ARQUITECTURA DE CONTROL DISTRIBUIDA SOBRE BUS CAN PARA ROBOTS MÓVILES

Rafael Martínez*, Houcine Hassan, Ginés Benet, Alfons Crespo.
Departamento de Informática de Sistemas y Computadores (DISCA)
Universidad Politécnica de Valencia. Camino de Vera s/n - 46071 Valencia
*rmartin@dsic.upv.es, {husein, gbenet, acrespo }@disca.upv.es

Resumen

Este articulo describe el diseño de una arquitectura distribuida sobre bus CAN y su aplicación en el control de dos robots móviles. Se presenta el trabajo realizado en los laboratorios de informática industrial del DISCA consistentes en la implementación de los nodos microcontroladores i80c592 que permiten la gestión de los subsistemas de control y sensorización, en el desarrollo del sistema de comunicaciones basado en el bus de campo CAN y en el diseño del software y herramientas de desarrollo y depuración para el diseño de aplicaciones para sistemas robotizados.

Palabras Clave: Robots móviles, sistemas de tiempo real, sistemas distribuidos, buses de campo.

1 INTRODUCCIÓN

Este articulo describe el diseño de una arquitectura distribuida sobre el bus de campo estándar *Controller Area Network* (CAN) [3]. Este bus es muy adecuado para su utilización en sistemas de control de tiempo real ya que las características temporales de los mensajes es predecible [8], lo que permitirá posteriormente realizar un análisis exhaustivo de la carga del sistema. En este sentido, se detallan los aspectos de implementación del hardware de los nodos que constituyen la arquitectura así como se expone el sistema de comunicaciones que permite la interconexión de los diferentes subsistemas de control y sensorización de los vehículos móviles. Asimismo, se describen las herramientas de desarrollo y depuración y el software de apoyo implementado para facilitar el diseño de las aplicaciones. La utilidad de la arquitectura es mostrada mediante su implantación en dos robots industriales. El trabajo descrito aquí forma parte de un proyecto de investigación en el que ha sido construido un prototipo de robot industrial *Yair* [1].

A continuación, en el apartado 2 se describe la organización global de la arquitectura, entrando más en los detalles de implementación de los nodos microcontroladores que la componen. El apartado 3 expone las características del sistema de comunicaciones sobre bus CAN. Los entornos de desarrollo y soporte software utilizados para la implementación de las aplicaciones son mostrados en el apartado 4. La utilización de dos variantes de la arquitectura en dos robots móviles es mostrada en el apartado 5. Finalmente las conclusiones y trabajo futuro son resumidos en el punto 6.

2 ARQUITECTURA HARDWARE

Una visión global de la arquitectura global se puede apreciar en la Figura 1. Dicha arquitectura se compone de diferentes nodos interconectados mediante el bus CAN.

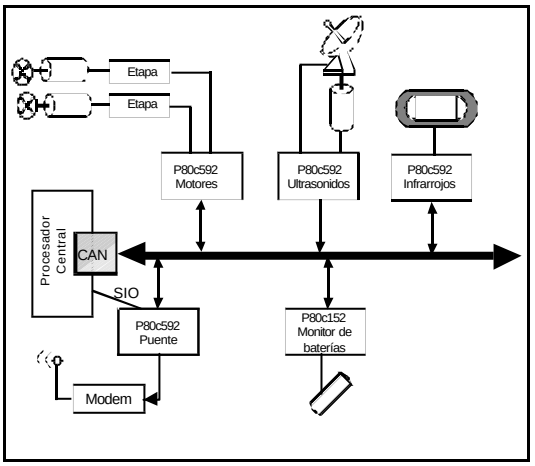

Figura 1. Arquitectura global.

La configuración de la arquitectura puede adaptarse a las necesidades de la aplicación o del robot en el que se vaya a implantar. Una configuración genérica podría estar compuesta por un procesador central dedicado a tareas de fusión de datos y de planificación de objetivos de alto nivel [2]. Además del procesador central existirán una serie de subsistemas de control (i.e. motores) y de sensorización (i.e. sonar) basados en el microcontrolador Intel 8XC592 [5]. La comunicación entre el procesador central y los distintos subsistemas

se establece a través del bus CAN. Asimismo, existe la posibilidad de comunicación mediante radiomodem con un PC-remoto.

2.1 DISEÑO DE LA PLACA CONTROLADORA BASADA EN EL i80C592

Los diferentes subsistemas de control y de sensorización del robot son gestionados por placas microcontroladoras basadas en el i8XC592 desarrolladas en los laboratorios del DISCA. El 8XC592 es un microcontrolador monopastilla de 8 bits basado en el estándar industrial de la arquitectura Intel 8051 que incluye un controlador CAN 80C200. Este microcontrolador incluye un buen número de nuevas prestaciones para satisfacer los requerimientos demandados por el mercado de la automoción y de las aplicaciones industriales. La placa diseñada puede ser apreciada en la Figura 2.

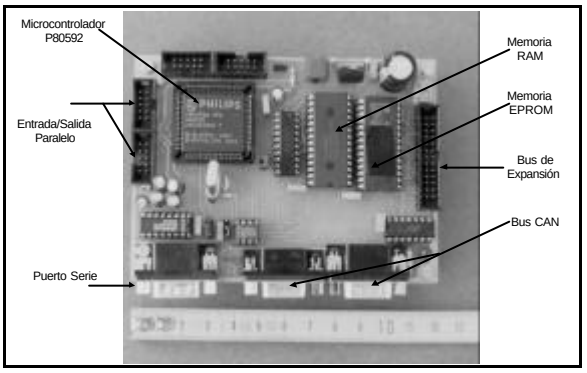

Figura 2: Fotografía de la placa de control.

Además del controlador CAN 80C200 encapsulado en el mismo microcontrolador 8XC592, éste incorpora 2 bloques de 256 bytes de memoria RAM, 1 temporizador, 1 conversor A/D, 2 salidas de modulación por ancho de pulso, un DMA y una UART. La frecuencia de reloj del microcontrolador es de hasta 16Mhz.

2.1.1 Diseño del sistema de memoria

En el diseño de esta placa se ha optado por una organización de memoria combinada. De este modo se posibilita fácilmente la coexistencia de memorias ROM y RAM y la ejecución de programas desde ambas. Esto proporciona bastante flexibilidad a la hora de la depuración de los programas.

De los 64Kbytes direccionables, se utiliza un encapsulado de EPROM de 32K que ocupan las direcciones 0000 -7FFFh, accesibles sólo para ciclos de captura de instrucción (FETCH) y un encapsulado RAM de 32K en las posiciones 8000 – FFFFh, accesible tanto para ciclos FETCH como para datos.

Llegada la necesidad se podría disponer de espacio de memoria para entrada/salida. Para tal fin, se dispone del extensor del bus del microcontrolador, fácilmente accesible.

2.1.2 Manejador y adaptadores del bus CAN

Este elemento adapta las señales del bus, emisiones en banda base, a las del microcontrolador, transmisión digital, y proporciona la corriente necesaria para la comunicación por el bus. Este driver tiene las siguientes características:

- Velocidades de transmisión de hasta 1 Mbit/s.
- Transmisión y recepción diferencial.
- Compatibilidad total con el estándar ISO/DIS 11898.
- Hasta 110 nodos conectados, según valor nominal.
- Modo de bajo consumo *Stand-By*.

En la Figura 3 se muestra el esquema de conexión al bus CAN de la placa.

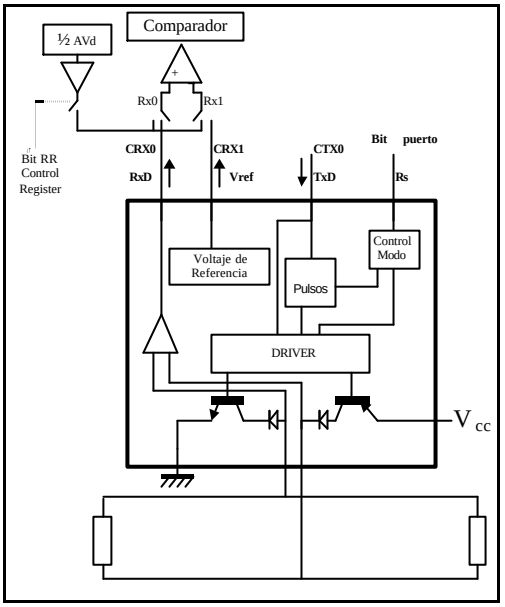

Figura 3. Esquema de conexión al bus CAN.

El modo de operación depende del voltaje introducido en la patilla RS:

- Modo operativo a máxima velocidad (VRs < 0.3 Voltios). Es el usado en nuestro caso, conectando la patilla directamente a masa. Los pulsos en el bus son directamente proporcionales a la corriente circulante.

- Modo control de pulso (0.3 V < VRs < 0.75 V). Se limita el alto de los pulsos en el bus, en el caso que el modo anterior ocasionase problemas de picos.

- Modo stand-by (VRs > 0.75 V). Se puede implementar en el controlador el Modo Wake-up a través de CAN, el driver en este modo detectara

actividad CAN, notificándoselo a la CPU mediante un cero en la línea RxD.

Por lo que respecta a los adaptadores, quedan dispuestos dos DB-9, uno macho y uno hembra, conectados entre sí, cuya finalidad es la conexión directa con el bus, como se aprecia en fotografía de la Figura 2. El bus se cierra en los extremos con un terminador de 75 ohmios, montado en un DB9.

3 SISTEMA DE COMUNICACIONES SOBRE BUS CAN

El controlador 80C200 permite gestionar de forma asíncrona los eventos del bus. Para ello dispone de 4 fuentes de interrupciones: transmisión, recepción, error y desbordamiento. Usando las interrupciones se descarga a la CPU liberando tiempo para otras aplicaciones en el sistema. Además, así se incluye la posibilidad de realizar transferencias por DMA desde los buffers del controlador a la memoria interna del microcontrolador.

Se ha desarrollado una librería de funciones que proporciona al programador la posibilidad de intercambiar información vía bus CAN entre los microcontroladores. El principio operativo es el mostrado en la Figura 4.

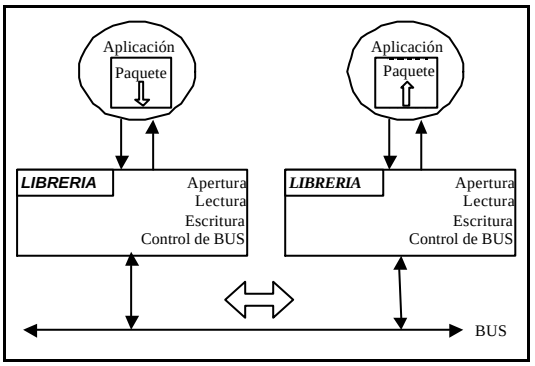

Figura 4. Interfaz de acceso a CAN.

La aplicación utiliza el interfaz sencillo de la librería en lenguaje C para enviar y recibir la información, y la librería gestiona todo el hardware de comunicación subyacente para hacer efectiva y eficiente la comunicación, pasando ésta de forma trasparente a la aplicación.

Los requerimientos de la librería se han enfocado desde dos perspectivas:

- Desde el Bus: se realiza una comunicación a 1 Mbit/seg, tramas de hasta 8 bytes de datos. Esto exige que las rutinas de recepción del bus sean muy rápidas, de lo contrario sufren desbordamientos, o se estará obligado a reducir la tasa real de transferencia.

- Desde la aplicación: La información tendrá asociada prioridad, pues en el sistema de control existirá información urgente que debe llegar dentro de unos márgenes temporales establecidos, e incluso si la comunicación de esa información no se realiza dentro del margen, esta puede ser no válida, por llegar en un momento en el que ya no sirve.

Además es necesario poder multiplexar el uso del bus CAN, de modo que se pueda ofrecer para un sistema multitarea la posibilidad de que cada tarea haga uso del bus.

Por lo tanto se requiere que las rutinas se ejecuten dentro de unos tiempos marcados por la velocidad del bus. Este tiempo debe permitir que un nodo pueda recibir una trama y prepararse para recibir la siguiente antes de que ésta llegase, aun en el caso en que se emitan de forma consecutiva. Estas exigencias han obligado a desarrollar partes del código directamente en lenguaje ensamblador, puesto que el compilador disponible no genera un código adecuado a nuestros objetivos. También el uso de memoria impone ciertas limitaciones, sobre todo en el acceso a memoria externa que es mucho mas lento. Esto se ha solventado reorganizando el uso de la memoria interna, liberando espacio para optimizar el uso del DMA, que sólo funciona con este tipo de memoria.

En la implementación de la librería se distinguen 2 partes, como se puede apreciar en la Figura 5: la capa superior y la inferior. La capa superior implementa el interfaz con la aplicación mediante primitivas sencillas en C, éstas son:

void ***Iniciar*** *(BYTE Vbit, BYTE Rango_entrada)*
Char ***Abrir*** *(BYTE rango, BYTE puerto, BYTE modo, BYTE modoop)*
Signed char ***Escribe*** *(BYTE desc, BYTE *buffer, BYTE nbytes)*
Signed char ***Lee*** *(BYTE desc, BYTE *buffer, BYTE nbytes)*

La aplicación abre un descriptor para el envío y decepción, asociado a una dirección CAN, en modo orientado a conexión o sin conexión, y con modo de operación bloqueante o no bloqueante. A través del descriptor y mediante la primitiva *Escribe*, se realiza el envío de paquetes de datos, con un tamaño limitado a 80 bytes. Será la capa inferior quien descompondrá el paquete en las tramas de datos CAN necesarias para el envío, siendo reconstruidas al otro lado en el paquete original. La aplicación del nodo remoto solo recibe información si ejecuta la primitiva *Lee*, de lo contrario la información quedará

almacenada en la capa inferior hasta que sea solicitada.

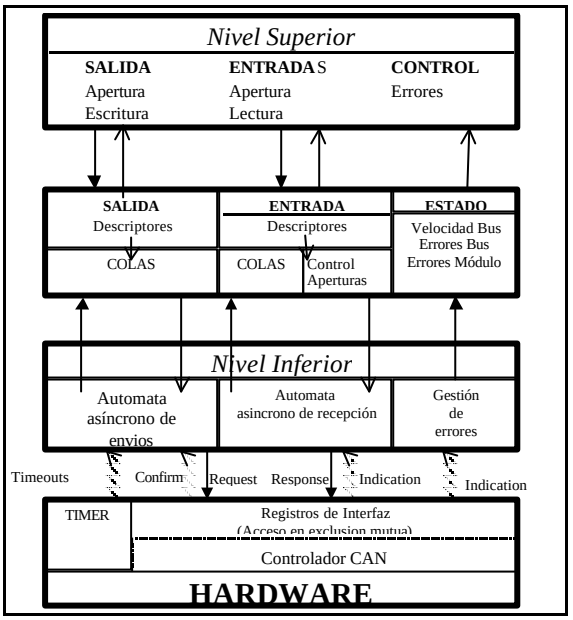

Figura 5. Estructura de la librería CAN.

Por lo que respecta a la gestión del hardware de control del bus, es desempeñada por la capa inferior de la librería, desarrollada en ensamblador para conseguir una implementación eficiente en el 80C592 de *Basic CAN*.Dado que el 80C952 es de 8 bits y los identificadores CAN de 11 bits, se produce una sobrecarga de computo a la hora del tratamiento de envíos y recepciones de trama para discriminación de las útiles de las no necesarias por el nodo. Además, 11 bits de identificación implica 2032 tipos de trama, lo cual incrementa las necesidades de memoria, escasa en nuestro microcontrolador, cuando se estima que un nodo de la red utilizara como limite superior entre 10 a 15 tipos de identificadores de trama distintos.

Estos contratiempos se solucionan introduciendo una descomposición jerárquica de las direcciones en dos campos, rango y puerto. De los 11 bits, los bits 0 a 6 conforman el primer campo, que presentan 128 posibles valores a los que se denomina puertos. El resto de bits, del 7 al 10, presenta 16 posibles valores a los que se denomina rangos.

La librería sólo usa uno de estos 16 rangos para utilizar el bus CAN, filtrando el resto de identificadores de trama mediante la mascara de filtrado por hardware del 80C200. El rango es elegido en la inicialización de la librería, por tanto, quedan disponibles para el nodo 128 identificadores, capacidad mucho mayor a las 10 a 15 necesarias a priori.

4 HERRAMIENTAS Y SOFTWARE PARA EL DESARROLLO DE LAS APLICACIONES

En este apartado se describen el entorno de operación de la plataforma móvil desde el PC remoto y el software de desarrollo de sistemas empotrados utilizado para los microcontroladores.

4.1 PC REMOTO

A través del interfaz gráfico de la Figura 6, el operario puede llevar a cabo el seguimiento y control del robot.

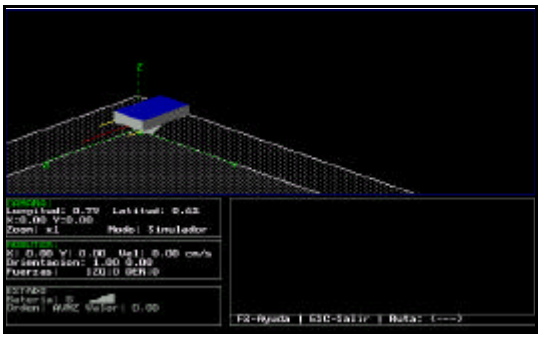

Figura 6. Interfaz de operación con el Robuter I.

El envío de las consignas se hace a través de bus CAN, RS-232 o radiomodem. Mediante un archivo de *script*, el operador trasfiere una lista de objetivos que se han de alcanzar y la velocidad a la que el robot debe circular. Asimismo, el interfaz recoge información de estado del robot consistente en la localización actual, velocidad y sentido del desplazamiento, fuerzas aplicadas a los motores, objetivo actual y estado de las baterías. Esta interfaz está disponible para Windows. Actualmente, se está en fase de culminación de la implementación de un entorno de desarrollo, depuración y generación de código más completo para Linux [6].

4.2 NÚCLEO BÁSICO DE LOS NODOS MICROCONTROLADORES

Para dotar al sistema desarrollado de una funcionalidad mínima, se ha incluido en la EPROM del sistema un pequeño núcleo de sistema operativo que proporciona las primitivas básicas para usar las tarjetas microcontroladoras. El núcleo se ubica a partir de la dirección 0000h de la memoria de programa y es el primer programa que se ejecuta tras un reinicio. Su función principal es inicializar la máquina para dejarla en un estado estable, concretamente:

- Inicializa todos los vectores de interrupción, banco de registros y pila.

- Inicializa el puerto serie SIO0.
- Establece una rutina para el puerto serie, y quedar a la espera de comandos a través de él.

Se dispone de un monitor residente que facilita la compilación cruzada de los programas, su instalación en los microcontroladores y su posterior depuración, de forma muy simple. Desde el PC, se puede acceder a la placa mediante la utilidad SERIN, para empotrar, ejecutar y depurar las aplicaciones.

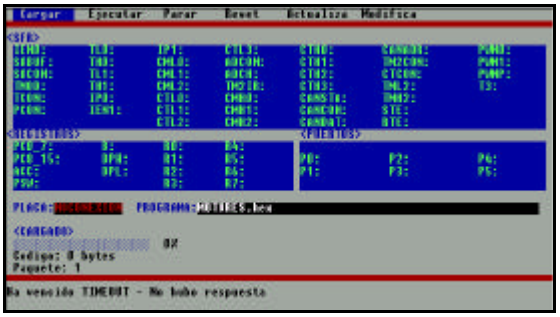

Figura 7. Pantalla del programa SERIN

En la Figura 7 se muestra una pantalla de dicho programa donde se pueden apreciar entre otras cosas los registros de función especial, incluidos los registros del controlador CAN (CANCON, CANDAT, CANADDR), los registros de propósito general y el contenido de los puertos de entrada/salida.

4.3 NÚCLEO EMPOTRADO DE TIEMPO REAL

La ejecución de los procesos que gestionan los subsistemas de control y sensorización es soportada por un núcleo de tiempo real específico, Tiny 51 de IAR systems [4] para los microcontroladores de la familia MCS-51. El uso de este núcleo permitirá en un futuro realizar un análisis temporal exhaustivo de la carga total ejecutada por el sistema distribuido. El núcleo es básicamente una librería de funciones para gestionar la ejecución de las tareas. Con el fin de poder gestionar el bus CAN, se ha modificado el código original de la versión 1.02 de este núcleo incluyendo un módulo específico.

4.3.1 Principios operativos

Básicamente Tiny51 esta formado por los siguientes elementos:

- Dispatcher.
- Scheduler.
- Funciones de usuario.

En la Figura 8 se muestra la organización interna e interfaz de Tiny51. Se utiliza un temporizador para organizar el tiempo que se cederá a cada una de las tareas. De modo que a intervalos regulares, denominados *quantums*, se activa el *dispatcher*.

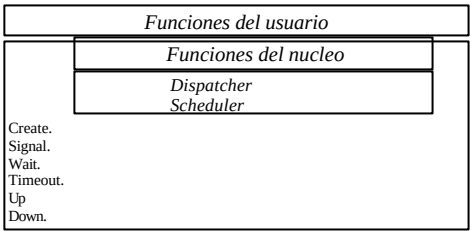

Figura 8. Organización interna de Tiny51.

El *Dispatcher* interrumpe a la tarea actual, cuando ésta ha consumido su quantum y procede a guardar su contexto e invocar al *Scheduler*. El *Scheduler* implementa la política Round-Robin, donde las tareas se organizan en una cola giratoria, y a cada una se le concede el mismo tiempo de procesador. Cuando expira este tiempo, se pasa a ejecutar a la siguiente tarea de la cola.

4.3.2 Creación de Tareas de usuario

Para que la tarea sea planificada por el núcleo debe ser creada. Para ello hay que realizar 3 pasos:

1. La tarea padre debe reservar un espacio para crear la pila de la hija, un descriptor de tarea y además definir un Identificador de Proceso (PID) para ella.
2. Invocar a Tiny592, mediante la función *create*, a la cual se le indicará la dirección de su descriptor. Esta función registra el nuevo descriptor enlazándolo en la cola circular del *Scheduler* y pasando la nueva tarea a estado “En creación”.
3. La tarea padre debe invocar la función principal de la tarea hija. La primera instrucción debe ser una llamada a la función *SetNewTask*. De este modo, se completa la creación de una pila para la nueva tarea y esta pasa ya a preparada.

4.3.3 Soporte del núcleo a bus CAN

Se ha incorporado al núcleo TINY51 un módulo que permite el control del bus CAN. Para ello, en los requerimientos del modulo de control de CAN, se incluye la gestión del bus, además de incluir la necesidad de multiplexar el bus de forma que distintas tareas pudieran usarlo en paralelo.

El sistema Tiny51, proporciona facilidades de gestión de recursos de forma exclusiva gracias a los semáforos y las señales. El control de CAN se lleva a cabo como si se tratase de un recurso compartido por el que compiten las distintas tareas. La petición del servicio detendrá al proceso invocante, pasando éste

a suspendido, y el scheduler elegirá otro proceso que pasará a ejecución mientras se complete la operación. Cuando esta se completa, el proceso suspendido pasaría de nuevo a preparado, y cuando pase a ejecución recibirá la respuesta del servicio.

5 APLICACIONES DE LA ARQUITECTURA

La arquitectura descrita ha sido probada satisfactoriamente en dos robots industriales, el Robuter I de Robosoft y el prototipo de robot industrial *Yair*, construido en los laboratorios del DISCA.

5.1 ROBOT *ROBUTER I*

El robot Robuter I [7] es una plataforma móvil provista de un tubo de motorización formado por una carcasa metálica para alojar los motores y realizar las conexiones correspondientes, de dos motores reductores de corriente contínua, cada uno de los cuales dispone de electrofreno y codificador óptico, también dispone de dos etapas de potencia integradas por dos servoamplificadores insertados en sus correspondientes placas de circuito impreso, que se encargan de adaptar las señales de control a los parámetros requeridos por los motores. La alimentación del sistema, está compuesta por cuatro baterías de 12 voltios cada una, conectadas en serie, de forma que proporcionan 48 voltios de corriente continua. El control de motores se realiza con un microcontrolador y dos tarjetas acondicionadoras que proporcionan las oportunas señales de control al sistema, al tiempo que recibe y adapta las señales de los codificadores. En la Figura 9 se aprecia la conexión de los diferentes subsistemas en el Robuter.

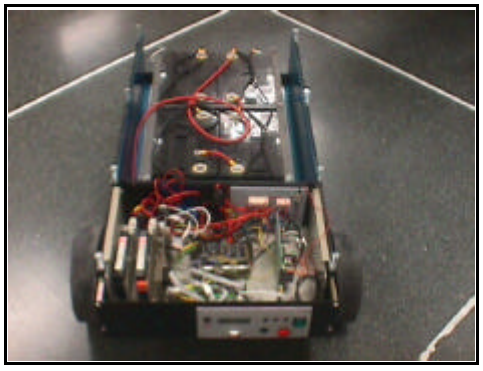

Figura 9. Robuter I.

El diagrama de bloques de la aplicación que ha sido probada sobre el Robuter I se puede apreciar en la Figura 10.

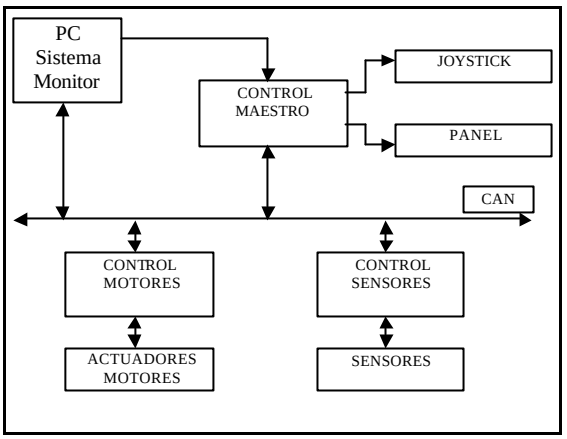

Figura 10: Elementos de la arquitectura del Robuter I

El PC sirve para enviar las consignas que debe ejecutar el robot y para llevar a cabo la monitorización de la ejecución. El microcontrolador maestro atiende al operario que puede enviar ordenes a través de un joystick y/o de un panel frontal. Otro microcontrolador sirve para el control de los motores y el tercer microcontrolador gestiona los sensores de ultrasonido ubicados alrededor del robot.

5.2 ROBOT *YAIR*

El robot *Yair* es un prototipo de robot industrial construido en los laboratorios del grupo de informática industrial del DISCA [1, 2]. El robot *Yair* se puede apreciar en la Figura 11.

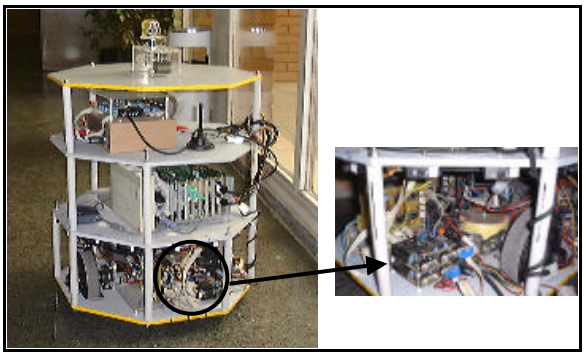

Figura 11. Robot *Yair* y detalle de los microcontroladores.

Los nodos microcontroladoras presentados en este articulo forman parte de la arquitectura de control basada en bus CAN que incorpora *Yair*. En dicho robot una de las unidades de microcontroladores se utiliza para realizar el control de los motores mediante controladores HCTL-1100. Un segundo microcontrolador se utiliza para la gestión de los sensores de infrarrorojos, y una variante de la placa diseñada soporta el control del subsistema de ultrasonidos.

La red de microcontroladores y el procesador central están interconectados mediante bus CAN. El procesador central de abordo se encarga de planificar las tareas del robot que consisten en ir a los objetivos de forma autónoma, teniendo la capacidad de sortear los obstáculos que se encuentran en su camino.

6 CONCLUSIONES

En el presente artículo se ha descrito una arquitectura distribuida para el control de robots móviles. En este sentido se ha detallado la implementación hardware de los nodos microcontroladores que gestionan los distintos subsistemas de control y sensorización. Asimismo, se han presentado las diferentes herramientas de desarrollo y soporte software utilizado en el diseño de las aplicaciones empotradas. Finalmente, se ha mostrado la utilidad de la arquitectura mediante la implantación de dos variantes de la misma en dos robots móviles.

Como trabajo futuro se pretende profundizar en el desarrollo del planificador de objetivos ubicado en el procesador central y de dar soporte a la ejecución de los procesos de este último mediante un núcleo de tiempo real *rt-linux*. Asimismo, se pretende integrar el entorno de desarrollo, depuración y generación de código para *rt-linux*, actualmente en fase de prueba, en el PC-remoto y realizar un análisis exhaustivo de la carga global ejecutada por el sistema.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto de investigación CICYT TAP98-0333-C03-02.

Referencias

[1] Benet, G., Blanes, F., Martínez, M., Simó, J., 1998. A Multisensor Robot Distributed Architecture. IFAC Conference INCOM'98. Metz-Nancy. June 1998.

[2] Blanes F. Benet G. Pérez P. Simó J. Map Building in an Autonomous Robot Using Infrared Sensors. IFAC Symposium on Intelligent Components and Instruments for Control Applications. Buenos Aires 2000.

[4] IAR systems embedded solutions 1999.

[3] ISO/IS 11898. Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High Speed Communication. Nov. 1993.

[6] Martínez R. Xsimu : entorno para el diseño de sistemas de control de tiempo real. Documento interno DISCA 2000.

[5] Philips Semiconductors, "P8XC592- 8-bit microcontroller with on-chip CAN", Data Sheet, January 1995.

[7] Robosoft (1997). "Refurbished Robuter I." Operating Manual.

[8] Tindell, K., Burns, A., "Guaranteeing Message Latencies on Controller Area Network", Proc. First International CAN Conference, Germany (Sept. 1994).