independiente, es decir, una conducta no necesita saber lo que otra esta percibiendo o la acción que está ejecutando. Este paradigma da una visión mucho más intuitiva a los desarrolladores a la hora de desarrollar sistemas en los que se tiene que realizar tareas complejas en los que están involucrados varios objetivos distintos. En la figura 2 podemos ver el ciclo de vida de este paradigma.

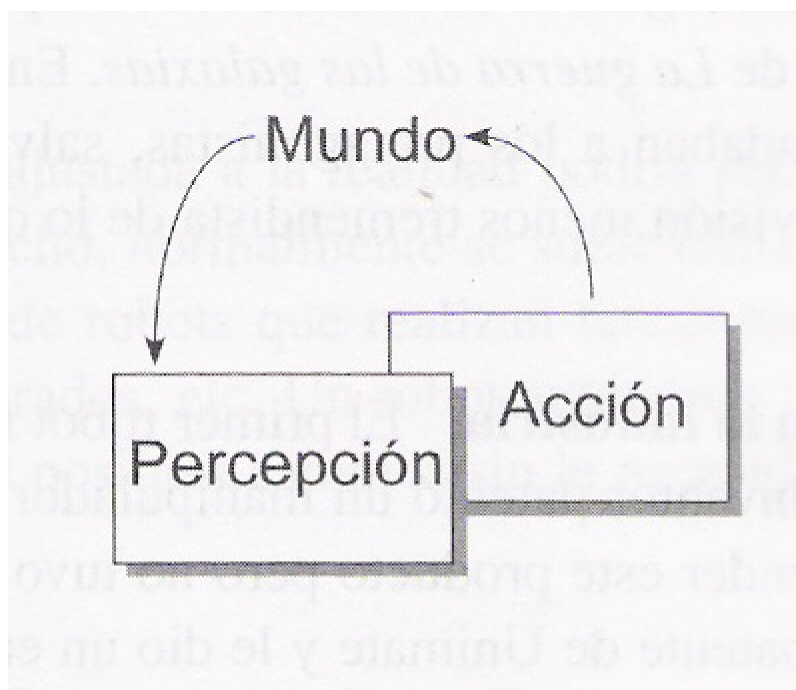

Figura 2. Ciclo del Paradigma Reactivo

2.3 Sistema operativo y Open-R

Sistema Operativo:

El nombre del sistema operativo del AIBO es "APERIOS". APERIOS es un sistema operativo empotrado basado en una arquitectura de meta-niveles cuyo diseño ha tenido gran influencia en la forma compleja en que se programa con OPEN-R. APERIOS se ha utilizado en diversos equipos de Sony además del robot AIBO, como por ejemplo el receptor de satélite DST-MS9, aunque al tratarse de secretos comerciales hay muy poca información al respecto y fue una de las barreras que hubo que superar para publicar OPEN-R.

OPEN-R:

El robot AIBO se comercializó inicialmente como un sistema de entretenimiento robótico. Su atractivo diseño y su fiabilidad enseguida atrajeron la atención de los desarrolladores, lo que llevó a Sony a publicar el API de su sistema operativo. Se trata de un API en C++ que corre sobre el sistema operativo APERIOS y que Sony ha proporcionado bajo el nombre de OPEN-R SDK (resumida en la figura 3).

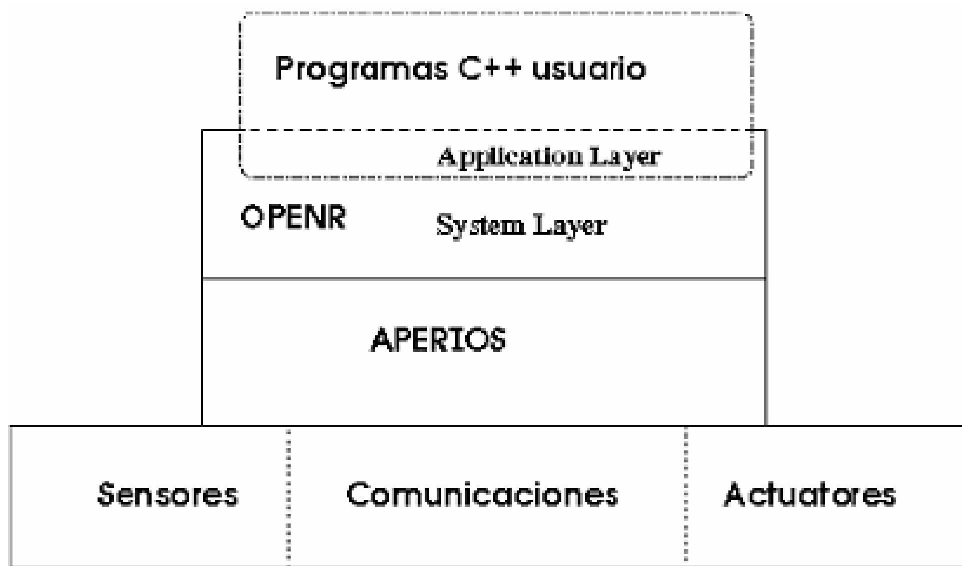

Figura 3. Diseño del Sistema AIBO

Todo en OPEN-R es un objeto. Cada objeto encapsula el estado, métodos que acceden a su estado y un procesador virtual que ejecuta sus métodos. La comunicación entre objetos se realiza mediante paso de mensajes y la ejecución está dirigida por eventos. Así, después de la inicialización un objeto está en estado *idle* y solo la llegada de un evento disparará la ejecución de alguno de sus métodos. Así se distinguen con prioridades eventos relacionados con los mensajes entre objetos y aquellos generados por el hardware. Un evento puede ser por ejemplo un mensaje enviado desde un objeto. La información que se pasan dos objetos se intercambia mediante una zona de memoria compartida. Cuando un objeto está realizando una operación (típicamente un método) no será interrumpido por ningún evento hasta que termina para evitar condiciones de

carrera. Los mensajes que no pueden ser tratados inmediatamente se almacenan en un *buffer* de memoria compartida. OPEN-R no proporciona métodos transparentes de protección de la memoria compartida, lo que complica la programación en OPEN-R directamente. OPEN-R diferencia dos niveles: el *system layer* y el *application layer*. El primero contienen los servicios necesarios para acceder al hardware del robot. El *application layer* es el programado por el usuario como muestra la figura 3.

Los tres objetos principales que proporciona el *system layer* son:

1) OVirtualRobotComm: que proporciona acceso a los sensores y actuadores del robot y a la cámara.
2) OVirtualRobotAudioComm: que proporciona acceso al micrófono y altavoz.
3) ANT: que proporciona la implementación de TCP/IP.

El concepto de objeto en OPEN-R es similar al de proceso en UNIX, con la diferencia de que los objetos son mono-hilo (*single-thread*) y que la comunicación entre ellos se realiza mediante paso de mensajes. Los mensajes contienen los datos (cualquiera en C++) y un identificador que especifica que método se ejecutará cuando llegue el mensaje. De esta forma, cada objeto OPEN-R tiene varios puntos de entrada por los que le pueden llegar mensajes que se especifican en tiempo de compilación en un fichero de configuración similar a los IDL de CORBA que se denomina STUB.CFG La exclusión mutua se resuelve utilizando la clase RCRegion de OPEN-R que puede acceder a un segmento de memoria compartida, llevando un contador de objetos que apuntan a ella.

El flujo de ejecución de los objetos OPEN-R es el que ilustra la figura 4 y que básicamente consiste en:

- El objeto se carga al arrancar el robot, se ejecuta el DoInit de cada objeto.
- Los objetos esperan la llegada de un mensaje.
- Cuando llega un mensaje, el método correspondiente es invocado.
- Cuando un método termina su ejecución, espera la llegada de un nuevo evento.

SUBJECT	***OBSERVER***
Datos Sent	
Evento Notify	
	Datos Recived
	Evento Ready

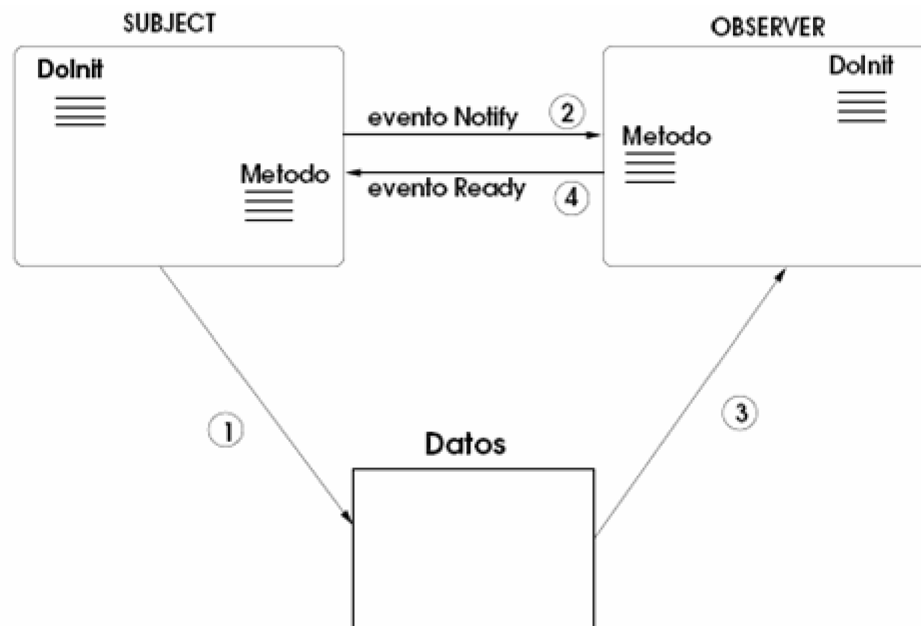

Figura 4. Flujo de interconexión de los objetos en OPEN-R

En la figura 4 se identifican dos objetos, uno que se ha marcado como *Observer* (el que recibe el mensaje) y otro como *subject* (el que lo envía) en la nomenclatura de OPENR.

Obviamente, un objeto puede tener tanto un rol como el otro. Un *observer* se da cuenta de que ha llegado un mensaje cuando recibe un evento Notify del *subject*.

Éste por su parte sabrá que el *observer* está dispuesto a recibir más datos cuando reciba un evento *Ready*, tal y como refleja la figura 4. Este mecanismo, completamente concurrente y orientado a objetos, hace difícil la implementación de aplicaciones en muchos casos, en especial en programadores noveles.

2.4 Lenguajes de programación e interfaces.

2.4.1 Inconvenientes de la programación con OPEN-R.

El principal problema de programar directamente en OPEN-R es que es de bajo nivel, por lo que hay que familiarizarse con el funcionamiento del sistema operativo APERIOS ya que la estructura de los programas es diferente a la que estamos acostumbrados. En APERIOS el sistema se estructura como una serie de objetos que interaccionan entre sí con ayuda de meta-objetos. Por esto, los programas son construidos como una colección de objetos OPEN-R que se ejecutan concurrentemente y que se comunican entre ellos gracias a unas determinadas políticas de paso de mensajes, defiriendo de la estructura habitual a la que estamos acostumbrados a programar. Así pues, tanto la estructura del programa como la de los objetos OPEN-R son distintas a las que estamos familiarizados y son las siguientes:

También hay que mencionar que se necesita una gran cantidad de ficheros de configuración para desarrollar un programa en OPEN-R: Archivos nombreObjeto.h,

nombreObjeto.cc, Makefile, STUB.CFG, nombreObjetoStub.OCF, nombreObjeto.OCF , OBJECT.CFG, CONNECT.CFG, WLANCONF.TXT.

Los objetos OPEN-R tienen una estructura especial que hay que respetar, y es que deben implementar los métodos:

DoInit().
DoStart().
DoStop().
DoDestroy().

Además la comunicación entre objetos OPEN-R no es trivial, ya que se debe establecer una especie de canal que defina el flujo de información que será transmitida entre dos objetos. Un objeto actúa como sujeto, generando datos y otro como observador, recogiendo estos datos y procesándolos. Necesitamos definir la comunicación en los distintos ficheros de configuración lo que supone otra complicación más.

Por otra parte, para acceder a los sensores y actuadores de AIBO debemos trabajar sobre unos objetos especiales encargados de estas funciones. Estos objetos actuarán como observadores o sujetos, en función de las necesidades, y sirven para: uso de la cámara, lectura de sensores y control de movimientos.

2.4.2 Lenguajes basados en Scripts: URBI (Universal Real-Time Behavior Interface).

URBI es un lenguaje de script de alto nivel, independiente de la plataforma destino y de desarrollo, y que ofrece mecanismos para paralelizar comandos.

La arquitectura URBI se basa en un motor interno y una serie de módulos remotos.

Para desarrollar programas en URBI, se puede utilizar el lenguaje de script, para después incluir un archivo de texto plano en lenguaje OPEN-R en la memory stick, y que el sistema operativo APERIOS puede interpretar y ejecutar.

Otra opción es realizar programas que controlen de forma remota el comportamiento del robot, para lo que se pueden usar las librería facilitadas en distintos lenguajes (Matlab, C++, java, Open-R).

En nuestro caso, estamos usando la librería en java para el desarrollo de un programa que controle el robot de forma remota enviándole los scripts de comportamientos según los datos que reciba el programa del robot.

La principal ventaja por la que decidimos utilizar URBI es que permite desarrollar programas de propósito general, integrando comunicación con los robots de forma sencilla, y con una estructura de programa con la que estábamos familiarizados.

2.5 Interfaces de manejo para AIBO

Una interfaz de manejo para un robot consiste en una aplicación que nos proporcione una interfaz gráfica a través de la cual podamos realizar diferentes funciones sobre los distintos elementos del robot. Un ejemplo de ello sería la posibilidad de modificar los valores de las articulaciones del robot a nuestro antojo, siempre y cuando los valores que le demos entren dentro del rango de los posibles valores que puede tomar dicha articulación.
Estas interfaces deben ser lo más amigables posibles y a su vez proporcionarnos el mayor número posible de funcionalidades para facilitarnos el manejo del robot.
Para el robot AIBO ERS-7 hemos encontrado dos interfaces que nos podían resultar de ayuda: Tekkotsu y Telecommande. De estas dos finalmente nos hemos quedado con Telecommande porque era la más amigable y la que más fácilmente se podía instalar en los PCs de que disponíamos. Además de estos dos motivos, hay que añadir que Tekkotsu no solo es una interfaz de manejo del AIBO, sino que también es una estructura que permite la programación a más alto nivel del robot, por lo que la complejidad de manejo del interfaz era un inconveniente que nos ayudó a tomar la decisión de descartarlo.

2.5.1 Tekkotsu. [3]

Tekkotsu significa 'Huesos de hierro' en japonés, a menudo utilizado en el contexto de los armazones estructurales de los edificios. Similarmente, este paquete de software pretende ofrecerte una estructura sobre la que construir, manejando tareas rutinarias de modo que puedas centrarte en la programación de alto nivel.
Tekkotsu es un armazón de desarrollo de aplicaciones para robots inteligentes, que utiliza una arquitectura orientada a objetos y de paso de eventos, haciendo un uso completo de las características de la plantilla y de la herencia de C++. Fue originalmente escrito para el Sony AIBO, pero desde entonces ha evolucionado para funcionar en una mayor variedad de robots.

Puesto que Tekkotsu está escrito en C++ industrial estándar, no hay nuevos lenguajes que aprender, y no hay grandes diferencias entre "alto nivel" y "bajo nivel", así pues se puede empezar a manejar rápidamente y poco a poco ir descubriendo las diversas funcionalidades que proporciona el API. Para el futuro, Tekkotsu pretende complementar, no suplantar, el acceso a la capa más baja de las características del hardware del host. Por ejemplo, aunque probablemente nunca se vaya a necesitar hacer una llamada OPEN-R directa cuando se programa con Tekkotsu en el AIBO, la posibilidad de hacerlo está ahí. Esto significa que las capacidades de manejo del robot están limitadas solo por el hardware en sí mismo, no por el API.

Algunos de los servicios que provee Tekkotsu incluye procesamiento visual básico, resolutores de la cinemática delantera y trasera, herramientas de monitorización y teleoperación remota, y soporte para la creación de redes wireless. Una larga librería de tutoriales y comportamientos de demostración están incluidos para ayudar a la inicialización en la aplicación, con una documentación de referencia muy extensa para el caso en que tengas preguntas específicas. Es código abierto, proyecto de software libre que se basa en varias bibliotecas de terceros, tales como ROBOOP (cinemática general), NEWMAT (matriz de las operaciones), libjpeg, libpng, libxml2, y zlib.

El panel de la GUI del Controlador mostrado más abajo es la interfaz principal para el control del robot desde el PC. También existe la posibilidad de manejar el robot a través de la consola con opciones similares si no se quiere utilizar una interfaz GUI basada en Java (ejemplo en figura 5).

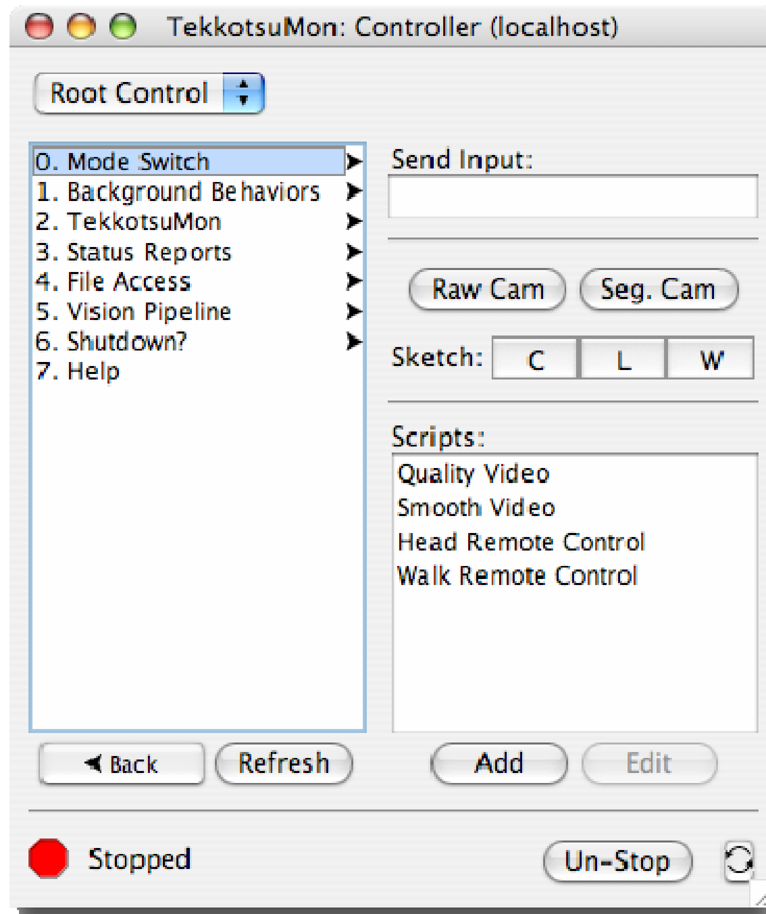

Figura 5. Menú de ejemplo de la GUI del Controlador

Se puede emitir el vídeo que capta el robot en tiempo real con RawCam y supervisar el rendimiento de la segmentación de color con SegCam, que son lanzados desde el ControllerGUI. Podemos observar un ejemplo en la Figura 6. También se pueden grabar estas emisiones y transformarlas en secuencias de imágenes para el procesamiento posterior. La ventana del RawCam también permite seleccionar qué espacio de color ver. La compresión, la resolución y las opciones de canal también pueden ser modificadas. Los códigos de protocolo de red de bajo nivel para todos los servicios están separados de la interfaz gráfica de usuario en clases independientes de la aplicación para facilitar la reutilización para otras aplicaciones, tales como mashups de GUIs o procesamiento off-board.

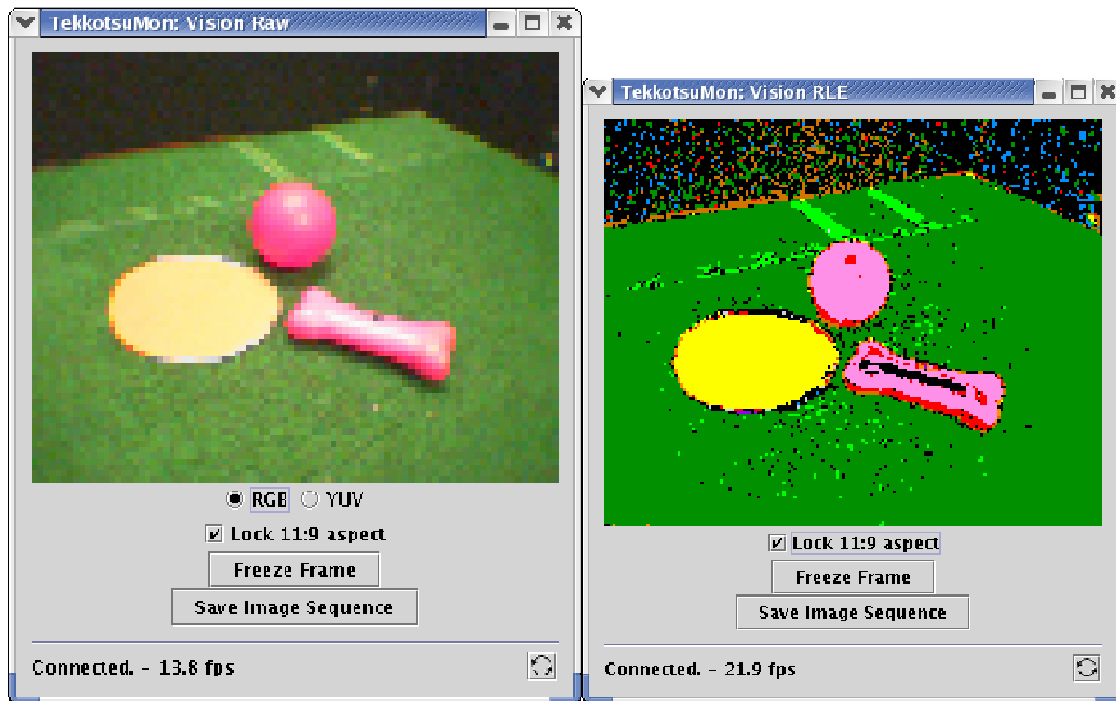

Figura 6. Imágenes de RawCam y SegCam

El AIBO podrá ser manejado utilizando la interfaz de Control Remoto de Movimiento (Fig. 7). Combinando esto con la monitorización de la visión, obtenemos una herramienta muy completa para el control remoto del robot.

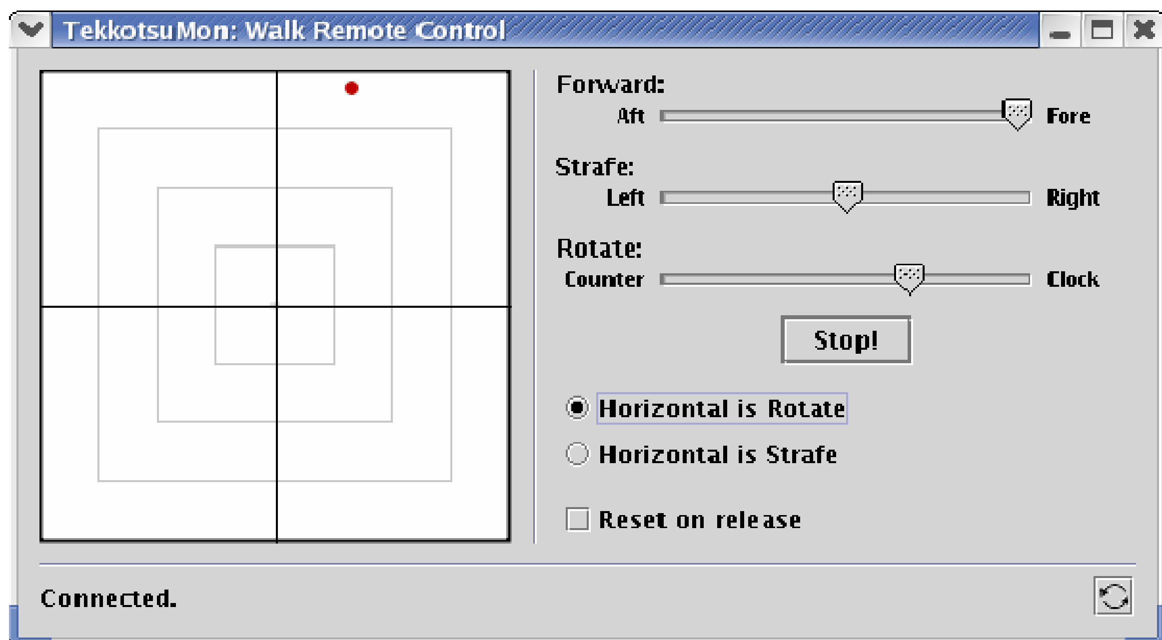

Figura 7. GUI de Control Remoto del Movimiento

2.5.2 Telecommande.

AIBO Telecommande es una interfaz gráfica para controlar un servidor genérico URBI. Esta interfaz es muy fácil de usar, y nos proporciona una gran cantidad de funcionalidades para manejar el robot AIBO.

Para ello, lo primero que hay que hacer nada más arrancar la aplicación es conectarse al robot a través del WiFi de que dispone, y mediante una red ad-hoc previamente configurada. Una vez conectado al robot, la aplicación nos ofrece una interfaz muy amigable con la que podemos realizar 8 tipos de funciones distintas:

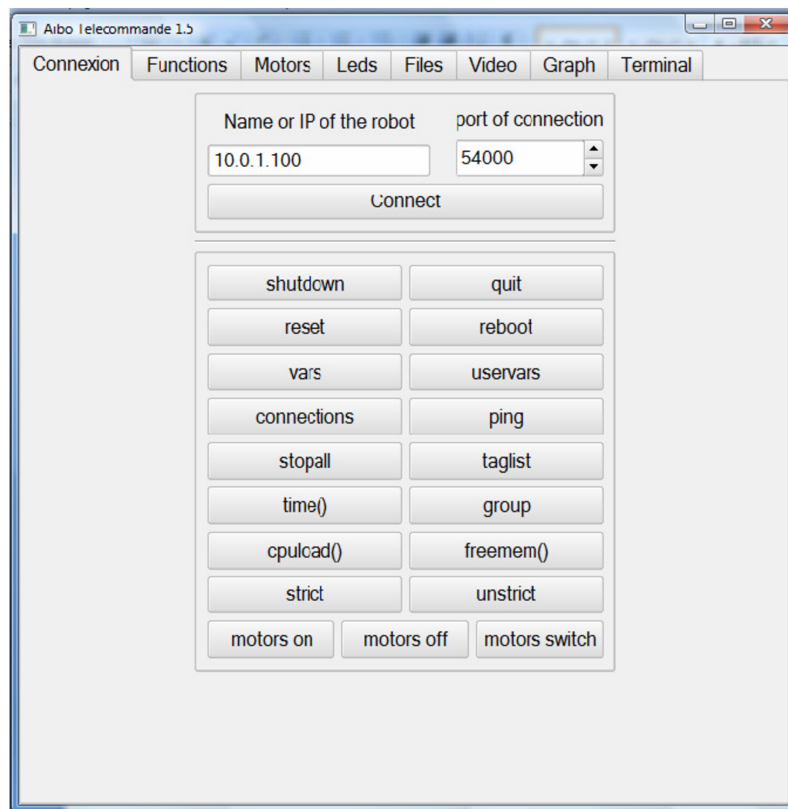

Figura 8. Funciones de conexión (Connexion).

En esta sección se pueden realizar las funciones primarias para inicializar el robot. Estas funciones son por ejemplo la función de conectar la aplicación con el AIBO, activar los motores, reiniciar el robot, apagarlo, parar todas las funciones que se estén ejecutando en el, etc. Los pasos a seguir siempre una vez se haya encendido el AIBO son los siguientes: conectar la aplicación con el robot, activar los motores y, finalmente empezar a mandar funciones o comandos para que los ejecute. En la figura 8 se puede observar la ventana que gestiona las funciones aquí descritas.

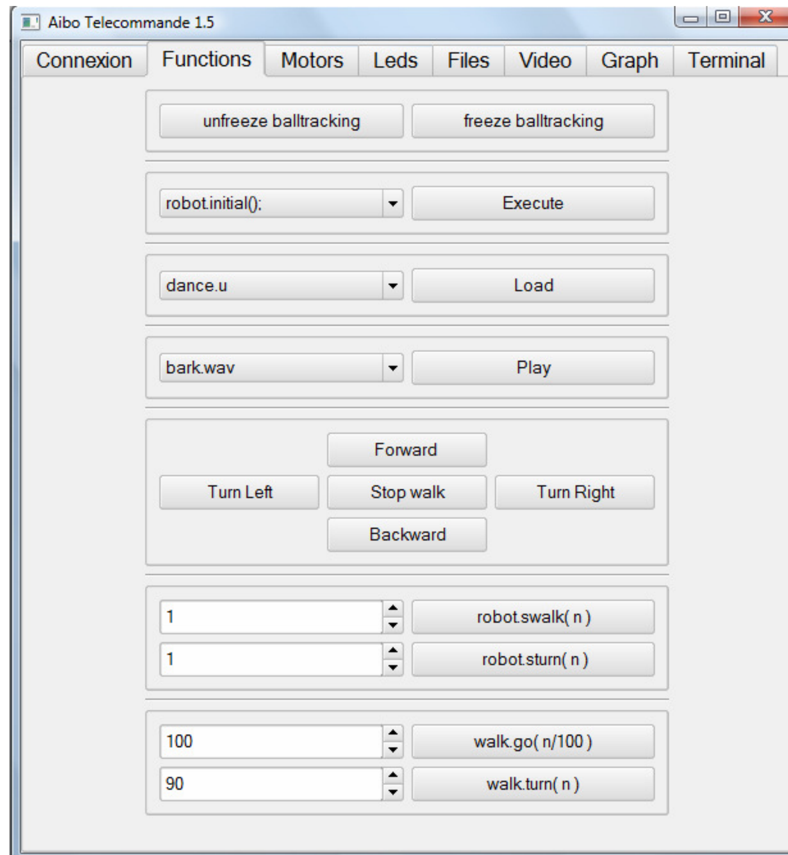

Figura 9. Funciones internas del robot (Functions).

En este apartado de la interfaz podemos ejecutar funciones ya implementadas en el robot como pueden ser las funciones de andar en todas direcciones, poner al robot en posición inicial, hacer que se tumbe, que se siente, etc.

También nos permite cargar comportamientos más complejos incluidos en el propio simulador en forma de ficheros .u como pueden ser bailes, activar el chat, etc.

Otra opción que tenemos disponible es la de hacer que el AIBO reproduzca sonidos disponibles también en el propio simulador como archivos .wav, como por ejemplo un ladrido.

En la figura 9 se muestra la interfaz del Telecommande relacionada con las funciones internas del robot.

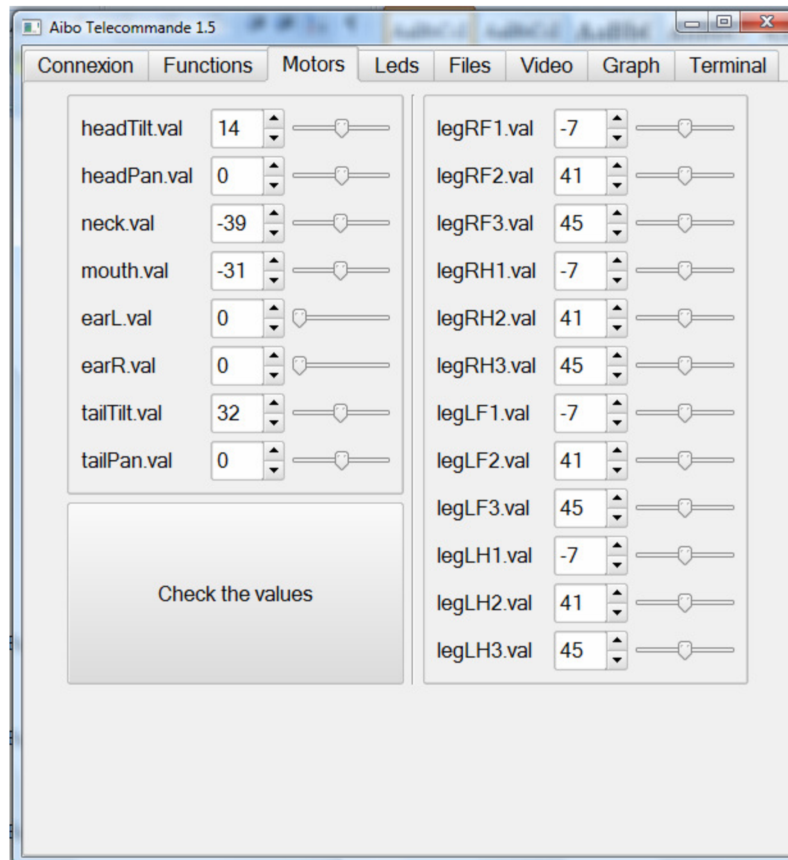

Figura 10. Funciones de manejo de motores (Motors).

En la ventana mostrada en la figura 10 podemos encontrar los valores de todas las articulaciones del robot y modificarlos ya sea introduciendo el valor directamente, o desplazando la barra que hay a la derecha de cada una de ellas. Al modificar el valor de una articulación en esta pantalla, el cambio se observa inmediatamente en el AIBO siempre y cuando esté encendido, conectado a la aplicación y con los motores activos.

Esta ventana nos ha sido de gran ayuda al crear las diferentes posiciones del robot para cada uno de los comportamientos que hemos implementado.

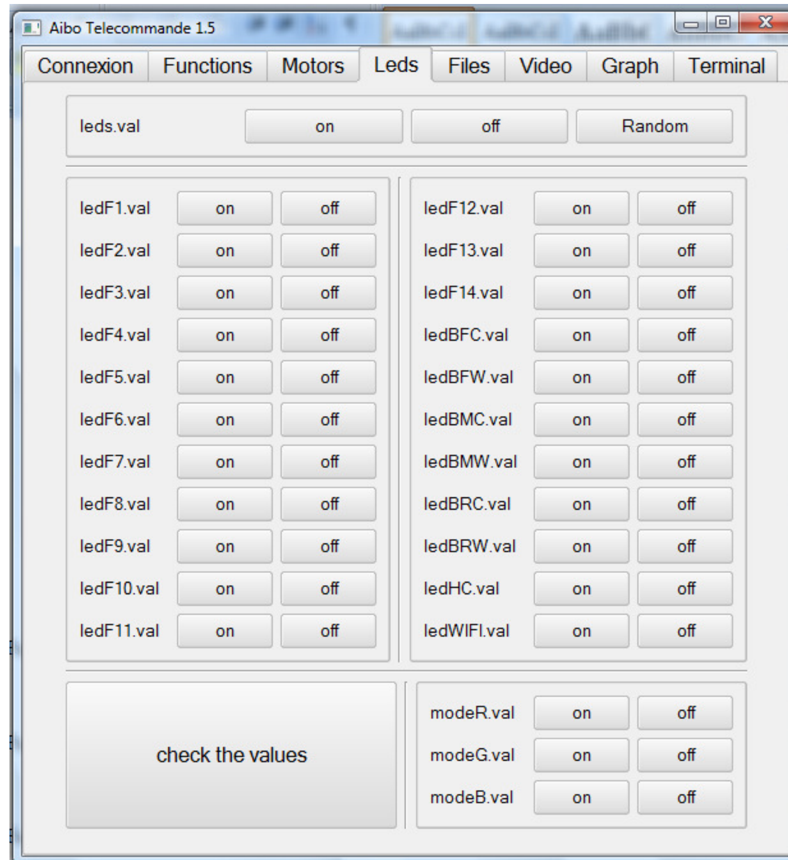

Figura 11. Funciones de manejo de los diferentes leds (Leds).

La ventana de leds que podemos ver en la figura 11 es muy parecida a la de los motores anteriormente explicada y mostrada en la figura 10. Análogamente a la anterior, esta muestra los nombres de todos los leds disponibles en el AIBO, y dos botones a la derecha de cada uno correspondientes a los valores de encendido y apagado. Así pues, si pulsamos alguno de estos botones de encendido o apagado el cambio realizado se observará en el AIBO inmediatamente siempre y cuando esté encendido y conectado con la aplicación.

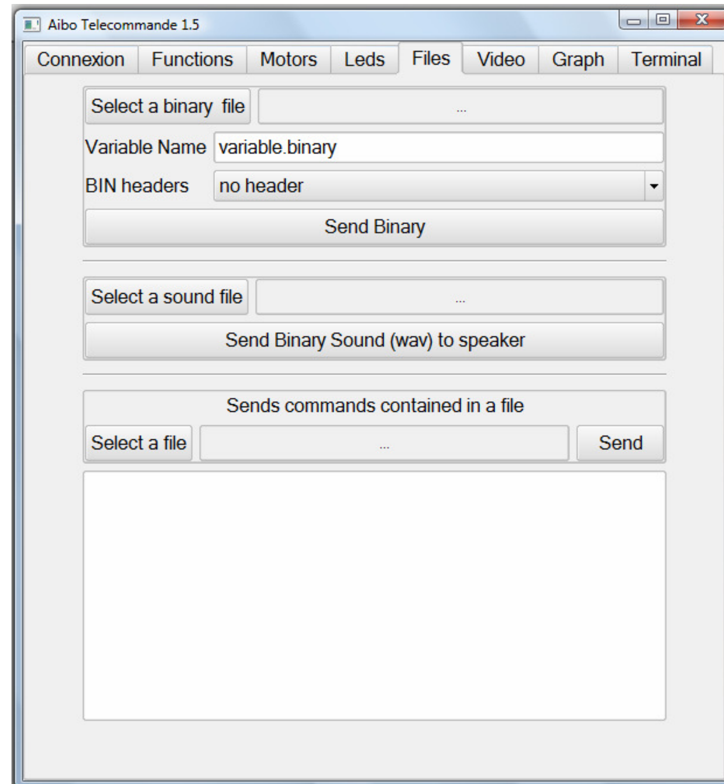

Figura12. Funciones de cargado de ficheros en el robot (Files).

Esta interfaz nos proporciona la posibilidad de cargar nuestros propios ficheros de comportamiento con extensión .u, de sonido con extensión .wav, y de archivos binarios. De esta forma le podemos enviar al AIBO ficheros que implementen un comportamiento, que reproduzcan un sonido, o que configuren variables del robot.

La figura 12 nos muestra la disposición de los diferentes elementos que podemos encontrar en esta interfaz.

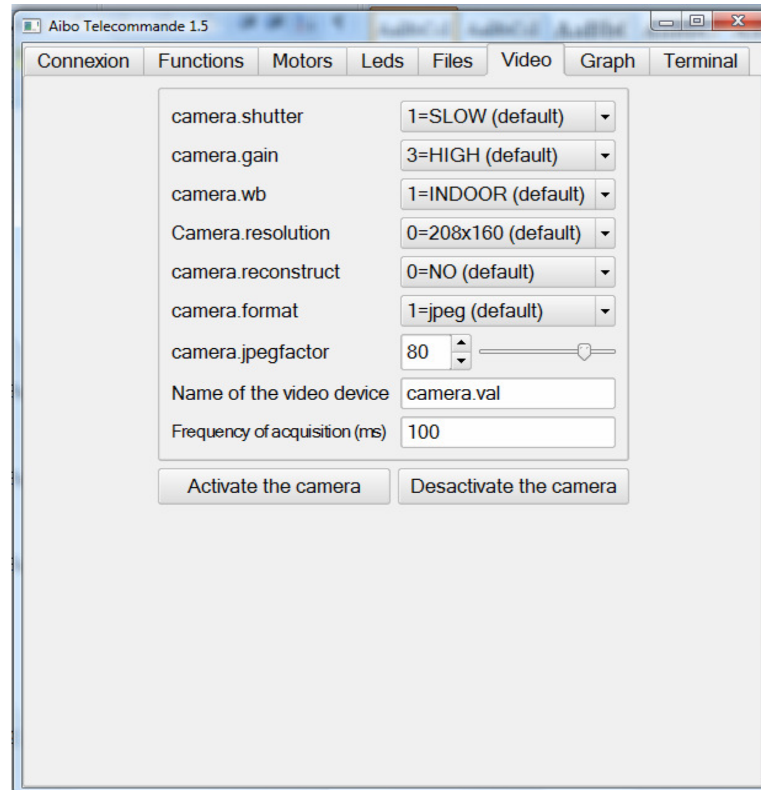

Figura 13. Funciones de configuración y manejo de la cámara del robot (Video).

En esta sección de la aplicación podemos configurar los diferentes parámetros de la cámara como son la resolución, el formato, los valores del obturador, etc. También podemos activar la cámara o desactivarla. Estas funcionalidades nos las proporcionan lo elementos de la interfaz que vemos en la figura 13.

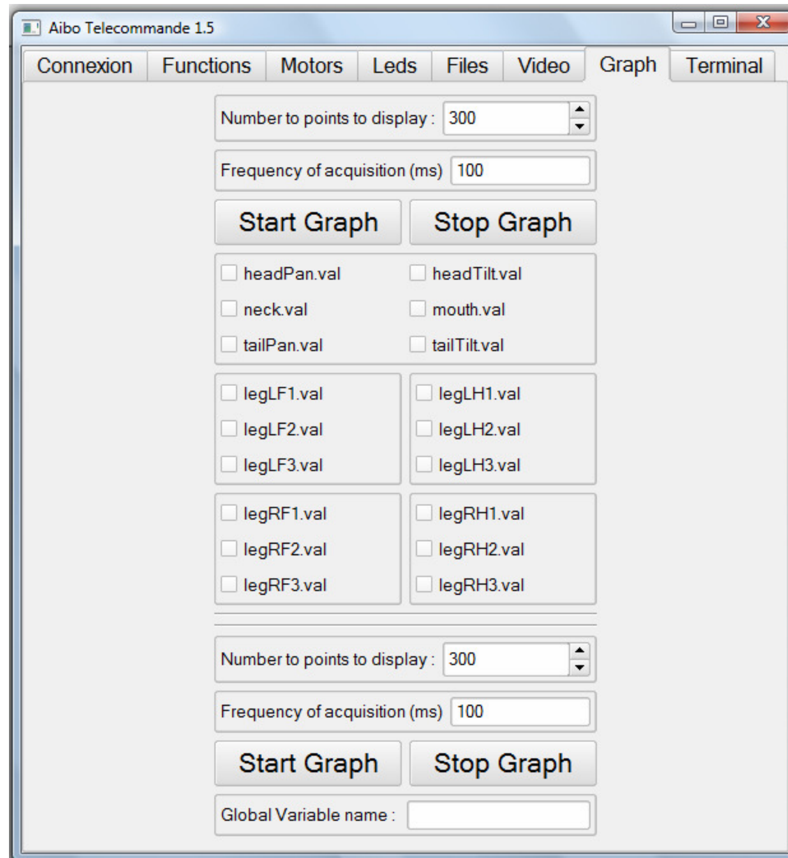

Figura 14. Funciones de obtención de diversas graficas (Graph).

La pestaña de gráficas plasmada en la figura 14 nos muestra una ventana que permite generar una gran variedad de gráficas mostrando los valores que van tomando un set de articulaciones que podemos definir nosotros a nuestro gusto. También podemos configurar el número de puntos que queremos mostrar y la frecuencia con la que queremos obtener los datos. También podemos obtener gráficas de variables globales que hayamos definido nosotros o que estén definidas en el AIBO como por ejemplo ball.visible.

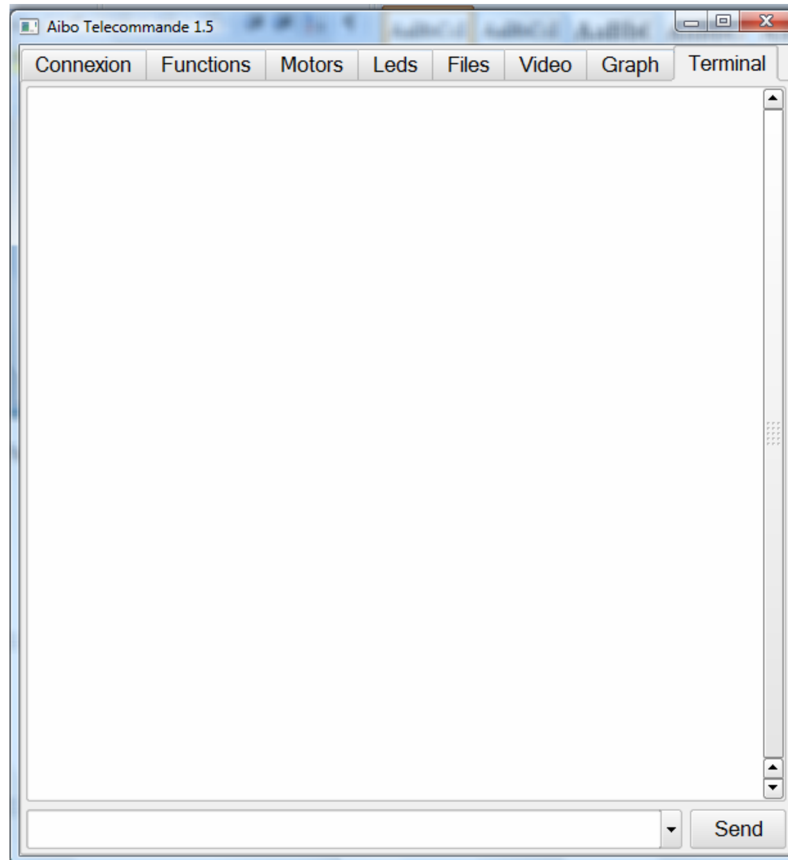

Figura 15. Funciones de envío de comandos (Terminal).

La pantalla de la terminal que tenemos en la figura 15 nos permite enviar comandos al robot para que los ejecute directamente, pero para ello tiene que estar encendido, conectado y con los motores activados.

Todas estas funciones se envían al robot de forma simple y eficiente. Quizá una de las funcionalidades más útiles que hemos encontrado en este simulador ha sido la de enviar segmentos de código URBI a través de la terminal, y poder comprobar simple y rápidamente si funcionaba correctamente según nuestras expectativas. Así a la hora de crear comportamientos como por ejemplo andar lateralmente, realizar una parada, etc. podíamos comprobar relativamente rápido si cada paso lo realizaba correctamente. En caso de que fuera erróneo podíamos modificarlo y volverlo a comprobar de una manera más o menos cómoda y rápida.

También nos ha sido de gran utilidad las funciones de manejo de los motores, ya que todos los comportamientos que hemos tenido que programar han sido a nivel de robótica, ajustando al máximo las articulaciones. Por esto, el hecho de tener esta opción de manejar los motores de una manera tan sencilla y poder ver los valores que nos da en cada momento, ha sido una gran ayuda, ya que si lo hubiéramos tenido que comprobar mediante generación de código, ejecución y observación, podríamos haber tenido serios problemas de tiempo.

Una de las funcionalidades que nos podría haber sido también de gran ayuda, pero que no conseguimos hacer que funcionara, es opción de visualizar en nuestra pantalla las

imágenes que capta la cámara del AIBO, y así poder saber si los fallos de funcionamiento del robot son debidos a errores introducidos en el código, o debidos que el dispositivo no tiene la suficiente calidad para nuestras necesidades.

2.5.3 Simuladores.

Una simulación es la experimentación con un modelo de una hipótesis o un conjunto de hipótesis de trabajo.
Hector Bustamante de la O la define así: "Simulación es una técnica numérica para conducir experimentos en una computadora digital. Estos experimentos comprenden ciertos tipos de relaciones matemáticas y lógicas, las cuales son necesarias para describir el comportamiento y la estructura de sistemas complejos del mundo real a través de largos periodos de tiempo".
Una definición más formal formulada por R.E. Shannon es: "La simulación es el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el comportamiento del sistema o evaluar nuevas estrategias -dentro de los límites impuestos por un cierto criterio o un conjunto de ellos - para el funcionamiento del sistema".
Para poder reproducir una simulación necesitamos un simulador. Un simulador es una aplicación que permite la simulación de un sistema, reproduciendo su comportamiento. Los simuladores reproducen sensaciones que en realidad no están sucediendo.
Un simulador pretende reproducir el comportamiento de los equipos de la máquina que se pretende simular. Para reproducir el entorno exterior se emplean proyecciones de bases de datos de terreno. A este entorno se le conoce como "Entorno Sintético". A las proyecciones de terrenos se las conoce como “mundos” y pueden contener diferentes objetos de muy diversas características. Además el mundo debe estar regido por unas leyes físicas que estarán programadas en el propio simulador, y que dotarán de más o menos realismo a la simulación. Así pues, cuantas más leyes se implementen y más se detallen, mejor será nuestro simulador.
En el dominio en el que estamos trabajando (el de los robots), los simuladores que hay disponibles permiten simular el comportamientos de los robots en diferentes mundos. Esto nos sirve para crear comportamientos y probarlos en el simulador rápidamente, sin necesitar tener disponible el robot, lo que conlleva tener la batería cargada; un espacio en el que probarlo que se parezca al que necesitamos; y las condiciones de iluminación adecuadas para nuestros propósitos.
Hay disponibles varios simuladores, pero la mayoría son de pago, por lo que nosotros en un principio decidimos investigar el funcionamiento de dos de ellos: UsarSim que es un simulador gratuito, y Webots que es un simulador de pago pero existe una versión de prueba gratuita. Tras un arduo proceso de familiarización de los simuladores, decidimos no utilizar ninguno de ellos. UsarSim lo descartamos porque solo se podía utilizar en uno de los ordenadores de que disponíamos, y no conseguimos hacer que funcionara correctamente siempre. La idea de utilizar Webots la desestimamos debido a que no conseguimos encontrar un controlador del robot AIBO ERS-7 que funcionara los mundos de que disponíamos, lo cual hacía infructuoso el uso de este simulador.
Vamos a explicar el funcionamiento en detalle del simulador Webots como ejemplo de un simulador de robots para que sirva de referencia.

Webots [4].

Webots es un paquete profesional de software de simulación para robots móviles. Ofrece un rápido entorno de prototipado que permite a los usuarios crear mundos virtuales en 3D con propiedades físicas como la masa, engranajes, coeficientes de fricción, etc. También se pueden añadir objetos pasivos simples u objetos activos llamados robots móviles. Esos robots pueden tener diferentes esquemas de locomoción (robots de ruedas, robots de patas o robots voladores). Además, pueden ser equipados con una serie de sensores y actuadores, tales como sensores de distancia, ruedas de conducción, cámaras, servos, sensores de tacto, pinzas, emisores, receptores, etc. Finalmente, el usuario puede programar cada robot individualmente para mostrar el comportamiento deseado.

Este simulador contiene un gran número de modelos de robots y ejemplos de controladores para ayudar al usuario a iniciarse en la aplicación. También dispone de un buen número de interfaces para robots móviles reales, por lo que una vez simulado en la aplicación el comportamiento que se desea y ver que funciona como se espera, puede ser cargado en el robot real como por ejemplo e-puck, Khepera, Hemisson, LEGO Mindstorms, AIBO, etc.

Webots está diseñado para investigación y proyectos educacionales relacionados con robots móviles. Varios proyectos robóticos han confiado en Webots durante años en las siguientes áreas:

- Prototipado de robots móviles (investigación académica, la industria automovilística, aeronáutica, la industria de limpieza del vacío, la industria del juguete, hobbies, etc.).
- Investigación de robots de locomoción (articulados, humanoides, robots cuadrúpedos, etc.).
- Investigación multi-agente (enjambre inteligente, grupos de colaboración de robots móviles, etc.).

Aunque no es necesario un conocimiento especial, simplemente para ver la demostración de una simulación de robots en Webots se necesita un mínimo de conocimientos técnicos para poder desarrollar simulaciones propias:

Un conocimiento básico de C, C++, Java, lenguaje de programación Python o MATLABTM es necesario para programar nuestro propio controlador del robot. Sin embargo, incluso si no se saben estos lenguajes de programación, se puede programar los robots e-duende y Hemisson usando un sencillo lenguaje de programación gráfica llamado BotStudio.

Si no se desean utilizar los modelos de robot que proporciona Webots y se quisiera crear modelos propios de un robot, o añadir objetos especiales en la simulación de entornos, se dispondrá de una base de conocimientos de gráficos por ordenador en 3D y el lenguaje de descripción VRML97 3D. Esto permite crear modelos en 3D en Webots o importarlos de software de modelado 3D.

Una simulación en Webots está compuesta por lo siguiente:

- Un fichero de un mundo de Webots que define uno o más robots 3D en sus entornos.
- Programas controladores para el/los robot/s mencionado/s en el punto anterior.
- Un supervisor opcional.

Un mundo en Webots, es una descripción en 3D de las propiedades de un robot y su entorno. Contiene una descripción de todos los objetos: su posición, orientación, geometría, apariencia (como color o brillo), propiedades físicas, tipo de objeto, etc. Los mundos están organizados como unas estructuras jerárquicas donde los objetos pueden contener otros objetos (como en VRML97). Por ejemplo, un robot puede contener 2 ruedas, un sensor de distancia y un servo que el mismo contiene a su vez una cámara, etc. El fichero de un mundo no contiene el código del controlador de los robots; sólo especifica el nombre del controlador que es requerido para cada robot. Estos mundos están almacenados en ficheros .wbt. Estos ficheros .wbt están almacenados en el subdirectorio de mundos de cada proyecto Webots.

¿Qué es un controlador?

Un controlador es un programa de ordenador que controla un robot especificado en el fichero de un mundo. Estos controladores puede ser escritos en cualquier lenguaje de programación soportado por Webots, y estos son: C, C++, Java, URBI, Python o MATLABTM.

Cuando una simulación comienza, Webots lanza los controladores especificados, cada uno de ellos como un proceso separado e independiente, y asocia los procesos de los controladores con los robots simulados. Hay que tener en cuenta que varios robots pueden usar el mismo código de controlador, aunque se lanzará un proceso distinto por cada robot.

Algunos lenguajes de programación necesitan ser compilados (C y C++), otros lenguajes necesitan ser interpretados (URBI, Python y MATLABTM) y algunos otros necesitan ser tanto compilados como interpretados (Java). Por ejemplo, los controladores de C y C++ son compilados a ejecutables binarios dependientes de la plataforma (por ejemplo .exe sobre Windows). Los controladores de URBI, Python y MATLABTM son interpretados por el correspondiente sistema run-time (el cual tiene que ser instalado). Los controladores en java necesitan ser compilados a un código de bytes (ficheros .class o .jar) y posteriormente interpretados por una Máquina Virtual de Java. Los ficheros fuente y binarios de cada controlador están almacenados juntos en el directorio del controlador. El directorio de un controlador está situado en el subdirectorio de los controladores de cada proyecto Webots.

¿Qué es un supervisor?

El Supervisor es un tipo privilegiado de robot que puede ejecutar operaciones que normalmente solo pueden ser realizadas por un operador humano y no por un robot real. El Supervisor está asociado normalmente al programa de un controlador también puede ser escrito en cualquiera de los lenguajes de programación anteriormente mencionados. De cualquier forma, en contraste con un controlador de robots normal, el controlador del Supervisor tendrá acceso a las operaciones privilegiadas. Las operaciones

privilegiadas incluyen control de la simulación, por ejemplo, moviendo los robots a posiciones aleatorias, hacer capturas de vídeo de la simulación, etc.

Con todo esto podemos decidir que Webots es un simulador muy completo y profesional que nos permite crear nuestro propio controlador del robot o utilizar alguno ya existente, y crear simulaciones muy variadas y realistas. Pero estas posibilidades tan avanzadas y completas que nos proporciona el simulador, conllevan también un alto grado de complejidad en el manejo y aprendizaje del simulador. Esto es debido a que muchas de las funcionalidades sencillas que nos proporciona Webots se llevan a cabo de la misma manera que funcionalidades mucho más complejas, y que requieren una mayor complejidad en la herramienta que las proporciona. Por todo esto creemos que Webots es una herramienta muy útil de simulación, pero que requiere unos conocimientos que si no se dispone de ellos, se deberá invertir bastante tiempo en el aprendizaje de su manejo, y habrá que evaluar muy detenidamente si conviene o no todo este gasto de recursos. En nuestro caso no fue necesario hacer todo esto puesto que al ser un programa de pago utilizamos una versión de prueba gratuita que no nos proporcionaba la funcionalidad para satisfacer nuestras necesidades, y nos llevó bastante tiempo conseguir instalarlo y aprender a manejarlo, por lo que descartamos la posibilidad de comprarlo y utilizarlo para nuestros propósitos.

2.6 Comparativa Urbi, OPEN-R y Tekkotsu. Ventajas e inconvenientes.

Realmente, son soluciones bastante distintas. OPEN-R es un SDK de Sony que permite programar tu robot en C++ usando las librerías y framework OPEN-R. Es bastante complicado de entender y obliga a usar el sistema operativo APERIOS y su paso de mensajes. Una vez lo comprendes la complejidad pasa a ser un problema menor pero aún así carece de portabilidad porque sólo funciona con robots Sony (en nuestro caso en concreto esto no es un problema, pero podría darse en otros proyectos). Además, debido a su naturaleza, es habitual que la depuración requiera de una cantidad de tiempo mayor que con lenguajes de más alto nivel. Como ventaja es claramente el lenguaje más potente y preciso, y puede ser necesario cuando el proyecto necesite explotar las capacidades del AIBO como robot al máximo.

Tekkotsu proporciona una capa de mayor nivel que OPEN-R, en principio en el marco de la AIBO robocup, pero extendiéndose a problemas robóticos más generales y a más tipos de robots. Al igual que OPEN-R la arquitectura es complicada y está basada únicamente en librerías C++. Permite capacidades de mayor nivel que URBI. Tekkotsu es código abierto y está bien mantenido por CMU.

URBI es un lenguaje basado en scripts que simplifica el modo en que se utilizan los motores y sensores. URBI permite ejecutar comandos en paralelo mediante el operador & a diferencia de otros lenguajes basados en scripts. Además se puede integrar en C++, Java y Matlab usando liburbi y la arquitectura cliente/servidor, se podría incluso utilizar distintos lenguajes a la vez, con distintos clientes.

Teniendo todo esto en cuenta la opción que más nos convenció fue utilizar URBI para el proyecto. URBI es generalmente más simple de usar que otras arquitecturas de control de robots; manteniéndose, a pesar de ello, potente. Te permite trabajar con distintos

lenguajes, con lo que pudimos elegir JAVA en vez de C++, esto nos permitió trabajar con un lenguaje que conocíamos mejor, facilitándonos la tarea. Es además bastante flexible, pudiendo ejecutar parte del código en el robot mientras que otras se ejecutan en un equipo remoto o incluso en varios en distintos lenguajes. Otras características notables son:

- Acceso a los sensores y manejo de los actuadores simple e intuitivo.
- Es mejor que otros lenguajes tradicionales por sus características orientadas a la robótica; procesamiento de comandos en paralelo/serie, manejo de eventos, asignaciones complejas (como asignar una trayectoria a un motor), modelos de mezclas para manejar conflictos en las asignaciones, etc.
- Existencia de Telecommande como cliente telnet, sencillo y potente, para el envío de comandos y recepción de mensajes, permitiendo pruebas rápidas y facilitando la codificación de scripts.

3. Metodología GMG de diseño de comportamientos para AIBO ERS7.

Una vez terminada la etapa de investigación y elección de los distintos lenguajes e interfaces que hay disponibles para trabajar en el desarrollo de comportamientos para el AIBO, el siguiente paso que era obligatorio abordar era la implementación de comportamientos de prueba para desarrollar nuestra propia metodología de trabajo en lo que a creación de comportamientos autónomos para el robot se refiere.
Es importante que cualquier persona que quiera trabajar en la creación de un módulo de Inteligencia Artificial para AIBO esté familiarizado con el uso de las librerías específicas que hay disponibles para ello, ya que si surge algún problema el usuario debe disponer de las herramientas necesarias para resolverlo, pero al tratarse de una herramienta utilizada en investigación y de código abierto, hay disponibles numerosos conjuntos de comportamientos ya implementados y testados que pueden ahorrar mucho trabajo al investigador. Como en todos los aspectos de la informática la reutilización de código ya desarrollado por los demás puede resultar muy provechosa, "no hay que reinventar la rueda".
En esta fase del trabajo se resolvieron problemas tales como los de comunicación terminal/AIBO y paso de mensajes para conocer el estado de los distintos sensores del perro, y se aprendió a manejar los distintos actuadores y motores de los que dispone el robot para conseguir la respuesta adecuada a los estímulos de entrada. Además nos ayudó a adquirir cierta experiencia en el manejo del lenguaje de programación URBI que fue el lenguaje elegido para la creación de comportamientos para el AIBO.

Ya desde un principio elegimos diseñar los distintos casos utilizando máquinas de estados debido a su facilidad de comprensión y creación, ya que es un método visual que puede entender hasta un usuario poco experimentado, y que es un modelo que resulta sencillo de trasladar a código URBI porque este lenguaje proporciona las herramientas necesarias, como pueden ser disparadores y temporizadores, para su posterior implementación.

3.1 Análisis de problemas.

A la hora de afrontar el análisis de un problema concreto es importante tener en cuenta que nuestra intención ha sido desarrollar comportamientos muy sencillos y que en un principio no definen una acción muy compleja. Esto es debido a que decidimos que para la creación de comportamientos más completos se pueden utilizar un conjunto de ellos más simples; además el tiempo invertido en el proceso de desarrollo de un comportamiento reutilizando unos más sencillos es mucho menor al empleado en crear el comportamiento pasando por todas las fases de la metodología empleada.

Para un desarrollo ordenado y eficiente de comportamientos para el AIBO, hemos creído oportuno seguir una serie de pasos:

- Planificación: Se trata del primer paso que hay que llevar a cabo. Consiste en la

identificación clara de las acciones que se quiere que realice el AIBO y como, y las herramientas de que disponemos para desarrollar dicho comportamiento. Esta etapa es de gran importancia ya que al ser una metodología iterativa, todas las etapas van a depender de la anterior, y al ser esta la primera, un error en ella provocará que dicho error se arrastre hasta la última fase, con lo que el trabajo realizado será en vano.

- Diseño e implementación de la máquina de estados: Una vez se tiene definido el comportamiento que se desea implementas, es necesario formalizarlo, para lo cual existen multitud de técnicas. Nuestra metodología emplea las máquinas de estados porque se adaptan perfectamente a los elementos que disponemos para realizar los cambios de estados del robot, y los propios estados son perfectamente identificables. Así pues, las transiciones entre estados vendrán disparadas por los cambios en los valores de los sensores del robot. Los estados los podremos identificar como posiciones del AIBO según sus articulaciones y/o sus percepciones del entorno.

- Traducción al lenguaje elegido: El siguiente paso consiste en traducir la máquina de estados al lenguaje elegido. Para ello es necesario tener conocimiento previo de las herramientas disponibles para implementar las acciones y transiciones de los estados. Una vez comprobado que disponemos de las herramientas necesarias, la mejor manera de programar este tipo de máquinas de estados de comportamientos en URBI es usar una conjunción de funciones y comandos “at” y “stop”.

- Prueba y vuelta a empezar: Finalmente, habrá que probar el funcionamiento del comportamiento creado cargándolo en el robot y ejecutándolo en las condiciones previstas. Si se observa que hay algún error en dicho comportamiento, habrá que anotarlo e identificar en qué parte de la máquina de estados falla para poder modificarla, o si no se trata de un error en las acciones, revisar el código generado en busca de las posibles erratas.

3.1.1 Planificación.

En el desarrollo de un comportamiento la buena planificación de lo que se quiere hacer y el manejo de las herramientas de las que disponemos para hacerlo son muy importantes. Esto es debido que al ser una metodología iterativa (planificación, diseño, implementación, pruebas y vuelta a empezar) todas las etapas dependen de la fase anterior de desarrollo. Al ser la planificación la primera etapa de esta metodología, y depender todas las demás de ella, si se ha realizado correctamente no será necesario volver a replanificar en iteraciones posteriores, pero si la solución al problema ha sido mal estructurada desde el principio el error se puede detectar en una iteración avanzada y todo el trabajo hecho hasta el momento tendría que ser desechado y habría que empezar desde el principio proponiendo una nueva planificación.
La primera decisión fundamental que hay que tomar y que tiene que estar muy bien definida es el objetivo principal del comportamiento, es decir, que queremos que haga el robot. Una vez definido la meta final, se pueden empezar a fijar objetivos intermedios, que son la semilla de lo que posteriormente serán los estados de la propia máquina de estados que se creará en la etapa siguiente. Siempre hay que tratar que estos subjetivos

sean lo más atómicos e independientes unos de otros como sea posible, ya que si se detecta que de uno de ellos pude haber varias transiciones posibles habrá que plantearse crear un comportamiento independiente para esta meta intermedia; este nuevo comportamiento será incluido en el diseño del comportamiento inicial. Esta manera de trabajar favorecerá la futura reutilización de código cuando se esté trabajando en el desarrollo de conductas para el AIBO.
Todas estas fases intermedias se alcanzan teniendo en cuenta los valores de los sensores del robot, por esta razón es importante definir claramente el set de sensores que van a estar implicados en la consecución de estos objetivos parciales. Una vez elegidos los sensores que van a entrar en juego hay que realizar una serie de mediciones empíricas para determinar los valores de estos sensores que se van a corresponder con cambios de objetivos parciales. Hay que tener en cuenta que los sensores de los que dispone el robot no son sensores de gran precisión, así que antes de tomar las mediciones comentadas anteriormente hay que conocer los márgenes de error de los distintos sensores e incluirlos en el cálculo de las mediciones.

3.1.2 Diseño de la máquina de estados.

Para formalizar el comportamiento de un sistema secuencial se pueden utilizar múltiples representaciones como pueden ser tablas o fórmulas, pero una de las representaciones que resulta más fácil de entender son las máquinas de estados.
Una máquina de estados es un grafo en el que los nodos representan los estados y los arcos representan transiciones entre ellos. En estas transiciones se debe especificar el valor de entrada que ha provocado el cambio de estado y en muchas ocasiones también se incluye el valor de las salidas correspondientes.
La representación mediante máquinas de estados se ajusta perfectamente a la metodología de trabajo que estamos siguiendo ya que los valores de los sensores del robot pueden servir como los datos de entrada del grafo y es dentro de los estados donde se desencadena la acción que deberá realizar el AIBO.

Las máquinas de estados utilizadas en esta metodología son máquinas de estados formados por una tupla ME:(S0, S, E, T, F, A).

- S: conjunto finito de estados
- S0: Estado inicial S0 ε S.
- E conjunto de valores de entrada que se corresponden con los valores de los sensores del robot. Los sensores más utilizados son la cámara y el sensor de proximidad.
- T: función de transición (T: S x E → A x S). Función de transición que decide el estado siguiente al que hay que navegar y la acción a realizar teniendo en cuenta el estado actual y la entrada proporcionada por el robot.
- F: Estado final o estado objetivo. Este estado es el estado que queremos alcanzar si todo ha salido correcto.
- A: Conjunto de acciones que dependen del estado actual y de los valores de entrada. Estas acciones pueden ser atómicas (andar, girar cabeza, mover alguna

articulación…) o más compleja como puede ser algún otro comportamiento definido previamente.

Un ejemplo de una máquina de estados con este formato puede ser el siguiente: “Sigue Pelota”.

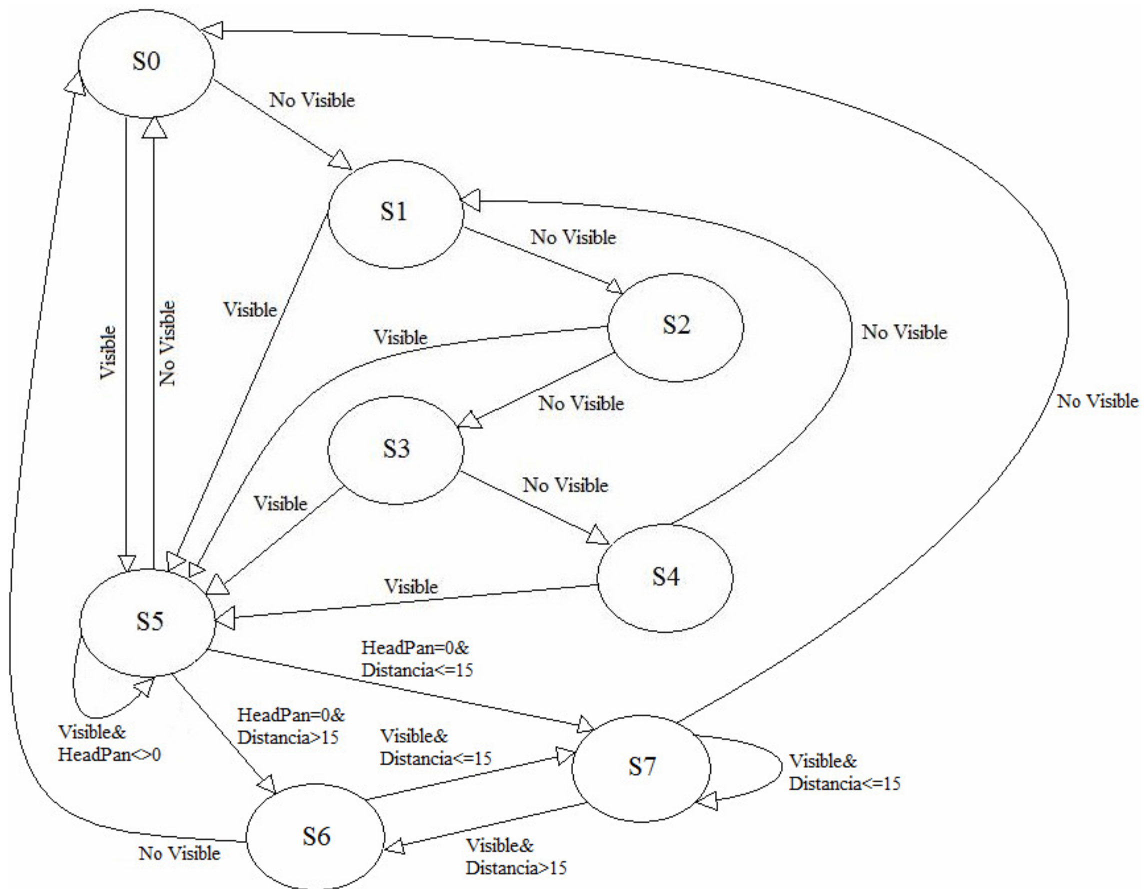

Figura 16. Máquina de estados de “Sigue Pelota”

Este ejemplo cuya máquina de estados se muestra en la figura 16 consiste en un script que dota al robot de un comportamiento mediante el cual, si no ve la pelota, la busca mediante movimientos circulares de la cabeza y en el momento en el que la tenga visible, girará a la vez la cabeza y el cuerpo (en sentidos contrarios) hasta posicionar su cuerpo en línea recta con la pelota. Una vez que tenga el cuerpo orientado hacia la pelota y siga viendo a esta, empezará a caminar hacia la misma si esta se encuentra a una distancia superior a los 15cm (lo cual detecta con el sensor de proximidad situado en el hocico). Si una vez visualizada la pelota la pierde de vista, el AIBO volverá a la posición inicial y volverá a realizar la búsqueda de la misma.

La descripción de los estados viene a continuación:

- S0: Posición inicial del AIBO para buscar la pelota una vez encendido, y a partir de la cuál puede empezar la búsqueda de forma adecuada.

- S1: El AIBO realiza el primer movimiento para buscar la pelota y consiste en un movimiento vertical de la cabeza hasta una posición determinada que

consideramos adecuada para empezar el barrido por la habitación, o hasta que encuentra la pelota con la cámara y pasaría al estado S5.

- S2: Segundo movimiento consistente en un movimiento horizontal de la cabeza hacia la derecha hasta la derecha del todo si no encuentra la pelota en una posición intermedia, con lo que pararía el movimiento horizontal y pasaría al estado S5.

- S3: Tercer movimiento consistente en un movimiento vertical hacia abajo hasta un punto que hemos creído conveniente o hasta que detecta la pelota con la cámara, con lo que pararía el movimiento vertical y pasaría al estado S5.

- S4: Cuarto movimiento consistente en un movimiento horizontal de la cabeza hacia la izquierda hasta la derecha del todo si no encuentra la pelota en una posición intermedia, con lo que pararía el movimiento horizontal y pasaría al estado S5.

- S5: Estado correspondiente a la situación en la cual el AIBO tiene enfocada la pelota en su cámara, y tendrá que girar el cuerpo para orientarse hacia la pelota, y la cabeza para no perder la pelota de vista. En el momento en el que se encuentre de frente a la pelota, empezará a caminar hacia ella.

- S6: Estado en el cuál el robot se encuentra orientado a la pelota y empieza a caminar en línea recta hacia ella. Si en su camino detecta que la pelota se encuentra a menos de 15cm mediante el sensor de proximidad situado junto a la cámara, entonces habrá alcanzado su objetivo y pasará al estado final S7.

- S7: Este es el estado objetivo en el cuál el AIBO tiene la pelota localizada con la cámara y a una distancia menor o igual a 15cm. En este punto, el robot seguirá en esa posición a no ser que la pelota se deje de ver, con lo que volverá al estado S0 (estado inicial) y volverá a realizar todo el proceso de búsqueda y seguimiento de la pelota.

3.1.2.1. Optimización de máquina de estados e integración de nuevos comportamientos:

Como hemos comentado anteriormente la máquina de estados obtenida en un primer momento puede ser optimizada extrayendo nuevos comportamientos independientes de lo que antes eran grupos de subobjetivos de la conducta inicial creando su propia máquina de estados que posteriormente puede ser utilizada para definir otras conductas. Estas optimizaciones favorecen la reutilización de código y ayudan al desarrollador en la creación de nuevos comportamientos.
Para realizar estas optimizaciones tenemos que localizar en el diagrama un subconjunto de estados que cumpla las siguientes condiciones:

El subgrupo de estados sólo puede tener transiciones externas a dos estados que se encuentren fuera del grupo, una transición de entrada al subconjunto desde un estado que podemos llamarlo “estado inicial” y otra de salida a un estado que podemos llamarlo “estado final”.

En el subconjunto tiene que existir al menos un camino entre algunos de los estados que lo forman, entendiendo por camino un conjunto de transiciones que salga de un estado llegando al mismo estado pasando por algún otro del subconjunto.

Siendo la máquina de estados que describe el comportamiento inicial ME:(S0, S, E, T, F, A), la nueva máquina de estados que obtenemos es un tupla ME:(S0’, S’, E’, T’, F’, A’) donde:

- S0’ es un nuevo estado inicial que es necesario incluir en la nueva máquina de estados y es el estado al que llegará la transición de entrada al subconjunto.
- S’ ⊆ S.
- E’ ⊆ E.
- T’ ⊆ T.
- F’: Es un nuevo estado incluido como estado final del nuevo diagrama y es de este estado del que partirá la transición externa de salida del subconjunto.
- A’ ⊆ A.

Utilizando el ejemplo anterior del “Sigue Pelota”, podemos observar en la figura 16 que, como acabamos de explicar, existe un subconjunto de estados relacionado entre sí y en el cuál existe un camino cíclico. Este subconjunto de estados se correspondería con los estados S1, S2, S3 y S4, y que determinarían el comportamiento de buscar la pelota con la cabeza (“Busca Pelota”). Dentro de este subconjunto de estados existe un camino que va desde el estado S1 hasta el S4 pasando por todos los demás, y que vuelve al estado S1. Esto quiere decir que todos los estados están interconectados ya sea directa o indirectamente. Además a este subconjunto de estados se llega únicamente desde el estado S0, y solamente hay un estado externo receptor de transición de estados del subconjunto, el S5. Con esto podemos asociar al estado S0 el valor de estado inicial de un nuevo diagrama de estados, al estado S5 el estado final y a los estados S1, S2, S3 y S4 los estados intermedios. De esta forma obtenemos un nuevo comportamiento que vamos a denominar “Busca Pelota” y cuya máquina de estados se muestra en la figura 17 que mostramos más abajo:

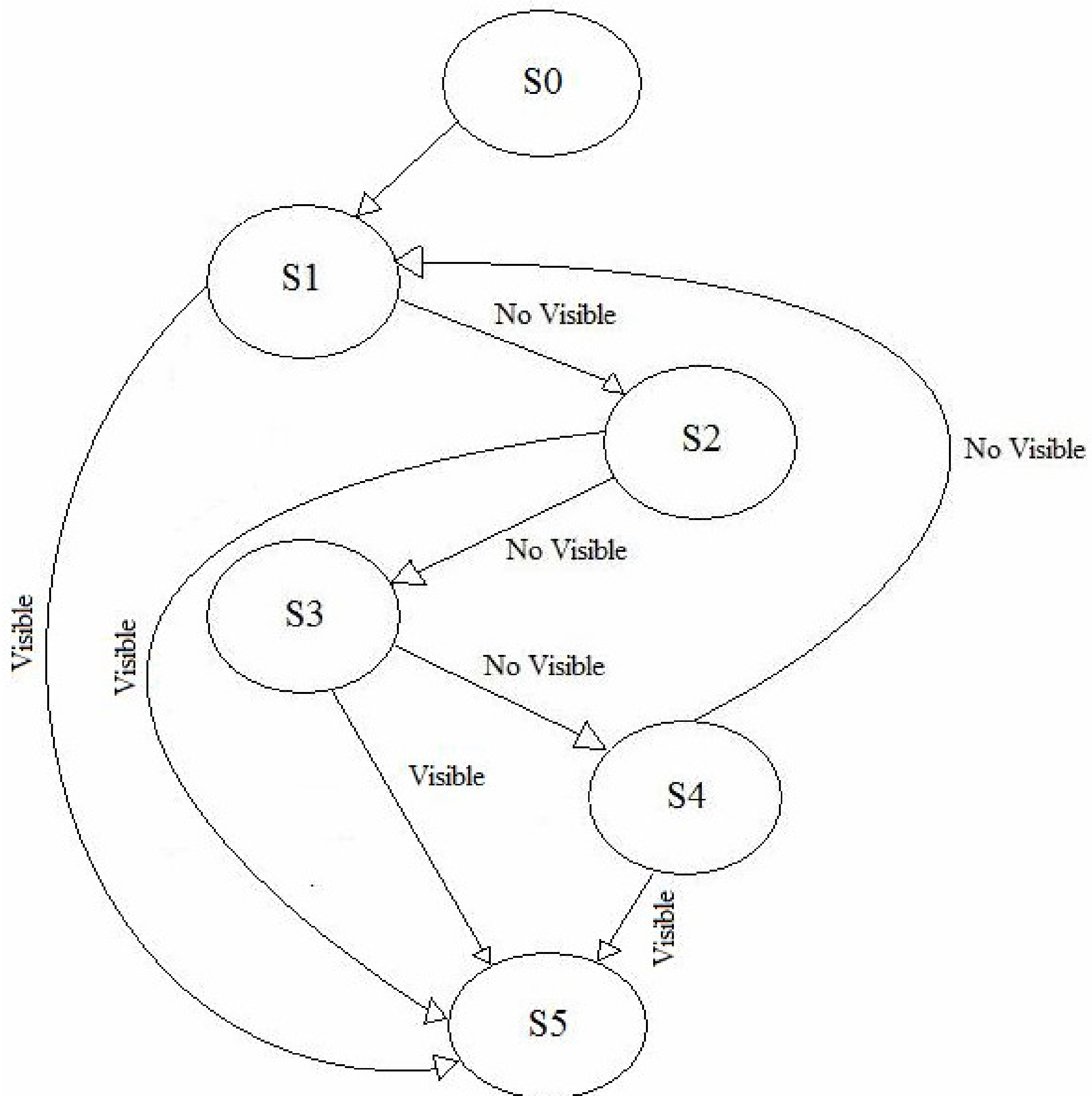

Figura 17. Máquina de estados de "Busca pelota".

En esta nueva máquina de estados podemos observar que los estados y las transiciones siguen siendo las mismas, y que por sí mismo puede aceptarse como un comportamiento independiente. La descripción de los estados es prácticamente idéntica que en el ejemplo anterior:

- S0: Estado inicial para comenzar la búsqueda de la pelota y que transitará automáticamente al estado S1.
- S1: El AIBO realiza el primer movimiento para buscar la pelota y consiste en un movimiento vertical de la cabeza hasta una posición determinada que consideramos adecuada para empezar el barrido por la habitación, o hasta que encuentra la pelota con la cámara y pasaría al estado S5.
- S2: Segundo movimiento consistente en un movimiento horizontal de la cabeza hacia la derecha hasta la derecha del todo si no encuentra la pelota en una posición intermedia, con lo que pararía el movimiento horizontal y pasaría al

estado S5.

- S3: Tercer movimiento consistente en un movimiento vertical hacia abajo hasta un punto que hemos creído conveniente o hasta que detecta la pelota con la cámara, con lo que pararía el movimiento vertical y pasaría al estado S5.
- S4: Cuarto movimiento consistente en un movimiento horizontal de la cabeza hacia la izquierda hasta la derecha del todo si no encuentra la pelota en una posición intermedia, con lo que pararía el movimiento horizontal y pasaría al estado S5.
- S5: Estado final correspondiente a la situación en la cual el AIBO tiene enfocada la pelota en su cámara y habrá alcanzado el estado objetivo del comportamiento.

Ahora que hemos creado un nuevo comportamiento que engloba un subconjunto de estados del ejemplo de “Sigue Pelota”, podemos optimizar dicho ejemplo sustituyendo los estados del subconjunto por un solo estado llamado “Busca Pelota”. Así pues el diagrama quedaría como se puede ver en la figura 18:

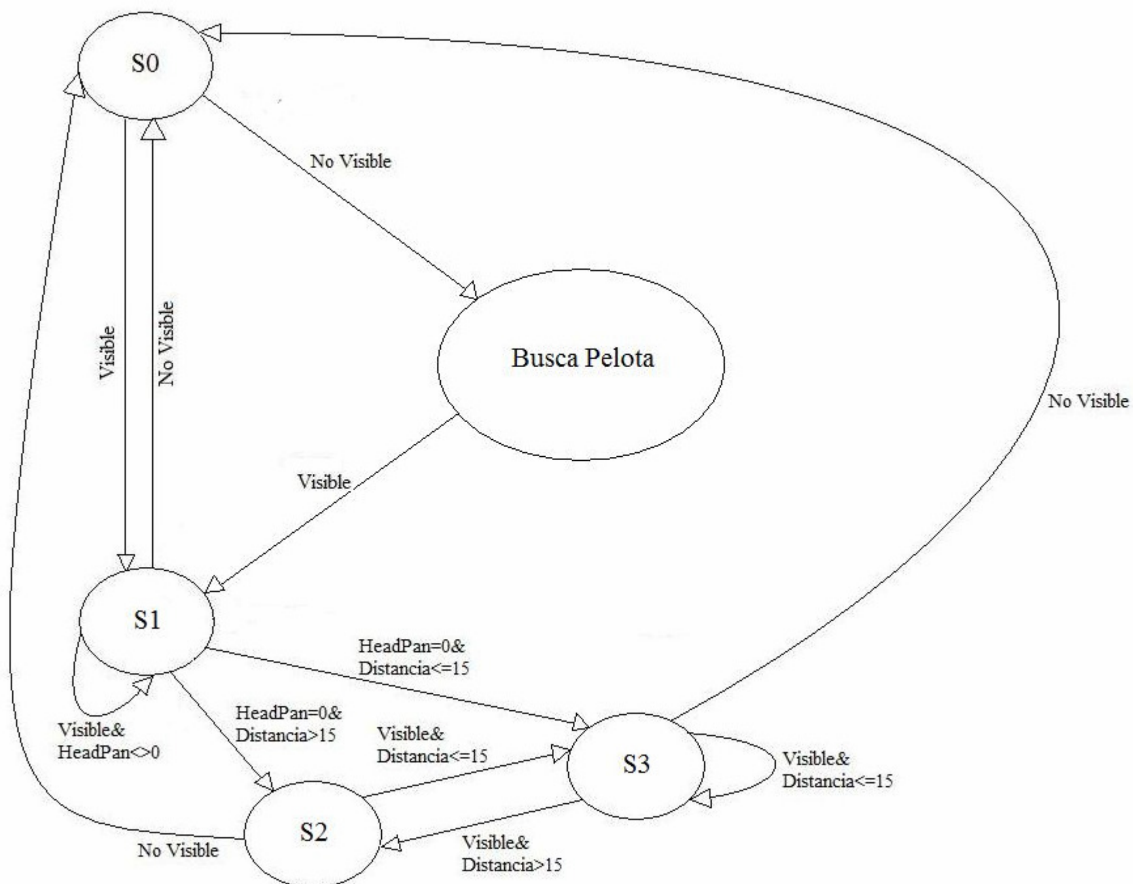

Figura 18. Máquina de estados optimizada.

3.2 Traducción del diseño al lenguaje elegido.

Una vez diseñada la máquina de estados es preciso buscar un modo de implementar las acciones y transiciones de los estados. La mejor manera de programar este tipo de máquina de estados de comportamientos en URBI es usar una conjunción de funciones y comandos "at" y "stop".

Para definir las acciones de cada estado, se usan técnicas de programación básicas teniendo en cuenta las particularidades de URBI. Este documento no incluye ni pretende ser ningún tutorial de URBI, sólo pretende explicar específicamente un modo correcto de programar comportamientos basados en máquinas de estados. Para un tutorial de programación en URBI recomendamos buscar en su página oficial www.gostai.com [5][6].

Una vez implementadas por separado las acciones de cada estado (para comprobar fácilmente su correcto funcionamiento) se define una función por estado que incluya todas las acciones a realizar en éste.

Después se traducen las transiciones, para ellas utilizaremos sentencias "at". Las sentencias "at" se ejecutan continuamente en background, ejecutando los comandos que contienen una sola vez cuando sus condiciones pasen de ser false a true. Las condiciones de estas sentencias obviamente tienen que ser los valores de entrada de las funciones de transición. El contenido de la sentencia "at" tiene que activar el estado objetivo de la transición, además es necesario desactivar el estado origen para evitar dejar cualquier sentencia residuo funcionando en background. Para ello etiquetaremos todas las llamadas a las funciones y detendremos la función del estado origen usando la sentencia "stop".

Veamos la traducción de una máquina de estados sencilla en el siguiente ejemplo cuya máquina de estados podemos apreciar en la figura 19:

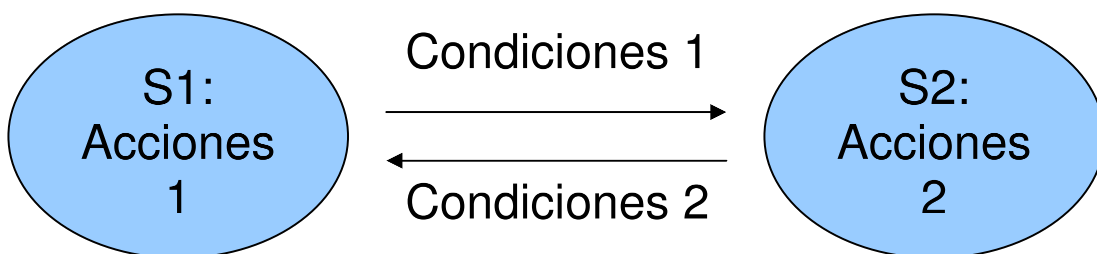

Figura 19. Máquina de estados del ejemplo.

Primero definiríamos las funciones que implementan las acciones a realizar en cada estado:

```
// Estado S1
def estado1() {
Acciones 1
};

// Estado S2
def estado2() {
Acciones 2
```

```
};
```

Luego escribiríamos las dos transiciones para “pegar” los estados entre sí.

```
// Transiciones
at (Condiciones 1) {
stop S1;
S2: estado2();
};

at (Condiciones 2) {
stop S2;
S1: estado1();
};
```

Finalmente sólo quedaría hacer la llamada al estado inicial.

```
S1: estado1();
```

Aunque para este ejemplo no se aprecie la necesidad de escribir código de este modo, en comportamientos complejos de muchos estados cada uno con varias transiciones demuestra ser la manera mejor y más segura de programar. Hace que el código sea modular, sencillo y fácil de mantener.

3.2.1 Comportamientos autónomos o guiados por equipos remotos.

El uso de URBI con un cliente Telnet es bastante limitado, para montar sistemas complicados es necesario poder mandar comandos y recibir mensajes utilizando un lenguaje de programación más potente. Para este propósito se utiliza liburbi. Liburbi es una capa TCP/IP para distintos lenguajes que permite las siguientes operaciones:

- Abrir una conexión hacia el AIBO desde otro lenguaje.
- Mandar comandos URBI desde ese lenguaje.
- Preguntar el valor de un dispositivo y recibirlo
- Escuchar mensajes mandados por el robot y reaccionar a ellos adecuadamente.

Al igual que con URBI no vamos a entrar en detalle sobre el funcionamiento de liburbi, toda la documentación oficial se puede encontrar en www.urbiforge.com [7].

La principal ventaja de usar un equipo remoto con liburbi para programar comportamientos es que el AIBO como computador es bastante limitado, teniendo un procesador lento y poca memoria, el utilizar nuestro equipo nos permite disminuir el tiempo que consumen los algoritmos complicados, utilizar algoritmos que usen más memoria RAM de la que dispone el AIBO o acceder y almacenar grandes cantidades de información.

No obstante para todo esto es necesario la comunicación entre el robot y otro computador, lo cual lleva cierto tiempo y aumenta la complicación del programa. Además para cada distinto tipo de mensaje que se quiera escuchar desde el equipo remoto es necesario abrir un hilo, lo cual puede hacer que se use mucha memoria RAM desde el equipo remoto.

Como regla general aconsejamos usar URBI como comportamiento autónomo siempre

que se pueda y no perjudique en exceso la velocidad de funcionamiento y usar liburbi solamente cuando sea realmente necesario.

3.2.2 Limitaciones del AIBO.

Cuando se trabaja con robots se debe tener en cuenta que las simulaciones o los supuestos teóricos no se trasladan con total precisión al mundo real, además cada mecanismo tiene unas características físicas que limitan sus posibilidades. Es por esto que hay que tener en cuenta las limitaciones de nuestro dispositivo y buscar soluciones cuando no se pueda alcanzar el modelo diseñado debido al choque contra la realidad.

Los principales problemas que hemos encontrado en nuestro proyecto son los siguientes:

- La cámara devolvía distintos valores sobre atributos como la visibilidad de la pelota o el porcentaje de la imagen que ocupaba en función de la iluminación o colores encontrados en el fondo. Para evitar estos problemas intentamos dos cosas: normalizar el entorno del AIBO, usando siempre el mismo entorno e intentando que estuviera bien iluminado y no depender de valores muy precisos de variables proporcionadas por la cámara, usando otros métodos más imaginativos para suplir a aquellas.

- Los movimientos del AIBO están limitados por la movilidad y potencia de sus motores. Intentar forzar un movimiento que exceda las capacidades del robot provoca un auto apagado para proteger los motores de un posible daño. Esto implica que los movimientos han de ser diseñados concienzudamente, teniendo en cuenta que las órdenes se pueden dar en distintas posiciones y que no se pueden realizar todas las secuencias de movimientos desde todas las posiciones. Para aliviar este problema diseñamos una serie de secuencias de movimientos que pusieran al AIBO en una posición inicial desde distintas posturas. Esta posición permitía al AIBO realizar sus movimientos con normalidad.

- Los movimientos del AIBO producen posturas distintas y recorren una cantidad distinta de distancia en función de variables que no se pueden controlar, como la adherencia del suelo o la presencia de objetos que obstaculicen la trayectoria. Para intentar resolver este problema se intenta trabajar siempre en el mismo entorno y se trabaja en que los comportamientos se adapten lo mejor posible a situaciones no ideales.

- Los sensores no siempre devuelven valores correctos. Por ejemplo, los sensores de distancia pueden devolver valores incorrectos a distancias demasiado cortas o lejanas, para esto usamos más de un sensor para la misma medida mejorando la precisión.

- El AIBO no es capaz de seguir movimientos de pelota muy rápidos. Se diseñan comportamientos capaces de volver a buscar la pelota si se pierde de vista.

- El AIBO cuenta con una cantidad de batería limitada, esto limita el tiempo de

trabajo y además hace que el robot se comporte peor cuando los niveles de batería son bajos.

Resumiendo, los resultados pueden no ser los esperados por razones ajenas al buen diseño o implementación del sistema en un contexto teórico o simulado. La solución a este problema es ser imaginativo a la hora de pensar en soluciones que circunvalen las limitaciones que nos impone el mundo real.

3.2.3 Sincronización.

Cuando se trabaja con procesamiento en paralelo surge el problema de la sincronización de las distintas partes del comportamiento. Hay que tener en cuenta además que la depuración en URBI es complicada y que no existe un programa que nos ayude en este aspecto. En URBI existen distintas maneras de encadenar comandos, mediante los operadores | & ; , se pueden conseguir distintos modos de procesamiento en paralelo/serie interesantes.

Además se tienen los métodos captadores de eventos que funcionan continuamente en background y que pueden interferir en el desarrollo de otras acciones causando colisiones de acciones inesperadas.

Por último hay que citar que cuando varios procesos intentan cambiar el valor de una misma variable en el mismo instante ocurre un conflicto, esto se resuelve mediante los modelos de mezclas.

Si se utiliza la metodología citada anteriormente estos problemas se minimizan, al menos entre distintos estados, aunque para el diseño de las acciones de un estado que sean realmente complejas se puedan dar. La experiencia y la documentación en la programación son posiblemente el mejor antídoto en este caso.

Es necesario decir que hemos detectado un problema en la instrucción “at” que es una piedra angular cuando se diseñan comportamientos con la metodología GMG. Si las condiciones pasan muchas veces de true a false y viceversa rápidamente o si cambia un instante y vuelve al estado anterior es posible que la sentencia sea llamada múltiples veces antes de resolverse o que se quede a mitad de ejecución y se bloquee, esto produce multitud de problemas, como por ejemplo estar en un estado múltiples veces, no cambiar de estado, etc. En nuestro caso usamos sentencias “at” adicionales y semáforos para asegurarnos de que la transición se ejecutase una sola vez y tratar de que se bloquease el menor número de veces posible. La solución mejoro significativamente el funcionamiento del programa.

También se da sincronización entre equipo remoto y AIBO cuando se utiliza liburbi, esto se hace mediante objetos UMessage y UCallback. Usando métodos oyentes que abren nuevos hilos para la recepción de los mensajes enviados por el AIBO, en el sentido contrario la comunicación se realiza mediante el envío de comandos.

3.3 Observación de errores y fallos y vuelta a empezar

Como en toda labor programativa es necesaria una etapa de debugeo o depuración en la

que se corrijan los fallos no previstos o aquellas partes que no se implementaron correctamente. En este sentido trabajar directamente con scripts URBI es difícil, puesto que no hay un compilador que tenga una aplicación diseñada con este propósito. Esta es una de las razones por las que la modularidad del sistema es importante y por lo que se elige el diseño de comportamientos como máquinas de estados. Esto nos permite comprobar las acciones de los estados en sí y simular transiciones de un estado a otro por separado. Cuando se trabaja con un lenguaje como java o c++ se puede depurar el programa normalmente, siempre teniendo en cuenta que al robot se le siguen enviando scripts URBI y que el robot funciona con multitud de rutinas en background que es necesario gestionar apropiadamente.

Independientemente de estos fallos, el sistema puede no funcionar por problemas de diseño o simplemente podemos querer mejorar ciertos aspectos o añadir características nuevas, teniendo esto en cuenta proponemos una etapa de evaluación en nuestra metodología en la que se analicen los resultados y se propongan los puntos a tratar en la siguiente iteración, donde se puede querer modificar la planificación, el diseño y/o la traducción.

De este modo el gráfico que representa el modelo de nuestra metodología, sería el mostrado en la figura 20:

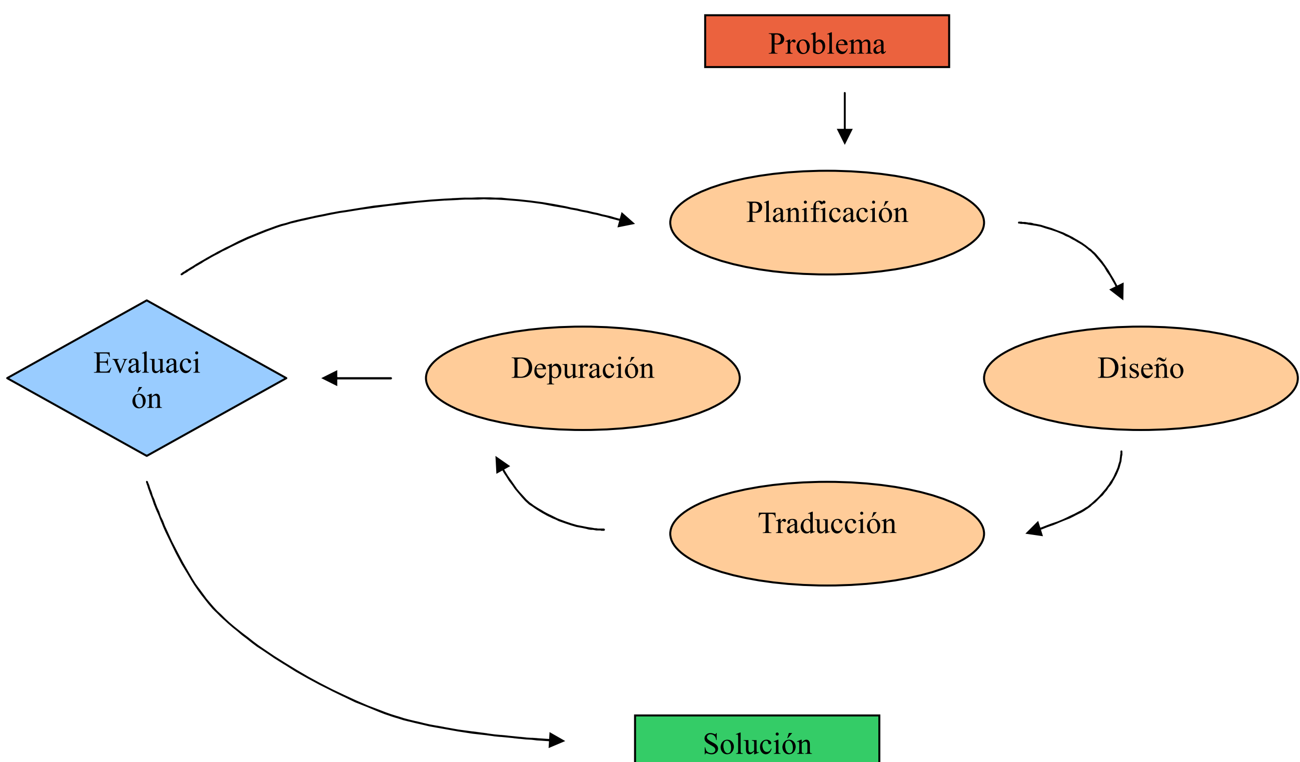

Figura 20. Metodología GMG.

3.4 Ejemplo. Persecución de una pelota.

Con el objetivo de simplificar el ejemplo hemos decidido explicar en este apartado la versión del comportamiento que persigue la pelota únicamente cuando ésta es visible.

En este ejemplo se verán las distintas etapas que se siguen a la hora de encarar el problema.

Planificación: La primera tarea es establecer cuál es el objetivo inicial, en este caso perseguir una pelota, como el problema está simplificado decidimos establecer que la perseguirá únicamente mientras sea visible y que no la buscará cuando lo sea. Lo siguiente será plantearnos objetivos intermedios para distintas situaciones que nos lleven a este objetivo. Nos planteamos las siguientes situaciones, que posteriormente se convertirán en estados:

- La pelota está aproximadamente en frente de nuestro robot; el robot avanzara en línea recta hacia ella.
- La pelota está a la derecha del robot; el robot girará sobre sí mismo a la derecha hasta que esté aproximadamente en frente de ella.
- La pelota está a la izquierda del robot; el robot girará sobre sí mismo a la izquierda hasta que esté aproximadamente en frente de ella.
- La pelota no es visible; el robot ladrará y esperará.

Diseño: A partir de estas premisas ideamos el diagrama de estados que tendrá la forma que podemos ver en la figura 21 mostrada más abajo:

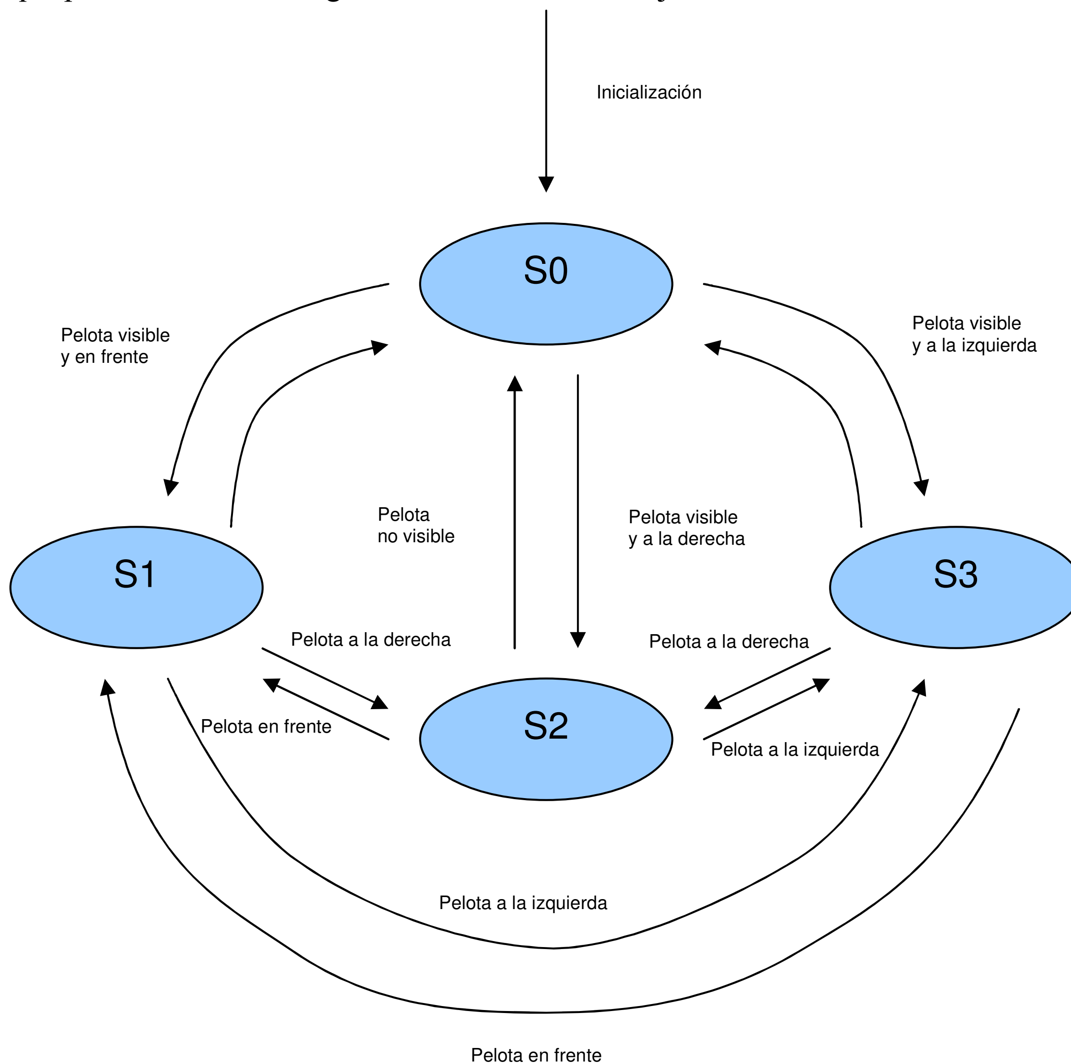

Figura 21. Máquina de estados de "Persecución de una pelota".

La descripción de los estados sería la siguiente:

- S0: La pelota no es visible, el robot ladra y espera.
- S1: La pelota es visible y está en frente del robot, el robot avanza.
- S2: La pelota es visible y está a la derecha del robot, el robot gira a la derecha.
- S3: La pelota es visible y está a la izquierda del robot, el robot gira a la izquierda.

Traducción: De la máquina de estados anterior y siguiendo las directrices explicadas en el punto 3.2 se pasa al siguiente código que ahora explicamos.

Primero definimos unas funciones auxiliares que nos permitirán inicializar al AIBO, también utilizamos una función previamente escrita por nosotros que nos da un

movimiento hacia delante mejor que la proporcionada por defecto por URBI (el robot avanza más deprisa).

```
//Función que hace que el robot siga la pelota con la cabeza
//siempre que esta sea visible
function mira() {

 whenever (ball.visible)

{

     ledF14=1;

     headPan  = headPan  + ball.a * camera.xfov * ball.x &

     headTilt = headTilt + ball.a * camera.yfov * ball.y;};

};

//Función que inicializa el robot preparándolo.
function inicial(){
motors on;
robot.initial();

legLH1.val = -30 & legRH1.val = -30;

legLH2.val = 15 & legRH2.val = 15;

legLH3.val = 90 & legRH3.val = 90;

legLF1.val = 0 & legRF1.val = 0;

legLF2.val = 7 & legRF2.val = 7;

legLF3.val = 100 & legRF3.val = 100;

neck.val = -27;

headTilt.val = 0;

headPan.val = 0;
mira();

};

//Función que permite al robot andar más rápido que la función de URBI
function andar(){

{legLH1.val = -15 & legLH3.val = 75 & legLF1 = 12 & legLF3 = 100 &

legRF1 = -12 & legRF3 = 90 & legRH1 = -45 & legRH3 = 90 &

legLF2.val = 7 & legRF2.val = 7 & legLH2.val = 15 & legRH2.val =
15}|wait(0.2s)|

{legRH1.val = -15 & legRH3.val = 75 & legRF1 = 12 & legRF3 = 100 &

legLF1 = -12 & legLF3 = 90 & legLH1 = -45 & legLH3 = 90 &

legLF2.val = 7 & legRF2.val = 7 & legLH2.val = 15 & legRH2.val =
15}|wait(0.2s)|ledHC.val = !ledHC.val;

};
```

Después pasamos a traducir la máquina de estados propiamente dicha, lo primero que implementamos son los estados y sus acciones.

```
// código para las acciones de S3

mov.gira1: izquierda(){

S3.bucle: robot.turn(-inf),

};

// código para las acciones de S2

functio derecha(){

S2.bucle: robot.turn(inf),

};

// código para las acciones de S1

function andando(){

 S1.bucle: loop {andar()},

};

// código para las acciones de S0

function novisible() {

ledF14.val = 0;

speaker.play("bark2.wav");

};
```

Ahora traduciremos las transiciones, como todas las transiciones hacia un mismo estado son iguales las agrupamos para evitar tener que escribir múltiples veces el mismo código.

```
// código para las transiciones a S3
at(headPan> 15 && ball.visible == true){
{stop S0 & stop S1 & stop S2};
S3: izquierda();
};

// código para las transiciones a S2
at(headPan< -15 && ball.visible == true){
{stop S0 & stop S1 & stop S3};
S2: derecha();
};

// código para las transiciones a S1
at(headPan < 15 && headPan > -15 && ball.visible == true){
{stop S0 & stop S2 & stop S3};
S1: andando();
};

// código para las transiciones a S0
at(!ball.visible ~100ms){
{stop S1 & stop S2 & stop S3};
S0: novisible();
};
```

Por último hacemos la llamada a la inicialización y vamos al estado inicial.

```
inicial();

S0: novisible();
```

Depuración: En esta etapa se harían distintas pruebas intentando averiguar que fallos se están dando y como arreglarlos. El código escrito arriba ya está depurado y consecuentemente ha superado esta etapa.

Evaluación: Por último se hace una valoración del grado de satisfacción en la consecución de los objetivos y de posibles fallos o mejoras que hay que tratar en la siguiente iteración. En nuestro caso se podría por ejemplo, ampliar la máquina de estados para que el robot buscase la pelota cuando la pierde o para que al acercarse a la pelota el AIBO la intentará controlar, utilizar semáforos para mejorar el comportamiento de las transiciones debido a los problemas de la sentencia at, escribir nuevos scripts que permitiesen un giro del robot sobre sí mismo más rápido, etc.

4. Reutilización de comportamientos con razonamiento basado en casos

4.1 Razonamiento Basado en Casos

4.1.1 ¿Qué es CBR?

Un sistema CBR (Case-Based Reasoning) o sistema de razonamiento basado en casos es un modelo de razonamiento que permite desarrollar sistemas de Inteligencia Artificial basados en la utilización de la experiencia previa. Básicamente permite construir razonadores que adapten soluciones ya utilizadas anteriormente y que hayan tenido buenos resultados, a nuevos problemas propuestos utilizando algún tipo de medida de similitud entre estos problemas [8].

Este paradigma apareció como una alternativa a los sistemas expertos basados en reglas ya que es un acercamiento mucho más natural a la manera de razonar que tenemos lo seres humanos, que utilizamos la experiencia previa que hemos adquirido en un dominio concreto en la resolución de nuevos problemas que surgen, adaptando estas soluciones e incorporándolas a lo que podemos denominar nuestra "base de conocimiento".

Algunos sistemas CBR desarrollados con éxito han sido:

- CYRUS: Considerado como el primer sistema CBR tal y como lo conocemos hoy en día. Desarrollado por Janet Kolodner, fue desarrollado para dar respuesta a consultas relativas a viajes y citas para una secretaría concreta de la administración pública estadounidense. [9]
- CASEY: Desarrollado para diagnosticar enfermedades cardiacas. [10]
- JULIA: Sistema desarrollado para diseñar menús en restaurantes. [11]
- PERSUADER. [12]
- MEDIATOR. [13]
- CHEF. [14]

En la actualidad estos sistemas CBR son muy utilizados en servicios de soporte técnico (Help-Desk) y comercio electrónico.

<u>Ventajas de los sistemas CBR frente a otros sistemas expertos.</u>

A diferencia de otros sistemas expertos que tradicionalmente han sido propuestos por la Inteligencia Artificial, como por ejemplo los sistemas basados en reglas, el razonamiento basado en casos no depende exclusivamente del conocimiento general del

dominio del problema, sino que utiliza el conocimiento específico de casos pasados para reutilizarlo como solución de nuevos problemas. Aquí radica precisamente la principal ventaja de los sistemas CBR frente a otros sistemas expertos, se elimina el cuello de botella que supone la adquisición del conocimiento que es necesario en otros sistemas expertos.

Según David B. Leake [15], existen 5 ventajas de los sistemas CBR frente a los sistemas expertos tradicionales:

- **Adquisición de conocimiento.** El mayor problema que tienen los sistemas expertos tradicionales es el proceso de adquisición de conocimiento ya que no es tarea trivial extraer reglas válidas a partir del conocimiento del dominio que tiene un experto en la materia. Es de vital importancia invertir una gran cantidad de tiempo y esfuerzo en el proceso de adquisición de conocimiento, ya que la rigurosidad con la que se aborde esta fase del diseño repercutirá directamente en la calidad del sistema final. Existen multitud de técnicas que permiten la extracción del conocimiento, pero ninguna de ellas se puede automatizar por completo, por lo que los desarrolladores tienen que estar muy implicados en esta etapa para poder continuar con las fases posteriores habiendo sentado unas bases firmes sobre las que poder seguir trabajando.

  En los sistemas CBR se elimina el cuello de botella que supone este proceso ya que utiliza experiencias previas para proponer soluciones a los distintos casos que se puedan presentar. Aunque en estos sistemas el esfuerzo en la adquisición de conocimiento sea mínima, hay que trabajar en lo que se denomina ingeniería de casos; La ingeniería de casos es la encargada de determinar la información que debe incluir cada caso, elegir una representación adecuada para cada uno de ellos, extraer esta información a partir de los datos disponibles y de intentar que la base de casos sea lo más completa y estable posible.

- **Mantenimiento del conocimiento**. En los sistemas expertos tradicionales hay que realizar un trabajo de mantenimiento a lo largo de todo el periodo de vida útil del sistema para intentar que la base de conocimiento no se quede obsoleta. Incluso el proceso de introducir nuevo conocimiento en la base puede implicar la redefinición de todo lo que se tenía anteriormente, convirtiéndolo en una tarea compleja y que requiere gran cantidad de trabajo y esfuerzo. Por el contrario, en los sistemas de razonamiento basados en casos, dada la naturaleza de aprendizaje incremental con el que se diseña la base de casos, esta puede ser limitada al principio y después se pueden añadir nuevos casos cuando sean necesarios de manera sencilla sin la necesidad de intervención de un experto en el dominio.

- **Eficiencia en la resolución de problemas.** La reutilización de casos previos permite que se puedan resolver problemas similares sin rehacer el proceso de razonamiento.

- **Calidad de la solución.** Como se ha señalado anteriormente, el trabajo de adquisición de conocimiento y su posterior transformación a algún tipo de reglas es un trabajo en el que se tiene que tener sumo cuidado para no obtener reglas imperfectas que nos den lugar a soluciones erróneas.

En sistemas en los que no se ha entendido correctamente el dominio de la aplicación, las aplicaciones CBR son mucho más eficientes ya que encontramos soluciones buscando en casos similares que hayan sido resueltos satisfactoriamente con anterioridad.

- **Aceptación del usuario.** En muchos sistemas expertos el proceso que se lleva a cabo desde que se propone un nuevo problema hasta que este proporciona una solución válida, puede ser incomprensible para el usuario de la aplicación, que no podrá comprobar la validez de la misma ya que no llega a entender por completo como se ha llegado hasta ella. En los sistemas CBR las soluciones están basadas en experiencias previas que pueden ser presentadas a los usuarios para apoyar las conclusiones a las que llega el sistema.

Elementos de un sistema CBR. [16]

A grandes rasgos, los elementos que forman parte de un sistema CBR son básicamente 3, los casos, la librería de casos y un mecanismo basado en alguna o algunas medidas de similitud para la recuperación de casos similares.

- **Los casos**.

  Los casos son los elementos principales de todo sistema CBR. “Un caso es un fragmento contextualizado de conocimiento que representa una experiencia y que enseña una lección importante para conseguir los objetivos del razonador.” [17].

  Al ser los casos un elemento tan importante dentro de los sistemas CBR, uno de los primeros pasos que hay que dar a la hora de empezar a diseñar una aplicación de este tipo es definir claramente una representación y un método de almacenamiento para los casos.

  Los casos tienen que estar representados como mínimo por la descripción del problema que se quiere resolver y por la solución de dicho problema. La descripción del problema puede ser un conjunto de atributos que caracterizan la situación dentro del dominio donde se presenta, es importante una buena elección de estos atributos para facilitar y hacer más eficiente y completo el posterior proceso de recuperación, por lo que el desarrollador tiene que tener un mínimo de conocimiento sobre el dominio sobre el que está trabajando.

  La solución del problema puede ser de muy distinta naturaleza, dependiendo del ámbito en el que estemos trabajando, se puede proporcionar como solución un fragmento de código, una serie de acciones, un conjunto de filas de una base de datos, etc. Teniendo en cuenta lo anterior podemos decir que un caso se comporta como una correspondencia entre elementos del espacio de posibles problemas con elementos del espacio de posibles soluciones. Un pequeño resumen esquemático de las relaciones entre el espacio de problemas y el de soluciones lo encontramos en la figura 22.

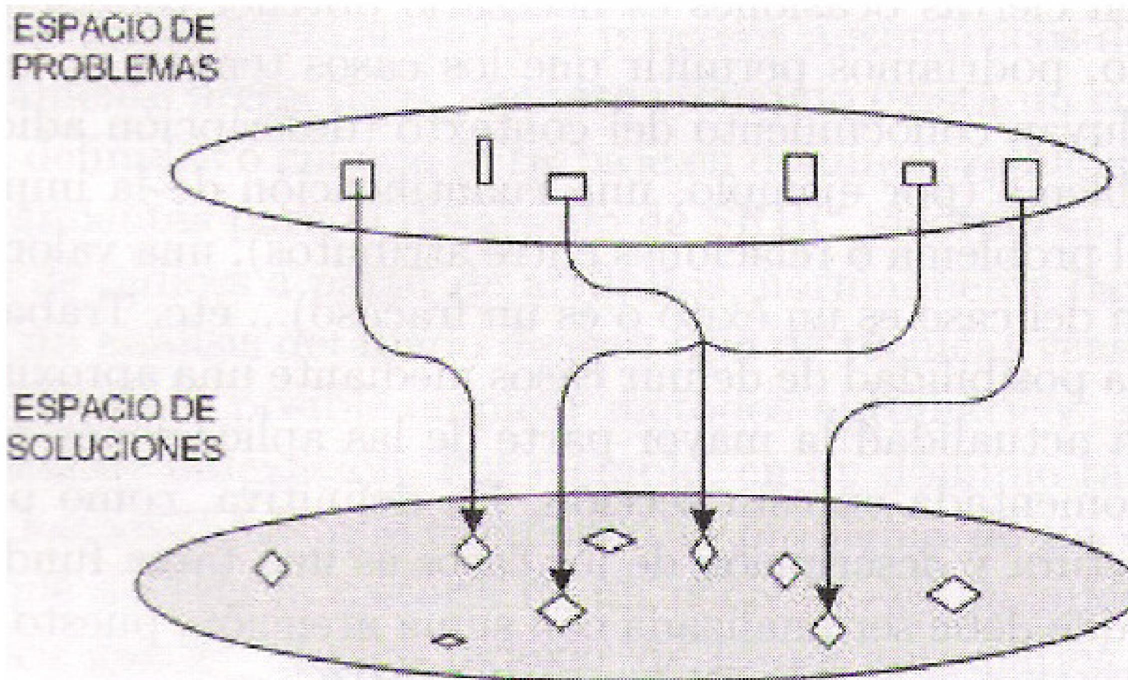

Figura 22. Espacio de problemas y soluciones.

- **La librería de casos**.

Dado que los casos es el elemento principal de todo sistema CBR, la manera de almacenarlos repercutirá directamente en el rendimiento del mismo. La librería de casos es el módulo encargado de almacenar y organizar todos los casos disponibles y su estructura es crucial para la fase de recuperación de casos similares.

En la creación de la estructura que formará la librería de casos hay que tener en cuenta aspectos fundamentales como puede ser el tamaño que queremos que tenga la base de casos, los requerimientos que nos impone el dominio específico en el que estamos trabajando y la inserción eficiente de nuevos casos.

Tradicionalmente se han propuesto dos estructuras principales para la el almacenamiento de casos:

- Memoria plana: En esta estructura se presentan los casos completos de forma secuencial utilizando algún tipo de indización ya sea manual o automática. Esta estructura tiene el principal inconveniente de que la búsqueda de casos es menos eficiente, por lo que no es recomendable utilizarla en sistemas que trabajan con una base de casos de gran tamaño y necesitan respuesta en tiempo real. Por otra parte la inserción de nuevos casos en este tipo de estructuras es muy sencilla ya que basta con incluir un nuevo registro con el nuevo caso respetando el método de indexación que se esté utilizando.

- Memoria jerárquica: En una estructura con memoria jerárquica se utilizan representaciones en forma de árbol similares a los árboles de

búsqueda ID3, en los que cada nodo interior representa un atributo del caso y en las hojas se almacenan las soluciones a los mismos. Cada recorrido desde la raíz hasta una de las hojas del árbol representa un caso completo.

Al igual que en los árboles ID3 se tiene que intentar que los atributos más discriminantes estén almacenados en los niveles superiores del árbol empezando desde la raíz. La gran ventaja de este tipo de almacenamiento es la eficiencia en la búsqueda pero a cambio se sacrifica la sencillez de inserción de nuevos casos.

En la siguiente figura 23 disponemos de un esquema en el que podemos observar las diferencias entre la estructura de una memoria plana y una memoria jerárquica.

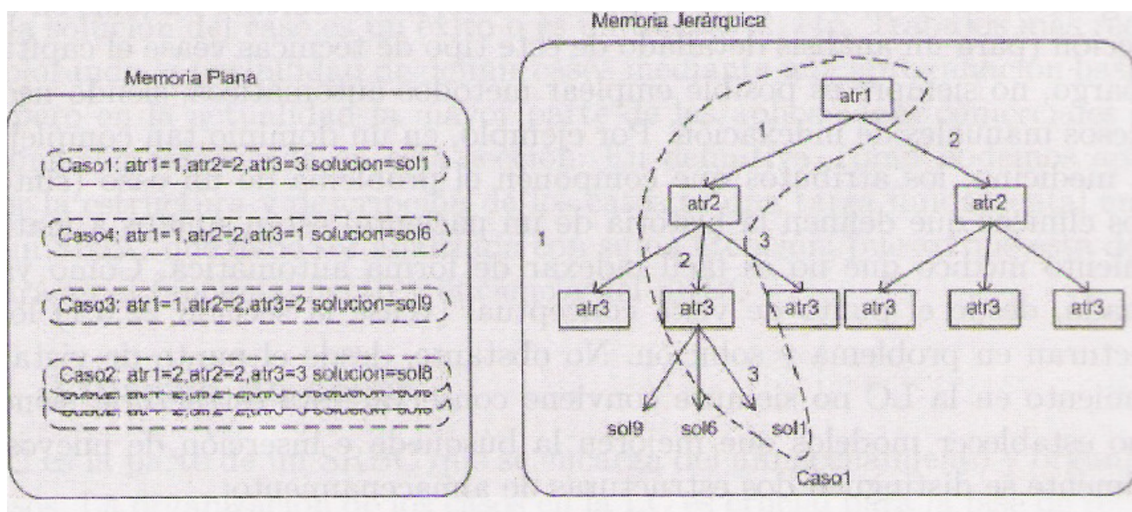

Figura 23. Memoria plana y memoria jerárquica.

Los desarrolladores de aplicaciones CBR tienen que llegar a un compromiso entre eficiencia de recuperación y eficiencia de inserción para acertar en la elección de una de las estructuras anteriores para crear la librería de casos. De la correcta elección de esta estructura dependerá en gran medida la eficiencia global de todo el sistema experto.

Mecanismos para determinar casos similares.

Cuando al sistema CBR se le presenta un problema para resolver es ingenuo pensar que va a encontrar en la librería de casos una situación con exactamente los mismos valores en todos sus atributos, por lo que es importante definir un mecanismo de similitud para poder comparar casos.

Este último elemento de los sistemas CBR es el encargado de encontrar en la librería de casos las situaciones más parecidas a la que se nos presenta como problema, y de esta manera proporcionar una solución que se adapte lo mejor posible a la situación actual.

La aproximación más intuitiva para comparar casos es comprobar la similitud de los distintos atributos que los conforman, pero hay que tener en cuenta que no todos los

atributos que definen un caso son del mismo tipo, nos podemos encontrar atributos representados por enteros, cadenas de caracteres o enumerados por ejemplo. Por esta razón es muy útil definir dos medidas de similitud distintas:

- Medidas de similitud local: Las utilizamos para cuantificar la similitud entre atributos del mismo tipo.

- Medidas de similitud global: Las medidas de similitud global recogen los datos obtenidos en las medidas de similitud local, asignando pesos para distinguir la importancia de los distintos atributos, y determinan la similitud entre casos completos. Casi todas las técnicas de similitud global utilizan alguna función de distancia entre los valores de los atributos ya que existe una relación inversamente proporcional entre el concepto de distancia y el de similitud. La función de distancia más utilizada y una de las más sencillas de aplicar es la distancia euclídea que mide la distancia entre dos puntos en el espacio eluclídeo.

  La distancia euclídea ponderada se define como:

$$D_{Euc}(C_i, C_j) = \frac{\sqrt{\sum_k (w_k d_k(at_k^i, at_k^j))^2}}{\sum_k w_k}$$

  donde:

  - w_k es el peso del atributo k
  - at_k^i es el valor del atributo k del caso i
  - $d(x,y)$ es la distancia que separa x de y

Existen otras funciones de similitud que miden la distancia entre atributos numéricos como pueden ser la distancia de Manhattan y la distancia de Minkowski, y para medir distancias de atributos de distinto tipo como pueden ser la distancia de Hamming o las distancias semánticas.

El ciclo CBR

Una vez definidos los elementos principales que forman parte de todo sistema CBR, es necesario especificar un modelo de trabajo en el que estos elementos se relacionen entre sí para poder ponerlo en marcha. Este modelo de trabajo es el denominado ciclo CBR. El ciclo CBR más conocido consta de 4 etapas principales [18]:

- Etapa de recuperación: en la que se utiliza algún método de similitud para determinar casos similares a la descripción del problema.

- Etapa de reutilización o adaptación: Se utiliza la solución sugerida por alguno de los casos similares que obtenemos de la etapa anterior, adaptándola para resolver el problema actual.

- Etapa de revisión: en esta etapa se comprueba que la solución proporcionada por

el sistema es correcta.

- Etapa de retención o almacenamiento: En la que se realiza el proceso de “aprendizaje” incorporando la descripción del problema y la solución adaptada a la librería de caos del sistema.

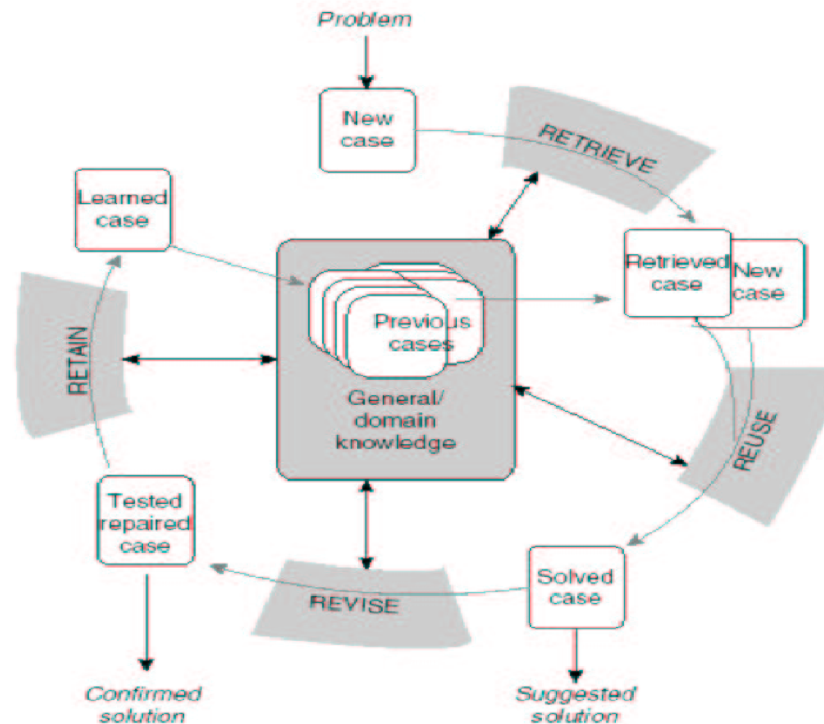

Figura 24. Ciclo CBR.

Este ciclo descrito y mostrado en la figura 24 es susceptible de sufrir modificaciones para adaptarlo al dominio concreto en el que se está trabajando y a los requisitos que debe cumplir el sistema, de esta manera se pueden añadir más etapas como puede ser una etapa de mantenimiento, o bien eliminar algunas de ellas hasta ajustar el funcionamiento del sistema a las exigencias de los desarrolladores.

4.1.2 ¿Por qué lo proponemos a nuestra metodología?

Como se ha expuesto anteriormente, los sistemas expertos CBR son muy útiles a la hora de desarrollar aplicaciones en las que la experiencia adquirida en el dominio sobre el que se trabaja es mucho más útil que el conocimiento que un experto nos pueda proporcionar. El dominio de la RoboCup, que ha sido el que finalmente se ha elegido para el desarrollo de este proyecto, se adapta perfectamente a esta situación ya que es mucho más fácil definir el comportamiento de los distintos jugadores basándose en casos similares que han obtenido resultados satisfactorios, que crear reglas para cada una de las situaciones que se puedan presentar dentro del campo de juego, ya que el abanico de posibles situaciones es muy amplio.

Por otro lado, el modelo de percepción/actuación con el que trabaja el robot AIBO se traslada fácilmente a la forma en la que se representan los casos dentro de un sistema CBR. Los valores de los sensores del robot pueden utilizarse para representar los distintos atributos que definen un caso y las actuaciones pueden proporcionarse en forma de scripts almacenados como la solución del caso. Debido a esta sencillez de adaptación de la manera de trabajar que tiene el robot con las exigencias que imponen los sistemas CBR, se elimina el cuello de botella de la ingeniería de casos que es una de las etapas más complicadas a la hora de crear una aplicación de este tipo.

Una vez expuestas las razones principales por las que un sistema de razonamiento basado en casos se acomoda perfectamente a la metodología de trabajo que hemos seguido a lo largo del proyecto, cabe decir que también presenta un gran inconveniente: la tarea de recolectar los casos. Si no se dispone de una librería de casos inicial que esté probada y sea estable, es tarea de los desarrolladores formalizarla, por lo que se tienen que estudiar un gran número de situaciones para que la base de casos inicial sea lo suficientemente extensa como para que la recuperación de soluciones tenga éxito en un porcentaje aceptable.

4.2 Metodología GMG con reutilización de comportamientos.

Para crear un proyecto de un modo ordenado y eficiente, es necesario disponer de una metodología que fije unas pautas a seguir que nos sirvan de guía y nos faciliten la labor del diseño global de la aplicación. La metodología que proponemos para desarrollar sistemas CBR consta de la siguiente serie de pasos:

Paso 0: Identificación y análisis del problema. ¿Es el CBR una buena solución?

En nuestro caso el problema consistía en conseguir que nuestro robot fuera capaz de comportarse de manera similar a un portero en el dominio del robo-fútbol, impidiendo que la pelota se introdujese en la portería. Esto implica una gran cantidad de posibles situaciones que requieren de distintas soluciones, además estas situaciones no tienen a priori una solución definida (no existe un algoritmo de portero perfecto).

El CBR nos proporciona la capacidad de adaptarnos a una situación nueva a partir de casos particulares ya experimentados que, en principio, dan una solución apropiada y que son sencillos de obtener. Además es capaz de, por medio de sensores, adaptarse a situaciones cambiantes como lo es un partido de robo-fútbol. Por lo tanto era adecuado usar un CBR.

Paso1: Análisis de sensores y actuadores.

Para poder diseñar un sistema de estar características es necesario conocer qué tipo de información es capaz de reconocer nuestro robot (sensores) y qué tipo de acciones puede ofrecer como respuesta (actuadores). En nuestro caso y como se explicó en el punto 2 teníamos conocimiento sobre las capacidades de nuestro AIBO ERS-7.

Paso 2: Dominio de los comportamientos. Diseño y búsqueda. Almacenaje y recuperación de aquellos.

Como hemos comentado nuestro robot tiene que ofrecer soluciones en función del ámbito en que se encuentre. Estos comportamientos deberían ser sencillos, de modo que al combinarse desarrollasen comportamientos más complejos.

En nuestro caso decidimos que nuestro portero debía ser capaz de realizar movimientos laterales básicos, tumbarse para cubrir la portería, despejar y buscar la pelota.

Una vez decidido que acciones se quieren realizar es necesario diseñarlas (en caso de que no estén disponibles), para esto ya definimos anteriormente la metodología GMG de diseño de comportamientos.

También es necesario determinar la forma en que se van a almacenar los comportamientos que se utilicen y de el modo de recuperación y uso. Nosotros decidimos definir los comportamientos como funciones cargados en el robot al inicializar la aplicación y utilizarlos mediante llamadas a dichas funciones.

Paso 3: Definición de los casos. ¿Qué atributos distinguen un caso?

Es importante tener bien definida la estructura de los casos. Los atributos que definen cada caso deben ser relevantes para distinguir una situación de otra, no es correcto almacenar atributos redundantes o que no aporten información. El conjunto de estos atributos tiene que definir además un caso unívocamente.

Los atributos que escogimos nosotros son: Si la pelota está siendo vista por el robot, la posición del robot respecto a la portería, la postura del robot (tumbado o de pié), la posición de la pelota con respecto al AIBO en el caso de que sea visible (determinada por la posición de la cabeza del robot que la sigue). Además a cada caso se le liga una solución que viene a ser una secuencia de llamadas a las funciones que implementan los comportamientos.

Paso 4: Almacenaje de los casos.

Hay que decidir un modo de almacenar la base de casos. En nuestro caso decidimos crear una base de datos MySql. Es adecuada debido a que los atributos de los casos eran o bien numéricos o bien booleanos y MySql nos permitía trabajar con ellos con rapidez y eficiencia. Además podíamos modificar, eliminar o crear casos precisos con sencillez y sin preocuparnos de la estructura de datos.

Paso 5: Base de casos inicial.

En todo sistema CBR es necesario tener una base de casos inicial. Esta base de casos debería ser lo más precisa y completa posible porque de ella depende en gran medida que las decisiones tomadas sean correctas.

Nosotros creamos nuestra base de casos manualmente tomando situaciones predefinidas recopilando los atributos en función de los sensores y decidiendo la solución que nos parecía oportuna en cada caso. Intentamos abarcar el mayor número de situaciones posibles entre las que existía alguna diferencia relevante (distintas soluciones y distintos valores de los atributos de los casos).

Paso 6: Algoritmo de recuperación de casos.

Es necesario definir un algoritmo que compare el caso problema definido por los sensores que se presenta con los casos almacenados en la librería de casos con el fin de decidir cuál es el más parecido.

En nuestro caso todos los atributos son del mismo tipo (numérico) por lo que hemos prescindido de medidas de similitud local. Para comparar casos se realiza una transformación de los datos de entrada (normalización) para formar puntos en el espacio euclídeo y comparar distancias entre ellos, se le ha asignado distinto peso a ciertos atributos para resaltar su importancia.

Paso 7: Implementación y pruebas.

La implementación está explicada en detalle en el punto siguiente, las pruebas determinan el grado de éxito obtenido en la resolución del problema. En función de este se puede tomar la decisión de volver a iterar en los distintos pasos en función de los problemas identificados. En nuestro caso decidimos, entre otras cosas, modificar una vez la base de casos inicial y reescribir ciertos comportamientos para que fueran más rápidos.

4.3 Implementación.

La estructura general del sistema que hemos creado está dividida en tres módulos principales: el robot AIBO, el sistema CBR y la comunicación entre estos dos módulos. En la figura 25 mostramos un diagrama de la arquitectura de nuestro sistema:

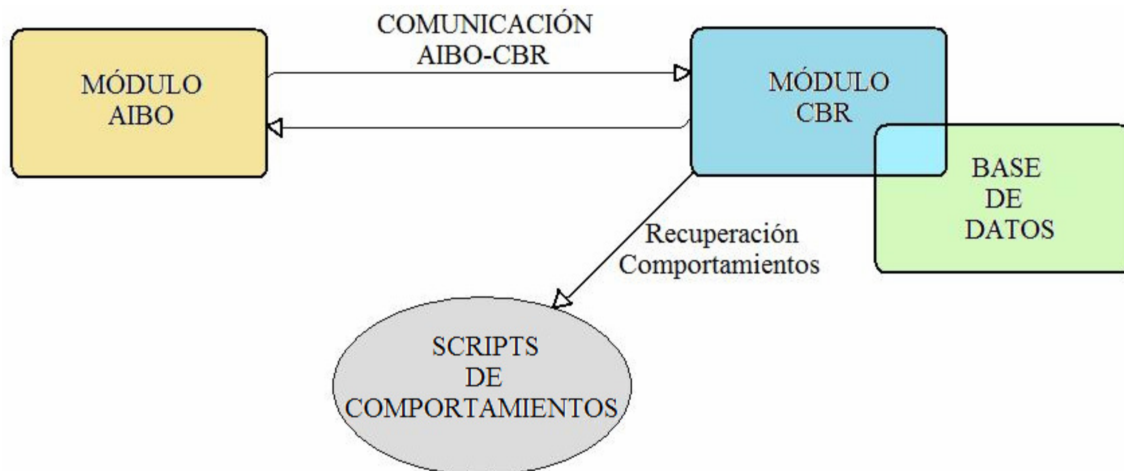

Figura 25. Diagrama de Módulos de la estructura general del sistema.

A grandes rasgos, el ciclo de funcionamiento de nuestro sistema es el siguiente:

- Percepción del entorno por parte del AIBO y recolección de valores a través de sus sensores. De esta forma, tendremos una idea de la situación en la que se

encuentra el robot en el campo y de la posición de la pelota con respecto al mismo.

- Envío de los datos adquiridos por el AIBO al módulo CBR, tratamiento de los datos para su utilización en los algoritmos de toma de decisión, y la recuperación del caso solución.

- Envío desde el sistema CBR del comando solución que forzará al AIBO a ejecutar el comportamiento recuperado.

Nuestra elección de esta arquitectura ha venido determinada por una serie de factores:

- Requeríamos de un tiempo de respuesta mínimo ya que se trata de un dominio en el cual las acciones se deben ejecutar en tiempo real. Esto solo lo podíamos conseguir utilizando un procesador suficientemente potente con el que no cuenta el robot. La solución fue realizar el proceso CBR desde un equipo remoto y comunicar la solución al AIBO.

- Para que la recuperación de casos fuese lo suficientemente precisa, necesitábamos una librería de casos considerablemente grande, y la memoria del AIBO limitaba mucho el tamaño de esta librería, por lo que este fue otro punto fundamental en nuestra decisión.

- Disponíamos de las herramientas necesarias proporcionadas por la librería LIBURBI para gestionar la comunicación entre el AIBO y el equipo remoto de forma rápida y eficiente, por lo que la arquitectura propuesta parecía adecuada.

4.3.1 Módulo AIBO.

Como hemos comentado anteriormente en el punto 1.1.2, al estar trabajando con un robot, nos vemos sujetos a trabajar con un modelo de sensores y actuadores, en el cual encontramos dos posibles paradigmas principales en los cuales basarnos: el paradigma jerárquico y el paradigma reactivo. Hemos elegido el paradigma jerárquico ya que la etapa de cognición la realiza un equipo remoto y no supone un cuello de botella temporal a la hora de realizar el ciclo percepción-cognición-reacción.

De este modelo jerárquico el robot AIBO realiza las etapas de percepción y actuación. En la etapa de percepción la lleva a cabo a través de sus sensores y de los valores que toman las articulaciones en un momento determinado. La etapa de actuación se basa en la ejecución de comandos correspondientes a comportamientos almacenados en el robot, que producen un cambio en la situación del AIBO dentro del entorno (campo de futbol) en el que se encuentra, o un cambio del entorno (desplazamiento de la pelota) con respecto al mismo.

4.3.1.1. Percepción.

En el dominio del futbol, y concretamente para desempeñar el rol de portero, hemos decidido que los factores más determinantes a percibir del entorno en todo momento son: la posición del robot respecto a la portería, posición de la pelota respecto al robot y el estado actual del robot (tumbado, de pié, despejando,etc.).

Para conocer estos datos, tenemos que tener en cuenta una serie de atributos, unos calculados por el propio CBR y otros que son obtenidos directamente de los sensores del robot. Los calculados por el CBR los explicaremos más adelante en el punto en que describiremos el funcionamiento en detalle del modulo CBR. Los sensores involucrados en la obtención de datos son los siguientes:

- La cámara: Se trata de un sensor de imagen CMOS de 350.000 pixeles. Uno de los atributos que utilizamos para describir los casos consiste en determinar si el AIBO detecta la pelota en su campo de visión, es decir, en la cámara. Para obtener este dato disponemos de una función URBI (ball.visible) que nos devuelve 1 si la pelota es detectada por la cámara y 0 en caso contrario. Esta función que nos proporciona URBI está configurada para que la cámara detecte una pelota de color rosa.

  El color que detecta viene determinado por un bitmap definido en un objeto incluido en la librería de LIBURBI (colormap). Este objeto recibe unos valores RGB y los interpreta como colores. Si necesitásemos utilizar una pelota de otro color, únicamente tendríamos que cambiar esta clase para que satisficiera nuestros propósitos. Una restricción que nos da la cámara es que, debido a la escasa resolución de la misma, la pelota no se detecta a una distancia superior a 1,7 metros, aunque esta distancia varía según la iluminación.

  Otra función que nos podría haber proporcionado la cámara es la función de ball.ratio, que nos indica la cantidad de pelota que entra dentro de la visión de la cámara, pero como hemos dicho antes, la iluminación influye mucho en los valores que nos proporcionan estas funciones, y sería un valor muy inexacto, y puesto que no es un valor que necesitemos para que la aplicación funcione correctamente, decidimos no utilizarla.

- Valor articulación HeadTilt: El valor de la articulación de HeadTilt nos proporciona datos relativos a la inclinación de la cabeza.

- Valor articulación HeadPan: El valor de la articulación de HeadPan nos proporciona datos relativos a la rotación de la cabeza respecto al cuello.

La combinación de los valores de HeadTilt y HeadPan nos ayuda a aproximar la posición relativa de la pelota respecto al AIBO. El sistema CBR toma estos atributos, los normaliza y los utiliza como atributos que describen los casos almacenados en la librería de casos que utiliza el CBR, y que servirán para la recuperación de casos. Puesto que estos valores solamente son útiles si la pelota es visible por la cámara, tendrán que tener un peso menor que el de la propia cámara en el cálculo de la similitud de casos.

4.3.1.2. Actuación.

Al iniciar la aplicación, esta enviará un fichero .u al AIBO que contendrá todos los comportamientos en forma de funciones, y que posteriormente solo tendrán que ser invocados para ser ejecutados. De esta forma agilizaremos el proceso de actuación, constando únicamente del envía de un comando desde el CBR al robot.

Este comando se encargará de invocar el comportamiento correspondiente a la situación en la que se encuentre el AIBO. Dicho comportamiento al finalizar su ejecución enviará al CBR una señal de fin de comportamiento mediante un objeto UCallback que explicaremos más adelante, y se podrá empezar el ciclo de percepción-cognición-actuación de nuevo. Este proceso de actuación es el más lento y costoso, ya que consiste en la ejecución de un comportamiento por parte del robot.

Debido a este hecho hemos intentado crear comportamientos lo más atómicos y rápidos posibles, del estilo de moverse un paso a la derecha, tumbarse para despejar, moverse dos pasos a la derecha, etc. De esta forma, el robot puede reaccionar mejor ante los cambios que se producen en el entorno, en nuestro caso el campo de futbol y la pelota.

Con esta medida parece que el sistema lleva a cabo el proceso de percepción-cognición-actuación relativamente rápido, pero puesto que no hemos tenido la ocasión de probarlo en un entorno completo de fútbol con más robots, no podemos decidir cuál es el nivel de perfeccionamiento alcanzado. Tenemos en mente algunas posibles mejoras como eliminar los comportamientos más complejos y dividirlos en comportamientos aún más simples, como por ejemplo las acciones de moverse dos pasos o tres pasos hacia algún lado eliminarlas, y quedarnos únicamente con las de moverse un solo paso hacia derecha e izquierda

4.3.2 Módulo CBR.

Este es el módulo principal del sistema. Se encarga de la recepción y tratamiento de los datos generados por el AIBO, y del envío de los comportamientos a realizar por el robot. En definitiva, es el módulo encargado interpretar el entorno percibido por el AIBO y de conseguir que este actúe en consecuencia de forma coherente dentro del dominio del futbol.

Este módulo está a su vez dividido en tres paquetes independientes e interconectados entre sí:

- **<u>Paquete de gestión de datos:</u>** Lleva a cabo toda la gestión de la librería de datos almacenada en la base de datos. Realiza la conexión con la base de datos al inicio de la aplicación y carga todos los datos en memoria para optimizar los tiempos a la hora de realizar los cálculos de similitud. De esta forma, en vez de tener que acceder a la base de datos cada vez que se quiera realizar un proceso de recuperación de casos, se consultará a un objeto almacenado en memoria que contendrá la librería de casos completa.

  Un trabajo futuro que tenemos planteado es el de aprendizaje automático del sistema, y este consistiría en la inserción de nuevos casos a la librería de casos según los fuese detectando el AIBO. Añadir esta funcionalidad a la aplicación sería relativamente sencillo ya que los datos generados por el robot y que habría que almacenar en la base de datos los tenemos disponibles en cada momento, con lo que únicamente habría que almacenarlos en la base de datos en el momento oportuno. Para la inserción en la librería de casos simplemente habría que crear una nueva función en la clase SGBD que conecte con la base de datos (lo cual requiere una única línea de código), y hacer la consulta pertinente.

- **Paquete CBR:** Es el paquete que realiza las tareas de recepción de datos y de cognición, por lo tanto es el paquete más importante de la aplicación. En este paquete hay disponibles funciones para el tratamiento de los datos recibidos del robot, y funciones para la realización del cálculo de similitud de casos necesarias para la correcta recuperación.

- **Paquete de comunicaciones:** En este paquete existen unos objetos llamados UCallbacks que son los necesarios para el establecimiento controlado de la comunicación con el AIBO, y mediante los cuales se enviarán los comandos de ejecución de los comportamientos, y se podrá detectar cuando ha terminado la ejecución de los mismos y si han finalizado correctamente. Hay un objeto UCallback para cada comportamiento, y son todos muy similares, por lo que si se quisiera insertar nuevos comportamientos, simplemente habría que crear un nuevo objeto por cada comportamiento.

4.3.2.1. Cognición.

Como hemos explicado anteriormente, la etapa de cognición consiste en tratar e interpretar la información recogida en la fase de percepción, y producir un resultado en forma de tareas a realizar en la etapa de actuación. Esta etapa concuerda con la etapa de recuperación de casos o razonamiento propio de los sistemas CBR. Para llevar a cabo este razonamiento adecuadamente, es necesario pasar por 3 etapas definidas:

- Tratamiento y transformación de los datos: En esta etapa se recogen los datos obtenidos en la etapa de percepción y se transforman en datos válidos para utilizarlos en el algoritmo de búsqueda del caso más parecido a la situación en la que se encuentra el robot. En nuestro caso vamos a disponer de dos datos que hay que nos proporciona el AIBO y que vamos a tener que tratar. Estos son: el valor de la articulación HeadTilt dada en valor real y que hay que normalizar a un valor entre 0 y 1; el valor de la articulación HeadPan que, al igual que el anterior, viene dada en valor real y que hay que normalizar a un valor entre 0 y 1. Estos datos no son los únicos valores que van a intervenir en el algoritmo de razonamiento, pero sí los únicos que tienen que ser tratados previamente.

- Búsqueda del caso más próximo: La etapa de búsqueda es la que realiza el cálculo más importante y en la que, según el algoritmo de similitud que se utilice, el resultado puede variar mucho de unos sistemas a otros. En la aplicación que hemos creado, utilizamos un algoritmo de similitud por proximidad de vectores. Este algoritmo crea para cada caso, así como para el caso actual, un vector que tiene tantas dimensiones como valores caracterizan los casos. Estos vectores están normalizados, por lo que serán todos vectores unitarios y con las mismas dimensiones. Una vez tenemos el vector del caso actual en el que se encuentra el AIBO, y los vectores de todos los casos almacenados en la librería de casos normalizados (lo cual se lleva a cabo en la etapa anteriormente explicada), comparamos el vector actual con cada uno de los vectores de la librería y nos quedamos con el más próximo. La proximidad de vectores la calculamos mediante resta de vectores, así pues el caso que obtenga el menor valor de resta con el vector asociado al caso actual, será el vector correspondiente al caso más parecido al caso actual.

- Identificación de la solución: Una vez recuperamos de la base de datos el caso más acorde a la situación en la que se encuentra el AIBO, el siguiente paso es identificar de este caso cuáles son las acciones a realizar. Estas acciones vienen almacenadas en el campo "acciones" del caso que está almacenado en la base de datos. Dicho campo contiene los nombres de las funciones que ejecutan los diferentes comportamientos que hemos creado anteriormente, y que ya están cargadas en el robot. Así pues, una vez tenemos los nombres de las funciones que tiene que ejecutar el AIBO, simplemente hay que enviarle los comandos pertinentes por orden y uno a uno, y esperar a que la última función a realizar acabe su ejecución.

4.3.2.2. Casos.

Como ya comentamos en el punto 4.1 en el que explicamos qué es un sistema CBR, los casos son los elementos principales de todo sistema CBR. Al ser los casos un elemento tan importante dentro de los sistemas CBR, uno de los primeros pasos que hay que dar a la hora de empezar a diseñar una aplicación de este tipo es definir claramente una representación y un método de almacenamiento para los casos. El método de almacenamiento ya lo hemos descrito anteriormente, pero no la representación de los casos.

Los casos tienen que estar representados como mínimo por la descripción del problema que se quiere resolver y por la solución de dicho problema. En nuestro caso, la descripción del problema consta de un conjunto de atributos, y estos se pueden agrupar en dos tipos según el origen de su valor: Obtenidos de los sensores del robot y calculados a partir de variables de la aplicación.

La solución a los diferentes problemas viene determinada por un atributo llamado "acciones", y consiste en una cadena de caracteres que contiene los nombres de las funciones que implementan los comportamientos que debe realizar el robot en cada situación o caso (estas acciones pueden constar de un solo comportamiento o varios).

La descripción del problema puede ser un conjunto de atributos que caracterizan la situación dentro del dominio donde se presenta, es importante una buena elección de estos atributos para facilitar y hacer más eficiente y completo el posterior proceso de recuperación, por lo que el desarrollador tiene que tener un mínimo de conocimiento sobre el dominio sobre el que está trabajando. La solución del problema puede ser de muy distinta naturaleza, dependiendo del ámbito en el que estemos trabajando, se puede proporcionar como solución un fragmento de código, una serie de acciones, un conjunto de filas de una base de datos, etc. Teniendo en cuenta lo anterior podemos decir que un caso se comporta como una correspondencia entre elementos del espacio de posibles problemas con elementos del espacio de posibles soluciones.

4.3.2.2.1. Atributos.

<u>Obtenidos de los sensores del robot</u>: Estos valores son proporcionados por los sensores y las articulaciones del AIBO, y son: "HeadTilt", "HeadPan" y "visible". "HeadTilt" y "HeadPan" son valores que nos dan las articulaciones que llevan esos nombres y que corresponden con la inclinación de la cabeza y el movimiento rotacional de la misma

respectivamente. El atributo “visible” es proporcionado por la cámara y nos indica si la pelota es detectada por esta o no.

Calculados a partir de variables de la aplicación: Los valores calculados son valores necesarios para posicionar al AIBO dentro del campo y determinar la posición en la que está, y que conseguirlos a través de valores que nos pueda proporcionar el robot sería muy costoso. Por ello creamos dos variables globales en la aplicación que inicializamos a 0, y que cambiaremos manualmente según el comportamiento que realice el AIBO. Estas variables son “aiboX” y “tumbado”. “aiboX” nos da la posición aproximada del robot respecto a la portería, y que modificamos cada vez que el AIBO lleva a cabo un movimiento lateral. “tumbado” es una variable booleana que nos indica si el robot está tumbado o está en posición inicial, y es necesaria debido a que cuando la pelota está cerca, el robot tiene que tumbarse para cubrir más portería, y no levantarse hasta que no vea que la pelota está lejos. Los demás comportamientos acaban en una posición inicial menos este, por ello necesitamos saber si se ha tumbado (ha realizado una parada) o está en posición inicial. Al igual que la anterior variable, la modificamos cada vez que el robot ejecute los comportamientos de parar o levantarse.

4.3.2.2.2. Solución.

La solución es el conjunto de acciones que debe realizar el AIBO según la situación en la que se encuentre. Esta viene determinada por un atributo llamado “acciones”, y consiste en una cadena de caracteres que contiene los nombres de las funciones que implementan los comportamientos que debe realizar el robot en cada situación o caso (estas acciones pueden constar de un solo comportamiento o varios). De esta forma, una vez realizado el razonamiento e identificado el caso más similar a la situación actual del robot, se recuperará de la base de datos el valor del campo “acciones” y se descifrará para extraer las funciones que habrá que mandarle ejecutar al AIBO. Un ejemplo de valor de este campo es:

“inicial;derecha;3;parada;”

Como podemos observar, en la cadena de caracteres podemos encontrar nombres de funciones y números. Los números corresponden al argumento de entrada a la función escrita inmediatamente antes que él, y solamente hay dos funciones que requieren de estos argumentos: “derecha” e “izquierda”. Los números indican el número de pasos que se quiere dar a derecha o izquierda. En este caso, las funciones que se reconocerían serían: ponerse en posición inicial; andar 3 pasos laterales a la derecha; ponerse en posición de parada.

Todas las funciones y los argumentos de entrada están separados por ‘;’ para poder identificar las diferentes funciones una vez recuperado el valor de la base de datos.

Así pues, una vez extraídos los nombres de las funciones que tiene que ejecutar el AIBO, se crearán los objetos correspondientes a esas funciones y se llamarán por orden y uno a uno. Estos enviarán un comando que invocará la función respectiva almacenada en el robot y que disparará la ejecución de un comportamiento específico. Cuando este comportamiento finalice mandará un mensaje a la aplicación para que esta pueda saber que ha terminado su ejecución y puede mandar el siguiente comando.

4.3.2.2. Librería Casos.

La librería de casos es el módulo que se encargará de almacenar la base de casos que vayamos a utilizar. En nuestro caso hemos elegido utilizar una estructura de memoria plana por dos motivos principales:

- Un inconveniente que tiene este tipo de estructuras es la recuperación de casos, pero puesto que en nuestro sistema únicamente vamos a hacer una recuperación de todos los casos al inicializar la aplicación, este inconveniente no va a ser relevante.

- Una ventaja que proporciona la memoria plana es la inserción de nuevos casos, y puesto que nosotros necesitábamos crear la base de casos desde cero y teníamos que insertar los casos uno a uno, esto era una ventaja que nos iba a proporcionar ligereza a la hora de la creación de la base de casos.

La implementación que hemos elegido para crear la memoria plana ha sido crear una base de datos MySQL en la que tenemos una tabla ("AIBO"), en la que cada fila corresponde con un caso distinto. Esta tabla está formada por registros que contienen los siguientes campos:

- id: Identificador del caso para disponer de un orden y poder saber fácilmente que caso se está recuperando en cada momento.

- visible: Atributo con valor booleano que indica si la pelota es detectada por la cámara o no (1 ó 0 respectivamente).

- tumbado: Atributo con valor booleano que indica la postura del robot, si está de pié o tumbado (0 ó 1 respectivamente).

- aiboX: Determina la posición del robot respecto a la potería. Puede tomar valores dentro del rango de los reales, pero en la base de casos inicial que creamos nosotros solo toma 3 valores distintos: -27, 0 y 27. Estos valores corresponden con las posiciones: pegado al poste izquierdo de la portería, en el centro de la portería, y pegado al poste derecho de la portería respectivamente.

- headTilt: Nos proporciona el valor de la articulación que indica la inclinación de la cabeza del AIBO.

- headPan: Contiene el valor de la articulación que representa el movimiento giratorio de la cabeza del robot respecto al cuello.

- acciones: Como ya hemos explicado en el punto anterior, almacena una cadena de caracteres en la cual aparecen los nombres de las funciones que debe realizar el AIBO en cada caso. Representa la solución del caso.

- ratio: Este es un atributo que en un principio pensamos en utilizar como descriptor de casos junto con los demás ya mencionados, y que indica la cantidad de pelota que se puede observar con la cámara. Finalmente la desechamos debido a que su valor depende mucho de la iluminación de la

estancia. Todos los valores son null.

- descripción: Es otro atributo que en un principio pensamos utilizar como ayuda para identificar los casos, pero que finalmente descartamos puesto que la labor de rellenar este campo en todos los casos que incluimos era muy tediosa y no era imprescindible. Todos los valores son null.

- fichero: Este campo representaría la ruta en la que estaría almacenado el fichero que contendría la implementación de la/s función/es que encontramos en el campo “acciones”, pero como al final decidimos tener todas las funciones cargadas en el AIBO, este campo quedó inservible. Todos los valores son null.

En la figura 26 presentamos un pantallazo que muestra el un fragmento del contenido de la librería de casos:

id	visible	tumbado	aiboX	headPan	headTilt	acciones	ratio	descripcion	fichero
1	0	0	0	0	0	buscar;	NULL	NULL	NULL
2	0	1	0	0	0	inicial;buscar;	NULL	NULL	NULL
3	1	0	0	40.1	24.8	izquierda;1;inicial;	NULL	NULL	NULL
4	1	0	27	35.6	27.8	izquierda;2;inicial;	NULL	NULL	NULL
5	1	0	-27	19.3	31.67	nada;	NULL	NULL	NULL
6	1	1	0	27.4	24	inicial;izquierda;1;inicial;	NULL	NULL	NULL
7	1	1	27	35.47	15	inicial;izquierda;2;inicial;	NULL	NULL	NULL
8	1	1	-27	20.7	16.5	inicial;nada;	NULL	NULL	NULL
9	1	0	0	36.1	27.5	izquierda;1;inicial;	NULL	NULL	NULL
10	1	0	27	44.08	25.95	izquierda;2;inicial;	NULL	NULL	NULL
11	1	0	-27	26.7	26.7	nada;	NULL	NULL	NULL
12	1	1	0	37.7	19.4	inicial;izquierda;1;inicial;	NULL	NULL	NULL
13	1	1	27	42.6	14.1	inicial;izquierda;2;inicial;	NULL	NULL	NULL
14	1	1	-27	25	16.4	inicial;nada;	NULL	NULL	NULL
15	1	0	0	2.5	32.06	nada;	NULL	NULL	NULL
16	1	0	27	14.9	35.87	izquierda;1;inicial;	NULL	NULL	NULL
17	1	0	-27	-6.7	32.8	derecha;1;inicial;	NULL	NULL	NULL
18	1	1	0	1	28.5	inicial;	NULL	NULL	NULL
19	1	1	27	14	23.66	inicial;izquierda;1;inicial;	NULL	NULL	NULL
20	1	1	-27	-8.22	23.28	inicial;derecha;1;inicial;	NULL	NULL	NULL
21	1	0	0	-1	36	nada;	NULL	NULL	NULL
22	1	0	27	12	36.25	izquierda;1;inicial;	NULL	NULL	NULL
23	1	0	-27	-14	31.29	derecha;1;inicial;	NULL	NULL	NULL
24	1	1	0	-2.1	26.71	inicial;	NULL	NULL	NULL
25	1	1	27	11	25.2	inicial;izquierda;1;inicial;	NULL	NULL	NULL

170 rows fetched in 0,0228s (0,0690s) — Edit — Apply Changes — Discard Changes — First — Last — Search

Figura 26. Librería de casos del proyecto.

Así pues, un caso está representado en la base de datos como una tupla de: id, visible, tumbado, aiboX, headPan, headTilt y acciones. Los demás atributos son de apoyo para su organización. Cada caso estará representado como una fila de la librería de casos y en total disponemos de 170 casos.

Para adquirir este conocimiento primero definimos la forma en que lo haríamos y posteriormente diseñamos un plan para recolectar los casos de una forma ordenada y

eficaz. La forma que se nos ocurrió fue la ir colocando la pelota en diferentes posiciones del campo que mostraremos más adelante en un esquema, posicionar al AIBO en 3 posiciones distintas de la portería para cada posición de la pelota, y tomar medidas de los atributos de la descripción de los casos (aiboX, visible, tumbado, headTilt y headPan) para las dos posibles posturas del robot. Es decir, para cada posición de la pelota en el campo, tomaríamos medidas para 6 posiciones del AIBO distintas: en el centro y de pié, en el centro y tumbado, pegado al poste derecho y de pié, pegado al poste derecho y tumbado, pegado al poste izquierdo y de pié, y pegado al poste izquierdo y tumbado. El esquema de las diferentes posiciones en las que colocamos la pelota y tomamos las 6 mediciones se muestra en la figura 27 mostrada aquí abajo:

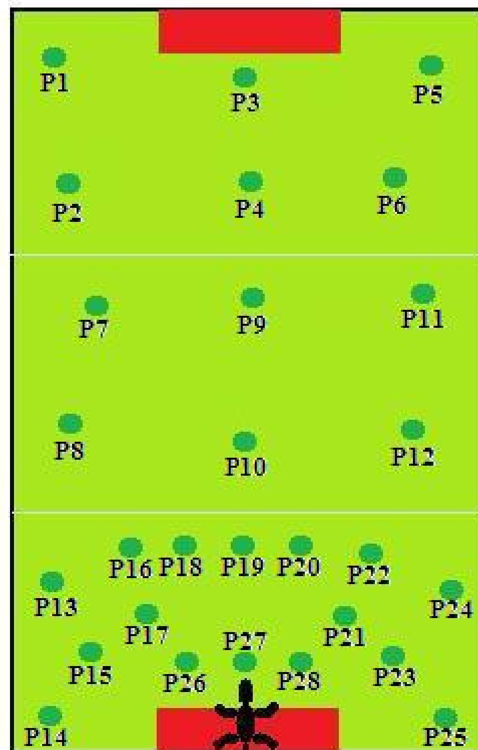

Figura 27. Posiciones de la pelota en la base de casos inicial.

Como se puede observar, el campo lo hemos dividido en 3 secciones longitudinales: una lejana (P1-P6), una intermedia (P7-P12) y una cercana (P13-P28). A su vez estas 3 secciones las hemos dividido en otras tantas transversales, aumentando el número de las mismas cuanto más cerca de la portería está la pelota, ya que existe mayor número de posibles comportamientos. Así pues, en las zonas lejana e intermedia hemos definido 6 posiciones de pelota en cada una, mientras que en la zona cercana decidimos tomar 16 posiciones ya que el comportamiento en esta zona tiene que ser más exacto y hay muchas más posibles acciones que en las otras dos zonas del campo.

A estos casos, hubo que añadir dos casos más, los cuales venían determinados por la visibilidad de la pelota. Estos son los casos en los que se dé la situación de haber perdido la pelota de vista, y estar tumbado o de pié. En estos casos dará igual los demás valores de los atributos de la descripción del caso ya que la acción será buscar la pelota, y en caso de estar tumbado levantarse y buscar la pelota.

Con esto, logramos generar una base de casos de 170 casos, que en un principio probamos y nos daba algunas soluciones erróneas, derivadas de fallos cometidos al introducir datos en la base de datos. Una vez detectados estos fallos, reeditamos la base de datos y conseguimos una base de casos inicial que creemos bastante completa y eficiente, y a la que se pueden incluir nuevos casos de forma sencilla.

4.3.2.3. Algoritmo recuperación casos.

Todo sistema CBR tiene que contar con un algoritmo propio de recuperación de casos que se adapte a su estructura de casos y a sus necesidades. La aproximación más intuitiva para comparar casos es comprobar la similitud de los distintos atributos que los conforman. Por ello, para elegir el mecanismo de similitud es importante tener en cuenta que no todos los atributos que definen un caso son del mismo tipo, nos podemos encontrar atributos representados por enteros, cadenas de caracteres o enumerados por ejemplo.

En nuestro caso los atributos que van a entrar en juego en el algoritmo son o bien valores reales, o bien valores booleanos. Puesto que estos últimos se pueden transformar en valores reales, decidimos que todos los atributos contendrían valores reales. Por esto, decidimos que el mecanismo que mejor se adaptaba a nuestros casos iba a ser el de “Medidas de similitud globales”, que recoja los datos obtenidos en las medidas de similitud local, asignando pesos para distinguir la importancia de los distintos atributos, y determine la similitud entre casos completos.

Como en la mayoría de las técnicas de similitud global, decidimos utilizar una función de distancia entre los valores de los atributos ya que existe una relación inversamente proporcional entre el concepto de distancia y el de similitud. La función de distancia que optamos por aplicar es la distancia euclídea que mide la distancia entre dos puntos en el espacio eluclídeo. La distancia euclídea ponderada se define como:

$$D_{Euc}(C_i, C_j) = \frac{\sqrt{\sum_k (w_k d_k(at_k^i, at_k^j))^2}}{\sum_k w_k}$$

dónde:

- w_k es el peso del atributo k
- at_k^i es el valor del atributo k del caso i
- $d(x, y)$ es la distancia que separa x de y

De esta forma, nuestra función de similitud compara el caso actual que representa la situación en la que se encuentra el AIBO con todos los casos contenidos en la base de casos. Una vez realizada la comparación, se quedará con la solución del caso más próximo y ordenará al robot realizar las funciones correspondientes a dicha solución.

Este es un algoritmo simple pero eficiente, ya que los cálculos a realizar son muy simples y, puesto que estamos lidiando con un sistema en tiempo real, cuanto más simples sean los cálculos mejor será el tiempo de respuesta. Por ello el mecanismo y el algoritmo de similitud elegidos creemos que son los más adecuados para la resolver nuestro problema.

4.3.3. Comunicación AIBO-CBR. [19]

La comunicación con del AIBO con la aplicación consiste en el paso de mensajes de una a otro ya que es bidireccional. Para poder realizar este paso de mensajes, hay que establecer un medio por el que transmitir la información. Para ello, el AIBO dispone de un módulo LAN inalámbrica WiFi. Los pasos a seguir para establecer la conexión entre el AIBO y el PC son los siguientes:

- Confirmar los requerimientos del sistema: Necesitamos disponer de un PC con un adaptador LAN wireless compatible con el estándar IEEE 802.11b que permita crear redes AD HOC.

- Habilitar las funciones de LAN wireless: Para poder habilitar las funciones de wireless del AIBO es necesario encender dicho dispositivo. Para ello hay que poner en ON una pestaña que podemos encontrar poniendo boca arriba al robot y quitándole la cubierta.

- Configurar las funciones de red del AIBO: Hay que modificar el fichero WLANCONF.TXT que podemos encontrar en la MEMORY STICK que contiene el AIBO MIND, o copiar dicho fichero e introducirlo en la tarjeta de memoria que vayamos a utilizar, y modificar los valores que se requiera. Para modificar este fichero recomendamos seguir los pasos que nos indica el documento de "Guía de configuración de red inalámbrica" cuya referencia se puede encontrar en el apartado de referencias.

- Configurar una red AD HOC en nuestro PC: El siguiente paso es configurar una red punto a punto en nuestro PC. Para ello también recomendamos seguir los pasos descritos en el documento de guía de configuración mencionado en el punto anterior.

- Conectar el PC con el AIBO: Una vez configurada la red AD HOC, podemos encender el robot e intentar conectar el PC a la red AD HOC creada. Si la conexión se realiza correctamente, se encenderán dos leds azules, uno encima de cada oreja de la cabeza del AIBO. Para realizar esta conexión disponemos de varios medios: podemos hacerlo a través de un interfaz de manejo del AIBO como por ejemplo el Telecommande, o a través de herramientas proporcionadas por las librerías de los distintos lenguajes de programación que nos permiten programar funciones para el robot como por ejemplo la librería Liburbi del lenguaje URBI.

Puesto que hemos estado trabajando con URBI, vamos a pasar a explicar las herramientas que nos proporciona Liburbi para establecer la comunicación y realizar el paso de mensajes entre el AIBO y el PC.

Para establecer la comunicación disponemos de una clase llamada UClient. En nuestro caso creamos una variable global estática que llamamos 'a' y que es un objeto de tipo UClient. Este objeto lo creamos al inicializar la aplicación, y la propia constructora del objeto intenta realizar la conexión con el AIBO. La creación de este objeto y realización de la conexión lo hacemos mediante el siguiente fragmento de código:

```
a = new UClient("10.0.1.100");
if (a.error() != 0)
{
System.out.println("Couldn't connect to the URBI server.");
return;
}
```

Este objeto de tipo UClient será el encargado de gestionar toda la comunicación con el robot. También disponemos de una interfaz de apoyo a la comunicación: “UCallbackInterfaze”. Esta interfaz nos sirve para crear objetos que implementan esta interfaz y que están constantemente pendientes de recibir mensajes generados por el robot. Estos mensajes normalmente contienen el valor de una variable declarada mediante una etiqueta y que corresponde con un sensor o una articulación del AIBO. Así pues podremos estar recibiendo constantemente el valor de los sensores y las articulaciones que necesitemos y almacenándolas en variables de nuestra aplicación para su tratamiento y utilización en el algoritmo de similitud. Un ejemplo de una clase que implementa este interfaz es la clase “RecogeVisible”, que se encarga de comprobar si la pelota está siendo vista por la cámara del robot o ha dejado de verla, en cuyo caso se ejecutaría la función de “BuscaPelota”:

```
package cbr.callbacks;
import liburbi.main.*;
import cbr.CBR;


public class RecogeVisible extends UCallbackInterface {


    private CBR c;
    public RecogeVisible(CBR c) {
      super();
      this.c = c;
    }

    public UCallbackAction onMessage(UMessage msg) {
      String s = msg.getRawMessage();
      float v = Float.valueOf(s);
      c.setVisible((int)v);
      if (c.getVisible() == 1) c.setBuscar(0);
      return UCallbackAction.URBI_CONTINUE;
    }
}
```

Esta sería la estructura de la clase, y la creación de un objeto instancia de la misma se haría mediante este fragmento de código:

```
RecogeVisible rv = new RecogeVisible(this);
a.setCallback(rv, "visible");
```

Como podemos observar son clases muy simples que únicamente implementan dos funciones: la constructora y la función “onMessage” que es la encargada de “escuchar” a través de la comunicación con el AIBO, pendiente de detectar la llegada de algún mensaje que corresponda con la etiqueta “visible”. Solamente recibirá los mensajes referentes a cambios en la variable “visible” porque al inicializar el objeto hemos definido que el “Callback” o “Devolución de la llamada” contenga la etiqueta visible.

De esta manera podemos crear un objeto para cada elemento del AIBO del que deseemos recoger su valor.

Así pues, disponiendo de estas herramientas para gestionar la comunicación entre el AIBO y el PC no tuvimos ningún problema de retardo en la transmisión de mensajes en tiempo real, y pudimos resolver el problema de la comunicación de manera satisfactoria.

5. Conclusiones y trabajo futuro

El desarrollo de sistemas con robots es una tarea ardua y compleja, a las dificultades propias del desarrollo de aplicaciones virtuales se añade la complejidad de tratar con un objeto físico en el mundo real y la limitación de trabajar con un dispositivo de características muy concretas. Este trabajo es, no obstante, muy gratificante, puesto que sus frutos son palpables y llamativos y esto hace mayor la satisfacción del trabajo bien hecho.

El hecho de trabajar con un robot programable nos ha brindado la posibilidad de trabajar en los campos de la Inteligencia Artificial y el de la Robótica a la vez, expandiendo nuestros conocimientos y experiencia en ambas disciplinas.

Conclusiones.

Las principales conclusiones y logros del trabajo son los siguientes:

- Se ha documentado las principales características del AIBO ERS7 y se informa sobre distintas plataformas para el desarrollo de aplicaciones en él. Se ha experimentado con estas plataformas a fin de evaluar sus características y posibilidades.

- Se ha desarrollado una metodología para el diseño e implementación de comportamientos, Ésta ha probado ser útil y sencilla, haciendo más simple y metódico este trabajo.

- Se ha investigado sobre los sistemas expertos y en concreto sobre CBR. Se ha aplicado los conocimientos adquiridos sobre nuestra metodología para reutilizar los comportamientos. Se eligió el dominio de soccerbots, concretamente el rol de portero, para el desarrollo de un prototipo que probase la eficacia de este sistema, obteniendo resultados bastantes satisfactorios. No obstante, nos hubiera gustado ampliar la funcionalidad del sistema, implementando otros roles e incluso la sincronización de varios miembros de un equipo.

Trabajo futuro

Aunque consideramos que los objetivos básicos propuestos al inicio del proyecto se han cumplido, consideramos interesante ampliar el alcance del proyecto en las siguientes líneas de trabajo:

- Ampliar la funcionalidad de todo el sistema experto para permitir que el robot pueda desempeñar cualquier rol dentro del campo de fútbol. Para ello habría que crear nuevos comportamientos y casos.

- Permitir que el AIBO aprenda los nuevos casos problema que han sido resueltos de manera satisfactoria (manual y automático).

- Mejorar el algoritmo de similitud, utilizando medidas de similitud local para reducir las búsquedas aumentando la velocidad de las mismas.

- Mejora de los comportamientos, optimizando la precisión y velocidad en que se realizan las acciones.

6. Índice de Figuras

- Tabla 1. Evolución física de las distintas generaciones de AIBO comercializadas por SONY. – Páginas 9 y 10.
- Figura 1. Ciclo del Paradigma Jerárquico. – Página 13.
- Figura 2. Ciclo del Paradigma Reactivo. – Página 14.
- Figura 3. Diseño del Sistema AIBO. – Página 15.
- Figura 4. Flujo de interconexión de los objetos en OPEN-R. – Página 17.
- Figura 5. Menú de ejemplo de la GUI del Controlador. – Página 20.
- Figura 6. Imágenes de RawCam y SegCam. – Página 21.
- Figura 7. GUI de Control Remoto del Movimiento. – Página 21.
- Figura 8. Funciones de conexión (Connexion). – Página 22.
- Figura 9. Funciones internas del robot (Functions). – Página 23.
- Figura 10. Funciones de manejo de motores (Motors). – Página 24.
- Figura 11. Funciones de manejo de los diferentes leds (Leds). – Página 25.
- Figura12. Funciones de cargado de ficheros en el robot (Files). – Página 26.
- Figura 13. Funciones de configuración y manejo de la cámara del robot (Video). – Página 27.
- Figura 14. Funciones de obtención de diversas graficas (Graph). – Página 28.
- Figura 15. Funciones de envío de comandos (Terminal). – Página 29.
- Figura 16. Máquina de estados de “Sigue Pelota”. – Página 38.
- Figura 17. Máquina de estados de “Busca pelota”. – Página 41.
- Figura 18. Máquina de estados optimizada. – Página 42.
- Figura 19. Máquina de estados del ejemplo. – Página 43.
- Figura 20. Metodología GMG. – Página 47.
- Figura 21. Máquina de estados de “Persecución de una pelota”. – Página 49.

- Figura 22. Espacio de problemas y soluciones. – Página 56.
- Figura 23. Memoria plana y memoria jerárquica. – Página 57.
- Figura 24. Ciclo CBR. – Página 59.
- Figura 25. Diagrama de Módulos de la estructura general del sistema. – Página 62.
- Figura 26. Librería de casos del proyecto. – Página 70.
- Figura 27. Posiciones de la pelota en la base de casos inicial. – Página 71.

7. Referencias Bibliográficas

- [1] Programación aplicada a la robótica – SIMD. http://system.albacete.dotnetclubs.com/ftp/Charlas%2006-07/Programación%20aplicada%20a%20robótica.ppt

- [2] Nilsson, Nils J.: “Inteligencia Artificial. Una nueva síntesis.” Mc-Graw-Hill 2001

- [3] Guía de Tekkotsu. www.tekkotsu.org

- [4] Guía de Webots. www.cyberbotics.com

- [5] Baillie, Jean-Christophe; Nottale, Mathieu; Pothier, Benoit y Despres, Nicolas: “URBI Tutorial for Urbi 1.5.” Copyright © 2006-2007 Gostai™

- [6] “URBI quick start guide Version 1.5” Copyright © Gostai™ SAS 2006 - 2007

- [7] Humbert, Remi: “Introduction to liburbi Java for Urbi 1.x.” Copyright © 2008 Gostai™

- [8] Escolano Ruiz, Francisco; Cazorla Quevedo, Miguel Ángel; Alfonso Galipienso, Mª Isabel; Colomina Pardo, Otto; Lozano Ortega, Miguel Ángel. Inteligencia Artificial. “Modelos, Técnicas y áreas de aplicación.” THOMPSON 2003.

- [9] Kolodner, Janet: "Reconstructive Memory: A Computer Model," Cognitive Science 7 (1983): 4.

- [10] Koton, P.: “Using Experience in Learning and Problem Solving”. Informe técnico MIT/ LCS/TR-441, MIT 1989.

- [11] Hinrichs, T.R.: “Problem solving in open worlds”. Lawrence Erlbaum Associates, 1992.

- [12] Sycara, K. (1988). Using case-based reasoning for plan adaptation and repair. “Proceedings Case-Based Reasoning Workshop, DARPA”. Clearwater Beach, Florida. Morgan Kaufmann.

- [13] Simpson, Robert L.: “A computer model of case-based reasoning in problem solving: An investigation in the domain of dispute mediation”. Technical Report GIT-ICS-85/18, Georgia Institute of Technology. 1985.

- [14] Hammond, Kristian J.: “Case-based planning”. Academic Press. 1989.

- [15] Leake, David B.: “CBR in Context: The Present and The Future”. En: David B. Leake (Ed). Case-Based Reasoning: Experiences, Lessons, and Future Directions, capítulo 2, pp.31-65. American Association for Artificial Intelligence, 1996.

- [16] Palma Méndez, José T. y Martín Morales, Roque: “Inteligencia Artificial. Técnicas, métodos y aplicaciones.” McGraw-Hill 2008

- [17] Kolonder, Janet L. y Leake, David B.: “A Tutorial Introduction to Case-Based Reasoning”. En: David B. Leake (Ed.), Case-Based Reasoning: Experiences, Lesons, and Future Directions, capítulo 2, pp. 31-65. American Association for Artificial Intelligence, 1996.

- [18] Aamodt, Agnar y Plaza, Enric: "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," Artificial Intelligence Communications 7 (1994): 1, 39-52.

- [19] Guía de configuración red inalámbrica. http://support.sony-europe.com/AIBO/downloads/en/PCNET_EN.pdf