Sistema software de acceso a dispositivos en tiempo real integrado en la plataforma MissionLab

Resumen de la Memoria

Autor

D. Rubén González del Pozo

Tutores

Dr. D. Raúl Alves Santos

Dr. D. Vidal Moreno Rodilla

Departamento de Informática y Automática

Universidad de Salamanca

1. Introducción

Dentro del sector productivo, la automatización de los sistemas logísticos del almacenaje constituye un desafío fundamental que busca optimizar el proceso productivo. El trabajo que se plantea en este proyecto se enmarca dentro de un proyecto de investigación más general en el que se pretende diseñar un sistema de guiado de vehículos terrestres que trabajen tanto en interiores como en exteriores, manejando diversos materiales. Este sistema se desplegará en una carretilla industrial con el fin de obtener un sistema autónomo de transporte para realizar tareas de logística y almacenaje. Dicha carretilla industrial posee una serie de dispositivos sensores y actuadores, gestionados por un sistema distribuido de microcontroladores conectados mediante un bus de campo CAN.

Este proyecto de fin de carrera aborda la capa de comunicación entre los datos del bus CAN y el software de especificación y control de misiones robóticas multiagente *MissionLab*. Para ello es necesario, por una parte, realizar la comunicación con el bus CAN, solicitando y adquiriendo los datos que proporcionan los sensores y enviando datos a los actuadores y, por otro lado, proporcionando los datos actualizados al sistema de control de misiones. Todo esto se realiza teniendo en cuenta que se trata de un sistema con una serie de tareas críticas y restricciones de tiempo real, por lo que se debe garantizar un tiempo de respuesta mínimo. Para conseguir este objetivo se utiliza *Xenomai*, un *framework* de desarrollo en tiempo real, que además permite convertir un *kernel* de Linux en un sistema de tiempo real.

Al finalizar el proyecto se dispone de un medio de comunicación de alto nivel que se puede utilizar entre el planificador de misiones de *MissionLab* y la carretilla industrial permitiendo el funcionamiento guiado de la misma.

2. Objetivo global

En este proyecto de fin de carrera se plantea como objetivo desarrollar un sistema software que permite el acceso a alto nivel, y en tiempo real, a una plataforma robótica real. Esta plataforma se trata de un vehículo de dimensiones industriales. Posee una serie de dispositivos (sensores y actuadores) gestionados por un sistema distribuido de microcontroladores conectados mediante un bus de campo CAN. El objetivo global del desarrollo es su integración en una arquitectura de control de alto nivel.

3. Aspectos relevantes

El propósito de este proyecto es crear una capa de comunicación abstracta (Figura 1) entre un robot físico y el driver utilizado por *HServer*, servidor de hardware de la arquitectura de control *MissionLab*, proporcionando una interfaz de alto nivel para acceder y controlar los dispositivos del vehículo que están conectados mediante un bus de campo CAN.

La capa desarrollada, denominada *SistemaSwAccesoDispositivosTiempoReal* en el esquema, es la encargada de configurar y establecer las comunicaciones con el robot, así como de solicitar periódicamente, y en tiempo real, los datos de los dispositivos conectados al robot físico para mantener esta información actualizada en la *Memoria compartida*, realizando el entramado, desentramado y paso de mensajes a bajo nivel. Estos datos actualizados serán solicitados mediante la interfaz de alto nivel desarrollada, desde la *Misión* creada mediante el editor de misiones de la arquitectura *MissionLab*, *CfgEdit*. Dicha misión es ejecutada por *mlab* a través del driver (*driver Carretilla*) creado para la carretilla industrial en el servidor de hardware *HServer*. Así mismo, las órdenes de control enviadas por la *Misión* de alto nivel al robot, son encapsuladas en mensajes por el *SistemaSwAccesoDispositivosTiempoReal*, encargado de su envío y actualización periódica en el robot físico.

Aunque se haya desarrollado para utilizarla específicamente con el servidor *HServer* y la carretilla elevadora industrial, esta capa de comunicación será independiente tanto de los dispositivos del robot como del servidor de hardware utilizado.

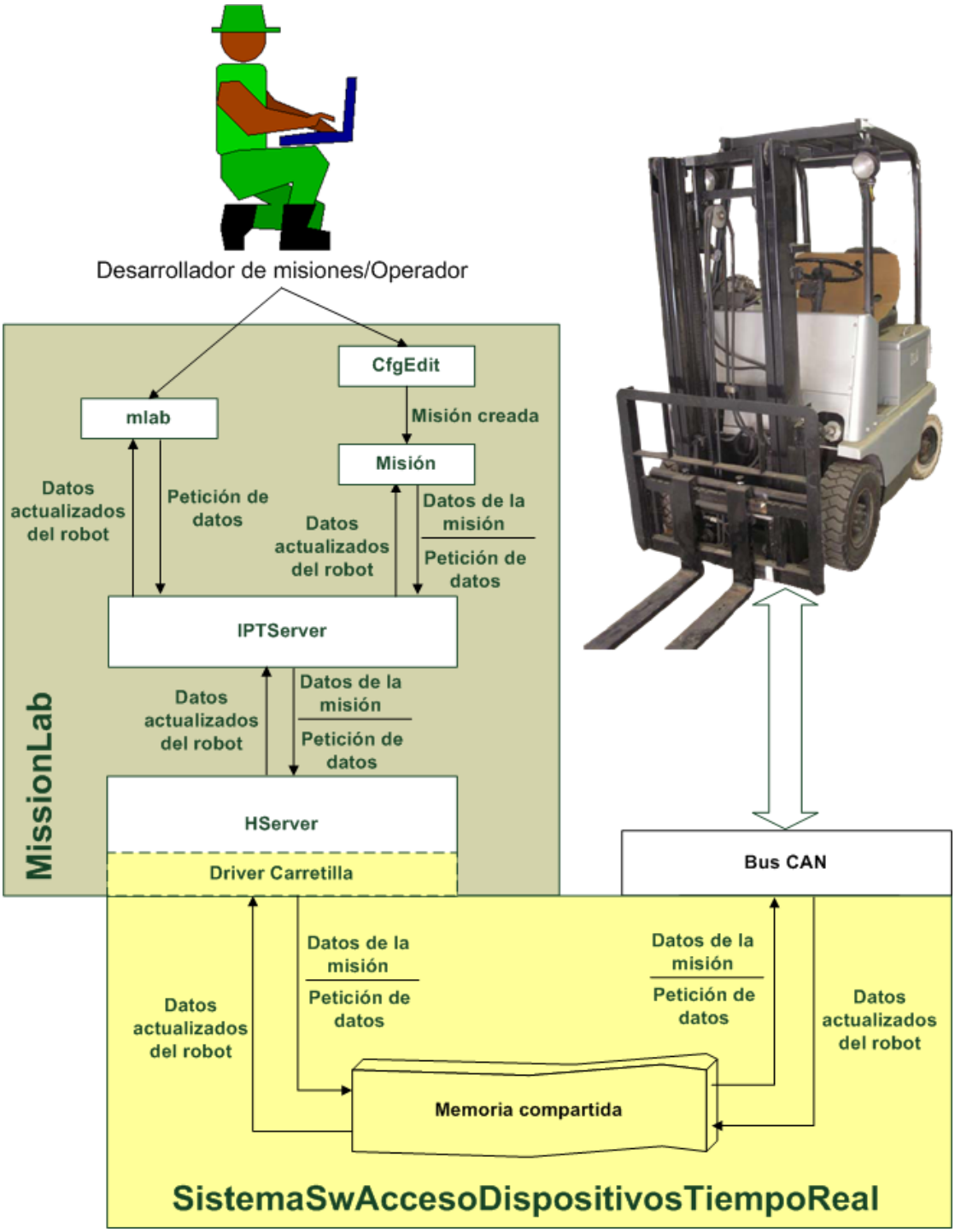

Figura 1– Arquitectura general del sistema

La arquitectura física de la plataforma (Figura 2) se ha diseñado siguiendo el modelo de capas de los sistemas de control distribuidos (DCS).

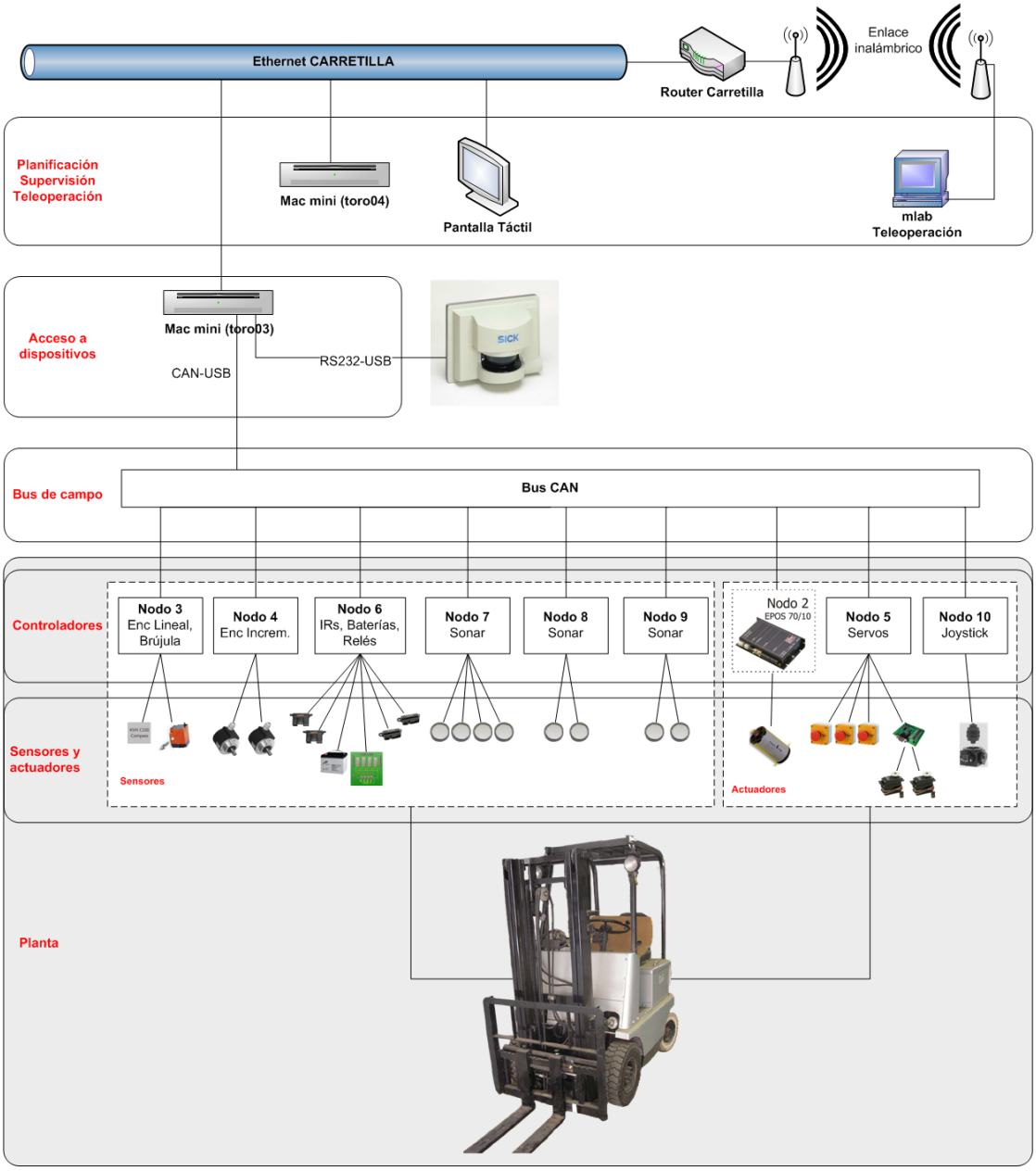

Figura 2 – Arquitectura de control distribuida de la plataforma

El nivel de *Planta* está constituido por las capas de acceso de bajo nivel a los dispositivos conectados directamente sobre la plataforma física, incluyendo tanto la capa de *Sensores y actuadores*, como la que permite el acceso a esta capa desde los niveles superiores de la arquitectura: *Controladores*.

La capa del *Bus de campo* permite sustituir las conexiones punto a punto entre los elementos de campo (instrumentación) y el equipo de control (capas superiores). En este sistema se ha utilizado un bus de campo CAN.

En el siguiente nivel, *Acceso a dispositivos*, se encuentra el puesto de control *toro03*, Mac mini ejecutando una distribución de Ubuntu parcheada con el *framework* de desarrollo en tiempo real *Xenomai*, que lo convierte en un sistema de tiempo real. Este será el encargado de ejecutar la capa de comunicación desarrollada en este proyecto, así como *HServer*, servidor de hardware de *MissionLab*. Está conectado al bus CAN de los dispositivos a través de un puerto USB mediante un adaptador CAN-USB. También se encarga de acceder al sensor Láser SICK LMS 221 a través de otro puerto USB mediante un adaptador RS232-USB.

En la capa *Planificación, Supervisión y Teleoperación*, nivel superior de la arquitectura, se encuentran tanto un Mac mini (*toro04*) como un PC industrial con *pantalla táctil*, además de cualquier puesto remoto para teleoperación a través del enlace inalámbrico disponible:

- El nodo *toro04* es el encargado de ejecutar el servidor *IPTServer* para la comunicación de procesos a través de redes TCP/IP. También ejecuta la misión deseada, que se comunica con el servidor de hardware *HServer* a través de una red TCP/IP privada que comparten todos los ordenadores.

- El PC industrial con *pantalla táctil* se encarga de ejecutar la consola de *MissionLab* (*mlab*), que monitoriza el progreso de ejecución de la misión, mostrando el movimiento del robot y el valor de los dispositivos. Permite al usuario crear nuevas misiones y modificar *offline* la misión en curso utilizando *CfgEdit*, e incluso realizar una teleoperación manual del robot.

El diseño del sistema software se realizó concibiendo la capa de comunicaciones independiente tanto de la plataforma robótica y dispositivos conectados al bus CAN como del servidor de hardware al que se proporciona la interfaz de alto nivel para el acceso a dichos dispositivos. El único requisito que se debe cumplir es la utilización del protocolo CAN seguido en el desarrollo del proyecto por parte de los dispositivos.

La arquitectura de este sistema, mostrada en la Figura 3, proporciona la independencia deseada gracias a dos aspectos fundamentales: utilización de una

memoria compartida para un acceso a los datos de alto nivel y acceso a los dispositivos a bajo nivel a través de un bus de campo CAN.

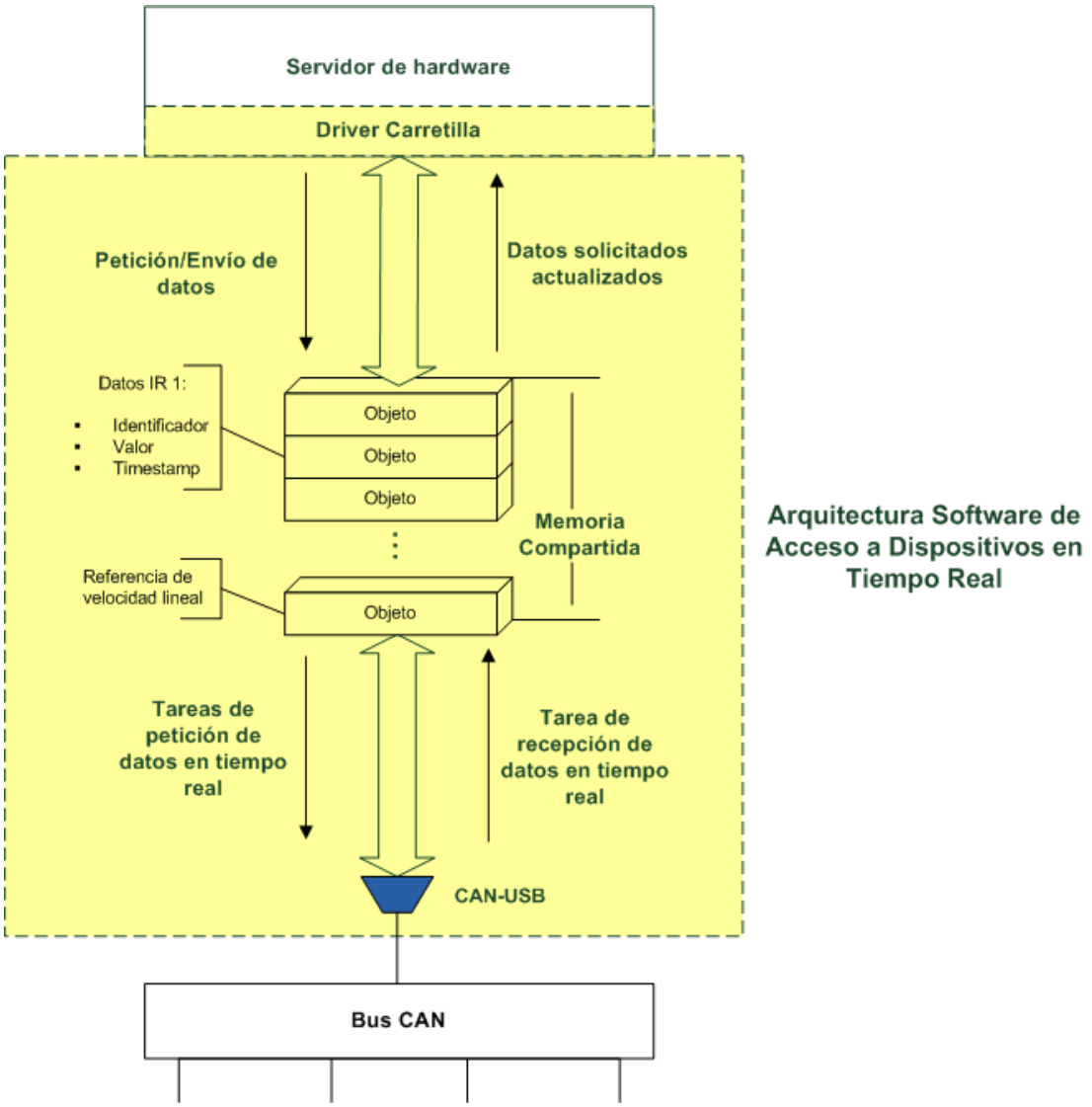

Figura 3 – Esquema general de la arquitectura software

Se utiliza una memoria compartida como mecanismo de comunicación entre procesos, lo que permite un acceso a los datos actualizados. El driver específico creado para el vehículo, en el servidor de hardware utilizado, permite el acceso a estos datos de una manera transparente a través de la interfaz de alto nivel proporcionada.

Para la comunicación del sistema ejecutado en el ordenador Mac mini *toro03* con los nodos de dispositivos del vehículo a través del bus de campo CAN, se han utilizado en este proyecto unos adaptadores de bus CAN a USB, debido a que este ordenador únicamente dispone de puertos USB. Estos adaptadores poseen un

controlador CAN y un microprocesador que maneja las comunicaciones entre la parte CAN y la parte USB.

El intercambio de datos con los dispositivos de la plataforma robótica es realizado por un sistema de tareas de tiempo real (Figura 4), desarrollado gracias al uso de las primitivas y la funcionalidad proporcionada por *Xenomai*. Estas tareas se ejecutan periódicamente y de manera concurrentemente, cumpliendo sus *deadlines*.

```
root@toro03:~# cat /proc/xenomai/sched
CPU  PID    CLASS  PRI      TIMEOUT   TIMEBASE   STAT       NAME
  0  0      idle    -1      -         master     R          ROOT/0
  0  5609   rt       0      -         master     X          hserver
  0  5618   rt      70      7ms912us  master     D          ReceiveTask
  0  5619   rt      60      -         master     X          SupervisionDataTask
  0  5620   rt      60      -         master     X          CalculationDataTask
  0  5621   rt      60      11s498ms273us  master     D          LowRefreshDataTask
  0  5622   rt      80      -         master     X          DataUpdaterTask
  0  5623   rt      60      -         master     X          SonarDataTask
```

Figura 4 – Planificador de tareas de *Xenomai*, con las tareas creadas en este proyecto

El sistema de tareas de tiempo real está compuesto por:

- ReceiveTask: esta tarea es la encargada de la lectura de los datos del bus CAN y su almacenamiento en memoria compartida. Su comportamiento depende del adaptador CAN-USB utilizado. En el caso de utilizar el adaptador CANUSB del fabricante *Lawicel*, lee el bloque de datos serie que se encuentra en el búfer de entrada del puerto y crea un hilo, al que envía el bloque leído. En este hilo, paralelamente a la ejecución de la tarea, divide el bloque en tramas serie, las desentrama y almacena la información que contienen en la memoria compartida. En caso de que se utilice el adaptador PCAN-USB de *PEAK Systems*, realiza la lectura del mensaje CAN que se encuentra en el búfer de entrada del puerto y obtiene directamente la información, introduciéndola en la memoria compartida. Se le ha asignado una periodicidad de 0.03 s en el caso del adaptador *Lawicel* y 0.025 s en el caso del adaptador *PEAK*, debido a que con el primero se lee un bloque que contiene varias tramas, mientras que el segundo lee tramas CAN individuales.

- SupervisionDataTask: se encarga de realizar periódicamente solicitudes bajo demanda de las mediciones de datos que permiten supervisar el control

automatizado del sistema. En el caso de la plataforma física utilizada, se realizan peticiones de medición de los sensores infrarrojos. Su periodicidad es de 0.25 s, ya que no se necesita una mayor actualización al tratarse de un vehículo experimental donde los movimientos son razonablemente lentos.

- CalculationDataTask: los datos solicitados por esta tarea son necesarios para realizar cálculos, como puede ser el cálculo de la velocidad lineal de avance del vehículo, gracias a los encoders incrementales conectados a las ruedas. Por lo tanto, esta tarea solicita los datos de los encoders incrementales, el encoder lineal y la brújula. Su periodicidad se ha establecido en 0.1 s debido a que, al tratarse de un vehículo experimental, se mueve a una velocidad muy baja, sin necesidad de obtener más datos por el momento aunque se tenga capacidad para ello.

- LowRefreshDataTask: esta tarea realiza las peticiones de datos de los dispositivos con una tasa de refresco muy baja, como pueden ser las baterías. Estos datos no varían de manera significativa, por lo que no es necesario actualizarlos constantemente. En el caso de este vehículo se solicitan únicamente los datos de las baterías y su periodicidad se ha establecido en 60 s, debido a la razón expuesta.

- SonarDataTask: esta tarea solicita específicamente los datos de los sonares. Se diseñó de manera separada como solución al problema de *crosstalking*, Esta solución fue solicitar la medida de cada uno de los sonares de manera secuencial realizando una espera de 0.06 s entre la petición de cada sonar individual, ya que es el tiempo que toma el sensor como máximo para recibir la señal reverberada. Las señales reverberadas que le lleguen en un plazo de más de 0.06 s las ignora. Este es el motivo por el cual la periodicidad de esta tarea se ha establecido en 0.8 s.

- DataUpdaterTask: La misión de esta tarea es actualizar las referencias (o *set points*) de los controladores PID que posee vehículo, permitiendo un control automatizado del mismo. Estas referencias son establecidas por el servidor de hardware al que esté conectado el sistema, procedentes de la misión ejecutada en las capas superiores de la arquitectura de control. Su periodicidad se ha establecido en 0.1 s debido a que la misión ejecutada por la arquitectura de

control utiliza los datos obtenidos por la tarea *calculationDataTask* para calcular las rectificaciones, y su periodo también se estableció en 0.1 s.

Para comprobar las capacidades de las comunicaciones, se ha realizado un análisis de la información transmitida a través del bus CAN durante el transcurso de una misión. La misión de prueba tiene una duración de cinco minutos, durante los cuales realiza desplazamientos máximos de la dirección manteniendo una velocidad de avance constante.

En la gráfica de la Figura 5 se muestra la evolución temporal del ancho de banda utilizado durante el transcurso de la misión, expresado en tramas/s. Teniendo en cuenta que los adaptadores utilizados soportan una tráfico máximo de 1000 tramas/s, se observa como, sin llegar a saturarse en ningún momento, se utiliza como término medio $1/3$ del ancho de banda disponible.

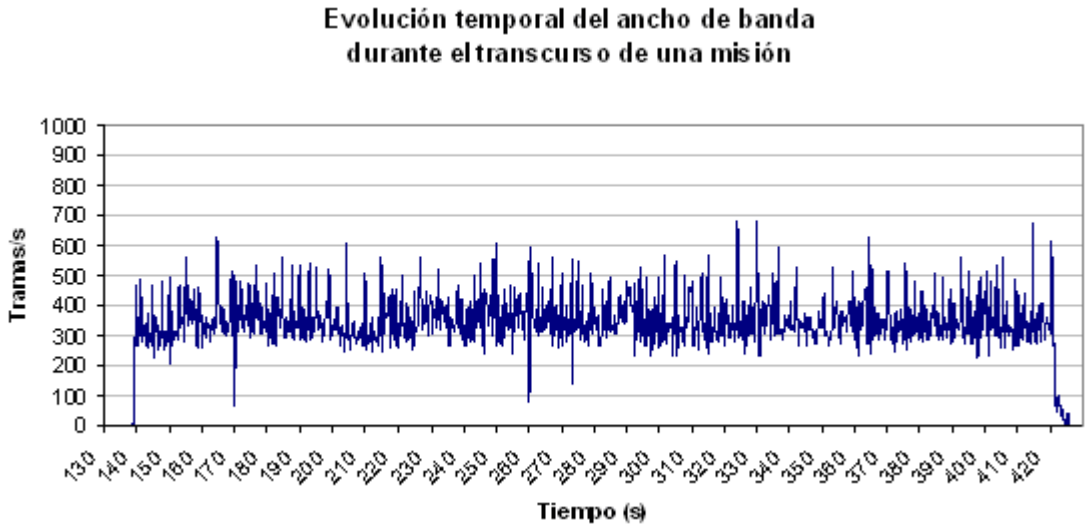

Figura 5 – Evolución temporal del ancho de banda durante la misión de prueba

Haciendo una comparativa del ancho de banda total utilizado por cada nodo (Figura 6), se obtiene que la mayor parte de las comunicaciones son utilizadas por las tareas críticas del sistema, como son las asociadas al control de la trayectoria del vehículo. La parte sensorial representa una carga muy inferior, de forma que se puede incrementar el número de sensores utilizando el esquema de comunicaciones propuesto, ya que se dispone de ancho de banda suficiente para posibles ampliaciones.

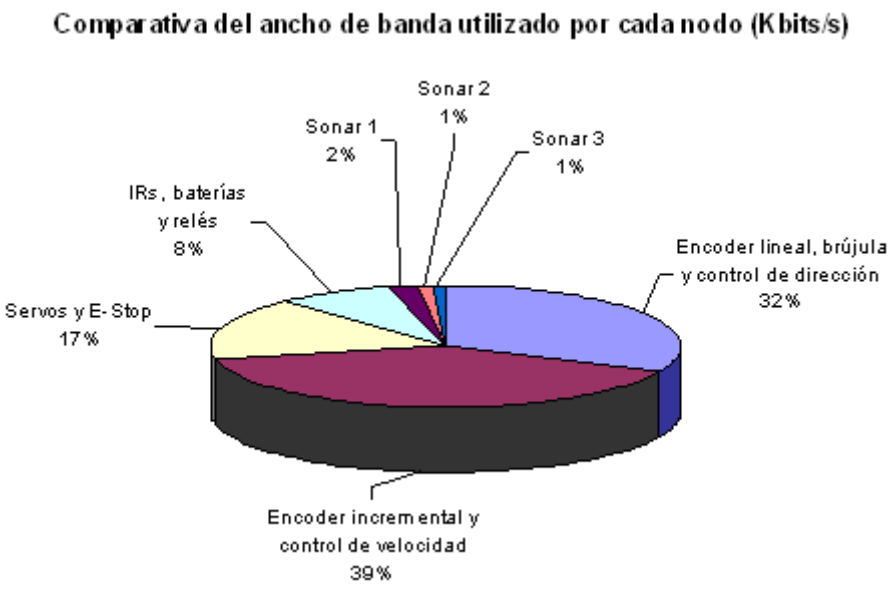

Figura 6 – Comparativa del ancho de banda utilizado por cada nodo (Kbits/s)

Finalmente, utilizando la interfaz de alto nivel que proporciona el sistema software de acceso a dispositivos en tiempo real desarrollado, se puede implementar un driver para la carretilla industrial en *HServer*, servidor de hardware de la arquitectura de control *MissionLab*.

El resultado del despliegue de la arquitectura, tanto software como hardware, es un sistema autónomo de transporte para realizar tareas de logística y almacenaje. En la Figura 7 se muestran diferentes vistas de la carretilla elevadora industrial tras el proceso de robotización descrito en esta memoria.

Figura 7 – Vista frontal y lateral de la carretilla elevadora industrial

4. Conclusiones

Tras finalizar este proyecto, cabe destacar la implementación satisfactoria del objetivo central marcado al comienzo del mismo: el desarrollo de un sistema software que permite un acceso en tiempo real, y a alto nivel, a los dispositivos conectados en un vehículo de dimensiones industriales.

Este sistema desarrollado se trata de una capa de comunicación independiente tanto de la arquitectura de control que se utilice, como de la plataforma robótica, siempre que los dispositivos conectados a esta plataforma se comuniquen a través del protocolo CAN seguido en este proyecto.

Como validación, se ha integrado en un proyecto de investigación cuyo objetivo es la automatización de una carretilla elevadora industrial. Se ha utilizado *MissionLab* como arquitectura de control, creando un driver específico para la carretilla elevadora utilizando la interfaz de alto nivel proporcionada por el sistema. El resultado de esta integración es la capacidad de funcionamiento guiado de la carretilla a través de las misiones diseñadas a alto nivel en *MissionLab*.