Control de un vehículo autoguiado a través de un PC

Este artículo describe las bases para implantar un sistema de control en un vehículo autoguiado (AGV), realizado en el laboratorio del Instituto de Investigación Tecnológica de la Universidad Pontificia Comillas. El sistema de control tiene una arquitectura basada en microprocesador, utilizando un PC convencional y Linux-RT, con el objetivo de minimizar costes y aumentar la flexibilidad en las acciones que el AGV puede llevar a cabo. Se darán unas breves nociones de Linux-RT y se expondrá la arquitectura de control.

Álvaro Sánchez Miralles

Ingeniero del ICAI especialidad electrónica.
Pertenece al área de Sistemas Inteligentes del Instituto de Investigación Tecnológica de la UPCO.
Áreas de trabajo: Robótica móvil, Comunicaciones y Control wireless, Sistemas de Diagnóstico y Modelado de sistemas no lineales.

Gonzalo Alonso de Ozalla Borras

Ingeniero del ICAI especialidad electrónica.
Pertenece al departamento de preventa de servicios profesionales de Lucent Technologies.
Áreas de trabajo: Robótica móvil, valoraciones económicas de servicios profesionales.

Introducción

Los vehículos autoguiados (AGV's) juegan un papel importante en el desarrollo tecnológico de los seres humanos, lo que ha suscitado la atención de muchos investigadores. Los AGV's se han creado con el objetivo de facilitar al ser humano un conjunto de tareas difíciles de llevar a cabo, monótonas o aquellas que conllevan un riesgo humano.

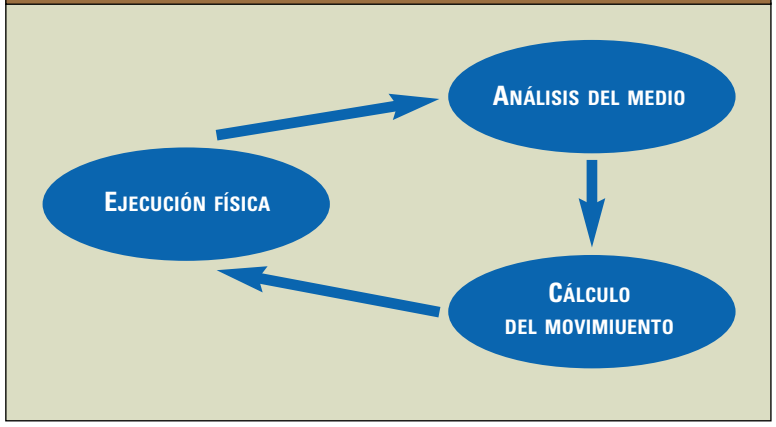
Se han desarrollado muchos tipos de AGV's, desde los caseros que se utilizan para experimentar nuevas sensaciones y juguetear hasta los más complejos que se emplean para labores de investigación

Sin embargo, todos los proyectos tienen tres pilares comunes que son imprescindibles para dotar de autonomía a un vehículo; un sistema motriz adecuado, la capacidad de análisis del entorno [1] y la posibilidad de calcular una reacción consecuente [2], tal como se indica en la Figura 1.

De estas tres habilidades básicas que debe reunir un vehículo autoguiado resulta especialmente importante la viabilidad para tomar decisiones acertadas en el momento oportuno, esto será lo que determine su integridad y operatividad. Considérese el ejemplo de una persona que conduce un automóvil y sabe que debe girar el volante pero no es capaz de hacerlo a tiempo.

En este artículo se va a describir una arquitectura de control basada en Linux [3] y Linux-RT o RT-Linux [4], que ofrece una

FIGURA 1. TAREAS BÁSICAS COMUNES A TODOS LOS VEHÍCULOS AUTÓNOMOS



gran flexibilidad en las acciones a realizar por el AGV, además de permitir ejecutar en tiempo real las tareas más críticas del control.

Descripción del sistema

A continuación se describe el vehículo que se utiliza, el sistema operativo RT-Linux y una arquitectura de control basada en Linux-RT.

El vehículo autoguiado

El AGV [5] que se utiliza fue desarrollado en el Instituto de Investigación Tecnológica de la Universidad Pontificia Comillas.

El AGV consta de tres planchas circulares de madera, ver Figura 2. La primera altura se ocupará de la tracción y la alimentación, la segunda de recoger información del entorno y la tercera de procesar las tareas de control y dar las órdenes pertinentes para el correcto funcionamiento del vehículo.

Control a través de un PC Linux/RT-Linux

Habitualmente las tareas de control de un robot recaen en un microprocesador. Si se añade a éste una serie de accesorios tales como memoria RAM y un dispositivo de almacenamiento de datos (disco duro) se dispondrá de una herramienta de control relativamente económica y accesible para un usuario familiarizado con la informática; para la programación del control pueden utilizarse todos los recursos que ofrecen los paquetes Software convencionales y añadir funcionalidades típicas de una plataforma PC.

Ahora bien, para aprovechar todas estas posibilidades es necesario elegir cuidadosamente un sistema operativo que aúne flexibilidad, economía y permita utilizar gran cantidad de Software que haya sido desarrollado previamente y pueda ser útil para mejorar la operatividad del robot. Además, debe ser capaz de operar en tiempo real para asegurar la correcta ejecución de las acciones críticas que afectan a los movimientos del vehículo.

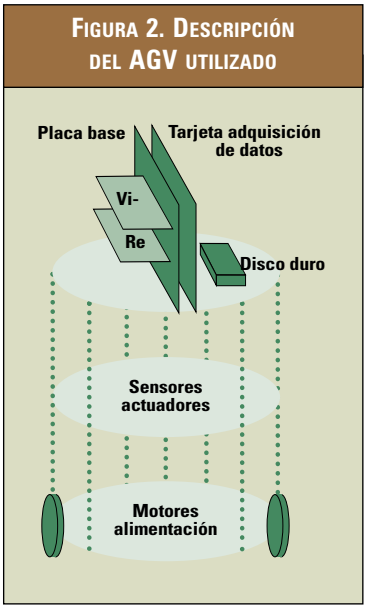
Un sistema operativo que reúne todas estas condiciones es el conocido Linux, con su extensión para operación en tiempo real RT-Linux.

Linux está basado en la filosofía UNIX y puede ser conseguido gratuitamente. Ofrece la posibilidad de instalar numerosas aplicaciones que permiten trabajar con gran eficiencia.

RT-Linux se puede conseguir de manera gratuita vía ftp accediendo a la página web www.rtlinux.com. Una de sus versiones, a la cual se refiere el presente artículo, es RTLinux-Beta16V2, que como su propio nombre indica es experimental y puede contener algún error que no se ha percibido.

El funcionamiento de RT-Linux consiste en un pequeño kernel de tiempo real que coexis-

FIGURA 2. DESCRIPCIÓN DEL AGV UTILIZADO



te con el kernel de Linux. El objetivo de este sistema es poder utilizar los servicios del Software para plataformas PC convencionales permitiendo el desarrollo de funciones en tiempo real en un entorno predecible y con el menor retraso posible.

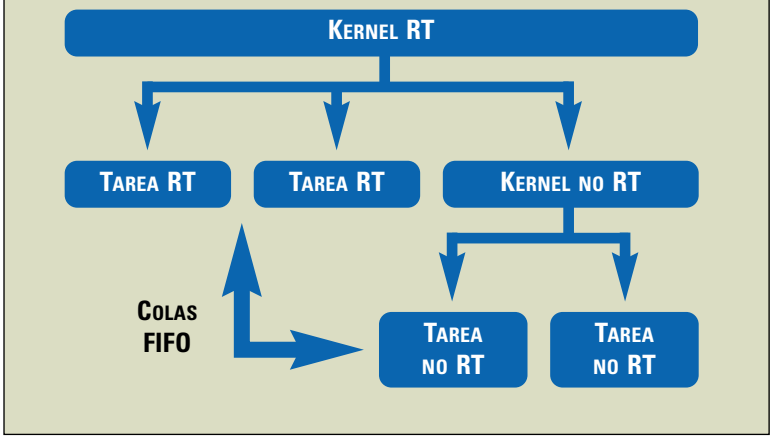
Como se aprecia en la Figura 3, la solución de RT-Linux es un modo de ejecución simple de tareas con un kernel que funciona en tiempo real actuando sobre otro kernel Linux, que no funciona en tiempo real (no RT), tomando a éste como su tarea de menor prioridad. Este último kernel es el que permite utilizar los programas existentes para Linux mientras que el anterior asegura la actuación en tiempo real.

Las tareas de usuario (Linux) se comunican con las de tiempo real (RT-Linux) a través de colas FIFO (first in first out). Desde el punto de vista del programador las colas se asemejan al manejo de dispositivos estándar de UNIX.

RT-Linux se apoya en Linux para arrancar, trabajar en red, manejar el sistema de ficheros, controlar procesos Linux, dispositivos externos y para cargar los módulos de tiempo real, haciendo el sistema extensible y fácilmente modificable.

Las aplicaciones RT consisten en tareas RT que son incorporadas en forma de módulos del sistema de tiempo real, mientras que los procesos Linux se ocupan de almacenar los datos, el interfaz gráfico, acceso a red y cualquier otro tipo de aplicación que no esté forzada por acciones RT.

FIGURA 3. ESQUEMA DE FUNCIONAMIENTO DE LINUX/RT-LINUX



En la práctica RT-Linux ha probado ser bastante eficaz. El mayor retraso de interrupción en un PC equipado con un microprocesador Intel 486 a 33 MHz se ha medido por debajo de los 30 microsegundos. RT-Linux es a la vez rígido y flexible de acuerdo con dos diseños de programación contradictorios.

La primera premisa de diseño es que los componentes de una aplicación RT que realmente están forzados por el tiempo no son compatibles con la localización dinámica de recursos, la sincronización compleja, el trasiego de datos significativos o cualquier otra actividad que introduzca retrasos en la disponibilidad de los mismos.

La configuración de RT-Linux más ampliamente utilizada ofrece tareas primitivas con memoria estáticamente localizada, un scheduler (elemento que se encarga de controlar la ejecución de procesos) simple de prioridad fija y sin protección contra "programas imposibles", difícil deshabilitación de interrupciones, memoria compartida como única primitiva de sincronización entre tareas RT y un rango limitado de operaciones en las colas FIFO conectando tareas RT y procesos Linux no RT.

Sin embargo, el entorno no es realmente tan austero debido a la gran variedad de servicios que ofrece el kernel no RT y a los que se accede fácilmente a través de las tareas Linux de usuario.

La segunda premisa de diseño se basa en que es poco lo que se conoce acerca de cómo deberían estar organizados los sistemas en tiempo real, y el sistema operativo debe permitir gran flexibilidad en asuntos tales como las características de las tareas RT, comunicación y sincronización.

El kernel RT ha sido diseñado con módulos reemplazables para su posterior mantenimiento, ampliación o mejora. En realidad existen módulos alternativos de scheduling (programación de las ejecuciones) de forma que pueda reconfigurarse el modo en que se distribuye el tiempo. Algunos

de estos módulos así como ciertos servicios RT, por ejemplo el uso de semáforos en tiempo real, son todavía experimentales.

Como se ha dicho, la eficacia de RT-Linux se basa en la óptima distribución del tiempo de ejecución. El kernel RT divide el tiempo entre todos los procesos de manera que estos siempre se ejecutan periódicamente. La organización de estas ejecuciones se lleva a cabo teniendo en cuenta en primer lugar qué tareas deben ejecutarse en un instante y, en segundo lugar, si dos tareas concurren en un mismo momento se opta por aquella a la que se haya asignado mayor prioridad. Se trata pues de un sistema periódico y jerarquizado.

En la siguiente Figura 4 se observa, mediante un ejemplo, de qué manera distribuye el tiempo RT-Linux.

El rectángulo exterior representa el transcurso del tiempo, durante el cual se ejecutarán las tareas. Se han supuesto dos tareas RT, T1 y T2 con distintos periodos de ejecución. La duración de la ejecución (representado por cada rectángulo oscurecido) es mayor en el caso de la tarea T2. La tarea T1 es la de mayor prioridad.

Como se ve, ambas tareas se procesan de manera periódica. El espacio sobrante del rectángulo exterior supone el tiempo durante el cual no se ejecuta ninguna tarea RT, y precisamente es éste el tiempo que se aprovecha para procesar las tareas Linux no RT, que son, por defecto, las de menor prioridad.

FIGURA 4. FUNCIONAMIENTO TEMPORAL DE RT-LINUX

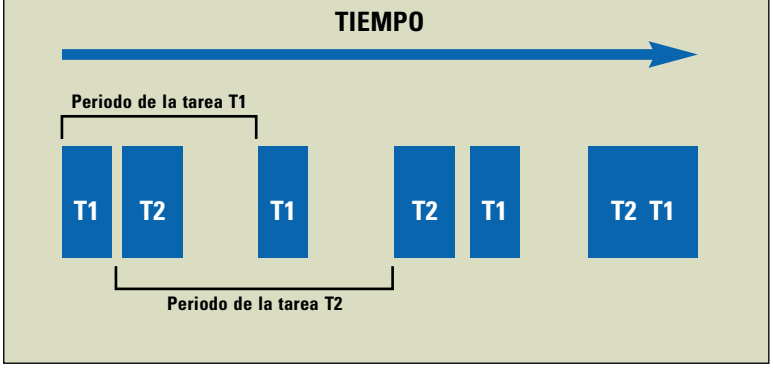
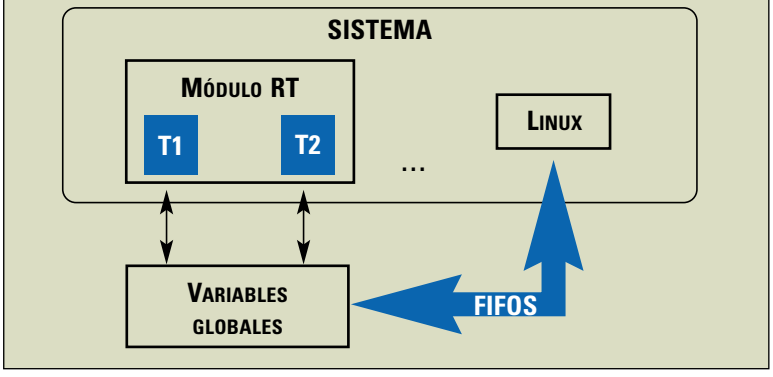


FIGURA 5. ESQUEMA DE PROGRAMACIÓN Y COMUNICACIÓN RT-LINUX



Cuando dos tareas RT concurren en un mismo espacio de tiempo, se ejecuta la de mayor prioridad hasta el final de sus instrucciones. La de menor prioridad proseguirá entonces su proceso de ejecución a partir del punto en el cual fue interrumpida.

La programación debe abordarse de forma modular. Los procesos que deben ser tratados en tiempo real han de ser introducidos en paquetes autónomos de funciones bien organizadas. Un módulo RT es independiente por sí mismo tanto del resto del sistema no RT, que en sí constituye otro módulo, como de otros módulos RT que puedan cargarse.

Ha de tenerse en cuenta que la ejecución de las tareas se realizará de manera periódica. Debe conocerse cuál será la prioridad de los procesos (Linux siempre tendrá la menor). Cada tarea debe tener bien definido cual será su periodo de actuación, esto es, cada cuánto tiempo debe ejecutarse. Puede controlarse el comienzo de la ejecución de las tareas así como su interrupción por software.

Las tareas se comunican entre sí por variables globales, si pertenecen al mismo módulo, y mediante colas FIFO si pertenecen a módulos diferentes, ver Figura 5.

Existe una entidad Software en este sistema llamada Manejador (Handler). Es una tarea que se relaciona con una determinada cola FIFO. El Manejador se ejecuta con la máxima prioridad cada vez que se detecta la llegada de un dato la cola, y suele emplearse en el control de ejecución de tareas (arranque, pausa y eliminación) o para la clasificación de los datos recibidos por la cola.

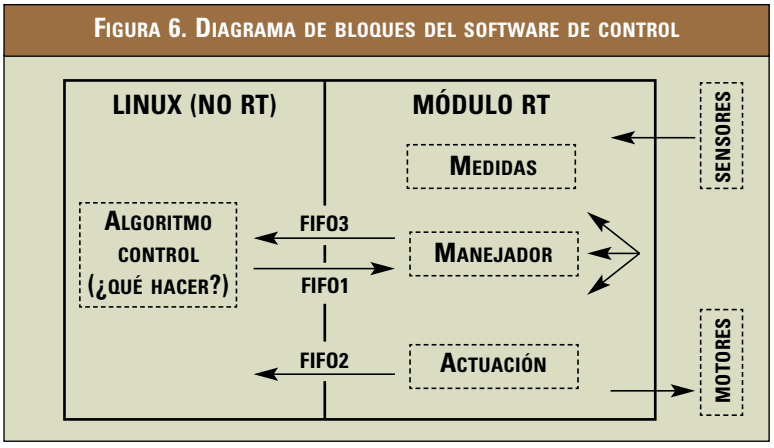
El scheduler distribuye el tiempo entre las tareas que están incluidas en los módulos observando cuál debe ejecutarse en cada momento: primero los manejadores, si ha habido algún acceso a las colas, luego las tareas por orden de prioridad y, finalmente, la parte Linux no RT, que utiliza por tanto el tiempo sobrante de la ejecución de las demás tareas RT del sistema.

Las aplicaciones en RT-Linux se programan en código C [6], de manera completamente estándar atendiendo algunas peculiaridades y normas de estructuración fácilmente asimilables.

Arquitectura básica del control

Una vez conocido el funcionamiento y las posibilidades del sistema operativo que se quiere emplear para el control del vehículo, debe realizarse una descripción estructural detallada del mismo. Una buena estructura de programación no es trivial y puede suponer un ahorro de tiempo y un aumento de la eficacia si se tiene claro cómo se utilizarán los recursos del sistema, en qué momento actuará cada tarea (recordemos que se trata de un sistema en tiempo real) y cómo se van a comunicar éstas.

FIGURA 6. DIAGRAMA DE BLOQUES DEL SOFTWARE DE CONTROL



A la hora de afrontar la programación siempre debe tenerse presente que hay dos partes bien diferenciadas: el código de tareas RT-Linux que se ejecuta en tiempo real o parte RT (módulo RT), y el código de tareas Linux que no se ejecutará en tiempo real o parte no RT (Linux no RT).

Como modelo de programación se propone el siguiente diagrama de bloques (Figura 6) que ilustra esquemáticamente el funcionamiento de un control propuesto.

En él ya se pueden observar las tres tareas que supondrán la base del diseño de programación:

- Lectura de la información de los sensores (análisis del entorno)
- Algoritmo de control (cálculo del movimiento)
- Control de actuación de los motores (ejecución física)

Como sistema de comunicación entre módulos RT se tienen tres colas FIFO, una de escritura, para enviar órdenes desde el control hasta la parte RT, y dos de lectura (siempre desde el punto de vista de la parte Linux no RT) para que el control reciba información de cada una de las tareas RT.

Por último se añadió un Manejador asociado a la cola FIFO1, que actúa en el instante en que se encuentra disponible algún dato en esta cola, para poder controlar el vehículo con precisión en el momento en que alguna orden es enviada a la parte RT.

Parte RT

La parte RT comprende aquellas tareas cuya ejecución ha de ser precisa en lo que al tiempo se refiere. Éstas son la tarea de lectura de medidas de los sensores y la de actuación de los motores.

Análisis del entorno

La tarea que recibe información de los sensores es crítica debido a que es necesario conocer, en todo momento, si existe algún objeto u obstáculo cerca-

no al vehículo que pueda interferir en su marcha. No se puede esperar una ejecución tardía de la tarea en cuestión, porque si esta se retrasa, y no se detecta algún objeto que esté demasiado cerca, el vehículo correrá el peligro de colisionar con el mismo.

En realidad esta tarea debe estar activa desde que entre en funcionamiento el control, ya que es imprescindible tener siempre información del entorno sea cual sea el estado del vehículo. Esto posibilitará una toma de decisiones congruente mediante la ejecución de algoritmos previos a la realización de las acciones de movimiento. Se trata de una primera decisión de diseño, que en la práctica supone estar recogiendo continua y periódicamente las medidas de los sensores.

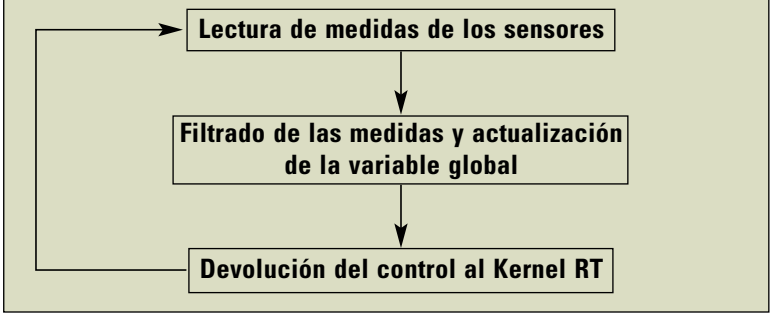
El objetivo principal es, por tanto, proporcionar de forma periódica una información robusta del entorno del vehículo a partir de las señales de medida que ofrecen los sensores de que disponga el vehículo.

El inicio de la ejecución debe tener lugar en el momento de carga del módulo RT y no se detendrá hasta que se elimine el dicho módulo.

No se requiere ningún parámetro para el correcto funcionamiento de la tarea.

Como resultado de la ejecución se obtendrán una serie de datos que se almacenan en forma de variable global. Cada uno de ellos debe corresponder a la medida de un sensor al que unívocamente quedará ligado. La relación

FIGURA 7. DIAGRAMA DE FLUJO DE LA TAREA DE LECTURA DE LOS SENSORES



quedará determinada tanto por el montaje del vehículo como por el programador del código.

Como se ve en la Figura 7, el proceso de análisis del entorno es sistemático y repetitivo, se trata de recoger las señales ofrecidas por los sensores y obtener una medida robusta mediante alguno de los múltiples algoritmos de filtrado existentes.

La tarea RT finaliza su ejecución devolviendo el control del sistema al kernel RT para que éste continúe organizando la ejecución de otros procesos con de menor prioridad, por ejemplo los procesos Linux no RT, si es necesario hasta que llegue el siguiente periodo de ejecución de la tarea.

Motores

La tarea que controla la actuación de los motores también es crítica por razones obvias. Si se retrasase la ejecución de esta acción no se tendría un control real del vehículo sino que, una vez dada la orden de actuación se debería esperar un tiempo indeterminado antes de ejecutarla. Esto ocasionaría desvíos de la ruta fijada, colisiones, giros imprecisos… Al declarar esta tarea como de tiempo real se tiene la posibilidad de controlar con precisión al menos su inicio (a través de un Manejador, capaz de iniciar la ejecución de una tarea al introducir un dato en la cola FIFO asociada), y tener perfectamente acotado el máximo error de movimiento al finalizar la ejecución. El control del vehículo será tanto mejor cuanto menor sea el periodo de actuación de esta tarea.

El cometido de este proceso es, por tanto, controlar con precisión el envío de órdenes de funcionamiento a los motores. Además posee la capacidad de requerir la detención del vehículo, inmediatamente y sin necesidad de recibir una orden del control, en el caso de que exista algún obstáculo con el que se pueda colisionar.

El sistema de control propuesto para los motores es temporal, es decir, cuando se desea recorrer una distancia, la tarea Linux de control calcula el tiempo que deben estar activos los motores y el sentido de giro, codifica esta información y la envía a la parte RT a través de la cola FIFO1 para que el Manejador (ver Figura 6) inicie la ejecución de la tarea y cargue los datos

Nuestra visión es ser el aliado de nuestros clientes en la nueva economía, lo que nos hace evolucionar continuamente para contribuir a su éxito y ayudarles a alcanzar sus objetivos.

Arthur Andersen ofrece a sus profesionales una actividad constantemente innovadora, proyectos vanguardistas en empresas líderes, alta especialización sectorial (con especial incidencia en finanzas, telecomunicaciones y energía), un excelente ambiente de trabajo, y una formación sólida y continuada.

Déjanos conocerte y podrás participar en un proyecto fascinante que contribuirá a tu desarrollo personal y profesional.

Cada uno de nuestros retos será una oportunidad para tí.

www.arthurandersen.es

ARTHUR ANDERSEN

necesarios. Una vez rebasado el tiempo de activación de los motores, la tarea de actuación da por completada la ejecución y se detiene el movimiento.

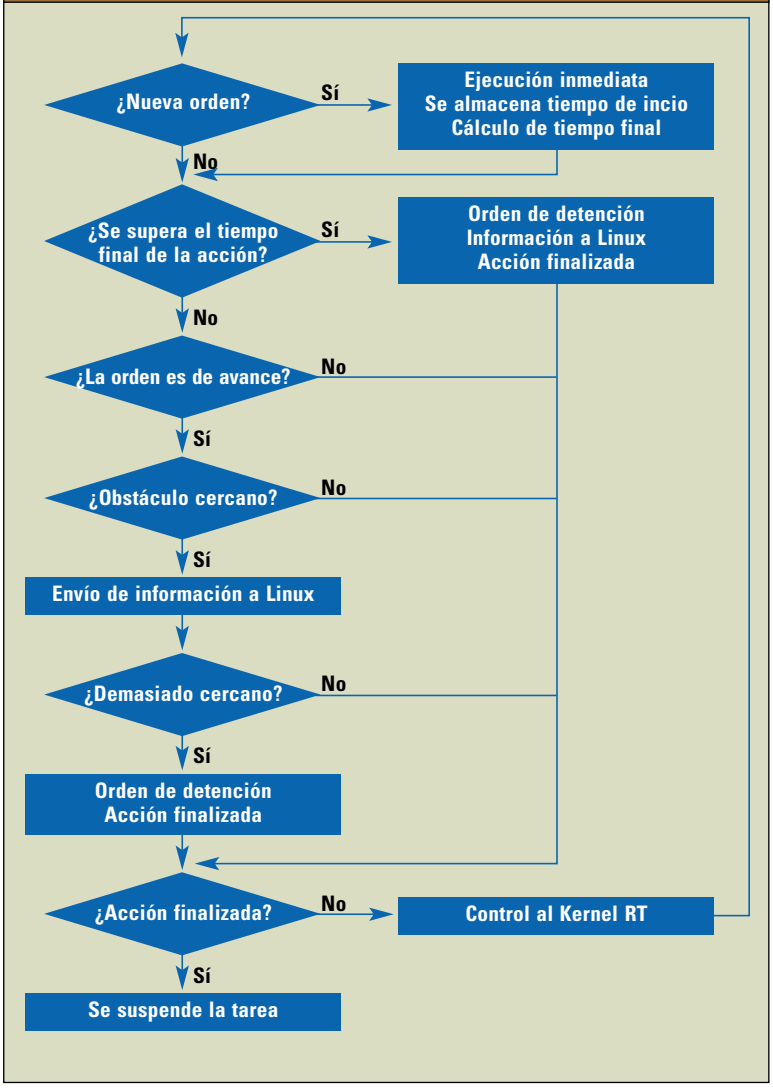
Por eso resulta especialmente crítica la elección del periodo de ejecución de la tarea, cuanto menor sea, más precisos serán los movimientos en general. La programación de esta función resulta poco compleja en lo que a número de instrucciones se refiere y, por tanto, su tiempo de ejecución en cada periodo no resulta demasiado largo. Esto favorece al sistema, pues en primer lugar supone que se puede reducir mucho el periodo de esta tarea y, en segundo lugar, permite una gran disponibilidad del procesador para la tarea de lectura de los sensores y el algoritmo de control no RT.

La prioridad de este proceso será la mayor del sistema de tiempo real, pues es considerado más crítico que el de recogida de información del entorno. Es imprescindible que el control de movimiento sea muy preciso.

El inicio de la ejecución de esta tarea tiene lugar en el momento en que el Manejador (ver Figura 6) recibe una orden del algoritmo de control (Linux no RT) y detecta que es una petición de movimiento.

La acción finaliza en el momento en que se rebasa el tiempo de giro estimado para los motores en la orden que se ejecuta, por tanto, el mayor retraso posible a la hora de detener la actuación de los motores coincide precisamente con el periodo de ejecución de la acción.

FIGURA 8. DIAGRAMA DE FLUJO DE LA TAREA DE ACTUACIÓN DE LOS MOTORES



Las acciones que se pueden llevar a cabo deben estar perfectamente definidas por el programador y dependen del montaje del vehículo.

El diagrama de flujo de esta tarea RT es el siguiente:

En la Figura 8 se puede observar que si existe una orden nueva, comienza a ejecutarse la misma, se almacena su tiempo de inicio y, añadiéndole la duración de la acción, se puede predecir su final. Si se ha superado el tiempo que marca el final de la acción simplemente se avisa a la parte encargada del control del vehículo y se suspende la tarea de actuación de los motores. En caso contrario se analiza el movimiento que se está llevando a cabo y se envía información del entorno a la parte no RT en el supuesto de que exista riesgo de colisión. Si

ésta es inminente se opta por detener el vehículo inmediatamente. Únicamente puede existir peligro en los movimientos de avance, pues se considera que el vehículo tiene la capacidad de girar sobre sí mismo, sin que se dé la posibilidad de un impacto.

Si la acción no ha finalizado al terminar el proceso, simplemente se devuelve el control al kernel RT hasta el comienzo del nuevo periodo de ejecución.

Parte no RT

Una vez se han determinado cuáles son las acciones críticas del robot, se definen aquellas que no necesitan ejecutarse en tiempo real. Estas son las tareas Linux no RT y se trata, principalmente, de algoritmos que calculen las pertinentes órdenes de actuación teniendo en cuenta la información del entorno recogida por los sensores; es decir, las tareas de planificación de trayectorias y de modelado del entorno.

Esta tarea no es crítica ya que su tiempo de ejecución será variable y además puede disponer del microprocesador todo el tiempo sobrante entre los periodos de las otras tareas, que no es poco, para precalcular movimientos y rutas posibles. En realidad se trata de la parte más flexible del control, y es muy variable en función del usuario o programador que actúe sobre ella y de la finalidad que persiga el vehículo autoguiado.

Se pueden utilizar todos los recursos que ofrezca el Software desarrollado para Linux unido a la posibilidad de utilizar cualquier periférico adicional para plataformas de tipo PC, esto posibilita la grabación de imágenes, transmisión de datos a través de una conexión existente y/o cualquier otra solución que pueda surgir.

Comunicación (colas y Manejador)

Ahora sólo queda definir correctamente la comunicación entre las tareas RT y Linux. En este caso se propone que las tareas RT estén incluidas en un mismo módulo, de modo que la posible comunicación entre la tarea de lectura de los sensores y la de actuación de los motores sea instantánea a través de variables globales. Por el contrario, la parte Linux siempre supone un módulo distinto a los de tiempo real, y por ello la comunicación debía realizarse a través de colas FIFO.

Dado que la tarea de control debe tomar decisiones teniendo en cuenta el entorno, una de las colas (FIFO1) debería ser capaz de hacer llegar órdenes de petición de medidas a la tarea de sensores, pero también órdenes de actuación a la tarea de actuación de los motores. Pudiera ocurrir que estas órdenes fueran críticas y que exigieran respuesta inmediata, como por ejemplo la orden de parada si el vehículo está a punto de chocar.

A tal fin se dispone un Manejador para esta cola, que como se ha dicho, se trata de una función que se ejecuta cada vez que está accesible un dato de la cola a la que está ligada. Esta función será capaz de discriminar hacia que tarea RT se dirige la orden, y si ésta es de actuación llevará a cabo una instrucción para que se ejecute inmediatamente, en caso contrario se tratará de una petición de medida, que será satisfecha devolviendo la información contenida en la variable global correspondiente.

Por supuesto, una vez que las órdenes procedentes de Linux han sido atendidas se tendrá que informar de sus resultados para que mediante el algoritmo de control se sigan tomando decisiones. Esta comunicación se realiza a través de dos colas más.

Una de ellas (FIFO2) informará del estado del vehículo, esto es, si se ha completado la acción ordenada o si ha tenido que interrumpirse por alguna causa y su manejo corresponde a la tarea de actuación.

La otra cola (FIFO3) comunicará las medidas tomadas por los sensores, y será controlada por el Manejador que esta ligado a la cola de transmisión de órdenes (FIFO1). En el momento en que se recibe la orden de recoger la medida de los sensores el Manejador colocará en la cola de respuesta a Linux (FIFO3) la última medida tomada por los sensores (que está almacenada en una variable global del módulo RT), como se muestra en la Figura 9.

Como se aprecia en la figura, si el Manejador recibe una petición de operación de los motores ("acción"), se cargan los datos correspondientes a la misma en la variable global correspondiente y se lanza la nueva actuación independientemente de que la anterior operación haya finalizado.

Conclusiones

Se ha desarrollado un sistema de control muy barato, gracias a que está basado en microprocesadores convencionales y en el sistema operativo Linux. Además posee la capacidad de ejecutar las tareas críticas del control en tiempo real, gracias a RT-Linux, como son la adquisición de medidas y la ejecución de ordenes de actuación.

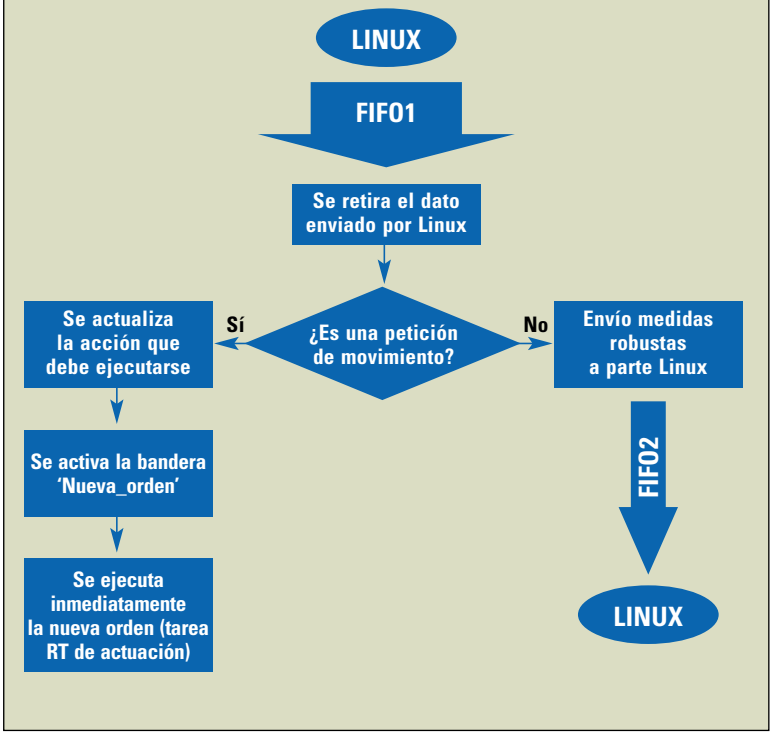
Futuros desarrollos

Una vez que se tiene desarrollado el esqueleto del sistema de control, se puede dotar de cierta inteligencia al vehículo.

A este respecto se están desarrollando modelos de entorno basados en técnicas de inteligencia artificial aplicados a algoritmos de planificación y localización sofisticados.

Además se está desarrollando un sistema de guiado por voz basado en técnicas de inteligencia artificial y un sistema de guiado autónomo por radiofrecuencia muy económico, lo que permite quitar el PC de control del AGV, con el consiguiente ahorro de espacio y de energía.

FIGURA 9. DIAGRAMA DE FLUJO DEL MANEJADOR



Agradecimientos

Los autores agradecen la colaboración de la UPCO-ETSI en la aportación de herramientas para poder desarrollar el sistema.

Bibliografía

[1] Robot motion planning. Jean-Claude Latombe. 1991 by Kluwer Academic Publishers.

[2] Integration, coordination and control of multi-sensor robot systems. Hugh F. Durrant-Whyte. Kluwer Academic Publishers. 1988.

[3] Running Linux. Matt Welsh adn Lar Kaufman. O'Reilly & Assocites, Inc. 1996.

[4] The RTLinux Manifesto. Victor Yodaiken. Department of Computer Science. New Mexico institute of tecnology. yodaiken@cs.nmt.edu.

[5] Diseño de un vehículo autoguiado controlado mediante un ordenador personal. Álvaro Sánchez Miralles, Gonzalo Alonso de Ozalla. UPCO-IIT 2000.

[6] El Lenguaje de Programación C. Brian W. Kernighan and Dennis M. Ritchie. Prentice Hall Hispanoamerica, S. A. 1992.