Universidad politécnica de Cartagena

Escuela Superior de Ingeniería de Telecomunicación

Máster TIC

Servicios software en robots humanoides

Director : Juan Ángel Pastor Franco

Codirector : Humberto Martínez Barberá

Autor : Juan José Alcaraz Jiménez

Fecha : 30/09/2008

Índice general

Resumen

El presente documento tiene por objeto condensar el trabajo realizado de cara a la puesta a punto del robot humanoide NAO para la disputa de un campeonato de fútbol robótico denominado RoboCup™. RoboCup es un proyecto internacional para promover el desarrollo en Inteligencia Artificial, robótica y campos relacionados. Es un intento de fomentar la investigación proporcionando un problema estándar donde se integran un amplio rango de teconologías.

Para que un equipo robótico realmente juegue al fútbol, se deben aplicar varias tecnologías, incluyendo: diseño básico de agentes autónomos, colaboración multi-agente, adquisición de estrategias, razonamiento en tiempo real, robótica y fusión de sensores.

Desde la Universidad de Murcia, representada por el equipo TeamChaos, se viene participando en la RoboCup desde el año 2002, aunque hasta ahora siempre se había hecho con el robot canino Aibo, desarrollado por Sony, que consituía la plataforma estándar seleccionada para la disputa del campeonato. En el año 2008, la organización quiso dar un salto más hacia la consecución de su objetivo final, y adoptó como plataforma estándar un robot con forma humanoide: Nao, fabricado por la empresa francesa Aldebaran Robotics.

De este modo, los principales retos que el equipo TeamChaos ha afrontado para esta edición de la RoboCup han sido los siguientes:

- Rediseñar la arquitectura del software concebida para el robot AIBO de manera

que se amolde a la nueva plataforma Nao.

- Rescatar aquellas funcionalidades del software diseñado para Aibo que pudiesen ser reutilizadas para el robot NAO y reimplementarlas, ajustándolas a la nueva arquitectura.

- Rehacer integramente aquellos módulos que no se pudiesen adaptar a Nao. En concreto, se han creado nuevas versiones del módulo de locomoción (la estructura mecánica del robot no es la misma) y del módulo de comunicaciones (con objeto de mejorar su funcionamiento).

- Tratar los problemas derivados del estádo prototípico del robot Nao. Según el caso, solucionando el problema, colaborando con Aldebaran Robotics para la solución del problema o atenuando su efecto.

Este documento persigue un doble objetivo. Por un lado, describir el funcionamiento general del sistema deteniéndonos brevemente en los diferentes módulos de software que lo componen, y por otro lado, exponer los trabajos de adaptación ejecutados y los problemas afrontados durante la puesta a punto del robot de cara a la participación en el campeonato RoboCup 2008.

Comenzaremos con una introducción que nos ayude a contextualizar el trabajo desarrollado, posteriormente pasaremos a describir los módulos que componen el sistema y terminaremos exponiendo algunas cuestiones relativas a la adaptación del software a la plataforma Nao.

Palabras clave: Nao, humanoide, robot, arquitectura, comunicaciones, RoboCup.

Capítulo 1

Introducción

1.1. Evolución de la robótica

La palabra "robot" viene del vocablo checo robota, "servidumbre", inicialmente utilizado para los llamados "trabajadores alquilados"que vivieron en el Imperio Austrohúngaro hasta 1848. El término fue utilizado por primera vez por Karel Čapek en su obra teatral R.U.R. (Rossum's Universal Robots).

Existen numerosos ejemplos de máquinas de funcionamiento robótico a lo largo de la historia, por ejemplo, en el principio del siglo XVIII, Jacques de Vaucanson creó un androide que tocaba la flauta, así como un pato mecánico que comía continuamente. En uno de los cuentos de Hoffmann de 1817, El Coco, presenta una mujer que parecía una muñeca mecánica.

Hacia 1942, Isaac Asimov da una versión más humanizada a través de su conocida serie de relatos, en los que introduce por primera vez el término robótica con el sentido de disciplina científica encargada de construir y programar robots. Además, este autor plantea que las acciones que desarrolla un robot deben ser dirigidas por una serie de reglas morales, llamadas las Tres leyes de la robótica:

- Un robot no debe dañar a un ser humano o, por su inacción, dejar que un ser

Figura 1.1: Robot Aibo de Sony

humano sufra daño.

- Un robot debe obedecer las órdenes que le son dadas por un ser humano, excepto si estas órdenes entran en conflicto con la Primera Ley.

- Un robot debe proteger su propia existencia, hasta donde esta protección no entre en conflicto con la Primera o la Segunda Ley.

Actualmente, no es posible aplicar las leyes de Asimov, dado que los robots no tienen capacidad para comprender su significado, evaluar las situaciones de riesgo tanto para los humanos como para ellos mismos o resolver los conflictos que se podrían dar entre estas leyes.

Entender y aplicar lo anteriormente expuesto requeriría verdadera inteligencia y consciencia del medio circundante, así como de si mismo, por parte del robot, algo que a pesar de los grandes avances tecnológicos de la era moderna no se ha llegado.

En 2002 Honda y Sony, comenzaron a vender comercialmente robots humanoides como “mascotas”. Los robots con forma de perro o de serpiente se encuentran en una fase de producción muy amplia, el ejemplo más notorio ha sido Aibo de Sony [1].

1.2. La robótica en la actualidad

Los robots son usados hoy en día para llevar a cabo tareas sucias, peligrosas, difíciles o repetitivas para los humanos. Esto usualmente toma la forma de un robot industrial usado en las líneas de producción. Otras aplicaciones incluyen la limpieza de residuos tóxicos, exploración espacial, minería, búsqueda y rescate de personas y localización de minas terrestres. La manufactura continúa siendo el principal mercado donde los robots son utilizados. En particular, robots articulados (similares en capacidad de movimiento a un brazo humano) son los más usados comúnmente. Las aplicaciones incluyen soldado, pintado y carga de maquinaria. La Industria automotriz ha tomado gran ventaja de esta nueva tecnología donde los robots han sido programados para reemplazar el trabajo de los humanos en muchas tareas repetitivas. Existe una gran esperanza, especialmente en Japón, de que el cuidado del hogar para la población de edad avanzada pueda ser llevado a cabo por robots.

Existen diferentes tipos y clases de robots, entre ellos con forma humana, de animales, de plantas o incluso de elementos arquitectónicos pero todos se diferencian por sus capacidades y se clasifican en 4 formas:

- Androides: robots con forma humana. Imitan el comportamiento del hombre, su utilidad en la actualidad es de solo experimentación. La principal limitante de este modelo es la implementación del equilibrio a la hora del desplazamiento, pues es bípedo.

- Móviles: se desplazan mediante una plataforma rodante (ruedas); estos robots aseguran el transporte de piezas de un punto a otro.

- Zoomórficos: es un sistema de locomoción imitando a los animales. La aplicación de estos robots sirve, sobre todo, para el estudio de volcanes y exploración espacial.

- Poliarticulados: mueven sus extremidades con pocos grados de libertad. Su utilidad es principalmente industrial, para desplazar elementos que requieren cuidados.

1.3. El robot humanoide Nao

Hasta ahora, los androides, o robots humanoides, han permanecido sobre todo en el dominio de la ciencia ficción. Sin embargo, en la actualidad ya existen algunos robots humanoides. El androide siempre ha sido representado como una entidad que imita al ser humano tanto en apariencia, como en capacidad mental e iniciativa. Antes incluso de haber visto un verdadero robot en acción, la mayoría de las personas asocian la idea de robot con la de androide, debido a su extrema popularidad como cliché de la ciencia ficción.

La actitud de base entre el público frente a los androides varía en función del bagaje cultural que posea dicho público. En la cultura occidental la criatura humanoide, fabricada casi siempre por un sabio, es con bastante frecuencia un monstruo que se rebela contra su creador y en ocasiones lo destruye como castigo por su arrogancia. De hecho, es tan notorio este fenómeno, que el reconocido experto en inteligencia artificial Marvin Minsky, llegó a narrar como en ocasiones llegaba a sentirse incómodo frente a una de sus creaciones, el androide Cog, cuando éste presentaba conductas inesperadas.

En otras culturas las reacciones pueden ser bastante diferentes. Un ejemplo meritorio es la actitud japonesa frente a los androides, donde el público no teme la antropomorfización de las máquinas, y aceptan por lo tanto con menos problemas la idea que un robot tenga apariencia humana, para poder así interactuar más fácilmente con seres humanos.

En general podemos resumir las ventajas de la robótica humanoide en dos:

- Facilitan la interacción con otros humanos.

- Se adaptan rápidamente a entornos y herramientas diseñados para humanos y perfeccionados a lo largo de la historia. Si bien, una forma humana no es óptima para ninguna tarea concreta, la generalidad de la aplicación del cuerpo humano para diferentes trabajos podría ser aprovechada por los robots humanoides, que heredarían la versatilidad para realizar diferentes tareas.

La plataforma sobre la cual vamos a llevar a cabo nuestras investigaciones consiste en un robot humanoide denominado Nao y fabricado por la empresa francesa Aldebaran Robotics [2]. Si bien el robot no se encuentra aún en fase de comercialización, si contamos con prototipos lo suficientemente avanzados como para poder llevar a cabo experimentos en áreas como locomoción, localización, reconocimiento de objetos y comunicación.

Nao parte de un diseño ambicioso en relación calidad a precio. Cuenta con un gran número de sensores y grados de libertad y su destino final pretende ser el público general, estando prevista su salida al mercado a lo largo del año 2009. A día de hoy, contamos con la segunda versión de venta a grupos de investigación colaboradores en el desarrollo del robot. Dicha versión está enfocada al uso del robot como plataforma estándar en la competición de fútbol robótico RoboCup, celebrada anualmente. Próximamente está previsto el reemplazo de esta versión por otra con capacidades mejoradas, especialmente en lo que se refiere a la estructura plástica empleada en la parte inferior del cuerpo del robot y al sistema eléctrico.

Nao ofrece una armoniosa y estilizada representación de la figura humana a tamaño reducido (57cm) y de bajo peso (4.5 Kg). El robot dispone de módulos de software embebido que permiten la reproducción de archivos de texto a voz, localización basada en sonido, reconocimiento de formas y creación de efectos visuales a través de sus LEDs.

En el plano computacional, el robot cuenta con un procesador x86 AMD Geode a

Figura 1.2: Robot Nao, fabricado por Aldebaran Robotics

500Mhz, 256 Mb de memoria SDRAM y 1 GB de memoria Flash. Además, el robot cuenta con dos interfaces de comunicación: Ethernet y Wifi 802.11g respectivamente. El sistema operativo, OpenNao, es una distribución abierta-embebida de Linux desarrollada por Aldebaran Robotics.

En cuanto a la estructura mecánica, el robot Nao posee 21 articulaciones, todas ellas de rotación. En la tabla 1.1 enumeramos sus nombres y valores máximos y mínimos de sus ángulos de giro. En la figura 1.3 podemos observar el emplazamiento sobre el cuerpo de Nao de cada una de estas articulaciones. Una de las articulaciones, HipYawPitch aparece en las dos piernas pero en realidad se trata de una sola articulación compartida.

Las secuencias de movimientos de las articulaciones que animan el robot no nos serían útiles si este no poseyese un conocimiento del medio en que se encuentra, para hacer efectiva la interacción. Por este motivo, Nao cuenta con varios tipos de sensores:

- Ocho sensores de fuerza emplazados en las esquinas de las plantas de los pies. Gracias a ellos podemos detectar donde se encuentra la proyección del centro de

Cadena	*Articulación*	*Movimiento*	*Rango[]*
Cabeza	HeadYaw	Head joint twist	-120 to 120
	HeadPitch	Head joint front back	-45 to 45
Brazo Izquierdo	LShoulderPitch	Left shoulder joint front back	-120 to 120
	LShoulderRoll	Left shoulder joint right left	0 to 95
	LElbowYaw	Left shoulder joint twist	-120 to 120
	LElbowRoll	Left elbow joint	-90 to 0
Pierna izquierda	LHipYawPitch*	Left hip joint twist	-90 to 0
	LHipRoll	Left hip joint right left	-25 to 45
	LHipPitch	Left hip joint front and back	-100 to 25
	LKneePitch	Left knee joint	0 to 130
	LAnklePitch	Left ankle joint front back	-75 to 45
	LAnkleRoll	Left ankle joint right left	-45 to 25
Pierna derecha	RHipYawPitch*	Right hip joint twist	-90 to 0
	RHipRoll	Right hip joint right left	-45 to 25
	RHipPitch	Right hip joint front and back	-100 to 25
	RKneePitch	Right knee joint	0 to 130
	RAnklePitch	Right ankle joint front back	-75 to 45
	RAnkleRoll	Right ankle right left	-25 to 45
Brazo Derecho	RShoulderPitch	Right shoulder joint front back	-120 to 120
	RShoulderRoll	Right shoulder joint right left	-95 to 0
	RElbowYaw	Right shoulder joint twist	-120 to 120
	RElbowRoll	Right elbow joint	0 to 90

Tabla 1.1: Nombre y rango de Articulaciones

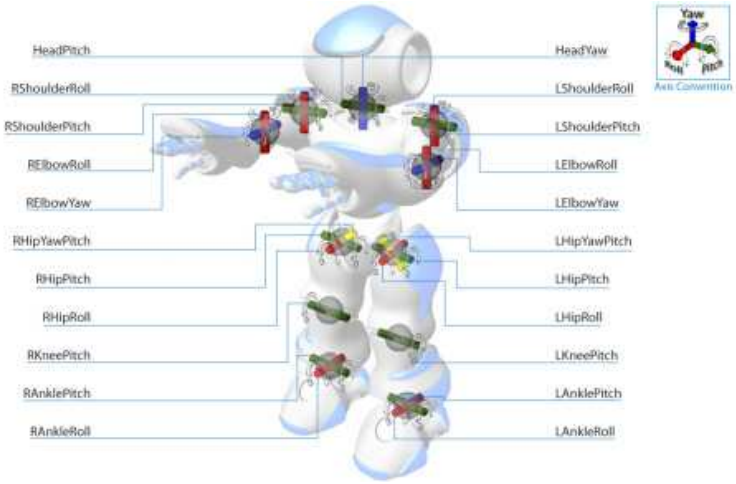

Figura 1.3: Las articulaciones de Nao.

masas y prevenir o corregir desequilibrios.

- Una unidad inercial situada en el torso y con su propio procesador. Esta unidad inercial está compuesta de dos girómetros y tres acelerómetros. Gracias a ella podemos obtener la velocidad de traslación del robot en cualquier dirección del espacio y la velocidad de rotación del mismo respecto a dos ejes situados en un plano horizontal. Es decir, podemos detectar cualquier movimiento del robot excepto una rotación sobre un eje vertical.

- Dos sensores de ultrasonidos que permiten al robot estimar la distancia a los objetos de su ambiente. El rango de detección varía de 0 a 70 cm, pero por debajo de 15cm no hay información de distancia, el robot solo es capaz de detectar la presencia de un obstáculo.

- Una cámara de vídeo capaz de captar 30 imágenes por segundo localizada en la frente del robot y capaz de ofrecer resoluciones de hasta 640x480 píxeles. Esta cámara se emplea para reconocer los objetos que rodean a la máquina.

- Dos micrófonos, situados a ambos lados de la cabeza, permiten al robot localizar fuentes de sonido, así como llevar a cabo labores de reconocimiento de sonidos.

Para completar la interacción con el medio, Nao también cuenta con dos altavoces que, en conjunto con los micrófonos permitirían a Nao la comunicación oral bidireccional, si bien en este caso el hardware va muy adelantado al software y es poco probable que veamos a Nao manteniendo conversaciones elaboradas.

Además de los altavoces, el robot dispone de LEDs en torso, pies, orejas y ojos para enriquecer el proceso de comunicación. Estos LEDs podrían expresar desde un simple testigo de carga de la batería a un conjunto de emociones.

1.4. RoboCup

RoboCup [3] es un proyecto internacional para promover el desarrollo en Inteligencia Artificial, robótica y campos relacionados. Es un intento de fomentar la investigación proporcionando un problema estándar donde se integran un amplio rango de tecnologías. Si bien se ha elegido el fútbol como tema central de investigación, el objetivo es que las innovaciones sean aplicables a problemas significativos de la sociedad y la industria.

El objetivo definitivo del proyecto RoboCup consiste en "Desarrollar, para el año 2000, un equipo de robots humanoides completamente autónomos que puedan ganarle al equipo campeón del mundo de fútbol".

Para que un equipo robótico realmente juegue al fútbol, se deben aplicar varias tecnologías, incluyendo: diseño básico de agentes autónomos, colaboración multi-agente, adquisición de estrategias, razonamiento en tiempo real, robótica y fusión de sensores.

Capítulo 2

Arquitectura del sistema

En este capítulo se explicarán la diferentes partes que componen el software que anima el robot, así como la herramienta externa empleada para su depuración. Existen cinco módulos distintos que tratan de aislar cinco tareas principales de manera prácticamente independiente [4]. Estas tareas son:

- Percepción de los objetos del medio (módulo PAM)
- Representación de un modelo del mundo según los datos percibidos (módulo GM)
- Toma de decisiones de tipo táctico y estratégico (módulo CTRL)
- Implementación de tareas de movimientos simples, como andar en una dirección y a una velocidad determinada (módulo CMD).
- Comunicación de un robot con otros robots y con herramientas de desarrollo (módulo COMMS).

Para poder ensamblar estas tareas entre sí y acoplarlas sobre la capa de software Middleware, hacen falta ciertos trabajos de adaptación, que se encarga de realizar la clase RobotManager, como veremos en el capítulo 3.

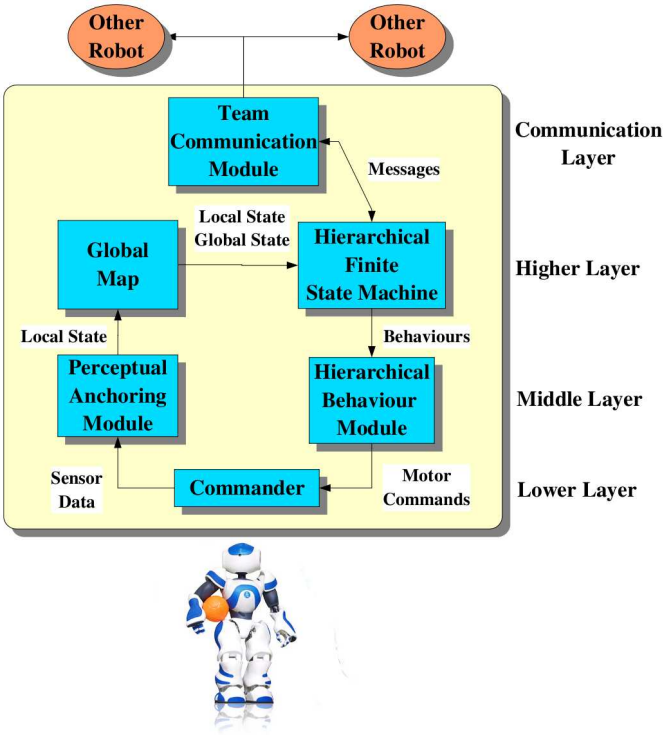

Figura 2.1: Variante de la arquitectura ThinkingCap

2.1. Arquitectura ThinkingCap

Cada robot emplea una estructura de capas como la mostrada en la figura 2.1. Se trata de una variante de la arquitectura ThinkingCap, que es un marco para construir robots autónomos conjuntamente desarrollado por la Universidad de Örebro y la Universidad de Murcia [5]. A continuación destacamos los principales elementos de esta arquitectura:

- La capa inferior, consiste en un módulo de abstracción de hardware (CMD), que proporciona una interfaz abstracta a las funcionalidades sonsorial-motrices del robot. El CMD acepta comandos abstractos de la capa superior y los implementa en términos de movimientos reales de los actuadores del robot. En particular, CMD recibe vectores con las velocidades deseadas $< v_x, v_y, v_\theta >$, donde v_x, v_y son

las velocidades hacia adelante y laterales y v_θ es la velocidad angular, y las traslada a un estilo de caminar apropiado controlando cada una de las articulaciones de las piernas.

- La capa media mantiene una representación consistente del entorno del robot (módulo de percepción, o PAM), e implementa un conjunto de comportamientos tácticos y robustos (módulo de comportamientos jerárquicos, o HBM). El PAM actúa como una memoria a corto plazo para la localización de objetos en torno al robot: en cada momento, el PAM contiene una estimación de la posición de esos objetos basada en la combinación de observaciones actuales y pasadas y de odometría. El PAM también es el módulo encargado del movimiento de la cámara, seleccionando el punto de fijación de acuerdo a las necesidades perceptivas [8]. La HBM realiza un conjunto de comportamientos de navegación y control de la pelota.

- La capa superior mantiene un mapa global del campo (GM) y realiza decisiones estratégicas en tiempo real basadas en la situación actual. La autolocalización del GM está basada en lógica difusa [14] [13].

- La capa de radiocomunicaciones se emplea para intercambiar posiciones e información de coordinación con otros robots (módulo de comunicaciones).

Esta arquitectura proporciona una modularización efectiva e interfaces claras, facilitando el desarrollo de las diferentes partes de ella [6]. Además, su implementación distribuida permite la ejecución de cada módulo en una computadora o robot indiferentemente. Por ejemplo, los módulos de bajo nivel pueden ser ejecutados en el robot y los módulos de alto nivel en un PC, dónde dispondremos de herramientas de depurado. Sin embargo, una implementación distribuida genera serios problemas de sincronización.

Esto genera retardos en las decisiones y los robots no pueden reaccionar suficientemente rápido a los cambios dinámicos del entorno. Por esta razón, se ha favorecido la implementación de una arquitectura de modo mixto: en tiempo de compilación se decide si se va a emplear una versión distribuida (cada módulo es un hilo) o una versión monolítica (la arquitectura entera es un hilo y el módulo de comunicación otro). Como veremos más adelante, esta última versión de la arquitectura (la implementación monolítica) es la que mejor rendimiento ofrece, por las dificultades que la temporización de los hilos acarrea en un sistema que no es de tiempo real.

2.2. Módulo de percepción

El módulo de percepción, o PAM (Perception Anchoring Module), agrupa las clases encargadas de reconocer objetos, y también gestiona las tareas de percepción activa [7], es decir, orienta la cámara para buscar o rastrear objetos determinados según unos parámetros de ajuste de la importancia de dichos objetos.

2.2.1. Detección de objetos

Dentro de las tareas de detección de objetos, podemos distinguir dos subprocesos. Por un lado, la creación de blobs y por otro la identificación de estos blobs como alguno de los objetos del terreno de juego.

Para crear un blob comenzaremos con la segmentación de una imagen. En el proyecto hemos contemplado dos técnicas para la segmentación, la segmentación por crecimiento de regiones [11] y la segmentación por umbral [9]. En la práctica, se ha mostrado más eficiente la segmentación por umbral, y es por ello la que usaremos habitualmente.

Es de destacar que ambas técnicas están implementadas mediante una tabla de búsqueda (Look-up table) para mejorar la eficiencia. Este será un tema recurrente y

al que centramos la máxima atención: la eficiencia del código, como veremos en el capítulo 3. Las tareas de percepción ocupan una de las mayores porciones de carga del procesador.

Una vez segmentada la imagen, se crea el blob teniendo en cuenta parámetros como una densidad y un tamaño mínimo determinados.

La última fase del reconocimiento de objetos consiste en tratar de identificar los blobs con elementos conocidos de nuestro juego. Por ejemplo, si hemos encontrado un blob azul, podemos comprobar su forma para decidir si puede ser un fragmento de portería. De este modo, cada robot tratará de identificar balizas, terreno de juego, líneas de campo, pelota, porterías y jugadores propios y contrarios.

Uno de los detalles a tener en cuenta para mejorar la eficacia de las tareas de reconocimiento de objetos, es la incorporación al algoritmo de una función de detección del horizonte. De esta manera podemos asegurar que los objetos que reconocemos están sobre el terreno de juego (y evitamos así confundir objetos del mundo exterior con elementos del juego).

Finalmente, el objetivo es obtener las coordenadas (ángulo y distancia) del máximo número de elementos del juego posibles, con la máxima frecuencia posible. Si bien, algunos elementos como la pelota pueden ser más importantes que otros como los jugadores, y por eso el robot tratará de buscar algunos elementos mientras que otros solo los registrará si durante el juego entran en su campo visual. En el siguiente apartado veremos como se produce este seguimiento de los objetos de interés.

2.2.2. Seguimiento de objetos

Para lograr el seguimiento de objetos, es preciso mover un conjunto de articulaciones de forma que la cámara fije en la fóvea (zona central de la imagen con poca distorsión de

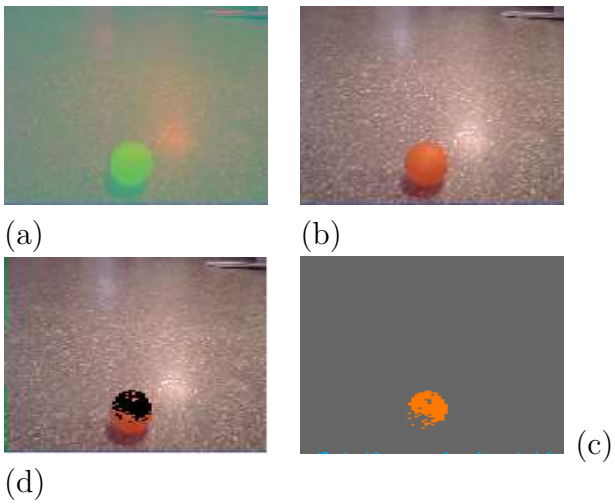

Figura 2.2: Distintas fases del reconocimiento de objetos. En (a), la imagen en el formato el formato nativo de la cámara (YUV). Transformado a RGB nos queda (b). En (c) se resaltan las semillas y en (d) se muestra la segmentación final.

la lente) el objeto que queremos seguir. Puesto que tanto el robot como el objeto están en movimiento, será preciso monitorizar la desviación del objeto respecto al centro de la imagen constantemente para obtener los ángulos de las articulaciones que es necesario modificar para tratar de acercarnos lo máximo posible en el siguiente ciclo.

Sin embargo, puede ocurrir que el objeto que necesitamos no se encuentre en el campo visual del robot, en cuyo caso habrá que buscarlo. Para maximizar la probabilidad de encontrar el objeto, se ha optado por realizar un barrido con lo cabeza que abarque los máximos ángulos de giro de las articulaciones del cuello y a la máxima velocidad posible que permita una detección fiable (a velocidades altas las imágenes captadas por la cámara resultan borrosas).

Una vez encontrado un objeto determinado, se plantea la cuestión de seguir o comenzar a buscar otro. Para ello se tendrá en cuenta la importancia asignada a cada objeto, el tiempo transcurrido desde la última vez que se detectó y el tiempo transcurrido desde la última vez que se ha buscado.

2.2.3. Asistencia a la calibración

La última de las funciones del módulo PAM consiste en asistir a la herramienta externa encargada de la calibración de los parámetros de detección de objetos. En secciones posteriores veremos más en detalle esta herramienta. Dependiendo del tipo e intensidad de la luz que ilumine un objeto determinado, éste reflejará unos colores distintos, por lo tanto es necesario ajustar los valores de los colores asignados a cada objeto según las condiciones ambientales en que nos encontremos.

El proceso de calibrado, es por lo tanto una tarea frecuente y resulta providencial que se realice de la forma más rápida y cómoda posible. Para lograr esto, contamos con una herramienta externa que solicita al robot las imágenes en diferentes formatos: YUV, RGB o incluso segmentadas para algún color. De este modo podemos ajustar las semillas (valores de los colores asociados a cada objeto), la densidad de los blobs y otros parámetros de la cámara y así optimizar las condiciones para la detección.

El módulo PAM será también la parte encargada en el sistema del robot de enviar las imágenes que esta herramienta externa solicite y de asimilar los parámetros con los nuevos ajustes que desde ella nos lleguen.

2.3. Módulo de generación de mapas

El módulo de generación de mapas globales, o GM (Global Map), es el encargado de generar un modelo del mundo a partir de las percepciones que cada robot tiene de su entorno[15]. Aunque por ahora el modelo generado solo se basa en las percepciones del propio robot, se está trabajando en la fusión de las percepciones del robot con las que le lleguen de sus robots vecinos, para así poder llegar a estimaciones más completas y precisas.

Para poder situar los elementos que rodean al robot, cuya posición relativa conocemos, en el mapa global, primero el robot debe haber sido capaz de encontrar su posición absoluta y orientación en dicho mapa. Posteriormente, a partir de las estimaciones odométricas y de las percepciones relativas obtenidas del módulo PAM, seremos capaces de situar dentro del mapa global los objetos reconocidos, aunque con una imprecisión considerable[13][14].

Las estimaciones odométricas consisten en calcular cuanto nos hemos desplazado desde la última vez que se estimó la posición del robot en el mapa. Existen diferentes técnicas para obtener dichas estimaciones. Por un lado, podemos suponer que si el robot pretende avanzar a 10 centímetros por segundo, después de un segundo habremos avanzado 10 centímetros. Sin embargo, lo anterior no siempre es cierto, ya que unas superficies tienen un rozamiento diferente a otras y a demás, los obstáculos que el robot puede encontrar en su camino pueden modificar su velocidad de avance. Por ello, una técnica interesante podría consistir en emplear los acelerómetros incorporados al torso del robot para estimar las distancias recorridas. Actualmente se está colaborando con la empresa desarrolladora del robot para que implemente, en una nueva versión del prototipo, una unidad inercial que cuente con el número suficiente de girómetros para estimar cualquier variación en la posición del robot.

Existen diferentes técnicas para lograr la localización de objetos dentro del campo. Nuestro sistema implementa dos de ellas. Una basada en filtros de Kalman y la otra en técnicas Markovianas. Finalmente, la que se ha mostrado más adecuada ha sido la localización Markoviana.

Para este tipo de localización, se representa la posición del robot como una función de probabilidad repartida a lo largo del mundo conocido, que se representa mediante una rejilla[12]. Entre sus ventajas destaca la capacidad para localizarse a partir de cero y de recuperarse de fallos de localización.

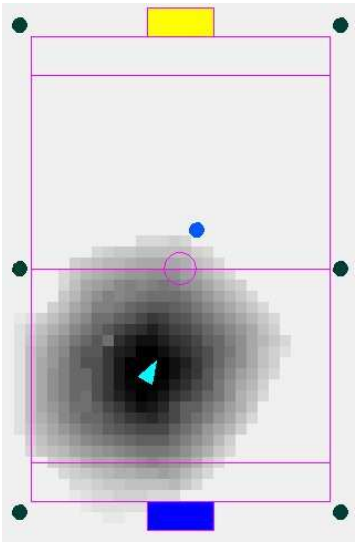

Figura 2.3: Representación de la estimación de la localización. Las zonas oscuras representan posiciones probables.

2.4. Módulo de comportamiento

El módulo de comportamiento, o CTRL, es el encargado de tomar las decisiones tácticas y estratégicas del robot. Es decir, por un lado, decide si resulta interesante ir hacia la pelota o hacia otro lugar, y, posteriormente, en el caso de que decida ir hacia la pelota, elige la dirección y la velocidad oportunas para colocar al robot con una orientación determinada respecto a la pelota, en un momento dado.

Los datos de los que se nutren las decisiones del módulo CTRL son los mapas de localización global generados por el módulo GM, y la salida del módulo son instrucciones de velocidad (lineal, lateral y de rotación) a interpretar por el módulo de locomoción (CMD).

En problemas cotidianos el abanico de comportamientos necesario puede ser muy rico. Por ejemplo, en el caso de disputar un partido de fútbol, los jugadores deben ser capaces de interceptar trayectorias, rodear la pelota para despejar en la dirección oportuna, orientar el jugador para lograr un buen disparo o desmarcarse para recibir

un pase. Todos estos comportamientos requieren de una cantidad importante de tiempo para su definición. Por ello, cuando se diseño el módulo CTRL se optó por emplear el lenguaje LUA para la definición de los comportamientos.

Además, para enlazar unos comportamientos con otros, es preciso definir unas reglas para las transiciones, y para ello empleamos una máquina jerárquica de estados finitos. A la hora de definir estos estados, nos ayudamos de una interfaz gráfica desarrollada por otro de los equipos colaboradores en la RoboCup [16].

LUA es un lenguaje de programación imperativo y estructurado, bastante ligero y que fue diseñado como lenguaje de script con una semántica extensible. La principal ventaja que nos ofrece sobre el lenguaje C++ es que podemos cambiar el código en tiempo de ejecución sin necesidad de recompilar, y esto es muy útil para la edición de comportamientos. El ciclo de trabajo consiste habitualmente en ordenar al robot que ejecute un determinado comportamiento, estudiar las posibles mejoras, enviar una modificación del script que lo define (ayudándonos de la herramienta externa Chaos-Manager) e indicarle al robot que vuelva a ejecutar dicho comportamiento.

Si redactásemos los comportamientos directamente en C++ habría que compilarlos y relanzar todo el programa en el robot, lo que ralentizaría considerablemente el ciclo de desarrollo. A cambio, tendría la ventaja de una mayor eficiencia. Sin embargo, para comportamientos estables, podemos compilar los scripts de LUA para lograr un menor coste computacional.

2.5. Módulo de abstracción de hardware

El módulo de abstracción de hardware, o CMD, es el encargado de gestionar las tareas de más bajo nivel que se ejecutan en el robot, especialmente, aquellas relacionadas con los movimientos de sus articulaciones o con la lectura de sensores. Podemos agrupar

estos movimientos dentro de dos subconjuntos, las articulaciones de la cabeza, encargadas de buscar o seguir objetos y las articulaciones del resto del cuerpo, encargadas de desplazar el robot en una dirección y a una velocidad determinadas.

Respecto a las articulaciones de la cabeza, el movimiento de la misma para seguir objetos viene especificado por el módulo PAM, con el que hay una comunicación constante para tratar este tema. Sí se encarga el módulo CMD de generar los puntos intermedios de trayectorias de escaneado de campo visual para las tareas de búsqueda de objetos.

Pero la parte más compleja sin duda del módulo CMD y probablemente de todo el sistema consiste en la generación de las secuencias de valores para los movimientos de las piernas utilizados para desplazar el robot. Existen tres tipos de movimientos: desplazamiento lateral, lineal y rotacional. Encontrar la secuencia de valores para las articulaciones que permiten conseguir dichos desplazamientos es una tarea compleja que requiere el estudio de la disposición de los centros de masas de cada fragmento entre articulaciones del robot y su comportamiento dinámico, además de resolver el problema cinemático inverso que permite desplazar los pies de un lugar a otro.

Entre los parámetros que se varían para conseguir las diferentes velocidades y estilos de desplazamiento están la altura a levantar el pie del suelo en cada paso, el desplazamiento lateral de la cadera (es necesario proyectar el centro de masas sobre un pie para poder desplazar el otro) y el ángulo de giro[18]. Combinando un ángulo de giro con un desplazamiento lineal, podemos obtener un desplazamiento curvilíneo, por ejemplo.

El hecho de que trabajemos con un robot bípedo y con una parte importante del peso en la parte superior del cuerpo, complica considerablemente el problema del desplazamiento, ya que el robot sufre una gran inestabilidad.

2.6. Herramientas externas: ChaosManager

Durante el proceso de configuración y mejora del funcionamiento de los robots, es inevitable el proceso de testeo y corrección, ya que ni los robots ni sus programadores son máquinas ideales. De hecho, las tareas de validación y reconfiguración abarcan una parte muy importante del tiempo empleado en el desarrollo del sistema. Para ayudarnos en todo este proceso, puede ser muy útil el empleo de alguna herramienta externa al sistema para que acelere su desarrollo.

En el TeamChaos contamos con una herramienta diseñada para tal fin: el ChaosManager. El ChaosManager permite acelerar las tareas de calibración de la cámara y parámetros de reconocimiento visual, definir y testear comportamientos, y monitorizar la tarea de localización.

2.6.1. Vision

El ChaosManager dispone de una ventana para la visualización de las imágenes tomadas por la cámara del robot. De manera adicional, podemos solicitar al robot que nos envíe las imágenes transformadas de algún modo que nos pueda ayudar a entender si el proceso de reconocimiento de objetos está funcionando adecuadamente. De este modo, podemos indicarle que las imágenes consistan solamente en aquello reconocido como portería, que resalte los píxeles pertenecientes a la pelota o que el formato sea YUV.

Si entendemos que el proceso de reconocimiento de objetos no está funcionando como era de esperar, podemos tomar diferentes medidas, como la modificación de las tonalidades en las que el sistema se basa para reconocer los objetos, o la densidad mínima exigida a los blobs que empleamos para reconocerlos. Otra acción necesaria para la calibración consiste en el ajuste de algunos parámetros de la cámara, como el

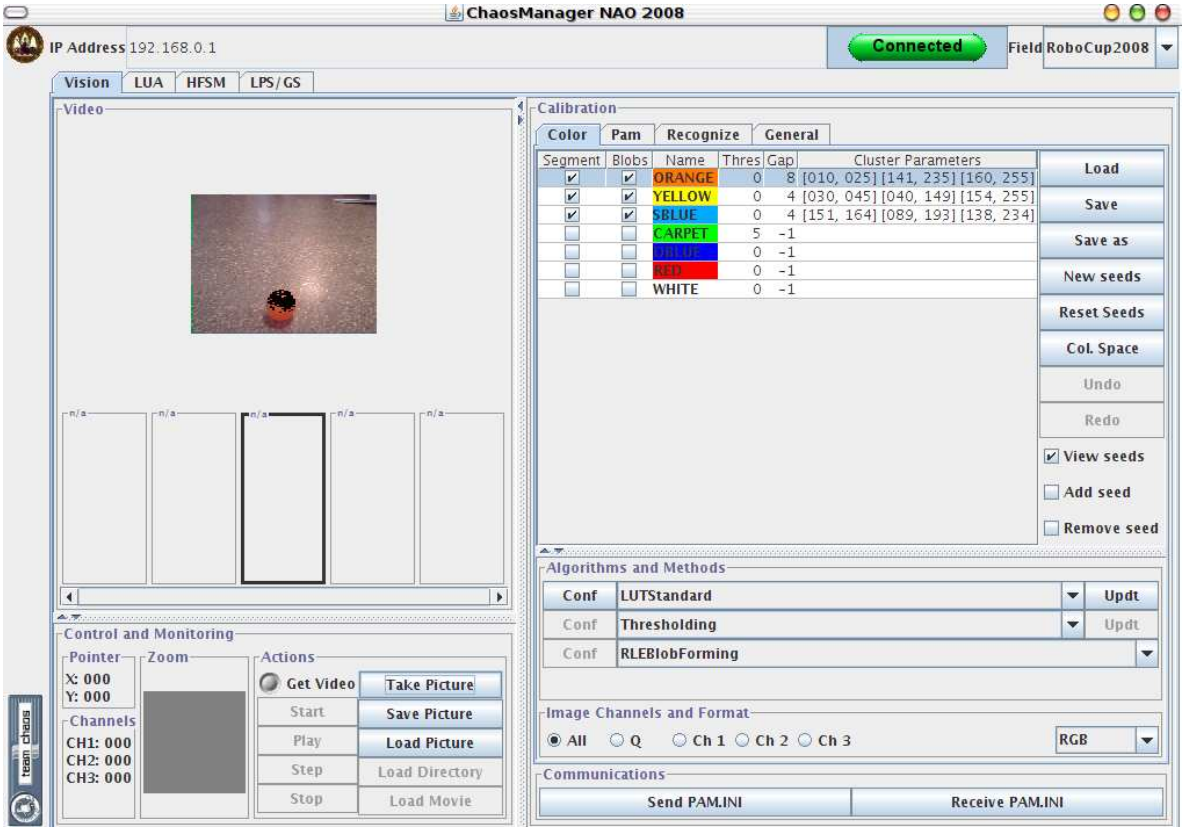

Figura 2.4: Aspecto de la ventana de calibración integrada en la herramienta ChaosManager.

balance de blanco o la ganancia.

Estas herramientas externas también nos pueden servir para monitorizar algunos errores impredecibles del funcionamiento del hardware. Por ejemplo, en la versión actual de hardware Nao, la cámara situada en la cabeza puede sufrir alteraciones del orden de los canales. Es decir, que los canales rojo y azul se inviertan. El ChaosManager está preparado para detectar y corregir este error.

2.6.2. Teleoperación

Desde el ChaosManager podemos manejar de forma remota algunas de las funcionalidades del robot. Podemos indicarle que se desplace en cualquier dirección del plano

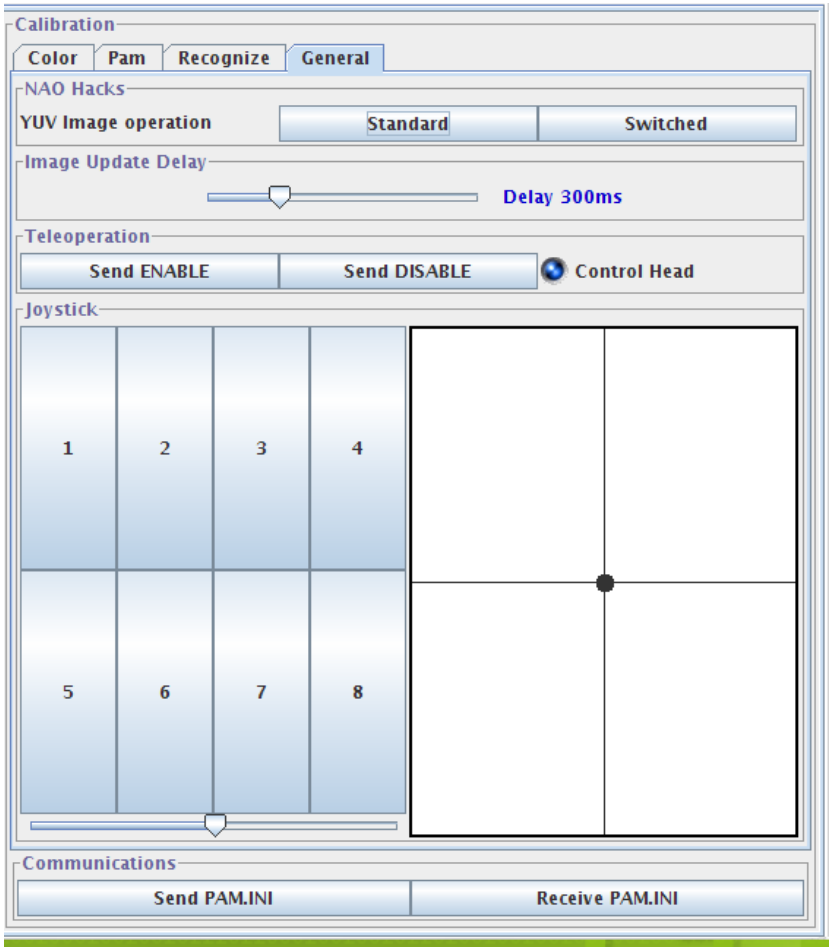

Figura 2.5: La teleoperación del robot puede ser muy interesante para depurar el módulo de locomoción.

horizontal o que gire su cabeza para orientarla hacia algún punto.

También podemos lanzar kicks, que son secuencias de movimientos de las articulaciones del robot que hemos definido de forma previa. Ejemplos de kicks podrían ser dar una patada, levantarse del suelo o celebrar un gol.

2.6.3. Comportamientos

Como comentábamos anteriormente, desde el diseño del módulo CTRL se hizo pensando en la hacer cómoda la labor de depuración de los comportamientos. La interfaz de

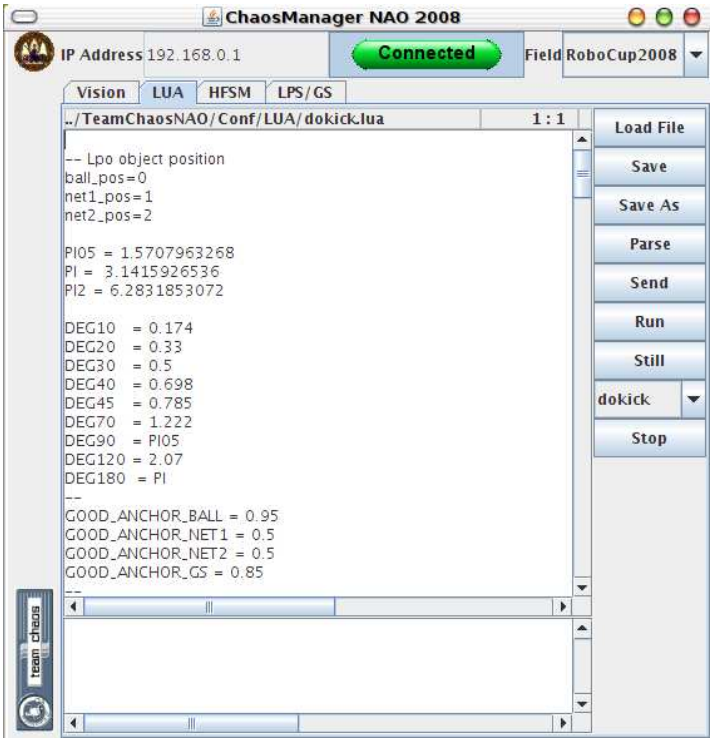

Figura 2.6: ChaosManager nos permite editar y validar comportamientos en tiempo real.

edición de scripts LUA con que cuenta el ChaosManager es el resultado de ese esfuerzo. Desde ella podemos editar los script de comportamiento y enviarlos al robot mientras el programa está corriendo. De esta forma podemos testear los script de una forma muy rápida.

2.6.4. Localización

La última de las funciones de la herramienta ChaosManager consiste en monitorizar los datos de localización que genera cada robot. Los robots actualizan constantemente su representación del mundo y envían un paquete UDP con la misma. El ChaosManager se encarga de recogerlas todas ellas y mostrarlas en un gráfico, lo que nos vale para comprobar la calidad de la localización.

Figura 2.7: Funciones de localización de la herramienta ChaosManager.

En realidad, los datos que el robot envía podrían dividirse en percepciones relativas y percepciones globales. Es decir, por un lado, los mensajes que los robots envían al ChaosManager incorporan información acerca de la posición relativa a la que el robot estima que está la pelota y por otro, los mensajes indican la posición y orientación que tendría el robot en el campo.

Esto nos permite tanto comprobar el funcionamiento del módulo PAM (percepción relativa) como del módulo GM (localización global).

Capítulo 3

Aportaciones y mejoras del sistema

En este capítulo veremos algunas de las tareas abordadas para conseguir llevar el robot Nao a un estado operativo, partiendo del sistema instalado en los equipos Aibo. Al margen de numerosos ajustes y pequeñas modificaciones repartidas por todo el sistema, las principales tareas llevadas a cabo en el marco de la realización de este Trabajo Fin de Máster se congregan principalmente en tres áreas:

- Creación de un módulo de comunicaciones robusto y adaptado a situaciones de congestión. Las comunicaciones servirán para dos actividades diferentes: la comunicación entre robots para acciones colaborativas y la comunicación de los robots con la herramienta externa ChaosManager para las tareas descritas en el capítulo 2.

- Diseño y generación de una serie de clases encaminadas a enlazar los módulos funcionales entre sí y con la capa de software que interactúa directamente con el robot (Naoqi). El sistema operativo Aperios [17], que contenía un núcleo privativo, empleado por la plataforma Aibo difiere sustancialmente de la distribución de Linux OpenNao embebida en los robots Nao. Por lo tanto, han sido necesarios un gran número de trabajos de adaptación a este nivel de la arquitectura software.

- Optimización de las tareas que se ejecutan de manera cíclica durante el funcionamiento del robot. Con el cambio de sistema operativo, ha sido necesaria una mayor atención a la duración de las mismas, ya que con el uso de Linux en lugar de Aperios se ha perdido eficacia en la sincronización y periodicidad de procesos, y estos efectos se amplifican con una carga computacional elevada.

3.1. Módulo de comunicaciones

El módulo de comunicaciones, o COMMS, es el encargado de definir los mensajes que los robots se van a intercambiar entre ellos y con el ChaosManager y de gestionar el envío y recepción de dichos paquetes.

Puesto que el sistema se asienta sobre una plataforma Linux, las comunicaciones se realizarán mediante sockets. Estos sockets pueden emplear los protocolos de comunicación TCP o UDP. Seleccionaremos uno u otro dependiendo del mensaje.

Los robots emplearán sockets UDP para todos aquellos mensajes que se intercambien entre ellos, mientras que basarán sus comunicaciones con el ChaosManager en sockets de tipo TCP.

Existen tres clases principales que dan forma a la implementación del módulo de comunicaciones: CommsModule, CommsManager y Message.

CommsModule

Esta clase define una interfaz que deben implementar todos aquellas clases que deseen ser capaces de enviar y recibir mensajes. Básicamente, incluye una definición de identificación de módulo de comunicaciones, una función para asignar el módulo de comunicación principal (CommsManager) y la definición del prototipo de la función de recepción de mensajes.

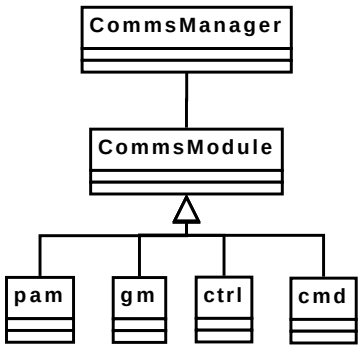

Figura 3.1: Asociaciones del sistema de comunicaciones

Para poder ser tenidas en cuenta como clases receptoras de mensajes, estas clases deben registrarse con anterioridad en la clase commsManager.

CommsManager

Es la clase principal en el proceso de comunicaciones. Una vez instanciada, crea dos hilos, uno para escuchar los mensajes UDP y otro para los TCP. Del mismo modo, atiende la peticiones de envío de mensajes que le llegan del resto de módulos.

Para establecer la comunicación TCP primero es necesario que un cliente (el Chaos-Manager) trate de conectarse, y posteriormente los mensajes se transmiten en formato flujo de datos. Es decir, ninguna de las dos partes sabe cuando empieza o termina un mensaje. Para ello se ayudan de los campos de la cabecera de los mismos, que indican el tamaño total del mensaje y también se basan en la propiedad de las comunicaciones TCP de no alterar el orden de los paquetes. Por lo tanto, para mensajes grandes o situaciones de saturación, es probable que sea necesario enfrentarse a situaciones de ensamblado de mensajes. Estas labores de ensamblado, se delegan a la clase Message, de la que heredan todos los mensajes y de la que hablaremos a continuación.

Message

Es la clase que define la forma en la que se van a almacenar los mensajes y añade cierta funcionalidad para su creación y para su reensamblado tras una recepción fragmentada. Todos los tipos de mensajes que el sistema emplea en sus comunicaciones serán clases derivadas de ésta.

Tipo	Módulo	Contenido
MsgBallBooking	ctrl	Intención de ir a por una pelota.
MsgBehaviour	ctrl	Estimaciones odométricas y perceptivas.
MsgDebugCtrl	ctrl	Solicitud de estado de depuración para el módulo ctrl.
MsgDebugGm	gm	Solicitud de estado de depuración para el módulo gm.
MsgFile	ctrl y pam	Fichero.
MsgGetFile	pam	Solicitud de fichero.
MsgImage	pam	Imagen de la cámara formateada (rgb, segmentada...).
MsgRunKick	ctrl	Nombre del comportamiento a ejecutar.
MsgTeleoperation	cmd	Órdenes de desplazamiento del robot o de su cabeza.
MsgTeleoperationEnable	cmd	Activación o desactivación del modo teleoperación.
MsgVideoConfig	pam	Solicitud de secuencia de imágenes formateadas.
MsgGetImage	pam	Solicitud de una imagen formateada.
MsgGrid	gm	Detalles del algoritmo de localización.

Tabla 3.1: Tipos de mensajes

Se pueden dar dos procesos diferentes de creación de un objeto derivado de la clase Mensaje. En recepción, creamos un mensaje a partir de un flujo de bytes, y en envío, creamos un mensaje a partir de un conjunto de campos.

De este modo, cuando un flujo de bytes es recibido, la clase CommsManager emplea

el método parseMessage de la clase Message, que es estático, para obtener un objeto de tipo Message que corresponda a dicho flujo de bytes. Para interpretar el flujo de bytes, la clase Message analiza los primeros bytes del flujo, que se corresponden con la cabecera, y a partir de ellos determina el tipo y tamaño del mensaje. Conocido el tipo, puede asignar a los campos de un mensaje de ese tipo los valores contenidos en el flujo de bytes. Gracias al conocimiento del tamaño del mensaje, podemos saber si aún quedan bytes por recibir para nuestro mensaje.

Tamaño (Bytes)	Campo	Descripción
4	MSGID	Identificador de tipo de mensaje
4	DESTINYMODULE	Subtipo de mensaje o módulo de destino
4	SOURCEID	Id de robot origen
4	DESTINYID	Id de robot destino
4	TEAMCOLOUR	Id de equipo de robots origen y destino
4	COMPRESSED	Indicador de compresión de los datos
4	LENGTHPAYLOAD	Tamaño de los datos
Variable	PAYLOAD	Datos del mensaje

Tabla 3.2: Formato de los mensajes

En caso de que así sea, la clase commsManager volverá a asignar al mensaje en cuestión el siguiente flujo de bytes, y desde la clase Message se volverá a analizar si el mensaje está completo. Una vez que el mensaje recibido esté completo, la clase commsManager desviará el mensaje a la clase registrada como receptora de ese tipo de mensajes. Que lo atenderá.

Por otro lado, puede ser que un flujo de datos contenga el final de un mensaje y el principio del siguiente, esta situación también es detectada por la clase Message durante el proceso de ensamblado de fragmentos. La clase commsManager interroga a Message sobre la existencia de bytes sobrantes en el último fragmento de un mensaje ensamblado, y en caso de que sea positiva, además de reenviar el mensaje completo a

su clase receptora también organiza la creación de un nuevo mensaje.

El proceso de envío de mensajes por parte del robot, es más sencillo. Simplemente se hace uso del constructor de la clase derivada de Message al que se le pasan todos los campos como parámetros. Posteriormente se emplea el método sendMessage para enviar dicho mensaje al exterior.

3.2. Adaptación al Middleware

La plataforma de desarrollo con la que trabajamos consiste en un robot humanoide NAO con sistema operativo Linux y middleware Naoqi. Esta capa de software llamada Naoqi y desarrollada por la empresa fabricante del robot (Aldebaran Robotics), está diseñada para hacer más cómoda la labor del programador a la hora de controlar el robot. Sin embargo, las peculiaridades de su diseño nos obligan a plantearnos cuidadosamente la forma en la que nuestro sistema va a descansar sobre este middleware. Lo que para el público general podría resultar cómodo, para los programadores de los equipos de robots participantes en la RoboCup puede resultar inconveniente, ya que Naoqi sacrifica en ocasiones eficiencia en pro de la comodidad y esto no es aceptable cuando se trata de sacar el máximo partido al hardware disponible.

3.2.1. Naoqi

Naoqi es un entorno distribuido que puede contar con tantos ejecutables binarios como el usuario desee, dependiendo de la arquitectura que se elija. Antes de cargar Naoqi y que éste lance nuestro módulo (ChaosModule) el sistema pasa por varias fases del proceso de carga. En primer lugar se lleva a cabo la inicialización de la placa base, posteriormente se da paso a GruB para que cargue kboot, que es una imagen mínima de linux que nos permite salvar la falta de soporte de la BIOS para el controlador NAND.

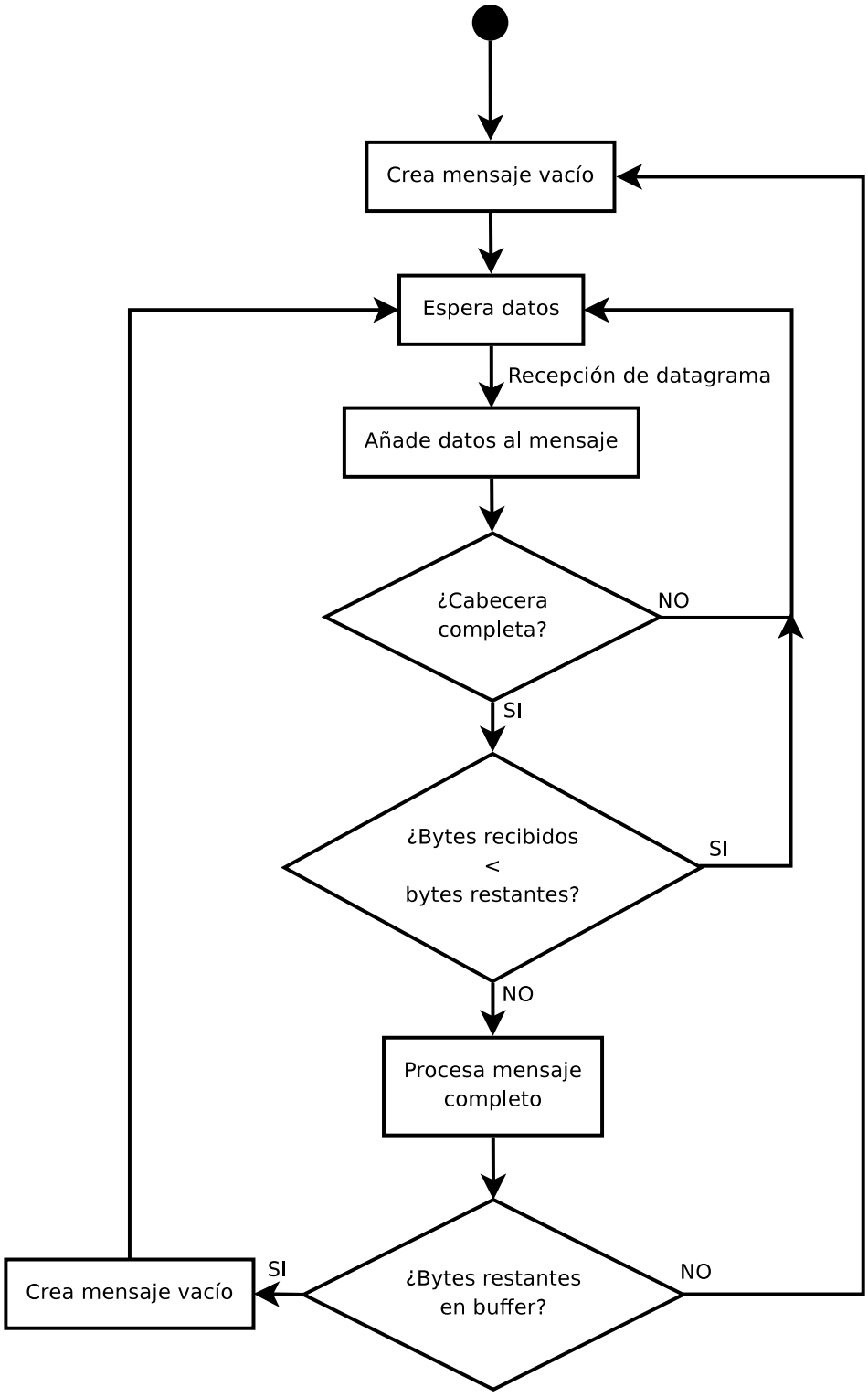

Figura 3.2: Recepción de mensajes vía TCP

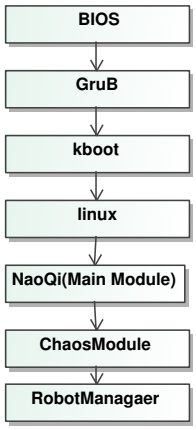

Figura 3.3: Distintas fases del proceso de inicialización

Finalmente se carga el sistema definitivo y sobre el la capa de middleware Naoqi, que automáticamente lanza nuestro módulo ChaosModule.

Naoqi permite estructurar nuestros programas en módulos y brokers. La implementación estándar de Naoqi incluye únicamente un broker principal y varios módulos para gobernar diferentes sensores y actuadores del robot. A continuación veremos en que consisten estas unidades arquitectónicas de Naoqi.

Módulos y brokers

Los brokers son ejecutables con un servidor que escucha una IP determinada. Todos los brokers están conectados con otros brokers excepto el broker principal.

Los módulos son clases derivadas de ALModule que pueden ser cargados por un broker.

Tanto los brokers como los módulos los podemos enlazar de forma local o remota. Cuando los enlazamos de forma remota se comunican con el resto del sistema mediante SOAP, lo cual hace que las tareas de comunicación sean bastante pesadas. Como beneficio, podremos ser capaces de conectarnos al sistema desde cualquier otra máquina

siempre que tengan conexión de red entre ellas.

La ventaja de enlazar los módulos o brokers de forma local reside en la mayor eficiencia lograda, al prescindir de las comunicaciones por SOAP.

A su vez, si usamos brokers en vez de módulos, obtendremos un mayor gasto computacional. La ventaja radica en que si un código que pertenece a un broker queda bloqueado, ese broker caerá con él pero no el resto de brokers. Si por el contrario, nuestro código esta asociado al broker principal, toda la capa de Naoqi caerá (probablemente también lo hará el robot) y será necesario reiniciarla.

Módulos del broker principal

La versión estándar de Naoqi incluye algunos módulos por defecto para interaccionar con el robot. Tres de estos módulos son necesarios para el funcionamiento del sistema (ALLogger, ALMemory y ALPreferences) mientras que el resto sirven para facilitar la tarea del manejo del robot. A continuación haremos una breve descripción de dichos módulos.

- ALLogger: permite usar el registro de Naoqi para mostrar información, advertencias o errores.

- ALMemory: se usa para compartir variables dentro de un mismo módulo o entre módulos distintos. Cualquier módulo puede añadir o consultar variables de ALMemory. Este módulo es Thread-safe, los módulos que hacen uso de ALMemory no se tienen que preocupar de emplear mutexs. Los datos que ingresamos en ALMemory son de tipo ALValue, que es un tipo genérico capaz de almacenar bool, int, float, double, string y vectores de ALValue. Para consultar el valor de una variable registrada en ALMemory podemos consultar directamente su valor o pedir a ALMemory que nos notifique cuando el valor de la misma cambie (se crea un

nuevo hilo). Entre los valores almacenados en ALMemory podemos encontrar los valores que toman todos los sensores, actuadores y códigos de error de las placas distribuidas por el robot.

- ALPreferences: podemos observar los valores asignados a algunos parámetros de configuración del sistema. Estos valores se toman de los ficheros de configuración contenidos en el directorio Naoqi
  Preferences
  cuando arranca el sistema.

- ALAudioPlayer: permite reproducir pistas de audio en mp3 o wav por los altavoces situados a los dos lados de la cabeza.

- ALLeds: Permite encender, apagar o fijar un nivel de intensidad de diferentes leds. Una de las opciones disponibles consiste en acceder a los leds mediante agrupaciones. Algunas de ellas ya vienen creadas para nosotros (todos los leds del pie derecho), pero podemos crear nuevos grupos si lo deseamos.

- ALMotion: es el módulo encargado de las funciones de locomoción. Su función principal consiste en fijar los valores de los ángulos de apertura de las articulaciones, así como consultar dichos ángulos. A la hora de establecer un valor para un ángulo, podemos indicar la velocidad a la que la articulación debe moverse para alcanzar dicho valor o asignar un tiempo de duración para la transición. Otras funciones más complejas que podemos expresar a través de este módulo son la modificación de la orientación del torso o de la posición del centro de masas. Las funciones más potentes, sin embargo, son aquellas que directamente hacen al robot andar hacia delante, lateralmente o girar sobre sí mismo. A estas funciones podemos configurarle diferentes parámetros como la distancia a recorrer, la lon-

gitud máxima del paso a efectuar, la altura máxima a la que levantar el pie, el máximo desplazamiento lateral, el máximo ángulo de giro y los desplazamientos en x y en z del zmp.

- ALTextToSpeech: Este módulo permite al robot hablar leyendo un texto. Envía comandos al motor sintetizador de voz y permite configurar la voz. Podemos cambiar el lenguaje o generar efecto de doble voz (habitualmente usado por los robots cinematográficos), en definitiva modificar el estilo de la voz para hacerla de nuestro agrado.

- ALVision: Este módulo es el encargado de comunicarse con la cámara situada en la cabeza del robot. Podemos usarlo para cambiar diferentes parámetros de la cámara como la ganancia, el balance de blanco o corregir ciertas aberraciones de las lentes.

- ALUltraSound: permite modificar la frecuencia con la que los sensores de ultrasonidos sondean el medio. Este parámetro debe ser superior a 240ms. Para obtener los valores leídos por los sensores es preciso recurrir al módulo ALMemory.

- ALWatchDog:

  - Monitoriza el nivel de carga de la batería, cambia el color del LED del pecho en consonancia.
  - Adopta la posición de reposo si el nivel de la batería es demasiado bajo.
  - Reproduce la palabra “Heat!” si el cuerpo del robot se sobrecalienta.
  - Gestiona las pulsaciones del botón del pecho de la siguiente manera:
    - Una pulsación: Nao saluda, se identifica y dicta su IP.

    - Dos pulsaciones: Desactiva las ganancias de todos los motores suavemente.
    - Tres pulsaciones: Apaga de forma ordenada todo el sistema.

- DCM: Este módulo está arquitectónicamente por debajo de todos los anteriores. Es decir, todos los anteriores hacen uso de él. Nos permite establecer una comunicación con el robot a más bajo nivel. Por ejemplo, si queremos mover una articulación, en lugar de emplear el módulo de locomoción podemos enviar al DCM un comando indicando que queremos establecer un valor determinado para el actuador correspondiente a la articulación que queramos mover, y que queremos que esta orden se haga efectiva en un instante determinado referido al reloj del sistema. El DCM, por tanto, nos permite tener un mayor control sobre el hardware, pero a cambio, la interfaz resulta más compleja.

3.2.2. Adaptación

En nuestros proyectos, optamos por un diseño basado en dos configuraciones diferentes del sistema. Ambas parten de una clase principal RobotManager que obtiene instancias de todos los proxies de los módulos que se van a usar (por ejemplo DCM, ALMemory, ALMotion, ALWatchDog ...).

La primera de las configuraciones, diseñada para el desarrollo cotidiano y para las versiones de competición consiste en compilar todo el sistema como una librería dinámica que representa un módulo asociado al broker principal de Naoqi y cargado automáticamente en la fase de arranque. Este módulo, a su vez, instancia la clase RobotManager, que se conecta a todos los módulos que necesite de forma local. De manera automática, cuando Naoqi arranca, se carga nuestro módulo y se llama a la función StartPlaying, que coloca al robot en un estado de preparación para jugar (cuerpo erguido y cabeza

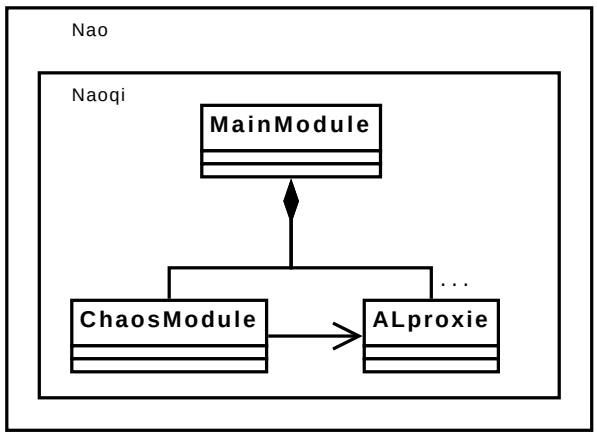

Figura 3.4: Configuración de máximo rendimiento

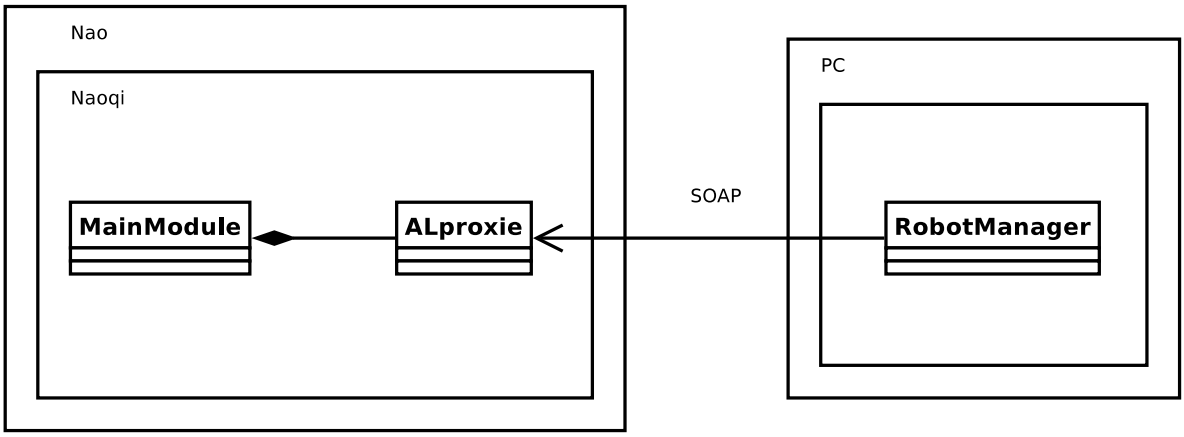

Figura 3.5: Configuración para depuración remota

en busca de la pelota) a la espera que una señal, ya sea telemática o bien de presión en el botón del torso, para comenzar a jugar.

La segunda de las configuraciones está pensada para la depuración exhaustiva. Consiste en compilar el sistema como un ejecutable que instancie la clase RobotManager y que ésta se conecte a los módulos que necesita de forma remota (a través de SOAP). Esta configuración no es apta para versiones de juego, ya que las comunicaciones por SOAP son muy costosas y hacen impracticable la comunicación fluida entre módulos necesaria para el juego. Sin embargo, esta configuración nos permite ejecutar todo nuestro código en otra máquina, facilitándonos las labores de depuración. Del mismo modo,

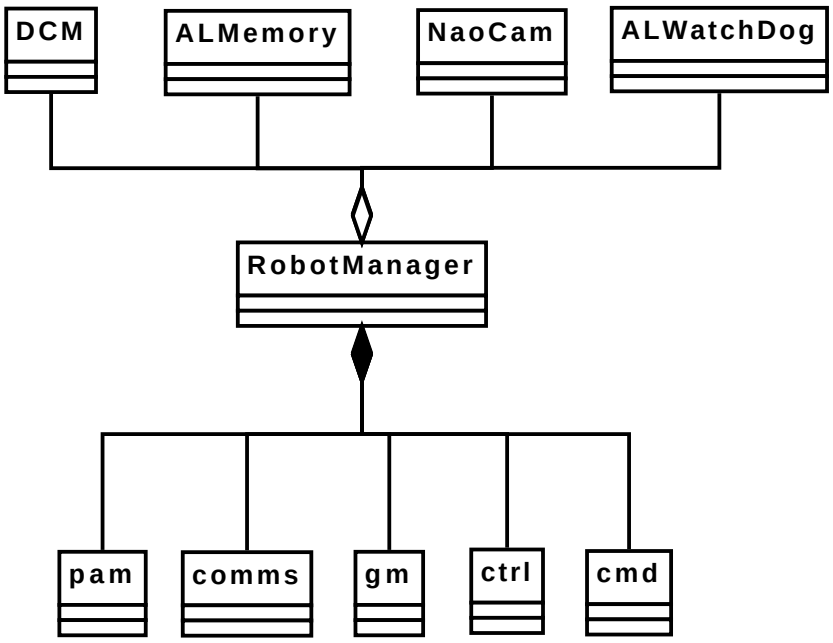

Figura 3.6: RobotManager adapta los módulos funcionales a la arquitectura Naoqi

si nuestro programa se cae, no arrastra consigo los demás módulos de Naoqi, lo cual puede resultar conveniente.

3.3. Optimización de la ejecución

El código con el que trabajamos en el equipo TeamChaos está basado en una versión redactada para el robot Aibo (de forma canina) y adaptada para el robot NAO (humanoide). Si bien, la forma de los robots supone un cambio importante, ya que hay que desechar todo el trabajo de locomoción, existen otros cambios igualmente profundos que no se aprecian a simple vista.

Mientras el robot Aibo de "Sony" empleaba un sistema operativo privativo y específico para dispositivos móviles (Aperios), la plataforma NAO, de "Aldebaran Robotics" emplea un sistema operativo Linux, mucho más genérico. La adaptación de este sistema operativo específico a otro más genérico y menos optimizado para requerimientos

de tiempo real supone un gran cambio y aún quedan retos pendientes. Algunos de los cuales describimos a continuación.

3.3.1. Optimización de la carga del procesador

Nuestro objetivo será procesar la mayor cantidad posible de datos provenientes de los sensores del robot (especialmente las imágenes de la cámara) y generar las decisiones con la mayor calidad posible. Sin embargo, la capacidad del procesador que empleamos es limitada, y además, nuestro programa no es el único que se ejecuta, sino que debe compartir la capacidad de procesado con el resto del sistema operativo y con los demás procesos del middleware que empleamos.

Es importante, por lo tanto, que mantengamos cierto margen de capacidad del procesador libre, ya que por ahora no somos capaces de controlar algunas de las operaciones de segundo plano del middelware Naoqi, y esto puede incurrir en la saturación del procesador durante un cierto período de tiempo.

Las consecuencias de esta saturación del procesador pueden ser desastrosas. Una de las consecuencias de esta excesiva carga del procesador es la aparición de temblores en los movimientos del robot. Puesto que el módulo de locomoción está preparado para generar las secuencias de movimientos de forma periódica, si este período se alarga, los movimientos del robot no se corresponden con los cálculos realizados, y es muy probable que se desestabilice si realiza alguna tarea de desplazamiento.

Por otro lado, los temblores del robot hacen aumentar el consumo de potencia de la batería. Cada vez que un motor inicia un movimiento se produce un pico elevado de consumo de potencia, y en situación de temblor, los motores arrancan y paran con cada ciclo en lugar de rotar de forma fluida, más o menos constante. Como resultado, la duración de la batería decrece drásticamente.

Como consecuencia última y más grave, puesto que todas las articulaciones reciben sus ordenes de movimiento en el mismo instante, todos los motores generan un pico de consumo en el mismo instante, y esto puede desestabilizar el sistema eléctrico del robot. Si la batería no se encuentra muy cargada o si el pico de consumo es muy elevado, el procesador puede no recibir suficiente corriente para alimentarse, lo que provoca que el sistema operativo del robot se reinicie repentinamente, provocando el desfallecimiento instantáneo del robot.

Las estrategias que desde TeamChaos hemos diseñado para afrontar este problema son:

- Análisis del tiempo de cómputo de los diferentes algoritmos empleados en nuestro programa, para detectar zonas críticas y mejorar su eficiencia.
- Predicción y control de actividades ajenas a nuestro programa que puedan acaparar la capacidad de carga del procesador.
- Colaboración con la empresa desarrolladora del robot para la mejora del sistema eléctrico.

Puesto que el tiempo de cómputo varía con cada iteración, se estudiaron aquellas que habían requerido mayor cantidad de procesado, y se trabaja sobre ellas para reducir las cotas máximas de tiempo de cálculo por periodo. En concreto, en aquellas tareas que implicaban una interacción con el robot, se trató de interaccionar con el mismo al más bajo nivel posible, de modo que se evitasen retardos ocasionados por capas de software de alto nivel. También se redujo la resolución de las imágenes a procesar con el fin de ahorrar tiempo de cómputo en el procesado de imágenes, siempre manteniendo la calidad suficiente para detectar los objetos de interés (pelota y porterías).

Método	Duración (μs)	Descripción
cmd.updateSensors	278	Caputra de valores de los sensores del robot (posición de articulaciones, valores de las unidades inerciales,...)
pam.updateOdometry	325	Actualización de representación relativa del entorno de acuerdo a las estimaciones de desplazamiento realizadas.
pam.updateImage	21320	Actualización de representación relativa del entorno de acuerdo a la imagen captada por la cámara
gm.process	16148	Generación de mapa global del entorno a partir de las representaciones relativas del entorno.
ctrl.process	12148	Evaluación de la situación y toma de decisiones tácticas y estratégicas.
pam.setNeeded	126	Actualización de las necesidades perceptivas.
cmd.updateMotion	5864	Generación de secuencias de ángulos y envío a las articulaciones.

Tabla 3.3: Tiempo de cómputo medido para las principales tareas de ejecución periódica en el robot

Por otro lado, se redujo a la expresión mínima la capa de middleware que se ejecutaba en el robot. Es decir, se evitó cargar aquellos módulos que no fuesen imprescindibles, como por ejemplo aquellos que sirven para controlar los leds o el sintetizador de texto. Tras la labor de adaptación, los únicos módulos de Naoqi cargados son aquellos relacionados con el módulo principal (MainModule), es decir: ALLoger, ALMemory y ALPreferences y el módulo DCM, que nos permite interaccionar con el robot a bajo nivel y por tanto puede actuar de sustituto del resto de módulos.

Para abordar los problemas de colapso del sistema eléctrico, se mantuvo una serie de reuniones con el Alexandre Mazel, ingeniero encargado del desarrollo de Naoqi de la empresa Aldebaran Robotics, durante la celebración del campeonato de fútbol robótico RoboCup en Suzhou, China. Entre las medidas tomadas, se monitorizó la carga de

las baterías en los puntos de colapso, encontrando cierta relación entre la capacidad de carga de las baterías y la frecuencia de colapso. Sin embargo, la baja carga de la batería, por sí sola, no justifica la caída del sistema, y fue necesario tomar en consideración la combinación de este factor con los mencionados anteriormente.

3.3.2. Paralelización de tareas

Puesto que tenemos tareas que deben realizarse con precisión cada cierto tiempo (análisis de imágenes, generación de movimientos) y otras tareas que no requieren tanta urgencia (planificación de movimientos a medio y largo plazo), sería interesante contar con dos hilos diferentes uno corriendo en segundo plano respecto a otro que disfrutase de una temporización precisa. Sin embargo, esto no es posible, ya que el sistema operativo empleado en el robot, OpenNao, no implementa funcionalidades de tiempo real, y por lo tanto, es imposible asignar prioridades de forma precisa a los procesos que se ejecutan en el mismo.

La opción elegida para resolver este problema consiste en la realización de un planificador, de forma que reparta el tiempo entre las diferentes tareas de la forma que más nos convenga. Para ello hemos optado por una implementación en un único hilo. Para repartir el tiempo entre las tareas, hemos dividido cada una de ellas en varias subtareas y monitorizamos en cada ciclo el tiempo empleado por ellas. Durante cada ciclo, ejecutamos primero las tareas de máxima prioridad, y posteriormente y mientras aún quede tiempo libre en el ciclo, vamos ejecutando las subtareas de baja prioridad. Una vez que el tiempo consumido se acerque al período del ciclo de control, suspendemos las tareas de baja prioridad y esperamos el comienzo de las tareas de alta prioridad. Y una vez que éstas terminen, retomamos las tareas de baja prioridad desde el punto en que se interrumpieron.

Este sistema permite al robot, por ejemplo, andar y buscar objetos simultáneamente, siendo uno de los pocos equipos participantes en las competiciones de fútbol robótico con el hardware NAO capaces de hacerlo en el último campeonato mundial.

Asimismo, también podemos destacar que nuestro objetivo de aumentar la precisión de la duración del periodo del ciclo de control también fue conseguido, y se consiguió disminuir la varianza del mismo en dos órdenes de magnitud. El error relativo medido descendió del 25 % al 2 %.

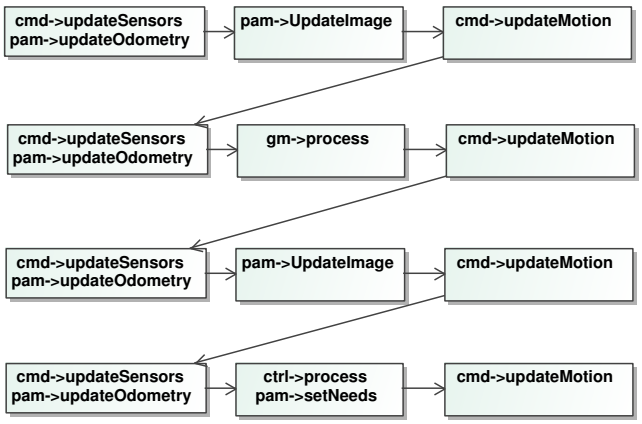

Figura 3.7: Ejemplo de planificación de tareas. El conjunto de acciones ejecutadas en cada ciclo varía.

Capítulo 4

Conclusión

El objetivo de este Trabajo Fin de Máster ha sido la puesta a punto de un robot humanoide NAO para la disputa de un campeonato de fútbol robótico. Gran parte de la dificultad de esta empresa radica en que el robot no es un producto comercial, sino que aún está en estado prototípico, y por lo tanto, exhibe numerosos problemas impropios de estas máquinas. Sin embargo, esto proporciona un interés mayor al proyecto, ya que algunas de sus aportaciones servirán para ayudar a la empresa fabricante del robot a lograr un mejor resultado final, y a cambio, nuestro grupo de investigación obtiene una plataforma de experimentación a un precio reducido.

Conseguir, en menos de un año, que el robot NAO fuese capaz de desempeñar el papel de un futbolista sería algo utópico si no hubiésemos contado con la experiencia del trabajo realizado durante años para los robots caninos AIBO. Hemos partido, por tanto, de la base del sistema desarrollado para los robots AIBO y a partir de él hemos creado una versión utilizable por el robot NAO. Del sistema diseñado para los robots caninos hemos tomado todos aquellos módulos software que se han podido reutilizar, esto es, los módulos de comportamiento, percepción y localización.

Sin embargo, debido a las diferencias mecánicas (especialmente la forma de los robots) y de sistema operativo entre los robots NAO y AIBO (con sistemas operativos

Linux y OPEN-R respectivamente), ha sido necesaria la creación de un nuevo módulo de locomoción, de un módulo de comunicaciones y de una adaptación al nuevo sistema operativo. Este trabajo en concreto, abarca la creación de un módulo de comunicaciones y los trabajos de adaptación al sistema operativo.

Si bien el sistema implementado en el robot AIBO era capaz e comunicarse, estas tareas estaban distribuidas por el resto de módulos, empleaban elementos propios de el sistema operativo OPEN-R y eran poco efectivas en condiciones de saturación. El nuevo módulo de comunicaciones tiene en cuenta el tipo de comunicación (inter-robot o robot-herramienta externa) y en función de ello selecciona un protocolo adecuado (UDP o TCP). Del mismo modo, se instaura un formato de mensaje común y se reduce el número de puertos empleados para la comunicación, entre otras características.

Otra de las vertientes de trabajo ha sido la adaptación al nuevo sistema operativo. El cambio de sistema operativo ha supuesto un gran tema de investigación y trabajo. Funcionalidades que podrían suponerse aseguradas como la precisión suficiente en el tiempo de espera para crear ciclos periódicos no están disponibles en la distribución de Linux implementada en el robot y este hecho supone un gran reto para el equipo de desarrolladores.

Por un lado no podemos realizar modificaciones de envergadura en el sistema operativo, ya que para controlar el robot precisamos del funcionamiento de una capa de middleware desarrollada por la empresa fabricante del robot y de código propietario. Por otro lado, no podemos permitir que el ciclo de control del robot tenga un período irregular ya que en este caso los movimientos de las articulaciones son discontinuos y el robot cae.

Por ello, ha sido vital encontrar una configuración que favoreciese la precisión de los temporizadores. Uno de los pilares de esta configuración ha sido la implementación en un único hilo de las tareas de control del robot. La presencia de varios hilos de ejecución

periódica provocaba una degeneración de la precisión del tiempo de espera solicitado al sistema operativo, para llevar a cabo las tareas de temporización. Además ha sido necesario encontrar una duración de período del ciclo de control adecuada que fuese lo suficientemente corta como para generar movimientos suaves y complejos en el robot y lo suficientemente larga para albergar dentro de un período las tareas que se repiten en todos los ciclos (percepción y locomoción).

Para dar con esta solución fueron necesarias cuantiosas medidas y pruebas, hasta que, finalmente, se optó por dividir las tareas de la forma indicada en el capítulo 3 y por emplear un ciclo de control de 40 ms.

Los resultados obtenidos fueron positivos, llegado el momento de disputar el campeonato de fútbol robótico celebrado en Suzhou (China), las comunicaciones funcionaron de manera eficaz y permitieron la calibración de robots y monitorización de su comportamiento. También es de destacar que el diseño modular de las comunicaciones favoreció la rápida creación de mensajes cuando estos fueron necesarios, como por ejemplo cuando se detectó que una versión del firmware de la cámara de algunos robots provocaba la inversión de los canales rojo y azul captados por la cámara.

De igual modo, resultó satisfactorio el rendimiento de la temporización de tareas, de acuerdo con las medidas de tiempos realizadas durante el campeonato. Aunque sin duda, la mejor manera de comprobar que el período del ciclo de control se ceñía a un escaso margen consistía en observar que el temblor de articulaciones de los robots durante el campeonato era despreciable y no los hacía caer. Como consecuencia también, la duración de las baterías era prolongada.

Finalizada la fase de puesta a punto inicial, surgen en este momento las diferentes alternativas de mejora del sistema, que son numerosas e interesantes. Una de estas vertientes podría consistir en el diseño de un sistema de percepción colectiva genérico. Mediante este sistema los robots podrían comunicarse entre sí para informar al resto del

equipo de sus percepciones y de esta manera completar su conocimiento del entorno.

Otro de los futuros temas de investigación podría ir encaminado a la implementación de algoritmos de aprendizaje en los robots. Especialmente interesante sería el aprendizaje aplicado a la locomoción, ya que lograría unos ajustes óptimos en los robots que no sería posible obtener de forma teórica debido a las imperfecciones de los mismos.

En definitiva, podemos afirmar que hemos dado un paso importante al conseguir que el robot humanoide Nao implemente una arquitectura de tipo "Thinking Capçompleta. Todos los módulos que la integran (percepción, localización, comportamiento, abstracción de hardware y comunicaciones) están funcionando, y los siguientes pasos que demos ya no irán tan encaminados a la solución de problemas de funcionamiento de los módulos, sino a la implementación de nuevas funcionalidades en los mismos.

Bibliografía

[1] Sony. Sony AIBO robots. http://www.aibo.com.

[2] Aldebaran Robotics. http://aldebaran-robotics.com.

[3] RoboCup. www.robocup.org.

[4] H. Martínez, D. Herrero, V. Matellán, F. Martín, C. Agüero, V. Gómez, M. Cazorla, M.I. Alfonso, A. Botía, B. Ivanov, A. Saffioti, K. LeBlanc. Robocup 2005, TeamChaos Documentation.

[5] H. Martínez-Barberá y A. Saffiotti. ThinkingCap-II Architecture.

[6] R. A. Brooks. A layered intelligent control system for a mobile robot. *IEEE T. Robotics and Automation*, 2:14-23, 1986.

[7] Active perception. *Preceedings of the IEEE*, 76(8):966-1005, 1998.

[8] A. Saffiotti and K. LeBlanc. Active perceptual anchoring of robot behavior in a dynamic environment. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3796-2802, San Francisco, USA, 2000.

[9] P. K. Sahoo, S. Soltani, A.K.C. Wong, and Y. C. Chen. A survey of thresholding techniques. *Computer Vision, Graphics, and Image Processing*, 41(2):233-260, 1988.

[10] J. Fan, D. Yau, A. Elmangermid, and W. Aref. Automatic image segmentation by integrating color-edge extraction and seeded region growing. *IEEE Trans. on Image Processing*, 10(10), 2001.

[11] R.Adams and L.Bischof. Seeded region growing. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16:647, 1994.

[12] W. Burgard, D. Fox, D. Henning, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proc. of the National Conference on Artificial Intelligence*, 1996.

[13] D. Herrero-Pérez, H. Martínez-Barberá, and A. Saffiotti. Fuzzy self-localization using natural features in the four-legged league. In *Proc. of the Int. RoboCup Symposium*, Lisbon, Portugal, 2004.

[14] P. Buschka, A. Saffiotti, y Z. Wasik. Fuzzy landmark-based localization for a legged robot. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, páginas 1205-1210, Takamatsu, Japón, 2000.

[15] A. Saffiotti and L.P. Wesley. Perception-based self-location using fuzzy locations. In *Proc. of the 1st Workshop on Reasoning wih Uncertainty in Robotics*, pages 368-385, Amsterdam, NL, 1996.

[16] V. Hugel, G. Amouroux, T. Costis, P. Bonnin, and P. Blazevic. Specifications and design of graphical interface for hierarchical finite state machines. In *Proc. of the Int. RoboCup Symposium*, Osaka, Japan, 2005.

[17] Sony. Sony AIBO SDE and Open-R SDK. http://openr.aibo.com.

[18] Carlos A. Zegarra, Humberto Martínez. Generación de trayectorias para la locomoción del robot humanoide Nao. *Trabajo Fin de Máster UMU*, 2008.