ARQUITECTURA DE CONTROL DISTRIBUIDA USANDO NODOS EMPOTRADOS CON RT-LINUX SOBRE EL PROTOCOLO DE COMUNICACIONES SCoCAN

J.O. Coronel, F. Blanes, P. Peréz, M. Albero, G. Benet, J.E. Simó
jacopa1@doctor.upv.es, pblanes@disca.upv.es, pperez@disca.upv.es, mialgil@doctor.upv.es,
gbenet@disca.upv.es, jsimo@disca.upv.es

Resumen

En este artículo se describe el diseño y la implementación de una arquitectura distribuida sobre SCoCAN para el control de un robot móvil[1]. En el desarrollo de esta arquitectura se han implementado nodos modulares empotrados encargados de gestionar tanto el control de sensores y actuadores como el control temporal del protocolo de comunicaciones SCoCAN (Shared Channel On CAN). El protocolo adoptado es un esquema TT (time trigger) puro, debido a que tiene un solo ciclo básico que me garantiza tiempos de respuesta aceptables y determinísticos, además de esto, SCoCAN utiliza algunas características de la comunicación por eventos en algunos de sus slots. Cada nodo se divide en tres partes, un primer módulo con microprocesador 8xC592 con controlador CAN, un módulo inteligente principal con microprocesador Ex386 empotrado con el sistema operativo Linux con extensión de tiempo real (RT-Linux) y un tercer módulo de acondicionamiento y digitalización. Toda la coordinación y planificación de las tareas de tiempo real encargadas del control del sistema así como de la configuración del protocolo de comunicaciones es llevada a cabo en el modulo 386 con RT-Linux

Palabras Clave: Robot móvil, sensores, actuadores, tiempo real, sistema empotrado, SCoCAN, TDMA.

1 INTRODUCCION

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 el control industrial automático, etc. en donde diversos sensores, actuadores y dispositivos de control, con diferentes niveles de complejidad, 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 ser planificada de forma que se cumpla con los requerimientos temporales necesarios, evitando de esta forma retardos de tiempo crítico y que los datos periódicos excedan su deadline y jitter máximo.

Los sistemas distribuidos por lo general usan buses de campo, tales como CAN, Interbus-S, LON, Profibus, entre otros. Un adecuado control e integración espacial y temporal del sistema requiere una respuesta en tiempo real, por lo que es importante tener en cuenta no solo las características del nodo sino además las del bus de campo a utilizar. Aunque hay una gran variedad de sistemas de buses de tiempo real que son usados para interconectar dispositivos, CAN (Controler Area Network) es una de las soluciones preferidas a la hora de comunicar sistemas empotrados distribuidos en espacios pequeños.

CAN [2] 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+AMP). Este protocolo de comunicaciones implementa un acceso al medio basado en prioridades fijas. Cada mensaje tiene un identificador único que le indica la prioridad, a menor identificador, mayor prioridad. De esta forma, si dos dispositivos intentan transmitir al mismo tiempo solo el de mayor prioridad logrará transmitirse. CAN implementa una técnica de planificación dinámica basada en prioridades fijas, las cuales se establecen dependiendo de las

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

características temporales de los mensajes, tales como periodicidad, deadline y tiempo de ejecución en el peor de los casos (WCET). Existen dos esquemas fundamentales de planificación con prioridades fijas: Rate Monotonic (RM) cuya asignación de prioridades se basa en la periodicidad de los mensajes, cuanto menor es el periodo mayor es la prioridad. En este esquema se asume que el deadline es igual al período. El otro esquema es Deadline Monotonic (DM) donde las prioridades se asignan de forma inversamente proporcional al deadline de los mensajes, considerándose esté menor que el período. Pero estos esquemas de planificación no garantizan un jitter mínimo ni la determinación del instante de transmisión, pudiéndose exceder los límites temporales de los mensajes. Esto se debe a que los tiempos de respuesta de los mensajes CAN tienen una gran variabilidad [6,7,9], que depende de las condiciones de error del canal así como de la carga del sistema. Esta incertidumbre tiene una repercusión negativa en el sistema, por lo que han sido propuestas varias extensiones de CAN para tratar de disminuir al máximo las latencias en la comunicación. Algunas de estas serán introducidas a continuación.

Flexible Time Triggered CAN (FTTCAN) [4] es una extensión de CAN basado en una planificación dinámica TDMA. FTTCAN tiene un ciclo elemental ó ciclo básico dividido en dos ventanas, una asíncrona utilizada para transmitir mensajes que no son de tiempo real y cuyo acceso al bus esta determinado por el protocolo regular CAN, y otra ventana sincrona en donde se transmiten los mensajes con características de tiempo real. En esta ventana el tráfico es planificado dinámicamente por un nodo central maestro lo que hace a FTTCAN un protocolo muy flexible pero un poco más complejo. En la figura 1 se muestra un ejemplo de un ciclo elemental TTCAN.

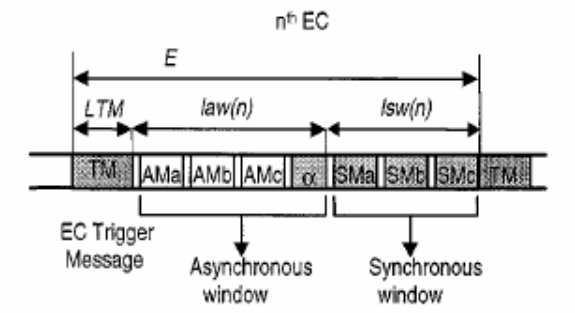

Figura 1: Ejemplo de un ciclo elemental de FTTCAN

Time Triggered CAN (TTCAN) [3] 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 privadas, que son utilizadas para transmitir únicamente un mensaje específico, ventanas compartidas, 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 la figura 2 se presenta un ejemplo de ciclo de matriz.

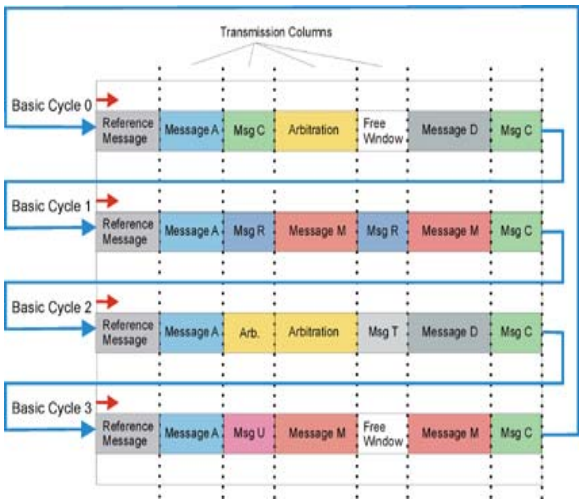

Figura 2: Ejemplo de un ciclo de matriz en TTCAN.

En este artículo presentaremos la descripción de una arquitectura distribuida para el control de un robot móvil híbrido denominado YAIR II, utilizando nodos inteligentes empotrados con RT-Linux, además, analizando el comportamiento y los requerimientos del sistema se utilizara como protocolo de comunicación una extensión básica de CAN, denominada Shared Channel on CAN (SCoCAN) [5] basado en una planificación estática TDMA, con lo que se tendrá una tabla de planificación de tiempos, pero a diferencia de TTCAN, esta tabla será la misma para cada ciclo básico. Se definirán ventanas privadas, para mensajes de tiempo real, y ventanas compartidas, en donde se compite por el acceso al bus CAN. En general este sistema comparte algunas características con TTCAN e implementa otras nuevas.

El artículo esta organizado de la siguiente forma. La sección 2 presenta la descripción general del hardware implementado en el sistema distribuido, detallando los nodos inteligentes empotrados. En la sección 3 se hace una presentación formal del protocolo de comunicaciones SCoCAN. La sección 4 hace referencia a las características temporales de los sensores y actuadores, que se deben tener en cuenta a la hora de programar los módulos. En la sección 5 se describen las tareas y mensajes utilizados en la implementación. Finalmente la sección 6 contiene las conclusiones y los futuros trabajos.

2 HARDWARE DE LA ARQUITECTURA

2.1 DESCRIPCION GENERAL DEL SISTEMA DISTRIBUIDO

El sistema implementado en el robot es un sistema distribuido con nodos inteligentes empotrados con el sistema operativo Linux con extensión de tiempo real (RT-LINUX). La configuración de la arquitectura adoptada es fácilmente adaptable y configurable a las actuales y futuras necesidades en el diseño del robot. En la figura 3 se puede apreciar una visión global de la arquitectura distribuida implementada en el robot.

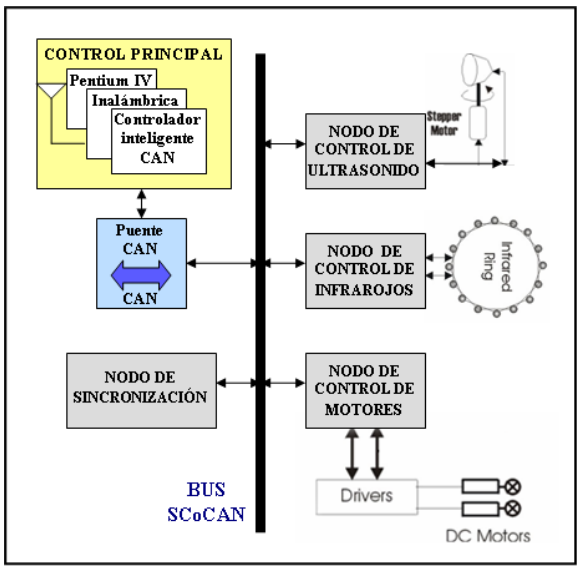

Figura 3: Esquema simplificado del sistema distribuido.

En la arquitectura hay un procesador central dedicado a tareas de recopilación de datos, fusión sensorial y planificación de objetivos de alto nivel [1,8], tales como evitación de obstáculos, seguimiento de trayectorias, levantamiento de mapas, etc. Este procesador central es un PC, cuyo software de desarrollado fue diseñado para una plataforma Windows. El sistema operativo que actualmente se esta utilizando es Windows XP, y debido a que este S.O. carece de características de tiempo real se ha diseñado un puente CAN-CAN para adaptar el PC al protocolo de comunicaciones utilizado (SCoCAN), esta tarjeta se describirá con detalle en el siguiente ítem. Adicionalmente se ha desarrollado un manejador (driver) de tiempo real que implementa el protocolo SCoCAN ofreciendo la alternativa de poder conectar directamente un PC con RT-Linux sin necesidad de utilizar un puente. 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.

2.2 PUENTE CAN-CAN

Este puente me permite adaptar el protocolo de comunicaciones CAN del PC al protocolo SCoCAN que es el utilizado en el bus de comunicaciones del sistema. En el puente se define la tabla de transmisión, de forma que los datos provenientes del PC sean transmitidos a la red en la ventana de tiempo correspondiente. Los datos provenientes de la red son retransmitidos directamente hacia el PC.

Esta placa esta formada por un PIC 16F876 y dos controladores CAN MCP2510, uno para la red del sistema y otro para el control de mensajes con el PC. Un esquema del puente de adaptación es mostrado en la figura 4.

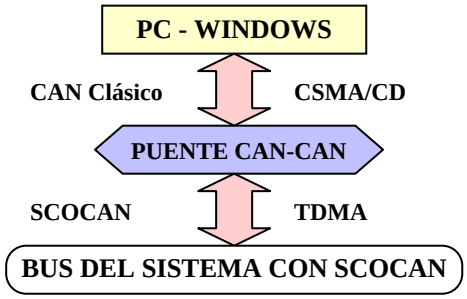

Figura 4: Esquema del puente de adaptación CAN-CAN.

2.4 DESCRIPCION DE LOS NODOS INTELIGENTES EMPOTRADOS

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

La plataforma modular empotrada mostrada en la figura 5, esta dividida en tres placas modulares. Un primer módulo basado en el microprocesador i8XC592, el segundo módulo es una plataforma empotrada con RT-Linux con un microprocesador i386EX y un tercer módulo de acondicionamiento y digitalización que me permite controlar los diferentes subsistemas.

El módulo inferior mostrado en la figura 5, 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 2x 256 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 6 se muestra el esquema general de los nodos.

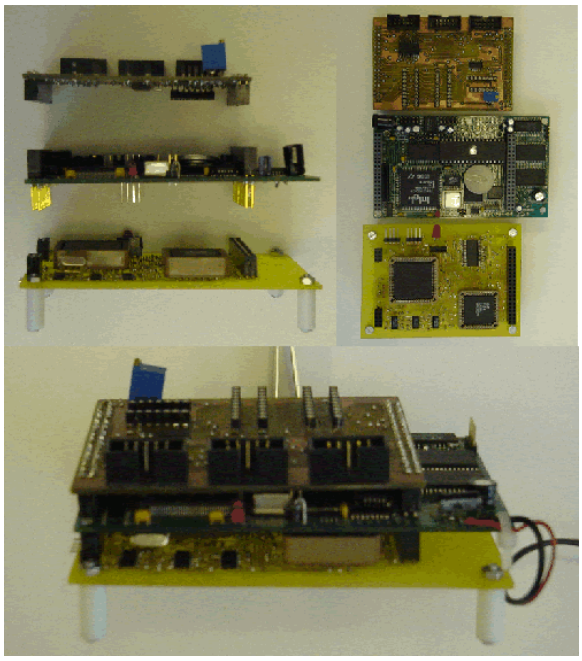

Figura 5: Ejemplo de uno de los nodos inteligentes empotrados.

La plataforma empotrada, que corresponde al módulo central en la figura 5, tiene como núcleo un procesador intel 386 con instrucciones extendidas a una velocidad de 66Mhz. La arquitectura de esta placa es similar a la placa base de un PC. Tiene

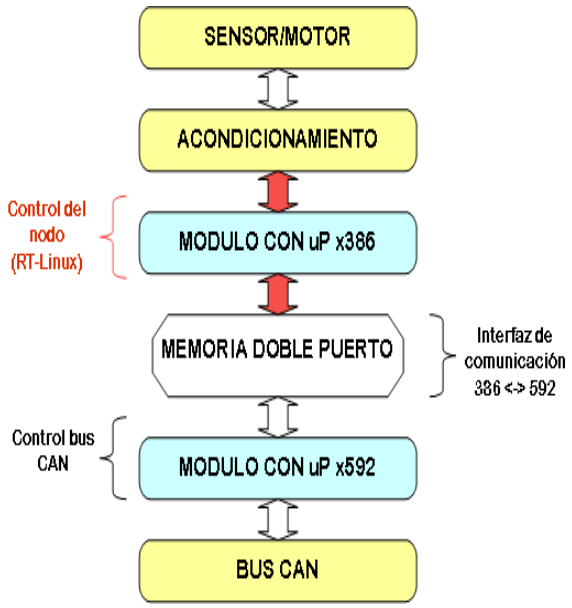

Figura 6: Esquema general de los nodos

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.

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.

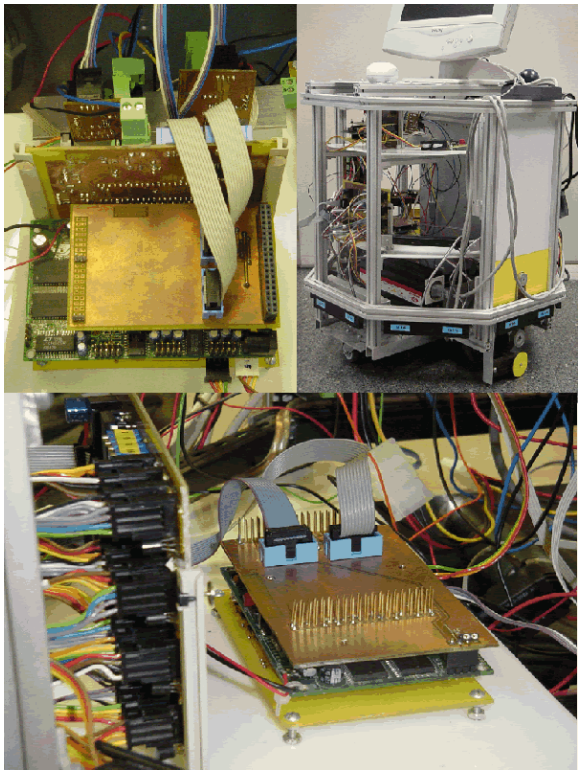

Figura 7: Nodo de infrarrojos y motores colocados en YAIR II.

En la figura 7 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.

3 DESCRIPCION DEL PROTÓCOLO DE COMUNICACIONES SCoCAN

Como se había descrito previamente, el protocolo de comunicaciones SCoCAN (Canal compartido sobre CAN) esta basado en una planificación estática TDMA, en la que se define previamente una tabla de planificación de tiempos. Cada ciclo básico del protocolo usará la misma tabla de planificación.

SCoCAN es un protocolo con un esquema TT (time trigger) puro, debido a que tiene un solo ciclo básico que me garantiza tiempos de respuesta aceptables y determinísticos, además SCoCAN utiliza algunas características de la comunicación orientada a eventos en algunos de sus slots.

Se han definido dos tipos slots de tiempo: **un 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. **Y un slot compartido:** en el que los nodos compiten por el bus utilizando el acceso CAN tradicional (CSMA/CDCA). Este slot es usado para mensajes de temporización no crítica y bloques de datos grandes.

Una de las características a destacar del protocolo, es que los slots privados pueden transformarse en slots compartidos. Esta transformación ocurre cuando en los slots privados no hay datos para transmitir, aprovechando de esta forma el ancho de banda del bus.

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 II se ha definido un ciclo básico de 10 ms, dividido en 20 slots de 500 microsengundos. 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 8 se aprecia la temporización del ciclo básico que se ha usado.

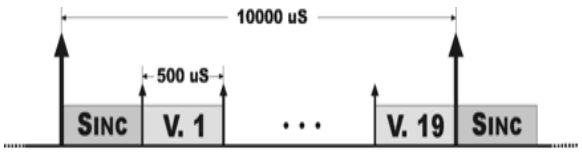

Figura 8: Temporización del ciclo básico de SCoCAN.

También se ha definido una tabla que determina el rango de identificadores que se pueden usar dependiendo del tipo de mensaje. Esta relación de ID´s se puede ver en la tabla 1, en la cual los mensajes del protocolo YCAL son definidos exclusivamente para mantener una compatibilidad con su generación anterior (YAIR), los cuales se transmitirían en slots compartidos.

0x000	0x00F	Mensajes de Sincronización.
0x010	0x01F	Mensajes para Alarmas.
0x020	0x0FF	Mensajes protocolo YCAL. (Compatibilidad).
0x100	0x2FF	Ventanas privadas.
0x300	0x3FF	Ventanas Compartidas (Alta prioridad).
0x400	0x4F8	Cargadores protocolo YCAL.
0x500	0x5FF	Ventanas Compartidas (Baja prioridad).
0x600	0x6FF	Ventanas Compartidas (S. de Ficheros).
0x700	0xXXX	Puentes y Terminales.
0x780	0xXXX	Configuración.

Tabla 1: Rango de identificadores en YAIR II.

4 CARACTERISTICAS TEMPORALES DE LOS SENSORES Y ACTUADORES.

4.1 SENSORES INFRAROJOS

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 9.

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. La primera medida tiene un tiempo de estabilización de 244 us y la segunda medida 348 us, por lo que se necesita un tiempo de 592 us por sensor y 9,472ms para cubrir todo el anillo.

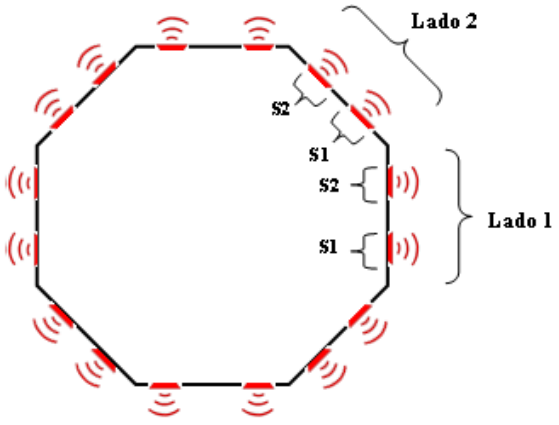

Figura 9: Anillo de infrarojos

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.

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 10, y tiene 200 pasos por vuelta (1.8º/paso) (ver figura 11). Para la adquisición de estos mensajes se utiliza un ADC de 10ksps de 12 bits y se toman 256 muestras por ecó en cada posición, aunque solo se tienen en cuenta 8 bits.

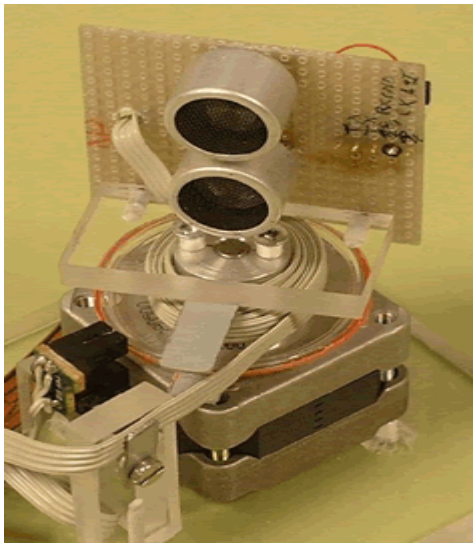

Figura 10. Sensor de ultrasonidos

Tomando 10000 muestras por segundo cada 25.6 ms generaría 256 bytes por posición y 51.2 kbytes por vuelta. Una carga como está 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.

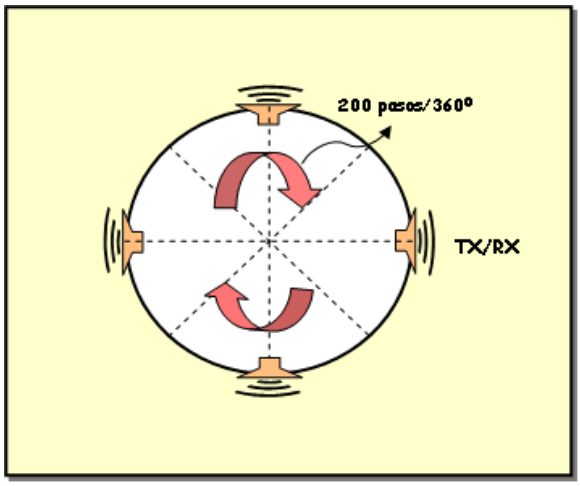

Figura 11. Esquema de ultrasonidos

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

Las tareas y configuración de los mensajes se basan en las siguientes características:

TRAMA: esta definida en la figura 12, cuya nomenclatura se basa en la figura 9.

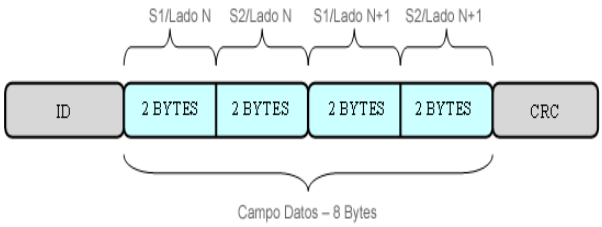

Figura 12: Definición de la trama del módulo de infrarrojos.

PERIODICIDAD: Los datos que conforman un mensaje se generan cada 2.5 milisegundos.

IDENTIFICADORES: se utilizan 4 ID, del 0x100 al 0x103. Esta asignación se basa en la tabla 1.

TIPO DE SLOT: Privados.

5.2 MOTORES

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

5.2.1 Estado

Estos mensajes se subdividen en dos, alarmas y acciones.

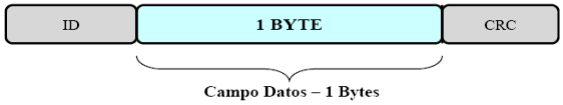

Figura 13: Formato de la trama de alarma.

Alarmas**:** Se utilizan para especificar errores, parada de motores inesperados, etc. Trama: ver figura 13. Periodicidad: Por evento. Identificador: 0x010. Esta asignación se basa en la tabla 1. Tipo de slot: Alarma.

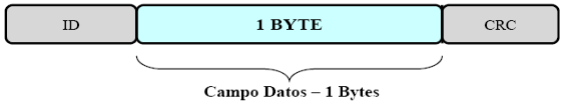

Figura 14: Formato de la trama de acciones.

Acciones: Se utiliza para encender o apagar los motores. Trama: ver figura 14. Periodicidad: Generada por PC. Identificador: 0x301. Tipo de Slot: Compartido.

5.2.2 Posición

En este mensajes se envían las cuentas de encoder. Trama: ver figura 15. Periodicidad: Cada 10 ms. Identificador: 0x10A. Tipo de Slot: Privado.

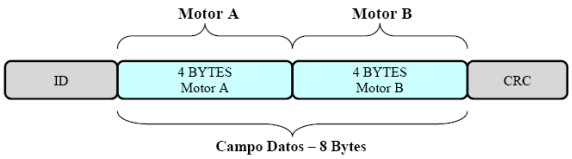

Figura 15: Formato de la trama de posición

5.2.3 Velocidad

Este mensaje es enviado por el PC para regular la velocidad de los motores. Trama: ver figura 16. Periodicidad: Generada por PC. Identificador: 0x10B. Tipo de slot: Privado.

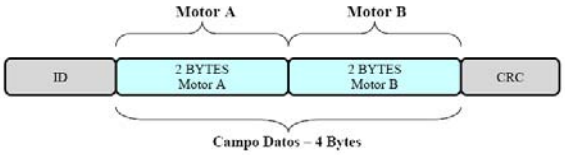

Figura 13: Formato de la trama de velocidad.

5.2.4 Modos

Es mensajes es usado para definir los modos de control de los drivers del motor. Trama: ver figura 17. Periodicidad: Generada por PC. Identificador: 0x300. Tipo de slot: Compartido.

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

Figura 13: Formato de la trama de modos con la relación de los posibles modos.

5.2.5 Aceleración

Este mensaje es usado para cambiar la aceleración de los motores del robot. Trama: ver figura 18. Periodicidad: Generada por PC. Identificador: 0x302. Tipo de slot: Compartido.

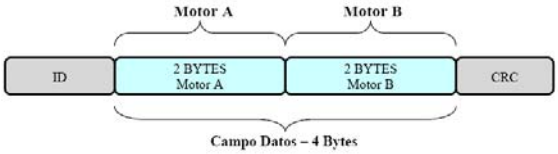

Figura 13: Formato de la trama de aceleración.

Especificación / Msj. / Módulos	Datos (byte)	Periodicidad de generación de información	Nº de msj	Tipo de mensaje	Identificadores
Infrarrojos	8	2.5 ms	4	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	Privado	0x10A
Motores-Velocidad	4	Generada x PC	1	Privado	0x10B
Ultrasonidos	8	--	32	Compartido	0x310-0x318

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

6 CONCLUSIONES

Usando protocolos con planificación TDMA, ya sea dinámica o estática, se pueden eliminar los efectos producidos por los retardos generados en CAN. Esto

se debe a que los tiempos de respuesta de los mensajes CAN tienen una gran variabilidad, que depende de las condiciones de error del canal así como de la carga del sistema.

El protocolo SCoCAN, descrito es secciones previas, es una buena opción, a la hora 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 otra característica especial es la flexibilidad de los slots privados que se transforman en slots compartidos cuando hay inactividad.

En la arquitectura del robot YAIR II 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 utilizando el protocolo de comunicaciones SCoCAN han proporcionado resultados satisfactorios en cuanto a la gestión de carga y temporización en el bus.

La implementación de delegación de código en la arquitectura distribuida del robot utilizando SCoCAN será un futuro trabajo.

Referencias

[1] 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.

[2] CiA (CAN in Automation), (1996) "CAN Application Layer for Industrial Applications", Documents No.: DS-201...DS-207, Version 1.1.

[3] Fuhrer, T., Muller, B., Dieterle, W., Hugel, R., (1999) "Time Triggerd Communication on CAN".

[4] Martins, E., Fonseca, J., Almeida, Luis., (1999) "A Coprocessor for Traffic Scheduling and Schedulability. Analysis in FTT-CAN".

[5] Pérez, P., Benet, G., Blanes, F., Simó J. E., (2000) "Communications jitter influences on Control loops using Protocols for Distributed Real-Time Systems on CAN bus".

[6] Rodríguez, G., Barranco, M., Proenza J., "Harmonizing Dependability and Real Time in CAN Networks".

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

[8] 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.

[9] Tindell, K., Burns, A. and Wellings, A. J., (1995) "Calculating controller area network (CAN) message response time", Control Engineering Practice, Vol. 3(8).