Revista Iberoamericana de Automática e Informática Industrial
CEA IFAC comité español de automática
RIAI
ISSN: 1697-7912. Vol. 3, Núm. 2, Abril 2006, pp. 71-78
http://riai.isa.upv.es

SCoCAN: UN PROTOCOLO DE COMUNICACIONES DE TIEMPO REAL PARA SISTEMAS EMPOTRADOS DISTRIBUIDOS. APLICACIÓN AL CONTROL DE ROBOTS

J.O. Coronel, F. Blanes, P. Pérez, M. Albero, G. Benet, J.E. Simó

Departamento de Informática de Sistemas y Computadores,
Universidad Politécnica de Valencia;
Camino de Vera, 14; 46022 Valencia, España.

Resumen: En este trabajo se describe el diseño y la implementación de una arquitectura distribuida para el control de robots móviles. En el desarrollo de esta arquitectura se han implementado tanto los nodos empotrados encargados del control del sistema, así como el protocolo de comunicaciones SCoCAN (Shared Channel on CAN). Este protocolo permite comunicaciones de tiempo real entre diferentes nodos distribuidos (sensores, actuadores y controladores). SCoCAN esta basado en un esquema de comunicaciones híbrido (Time Triggered-Event Triggered) que garantiza un jitter mínimo en el lazo sensor-control-actuador. Los nodos distribuidos utilizan RT-Linux como plataforma de gestión, control y planificación del sistema de tiempo real. *Copyright © 2006 CEA-IFAC*

Palabras Clave: Control distribuido, sistema empotrado, tiempo real, CAN, robótica.

1. INTRODUCCION[1]

Los sistemas distribuidos con sistemas inteligentes empotrados son en la actualidad cada vez más utilizados en arquitecturas complejas y/o espacialmente dispersas. Es común encontrarlos en controladores de vuelo, vehículos autónomos, robots, automóviles, circuitos de vigilancia, teleoperación, en control industrial, etc. en donde diversos sensores, actuadores y dispositivos de control se encuentran dispersos. Los sistemas distribuidos con múltiples nodos de procesamiento pueden organizarse según posean un control y procesamiento centralizado, un control centralizado y un procesamiento distribuido ó un control y procesamiento distribuido. Con estos sistemas se consigue no sólo una disminución del cableado sino también una reducción del ancho de banda requerido.

Los datos generados en esta arquitectura pueden dividirse en tres categorías: de tiempo crítico, periódico y con plazo largo de tiempo y grandes bloques de datos. La transmisión de estos datos por el mismo medio debe planificarse de tal forma que se cumplan los requerimientos temporales necesarios para el correcto funcionamiento de los procesos de control en el sistema, siendo deseable la eliminación de retardos en la comunicación (*jitter*) (Peréz, *et al*., 2003).

Por lo general, los sistemas distribuidos usan buses de campo, tales como CAN, Interbus-S, LON, Profibus, entre otros. Pero para un adecuado control e integración espacial y temporal del sistema se requiere una respuesta en tiempo real, por lo que es importante tener en cuenta tanto las características del nodo como las del bus de comunicaciones. Aunque para la interconexión de dispositivos hay una gran variedad de buses de tiempo real, CAN (Controler Area Network) es una de las soluciones

[1] Este trabajo cuenta con la financiación del proyecto CICYT DPI2002-04434-C04-03.

preferidas a la hora de comunicar sistemas empotrados distribuidos en espacios pequeños.

CAN (CiA, 1996) es un bus serie con características de tiempo real, funciona en ambientes hostiles, es fácilmente configurable y modificable, tiene detección de errores y implementa un acceso al bus no destructivo (CSMA/CD+CR). Este protocolo de comunicaciones implementa un acceso al medio basado en prioridades fijas las cuales se establecen dependiendo de las características temporales de los mensajes, tales como periodicidad, *deadline* y tiempo de ejecución en el peor de los casos (WCET). Cada mensaje tiene un identificador único que le indica la prioridad, a menor identificador, mayor prioridad. De esta forma, la transmisión de un mensaje puede ser retrasada si un mensaje de mayor prioridad compite al mismo tiempo por el acceso al bus ó si un mensaje ya está en proceso de transmisión. Por consiguiente, aún los mensajes con la mayor prioridad del sistema podrán presentar un cierto *jitter* y a medida que baja la prioridad de los mensajes aumentan las posibilidades de que éste se incremente. Y como es demostrado en (Peréz, *et al*., 2003), el impacto del *jitter* de comunicación en sistemas distribuidos de tiempo real es perjudicial para la correcta ejecución de lazos cerrados de control. Este retardo tiene una variabilidad que depende de las condiciones de error del canal así como de la carga del sistema (Rufino, *et al*, 1998; Tindell, *et al.,* 1995; Tindell,1994). Por lo que para tratar de disminuir al máximo las latencias en la comunicación, han sido propuestas varias extensiones de CAN, tales como FTTCAN y TTCAN.

Flexible Time Triggered CAN (FTTCAN) (Almeida, *et al.,* 2002) es una extensión de CAN basada en una planificación dinámica TDMA. FTTCAN tiene un ciclo elemental ó ciclo básico dividido en dos ventanas, una asíncrona: utilizada para transmitir mensajes con características temporales flexibles y cuyo acceso al bus esta determinado por el protocolo nativo de CAN; y otra ventana síncrona: en donde se transmiten mensajes de tiempo real estricto y cuyo acceso al bus es igualmente heredado de CAN pero con tráfico acotado y planificado dinámicamente por un nodo central maestro. Por consiguiente, FTTCAN puede resultar un protocolo bastante flexible pero complejo de implementar y con cierto jitter de comunicación.

Time Triggered CAN (TTCAN) (Fuhrer, *et al.,* 2000) es otra extensión de CAN basado en planificación estática TDMA. TTCAN utiliza un mensaje de referencia para indicar el comienzo de cada ciclo básico. Un ciclo básico estará dividido en diferentes tipos de ventanas: ventanas exclusivas, que son utilizadas para transmitir únicamente un mensaje específico, ventanas arbitradas, en donde los nodos compiten por el acceso al bus como en una comunicación normal de CAN, y ventanas libres, usadas para futuras ampliaciones. En este protocolo los ciclos básicos no son siempre iguales, el patrón completo de tráfico de TTCAN está compuesto por un número consecutivo de ciclos básicos que conforman un sistema de matriz o ciclo de matriz.

En este artículo se presenta la descripción de una arquitectura distribuida para el control de un robot móvil híbrido denominado YAIR, utilizando nodos inteligentes empotrados con RT-Linux (Yodaiken y Barabanov, 1996), además, analizando el comportamiento y los requerimientos del sistema se propone como protocolo de comunicación una extensión de CAN denominada Shared Channel on CAN (SCoCAN) (Coronel, *et al*, 2005), el cual esta basado en una planificación estática TDMA (*off-line*), pero con recuperación dinámica de ancho de banda (*on-line*).

El artículo esta organizado de la siguiente forma. En sección 2, se describe el protocolo de comunicaciones SCoCAN. La sección 3 presenta una descripción general del hardware del sistema distribuido, detallando los nodos inteligentes empotrados. La sección 4 hace referencia a las características temporales de los sensores y actuadores, las cuales deben ser tenidas en cuenta en el proceso de planificación. En la sección 5, se detallan las tareas y mensajes utilizados en la implementación. Finalmente, en la sección 6 se presentan las conclusiones y futuros trabajos.

2. DESCRIPCION DEL PROTÓCOLO DE COMUNICACIONES SCoCAN

SCoCAN (Shared Channels on CAN, Canales Compartidos sobre CAN) (Coronel, *et al*, 2005) es un protocolo de capa alta de CAN (sobre la capa de enlace de datos), cuyo objetivos son eliminar ó minimizar el *jitter* de comunicación, proporcionando determinismo sobre el bus, y al mismo tiempo, aprovechar al máximo la capacidad de transmisión del bus utilizando mecanismos de reciclaje de slots para la recuperación dinámica de ancho de banda.

SCoCAN sigue un esquema de comunicación híbrido, combinando tráfico Time Triggered (TT) y Event Triggered (ET), pero con separación temporal de ambos tipos de tráfico. Y esta separación es lograda por la asignación exclusiva de ancho de banda de bus para cada tipo de tráfico, la importancia de este aislamiento es presentado en (Almeida, *et al.,* 2002). Una implementación clásica hace uso de un ciclo básico de bus, dividido en slots de tiempo dedicados a tráfico TT y ET respectivamente, pero adicionalmente SCoCAN, permite que en instantes de tiempo determinados, los slots TT sean transformados en ET dinámicamente, consiguiendo de esta forma un aprovechamiento más eficiente del ancho de banda del bus.

Este protocolo esta basado en una planificación estática TDMA (Time Division Multiple Access) con

una ligera flexibilidad, en el que una tabla de tiempos es previamente definida y deberá ser guardada en cada uno de los nodos de la red con sus respectivas asignaciones. Pero adicionalmente, este protocolo soporta cambios de modo de operación, ya sean previamente ó dinámicamente definidos.

Se han definido dos tipos slots de tiempo:

- *Slot privado*: en donde solo uno de los nodos podrá transmitir y es usado para mensajes con características de tiempo real, mensajes de sincronización y mensajes de configuración.
- *Slot compartido*: en el que los nodos compiten por el bus utilizando el acceso CAN tradicional (CSMA/CDCA). Este *slot* es usado por mensajes de temporización no crítica y bloques de datos grandes.

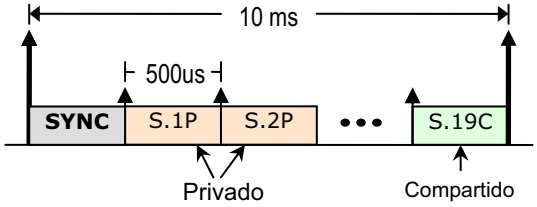


Figura 1. Temporización del ciclo básico de SCoCAN.

Se han especificado dos tipos de nodos:

- *Nodo tipo A:* placas inteligentes propias, tienen la capacidad de muestrear señales en el bus CAN y de esta forma pueden detectar inactividad vía hardware, lo que les permite transmitir cuando hay inactividad en los slots privados.
- *Nodos tipo B:* sin capacidad de muestreo de señales CAN, controlador CAN basado en tarjetas, y en caso de inactividad para que estos nodos también puedan transmitir habrá un nodo que transmitirá un mensaje con identificador igual a 1 que les indicará la condición.

Para YAIR se ha definido un ciclo básico de 10 ms, dividido en 20 slots de 500 microsegundos. El primer slot está reservado para el mensaje de sincronización. Siendo este mensaje el más importante de la red (identificador 0), pues es quien señala el comienzo de cada ciclo básico y con el que se sincronizan todos los nodos de la red. Para la generación de este mensaje se utiliza un nodo dedicado. En la figura 1 se aprecia la temporización del ciclo básico que se ha usado.

También se ha definido una tabla que determina el rango de identificadores que se puede usar dependiendo del tipo de mensaje. Esta relación de ID´s se puede ver en la tabla 1.

Tabla 1: Rango de identificadores en YAIR II.

ID	Tipo de Mensaje
0x000 : 0x00F	Sincronización
0x010 : 0x01F	Alarmas
0x020 : 0x0FF	Protocolo YCAL
0x100 : 0x2FF	Ventanas privadas
0x300 : 0x3FF	Ventanas compartidas (Alta Prioridad)
0x400 : 0x4F8	Protocolo YCAL
0x500 : 0x5FF	Ventanas compartidas (Baja prioridad)
0x600 : 0x6FF	Ventanas compartidas (Sistema de Ficheros)
0x700 : 0x779	Puentes y terminales
0x780 : 0xXXX	Configuración

3 HARDWARE DE LA ARQUITECTURA

3.1 Descripción general del sistema distribuido

El sistema implementado en el robot es un sistema distribuido con nodos inteligentes empotrados con el sistema operativo Linux y con una extensión de tiempo real (RT-LINUX) (Yodaiken y Barabanov, 1996). La configuración de la arquitectura adoptada es fácilmente adaptable y configurable a las actuales y futuras necesidades en el diseño del robot (Pérez, *et al.,* 2003; Simó, 1997) En la figura 2 se aprecia de forma global la arquitectura distribuida del robot.

La arquitectura es compuesta por un procesador central dedicado a tareas de recopilación de datos, fusión sensorial y planificación de objetivos de alto nivel, tales como evitación de obstáculos, seguimiento de trayectorias, levantamiento de mapas, etc.

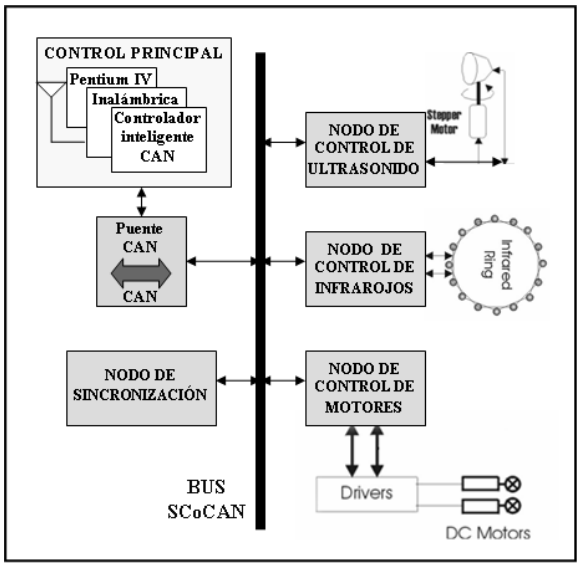


Figura 2. Esquema simplificado del sistema distribuido.

Para adaptar cualquier sistema que en principio utilice el protocolo de comunicaciones CAN nativo al protocolo SCoCAN, se ha diseñado un puente CAN-CAN, este puente se describirá con detalle en el siguiente apartado. Adicionalmente, se ha desarrollado un manejador (*driver*) de tiempo real que implementa el protocolo SCoCAN, el cual ofrece la alternativa de conectar directamente un PC con RT-Linux al bus de comunicaciones sin necesidad de utilizar el puente anteriormente mencionado.

Todos los nodos distribuidos inteligentes son modulares, los cuales se dividen en tres partes, un módulo con microprocesador 80C592 con controlador CAN integrado, un módulo con microprocesador i386EX con RT-Linux empotrado y un tercer módulo de acondicionamiento y digitalización que dependerá del sensor ó actuador a controlar.

3.2 Puente CAN-CAN

Este puente permite adaptar un PC con el protocolo de comunicaciones original de CAN al protocolo de comunicaciones SCoCAN. En el puente se define una tabla de transmisión global, de tal forma, que los datos provenientes del PC sean transmitidos a la red en el correspondiente *slot* de tiempo, evitando de esta manera, que el esquema de tiempos de SCoCAN se estropee. Los datos provenientes de la red son retransmitidos directamente hacia el PC.

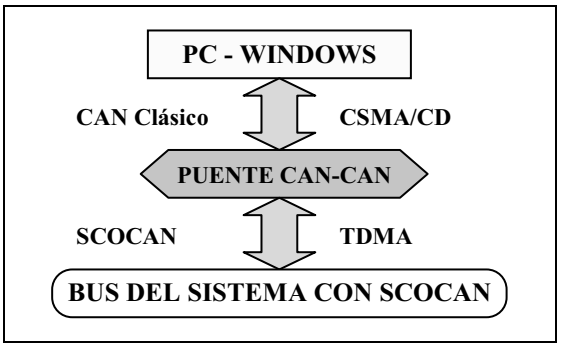


Figura 3. Esquema del puente de adaptación CAN-CAN.

Esta placa esta formada por un microcontrolador y dos controladores CAN MCP2510, uno para la red del sistema y otro para el control de mensajes con el PC. El esquema del puente de adaptación es mostrado en la figura 3.

3.3 Descripción de los nodos inteligentes empotrados

Para gestionar los diferentes subsistemas de sensores, actuadores y dispositivos de control del robot, así como para el control temporal del protocolo de comunicaciones SCoCAN, se han diseñado nodos modulares empotrados.

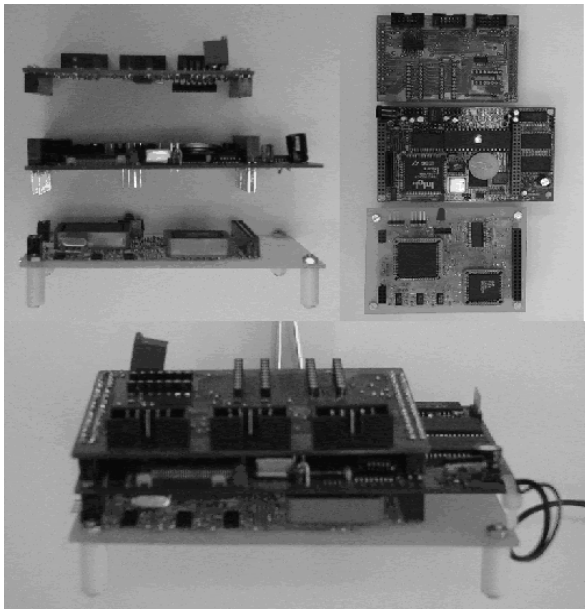


Figura 4. Ejemplo de uno de los nodos inteligentes empotrados.

La plataforma modular empotrada mostrada en la figura 4, esta dividida en tres placas modulares. Un primer módulo basado en el microprocesador i8XC592, el cual controla el protocolo de comunicaciones SCoCAN; el segundo módulo, es una plataforma empotrada con RT-Linux con un microprocesador i386EX, en donde se llevan a cabo los procesos de control propios del robot; y un tercer módulo de acondicionamiento y digitalización que permite adecuar los diferentes subsistemas de sensores y actuadotes.

El módulo inferior mostrado en la figura 4, tiene un microprocesador 80C592 philips de 8 bits, basado en la arquitectura estándar intel 8051 que incluye un controlador CAN PCA82C500 con DMA a la memoria RAM interna. Este chip tiene 2x256 bytes de RAM interna y usa una memoria RAM externa de 32 Kbytes, la frecuencia de reloj utilizada es de 16Mhz. A esta placa también se le ha incorporado una memoria RAM de doble puerto (DPRAM) de 4Kb, la cual se usa como interfaz de comunicación entre el 386 y el 592. En la figura 5 se muestra el esquema general de los nodos.

El módulo central en la figura 4 se corresponde con la unidad inteligente empotrada, la cual tiene como núcleo un procesador intel 386 con instrucciones extendidas y con 66Mhz de frecuencia de reloj. La arquitectura de esta placa es similar a la placa base de un PC, tiene un watchdog, 24 líneas de I/O, una unidad de chip-select, 1.5 MB de memoria NV-RAM, 1.5 MB de memoria flash, 512k de memoria EEPROM, un reloj de tiempo real, un conversor A/D de 12 bits de 1.25 Msps, 2 puertos serie con DMA y un tercero con SCC2690. Con esta placa se puede desarrollar fácilmente dispositivos de I/O porque el control de acceso lógico es programable. En esta

plataforma se ha empotrado el S.O. Linux con el kernel 2.1.4 y la extensión de tiempo real versión 3.0.

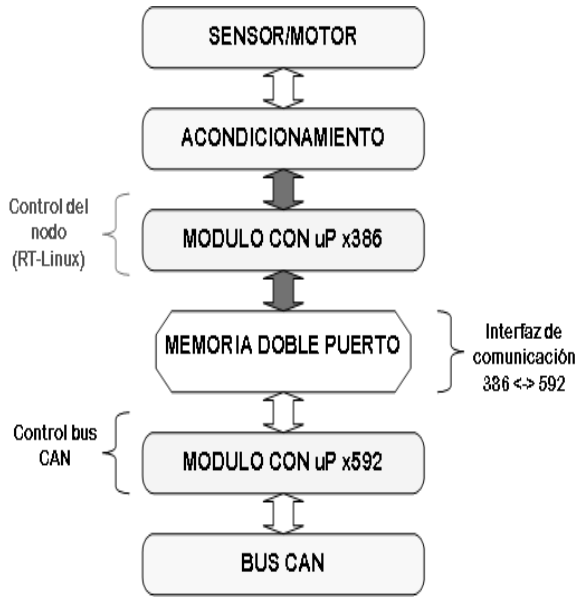


Figura 5. Esquema general de los nodos

La placa de acondicionamiento y digitalización adaptan las señales del 386 a el dispositivo (sensor, actuador) a interactuar. Esta placa varía de acuerdo con el sensor o actuador que se utilice.

En la figura 6 se muestran los módulos de infrarrojos y motores colocados en YAIR II. Los nodos de la arquitectura son lo bastante robustos para cumplir con los requerimientos necesarios del robot móvil.

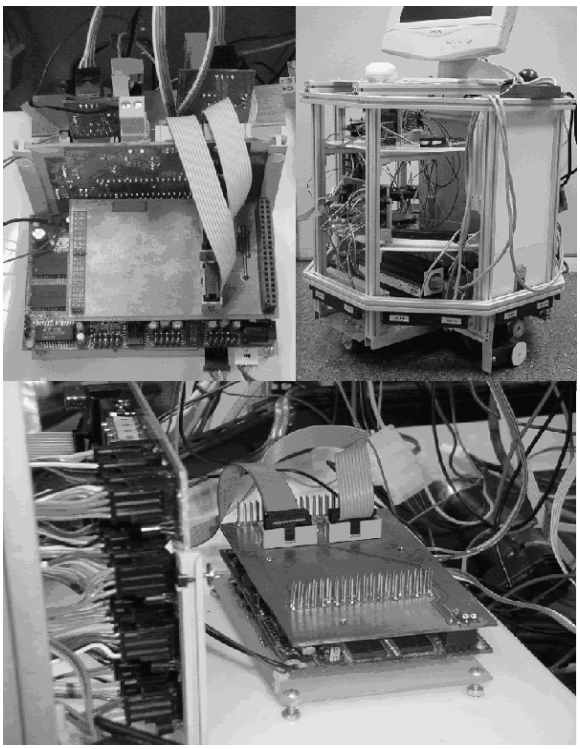


Figura 6. Nodo de infrarrojos y motores colocados en YAIR II.

4. CARACTERISTICAS TEMPORALES DE LOS SENSORES Y ACTUADORES

4.1 Sensores Infrarrojos

En YAIR II el anillo de infrarrojos esta compuesto por 16 pares de diodos uno para transmisión y otro de recepción, los cuales están distribuidos en un anillo octagonal, como se muestra en la figura 7.

Con cada uno de los sensores se toman dos medidas, una sin emisión, para determinar la influencia de la luz externa, y otra emitiendo, para determinar la cantidad de luz reflejada, basados en (Blanes, *et al.*, 2000). La primera medida tiene un tiempo de estabilización de 244 µs y la segunda medida 348 µs, por lo que se necesita un tiempo de 592 µs por sensor y 9,472ms para cubrir todo el anillo.

Las medidas se realizan con un conversor de 1.2 Msps de 12 bits, de modo que cada sensor generará un dato de 12 bits que se encapsulará en 2 bytes, por lo que se envía en un mismo mensaje los datos de 4 sensores.

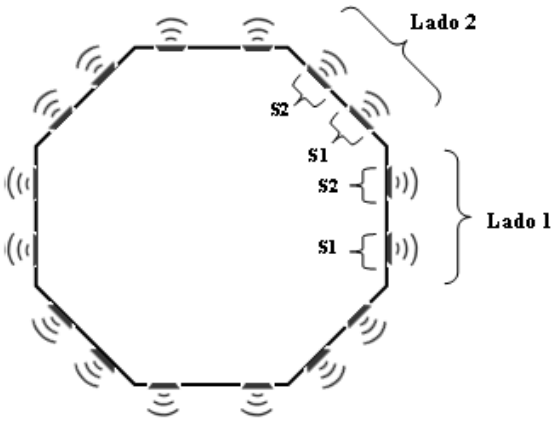


Figura 7: Anillo de infrarojos

4.2 Motores

En YAIR II uno de los módulos 386 se encarga de configurar y controlar los drivers de los motores. Se utilizan 2 drivers HCTL 1100 los cuales pueden configurarse en 4 modos de control: control de posición, control de velocidad proporcional, control trapezoidal y control de velocidad integral.

Las cuentas de enconder de cada motor tienen un tamaño de 4 bytes por lo que en un mismo mensaje se transmiten la de los dos, este mensaje es enviado cada 10 ms. El nodo de motores también gestiona los mensajes de control y configuración provenientes del PC, que pueden ser: de configuración de modo, encendido ó apagado de los motores y valores de aceleración y velocidad.

4.3 Ultrasonidos

El robot YAIR dispone de un sensor rotatorio con 2 transductores (Tx-Rx), mostrado en la figura 8, y tiene 200 pasos por vuelta (1.8º/paso) (ver figura 9). Para la adquisición de estos mensajes se utiliza un ADC de 10ksps de 12 bits y se toman 256 muestras por eco en cada posición, aunque solo se tienen en cuenta 8 bits.

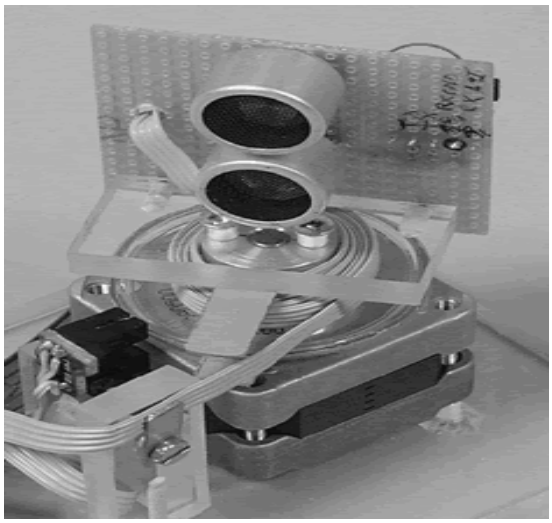


Figura 8. Sensor de ultrasonidos

Tomando 10000 muestras por segundo cada 25.6 ms generaría 256 bytes por posición y 51.2 kbytes por vuelta. Una carga como esta en el bus, utilizando CAN convencional, generaría jitters y retrasos en los datos periódicos. Con SCoCAN estos datos son enviados en ventanas compartidas, asegurando que no se afecte la periodicidad de los otros mensajes.

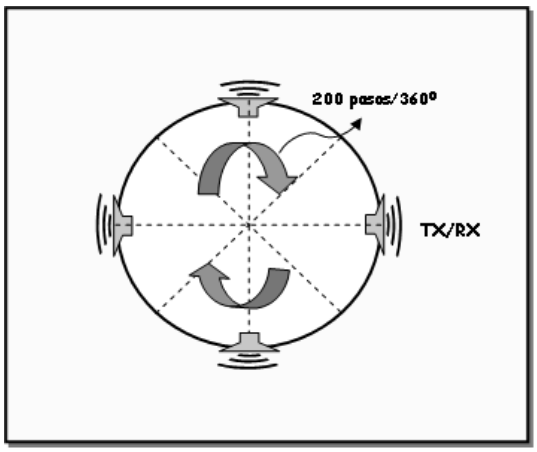


Figura 9. Esquema de ultrasonidos

5. DESCRIPCION DE TAREAS Y TIPOS DE MENSAJES DEFINIDOS EN SCoCAN

En la tabla 2 se presenta la relación de los identificadores y la temporización de los mensajes utilizados, necesarios para la planificación tanto de las tareas de tiempo real como la planificación del bus de comunicaciones.

5.1 Sensores Infrarojos

En estos mensajes se envía el nivel de luz reflejada menos la luz ambiente. Trama: esta definida en la figura 10, cuya nomenclatura se basa en la figura 7. Periodicidad: Los datos que conforman un mensaje se generan cada 2.5 milisegundos. Identificadores: se utilizan 4 ID's, del 0x100 al 0x103. Esta asignación se basa en la tabla 1. Tipo de slot: Privados.

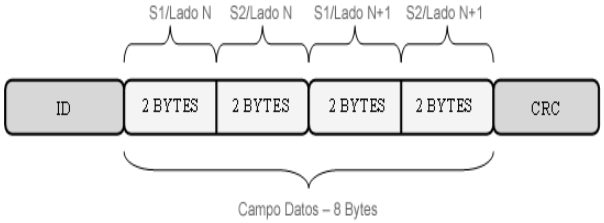


Figura 10. Definición de la trama del módulo de infrarrojos.

5.2 Motores

Para el módulo de motores se han definido 5 tipos de mensajes, cuyas características se describen a continuación.

Tabla 2. Relación de identificadores y temporización de los mensajes utilizados

Espec. / Msj. / Módulos	Datos (byte)	Período de generación información	Nº de mjs	Slots SCoCAN	Tipo de mensaje	ID
Infrarrojos	8	20 ms anillo	4	18,14,10,6	Privado	0x100 - 0x103
Motores-Alarmas	1	Por evento	1	-	Alarma	0x010
Motores-Modos	1	Generada x PC	1	-	Compartido	0x300
Motores-Acciones	1	Generada x PC	1	-	Compartido	0x301
Motores-Acelera.	4	Generada x PC	1	-	Compartido	0x302
Motores-Posición	8	10 ms	1	1	Privado	0x10A
Motores-Velocidad	4	Generada x PC	1	16	Privado	0x10B
Ultrasonidos	8	--	32	-	Compartido	0x310-0x318

Estado: Estos mensajes se subdividen en dos, alarmas y acciones.

Alarmas: Se utilizan para especificar errores, parada de motores inesperados, etc. Trama: ver figura 11. Periodicidad: Por evento. Identificador: 0x010. Esta asignación se basa en la tabla 1. Tipo de slot: Alarma.

Figura 11. Formato de la trama de alarma.

Acciones: Se utiliza para encender o apagar los motores. Trama: ver figura 12. Periodicidad: Generada por PC. Identificador: 0x301. Tipo de Slot: Compartido.

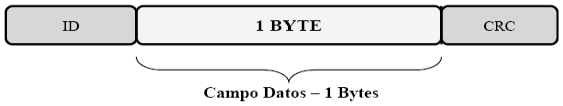


Figura 12. Formato de la trama de acciones.

Posición: En este mensajes se envían las cuentas de encoder. Trama: ver figura 13. Periodicidad: Cada 10 ms. Identificador: 0x10A. Tipo de Slot: Privado.

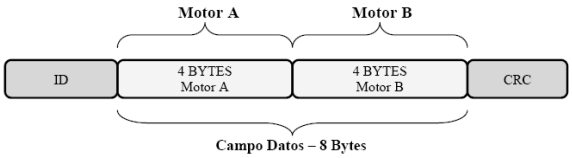


Figura 13. Formato de la trama de posición

Velocidad: Este mensaje es enviado por el PC para regular la velocidad de los motores. Trama: ver figura 14. Periodicidad: Generada por PC. Identificador: 0x10B. Tipo de slot: Privado.

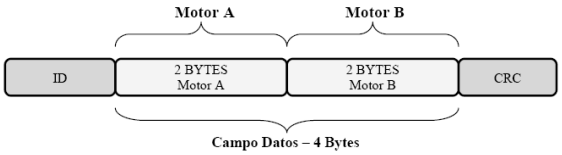
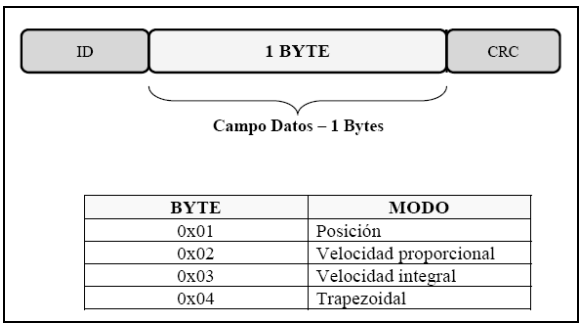


Figura 14. Formato de la trama de velocidad.

Modos: Este mensajes es usado para definir los modos de control en los drivers de motores. Trama: ver figura 15. Periodicidad: Generada por PC. Identificador: 0x300. Tipo de slot: Compartido.



BYTE	MODO
0x01	Posición
0x02	Velocidad proporcional
0x03	Velocidad integral
0x04	Trapezoidal

Figura 15. Formato de la trama de modos con la relación de los posibles modos.

Aceleración: Este mensaje es usado para cambiar la aceleración de los motores del robot. Trama: ver figura 16. Periodicidad: Generada por PC. Identificador: 0x302. Tipo de slot: Compartido.

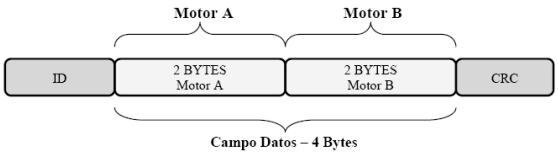


Figura 16. Formato de la trama de aceleración.

6. RESULTADOS

Un controlador digital PID ha sido diseñado para analizar los efectos de las latencias en el protocolo CAN tradicional en comparación con el protocolo SCoCAN. Este controlador ha sido simulado con el paquete Simulink de Matlab.

Se ha simulado un control de lazo cerrado en un escenario con múltiples cargas fijas, como es descrito ampliamente en (Peréz, *et al*., 2003). La carga esta compuesta por varios lazos de control, en el cual las acciones de control y las muestras de la señal de salida son enviadas en mensajes con diferentes prioridades. En el experimento se cambian las prioridades de los mensajes del lazo de control de un máximo (mensaje con identificador 0), a un mensaje con una latencia cercana a un milisegundo. El bus del sistema, en un escenario para el peor caso, ha obtenido un resultado positivo en el test de planificabilidad, que se calculó usando la ecuación presente en (Tindell,1994). Por lo que la planificación en el bus es factible, pero en la dinámica del control de lazo cerrado se ha producido overshoot. En la figura 17 se muestra una familia de curvas con los efectos producidos por la variabilidad del jitter. Para cada valor del eje Z (máximo jitter) una respuesta a la entrada escalón es obtenida, y a mayor jitter obtendremos mayor overshoot.

Comparando este comportamiento con la transmisión de éstos mensajes usando una aproximación time triggered (SCoCAN), en la que se tendrá jitter cero, encontraremos que todas las latencias serán las mismas para todos los lazos de control. Y de esta forma, siendo el sistema planificable, la dinámica del sistema tendrá la dinámica esperada.

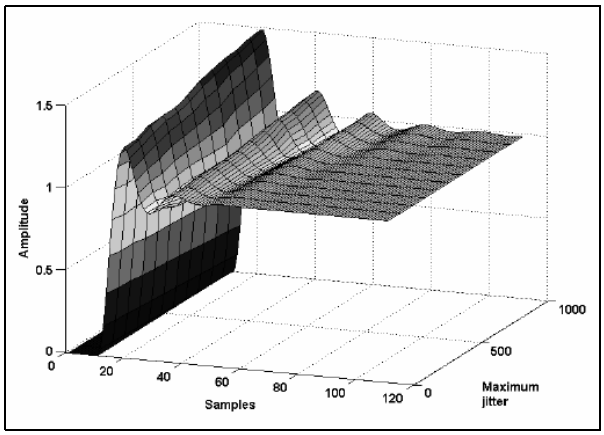


Figura 17. Respuesta del sistema a una entrada escalón vs. retardos

7. CONCLUSIONES

El artículo presentado aporta una solución global a los sistemas distribuidos de tiempo real que va desde el hardware específico, el protocolo de gestión del bus, hasta una adaptación del SO RT-Linux para sistemas empotrados basados en Intel-386.

El uso de extensiones CAN que utilicen una planificación TDMA (dinámica o estática), puede eliminar los efectos producidos por el *jitter* de comunicación en el protocolo CAN original, el cual depende de las condiciones de error del canal así como de la carga del sistema.

El protocolo SCoCAN es una buena opción en el momento de escoger un protocolo con planificación TDMA, comparado con otros protocolos como TTCAN ó FTTCAN, ya que además de ser sencillo de implementar, soporta tráfico de tiempo real con un jitter mínimo aceptable, no es necesario un planificador maestro como en FTTCAN y adicionalmente, la gestión del ancho banda es eficiente.

En la arquitectura del robot YAIR el bus de comunicaciones debe soportar grandes cargas de información, introducida principalmente por el nodo de ultrasonidos, siendo SCoCAN un protocolo adecuado para la gestión de grandes bloques de datos.

Los módulos inteligentes empotrados que controlan los sensores y actuadores del robot móvil han proporcionado resultados satisfactorios en cuanto a la gestión de carga, cumplimiento en las restricciones temporales de procesos y en el control temporal del bus.

Un futuro trabajo es la introducción de delegación de código en la arquitectura distribuida del robot utilizando el protocolo SCoCAN.

REFERENCIAS

Almeida, L., Pedreiras, P., Fonseca, J. A. - "The FTT-CAN Protocol: Why and How", IEEE Transactions on Industrial Electronics. Volume 49, Number 6. December. 2002.

Blanes F., Benet G., Pérez P., Simó J., (2000) “Map Building in an Autonomous Robot Using Infrared Sensors”. IFAC Symposium on Intelligent Components and Instruments for Control Applications. Buenos Aires.

CiA (CAN in Automation), (1996) “CAN Application Layer for Industrial Applications”, Documents No.: DS-201...DS-207, Version 1.1.

Coronel, J.O, Blanes, F., Benet, G., Pérez, P., Simó, J.E., “CAN-based Distributed Control Architecture using the SCoCAN Communication Protocol”, Proc. IEEE Int'l Conf. on Emerging Technologies and Factory Automation, ETFA'2005, vol 1.

Fuhrer, T., Muller, B., Dieterle, W., Hugel, R. “Time Triggerd Communication on CAN”. Cia CAN. 7th international CAN Conference 2000.

Pérez, P., Benet, G., Blanes, F., Simó J. E. “Communications jitter influences on Control loops using Protocols for Distributed Real-Time Systems on CAN bus”. 5th IFAC International Symposium. SICICA’03. Aveiro (Portugal). 2003

Pérez, P., Posadas, J.L., Benet, G., Blanes, F., Simó, J.E., “An Intelligent Sensor Architecture for Mobile Robots” 11th International Conference on Advanced Robotics - IEEE ICAR’03. University of Coimbra (Portugal). 2003

Rufino, J., Veríssimo, P., Arroz G., Almeida, C., Rodrigues, L., (1998) “Fault-tolerant broadcasts in CAN”, Digest of papers, The 28th IEEE International Symposium on Fault-Tolerant Computing.

Simó, J. et al. “Behaviour Selection in the YAIR Architecture”. In proceedings of IFAC Conference on Algorithms and Architectures for Real Time Control. Vilamoura, Portugal. AARTC'97.

Tindell, K., Burns, A. y Wellings, A. J., (1995) “Calculating controller area network (CAN) message response time”, Control Engineering Practice, Vol. 3(8).

Tindell, K., (1994) “Analysing Real-Time Communications: Controller Area Network (CAN)”. Proceedings of IEEE Real-Time Systems Symposium RTSS'94., IEEE Computer Society Press, pp. 259-263.

Yodaiken, V., Barabanov, M., (1996) “Real Time Linux”, Linux Journal.