Escuela Superior de Ingenieros.

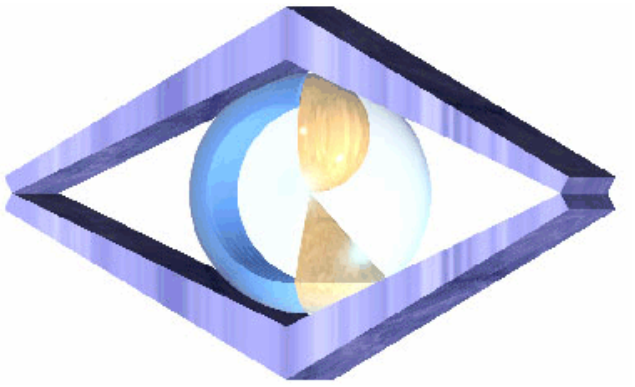
Universidad de Sevilla.

Ingeniería de Telecomunicación.

Proyecto de Final de Carrera.

Memoria.

Automatización de una Carretilla Elevadora.



Eduardo Gómez – Elegido Pedraza.

Índice.

Capítulo 1: Introducción.

1.1.- Antecedentes.

1.1.1.- El proyecto.

Este proyecto se engloba dentro del proyecto más general denominado "Robótica Móvil con Múltiples Articulaciones", financiado por la Comisión Interministerial de Ciencia y Tecnología (CICYT).

En este proyecto se desarrollan nuevos métodos y tecnologías para el control de robots móviles con componentes articulados. Se consideran tanto vehículos articulados como vehículos convencionales con elementos articulados a bordo entre los que se encuentran sistemas para orientación de cámaras, dispositivo de carga y descarga y brazos articulados. El proyecto integra cuatro subproyectos diferentes. En el primero, se investiga en robots móviles con elementos multiarticulados y sus aplicaciones en maquinaria forestal, guiado de vehículos pesados y vehículos de exploración. El subproyecto 2 se dedica específicamente al control de vehículos autónomos articulados. El subproyecto 3 es de carácter más básico, tratándose métodos de teoría de control para el seguimiento de trayectorias y maniobra con incertidumbres en el modelo. Finalmente, el subproyecto 4 se dedica a vehículos articulados para la exploración.

La memoria que ahora nos ocupa relata el fruto del trabajo realizado entre principios de septiembre de 2001 y finales de octubre de 2002 en el laboratorio del Instituto de Automática y Robótica, situado en las dependencias de los laboratorios del Departamento de Ingeniería de Sistemas y Automática de la Escuela Superior de Ingenieros de Sevilla. El proyecto se fraguó inmerso en el Grupo de Robótica, Visión y Control, trabajando en él como becario, bajo la tutela de D. Guillermo Heredia Benot.

1.1.2.- Generalidades.

Las carretillas elevadoras son, en la actualidad, un tipo de máquina muy utilizado en entornos industriales, con el objeto de trasladar mercancía de un lugar a

otro, dentro de una factoría, o bien para cargar vehículos con mercancías procedentes de la fábrica en vehículos de transporte destinados a trasladar dichas mercancías al exterior de la fábrica, es decir, a otras fábricas, o bien a almacenes para su distribución comercial. Precisamente en estos almacenes son también muy importantes estos vehículos, realizando un cometido muy similar al que realizan en las fábricas.

Para su traslado, las mercancías se almacenan en lo que se llaman palés, que no son más que unos soportes, de madera generalmente, cuya finalidad es mantener separada la carga del suelo, permitiendo de esta forma ser enganchados por maquinaria para su transporte. Esta situación es ideal para las carretillas, ya que son diseñadas especialmente para este traslado.

Hasta ahora, las carretillas elevadoras han sido manejadas por un operador humano, que va montado en la máquina y se mueve con ella accionando a voluntad distintos mandos ubicados en la cabina de la carretilla, que le permiten manejarla en cuestiones como la dirección que toma la carretilla en su movimiento, su velocidad, y otras variables características de la carretilla, como son la elevación de las palas de la carretilla, y la inclinación del mástil de la misma. Para accionar estos mecanismos, el operador dispone de una serie de mandos en la cabina, es común un volante para la dirección, y pedales para la velocidad y el freno. Para accionar todo lo relacionado con el mástil, se tiene una o varias palancas o joysticks, que controlan inclinación del mástil y altura de las palas a voluntad del conductor de la máquina. Así, con este procedimiento, se puede llevar la carretilla hasta donde se encuentra el palé que se desea trasladar, cogerlo con las palas, metiéndolas debajo del mismo, y levantándolas e inclinándolas hacia la carretilla, llevar la mercancía hacia otro punto de la fábrica, y, controlando la inclinación y la elevación de las palas, depositar el palé en otro lugar de la factoría.

En cuanto al vehículo, a la máquina en sí, hemos de decir que suelen ser máquinas pesadas, ya que tienen la tarea de llevar grandes masas de mercancía, observándose que no las llevan sobre el centro de gravedad del vehículo, sino desplazadas al lugar donde se encuentran las palas. Se necesita por esto que la carretilla tenga un peso suficiente para contrarrestar el peso de la mercancía transportada, sin vencerse hacia la carga por efecto del peso de ella. Referido a los medios de propulsión de las carretillas, diremos que las hay con propulsión eléctrica, existiendo también carretillas de propulsión mediante combustible, en general diesel. Por regla general, el tipo de propulsión está relacionado con el peso máximo que pueden transportar, así, las

diesel están destinadas a palés más pesados que las eléctricas. Para mover la dirección y todo lo relacionado con el mástil, se tiene un sistema hidráulico, con una bomba que mueve el fluido confinado en el interior del circuito hidráulico, que por efecto de una serie de válvulas, se mueve hacia los cilindros que accionan tanto la dirección, como el movimiento del mástil.

Otro aspecto interesante que cabe destacar de las máquinas que nos ocupan es que su movimiento suele estar restringido al interior de un cierto entorno, una fábrica, pudiendo moverse libremente en su interior. Por otra parte, no se necesita una actividad de dichas carretillas fuera del entorno de la fábrica, ya que todo su trabajo lo realizan dentro de dicho entorno. Así, podemos decir que se mueven en un entorno más o menos conocido o estructurado.

1.1.3.- Una aproximación a la robótica móvil.

Decíamos anteriormente que las carretillas han sido manejadas por un operador humano en su gran mayoría. Pero eso no es del todo cierto, existen antecedentes de vehículos de estas características, denominados AGV´s (Automatic Guided Vehicles), en los que se aprovecha la característica de trabajo en un entorno estructurado para mover estos vehículos de manera parcialmente autónoma. Así, se tiene un conjunto de vehículos de este tipo, y se mueven dentro de su entorno de trabajo sin un conductor que los maneje, al dictado de un ordenador que les transmite órdenes del tipo “lleva el palé A del punto B al punto C”. Esto resulta muy interesante, porque, sin la ayuda de un operador humano, se realiza un trabajo con estos vehículos, ahorrando dinero, y pudiendo llevar a cabo trabajos de traslado de mercancías peligrosas para el operador, así como trabajos en entornos agresivos, o, simplemente, evitando el peligro de trabajar con este tipo de máquinas al hombre.

Pero estas máquinas no son del todo un robot, ya que en la esencia de su funcionamiento se encuentra el guiado automático mediante procedimientos relativamente sencillos, como puede ser el de seguir unos raíles, bien pintados en el suelo, bien en forma de cables metálicos enterrados en el suelo, y por los que pasa una corriente, y otros métodos similares. En todos estos casos, el vehículo no se mueve de forma autónoma, sino siguiendo un camino ya trazado, y el procesamiento que necesita para su movimiento es relativamente pequeño. Se necesita, eso sí, algún tipo de sensor que mande información al control de la carretilla de su posición relativa al raíl que ha de

seguir, y en el caso de las carretillas, sensores que permitan al control conocer la posición de las palas y la inclinación del mástil. Para conocer la posición respecto al raíl se han usado sensores de tipo magnético, que detectan la variación del campo producido por una corriente que circula por el raíl, y, conforme a ese valor, conocen la desviación de la carretilla respecto al mismo. También se han usado otros sensores de tipo óptico, para guiar la carretilla cuando el raíl está pintado en el suelo. Sensores láser o sensores que detectan la variación de la intensidad de la luz recibida tras ser emitida por una fuente y rebotar en el suelo marcado por los raíles, además de las cámaras de vídeo, más caras y con un procesamiento mayor, son sensores de este tipo.

Estas máquinas no parecen muy inteligentes. La inteligencia del sistema reside principalmente en el ordenador que controla a todas las carretillas, y les da consignas de la siguiente operación a realizar. Este ordenador ha de establecer tareas para las máquinas que se encuentren inactivas, mandarles el trabajo a realizar, planificar la manera óptima, mediante algún criterio concreto, de cumplir las tareas a realizar, y asignar algún tipo de prioridad entre tareas y máquinas, para que no se interfieran en su trabajo. De todas maneras, este tipo de sistema presenta una serie de dificultades:

- La rotura de los cables magnéticos o raíles, puede ser difícil y lenta de detectar, produciendo retrasos en el correcto funcionar de la fábrica, y retrasos considerables en el proceso productivo de la misma.
- Los vehículos y sus zonas de trabajo han de ser limpiados de manera regular, ya que si no se puede dar lugar a un mal funcionamiento de los sensores, por efecto de la suciedad depositada. En un ambiente industrial, a veces es difícil mantener la limpieza adecuada para este propósito.
- La planificación de las rutas ha de ser cuidadosa, debido a que esas rutas son realizadas de manera física en cables, y difíciles y lentas, por tanto, de modificar.
- La implementación es especialmente difícil en áreas de baja maniobrabilidad. Este caso es común cuando se habla de carretillas, ya que en el momento de recoger un palé, la zona de maniobra es pequeña.
- Cuando el AGV pierde la banda de guiado, el raíl, no puede recuperarla de nuevo sin ayuda manual, un problema especialmente crítico en superficies deslizantes. El fluido hidráulico con el que funcionan es un aceite que se puede derramar por avería, o al reponerlo o cambiarlo. Este fluido hace que el suelo se vuelva extremadamente deslizante.

Con todos estos problemas, lo que pretende minimizar la presencia humana parece que no lo consigue del todo.

Pese a todos estos inconvenientes, actualmente se encuentra implantado este tipo de sistema, o similar, en algunas fábricas en España, incluso lleva ya tiempo trabajando, con buenos resultados. Como ejemplo de implantación de este sistema está la fábrica de Tetra Brick de Arganda del Rey en la provincia de Madrid.

1.1.4.- Robótica móvil aplicada a carretillas.

A partir de los anteriores AGV´s, cabe pensar en dotar a los vehículos de una inteligencia mayor, es decir, que estos vehículos no se remitan sólo a ser guiados conforme a una serie de estructuras a modo de raíles, sino que sean capaces de moverse de manera libre, por lo menos en un entorno conocido, eliminando gran parte de los problemas que se producen cuando se usan los AGV´s.

De esta manera surgen proyectos de automatización en universidades españolas, los de la Universidad de Valladolid, como se puede leer en los artículos "Automatic and Flexible Transport System Based on Mobile Robots" (G. Méndez y otros autores, 1995) y "Mobile Robot for an Industrial Environment" (A. Lara y otros, 1995); el de la Universidad de Alcalá, "Automatización de una Carretilla Industrial" (Manuel Mazo y otros, 2000) y otros artículos que hablan de sensores y de navegación de la carretilla en entornos balizados.

En todos estos sistemas expuestos, la carretilla se mueve en un entorno conocido, convenientemente balizado, y necesita una serie de sensores que informen a su controlador de su posición exacta dentro del entorno. Vemos que con esto no se alcanza una total posibilidad de movimiento autónomo, pero sí que se consigue ese movimiento dentro del entorno conocido de trabajo, al fin y al cabo, no deseamos otra cosa que eso.

Parece lógico pensar que el hecho de que la carretilla tenga un movimiento más inteligente y autónomo dentro del entorno hace necesario un conocimiento mayor del mismo, y un procesamiento mayor por parte del control del vehículo. Así, se necesitan más sensores, y más sofisticados, y controladores mejores, tanto en cuanto a la capacidad de procesamiento, es decir, a la máquina que realiza la labor de control, como en cuanto a la complejidad de los algoritmos de control empleados para el guiado de la carretilla. En los casos de los vehículos anteriormente citados, esto se resuelve con

controladores implementados con un ordenador situado en el propio vehículo, así como con sensores destinados a la navegación en un entorno balizado. Posteriormente profundizaremos en todos estos conceptos.

Viendo todo esto da la impresión de que hay ya muchas cosas hechas en este campo, pero siempre se puede avanzar en la realización de controladores más pequeños, de algoritmos de control mejores, así como procurar, a pesar de que las carretillas estén dotadas de una inteligencia mayor, establecer una forma de relación entre distintas carretillas automáticas, para combinar sus esfuerzos en una tarea común. Se trataría de planificar estas tareas, pero con una mayor libertad para las carretillas implicadas en el sistema, necesitándose, como es lógico, varias carretillas automáticas. El hecho de que las carretillas sean más autónomas, minimizaría con mucha probabilidad el número de las mismas para igualar las prestaciones de un sistema de AGV´s, ya que el no tener restricciones en su movimiento hace que se puedan realizar los trabajos más rápido, y podría abaratar en algunos casos el sistema.

1.2.- Objetivos del proyecto.

1.2.1.- Situación inicial.

Para establecer de manera real y no demasiado ambiciosa los objetivos del proyecto, es necesario conocer la situación inicial del mismo, es decir, de lo que se disponía al iniciarse el proyecto, y en el estado en el que se encontraban los distintos dispositivos y máquinas implicadas en el mismo, siempre referido este estado a su utilización para el fin del proyecto. Aunque posteriormente se tratarán todos estos dispositivos con mayor profundidad, sirva esta introducción como resumen.

Este proyecto tiene como objetivo la automatización de una carretilla elevadora, por lo tanto, como es obvio, existe una carretilla objeto de la automatización. La carretilla es eléctrica, de marca Linde, y modelo E 16C. Es una carretilla con tracción producida por un motor eléctrico, y movimiento de la dirección, así como de las palas y el mástil, producido por un sistema hidráulico. En principio, las válvulas que gobernaban el sistema hidráulico eran manuales, pero se hubo de proceder a un

duplicado del sistema hidráulico, para permitir el control eléctrico de las válvulas por parte del controlador. ¿Por qué duplicación, y no sustitución? La duplicación se llevó a cabo debido a la necesidad de mantener un funcionamiento manual en la carretilla. Se hubo también de montar los mecanismos que permitieran conmutar entre ambos circuitos hidráulicos, el manual y el automático. Posteriormente hablaremos con mayor profundidad de todo esto.

Hemos hablado ya de un controlador, con el que actuar en la carretilla, pero, ¿qué controlador? Lo primero que hemos de decir es que el controlador tiene una parte hardware y una software.

La parte hardware está compuesta por un ordenador PC, con procesador Intel Pentium MMX a 200 MHz, pero tiene una característica especial respecto a los ordenadores comunes de escritorio, y es el formato que tiene. Este formato responde al estándar PC104, que permite la implementación del controlador en un reducido espacio físico, ya que este estándar da lugar a ordenadores más pequeños que los comunes. Necesitábamos también distintas formas de comunicar el ordenador con la carretilla y los sensores, por eso se conectan al ordenador varias tarjetas digamos que de interacción con el entorno, como tarjetas de control de motores, tarjetas de entrada y salida de datos, tanto analógico, como digital, tarjeta de puertos serie, de temporización y tarjeta de red. El ordenador estaba sin montar, hubo que montarlo y completar lo que le faltaba, disco duro, etcétera.

De la parte software hemos de decir que el ordenador debía de llevar instalado el sistema operativo Linux, en su distribución Debian. El motivo de instalar este sistema, aparte de las ventajas propias de Linux, de las que ya hablaremos, es que se pretende que la carretilla funcione con el software desarrollado para el vehículo Romeo4R bajo este sistema operativo, debidamente modificado. Para una mayor información sobre todo este software, se ha de consultar el proyecto final de carrera de Rafael Martín de Agar, del año 2002, en el Dpto. de Ingeniería de Sistemas y Automática de la Escuela Superior de Ingenieros de Sevilla.

Por último, debemos hablar del sensor de navegación Robosense de la empresa Siman, se dispone de él para ser instalado en la carretilla, y determina la posición del sensor en un entorno balizado, así como la orientación del mismo respecto a un eje x determinado por las balizas. En un principio, el sensor no tenía mucha precisión hasta pasado un periodo de calentamiento de unas dos horas, y se hubo de solucionar este problema. Posteriormente también dedicaremos más tiempo a este sensor. Obviamente,

el sensor se monta en la carretilla, de manera que determine la posición y orientación de ésta. Las balizas son cilíndricas, recubiertas de un material reflectante, y el sensor dispone de un computador en su interior, que le permite calcular su posición mediante algún tipo de algoritmo de triangulación.

1.2.2.- Ahora sí, objetivos del proyecto.

El proyecto que nos ocupa tiene el objetivo de la automatización de la carretilla de la que ya hemos hablado, es decir, se ha de lograr que se mueva de manera autónoma dentro de un entorno balizado, este objetivo no es un objetivo claro o cerrado del todo, ya que el hecho de que la carretilla se mueva de manera autónoma tiene distintos niveles de realización, distintos sensores a integrar para recoger datos del entorno y navegar en él, y muchos algoritmos de control que se pueden realizar. De hecho, si la carretilla se moviera en bucle abierto, es decir, sin conocimiento alguno del exterior, sólo con algún tipo de consigna recibida del controlador, se estaría moviendo autónomamente, sin intervención humana, otra cosa sería el probablemente desastroso comportamiento ocasionado por este movimiento.

Así, teniendo en cuenta los sensores con los que podemos contar, debemos ver que el objetivo del proyecto es procurar un movimiento autónomo de la carretilla, controlada con un software instalado en el ordenador de que disponemos, con sistema operativo Linux, apoyándonos en el único sensor que tenemos, el láser Robosense.

Para realizar todo esto, debemos estudiar el funcionamiento de la carretilla, de las electroválvulas instaladas en la carretilla, entendido esto como las señales eléctricas que gobiernan todo esto, así como fabricar los circuitos necesarios para la adaptación de las señales del controlador a la carretilla. Se ha de montar también el ordenador de control, probarlo, instalar en él el sistema operativo Linux, programar en él los distintos drivers de las tarjetas de control que se utilizan, así como el software de control y el de comunicación con el sensor Robosense, software de procesado de los datos del sensor, algoritmos de control, etcétera. Se ha de montar también el sensor Robosense, y balizar el entorno en el que se realizarán las pruebas. Por último, se ha de procurar diseñar y realizar circuitos que aseguren la alimentación de los distintos dispositivos implicados, así como la conmutación entre el modo manual y el automático.

Si analizamos cada tarea veremos que la tarea de análisis del funcionamiento de la carretilla es una de las que más tiempo nos llevará, es esto debido a que no se dispone

de información muy detallada en los manuales de la misma sobre su funcionamiento, ya que no están orientados a ello, sino más bien al recambio de piezas y al mantenimiento. Esta tarea no da lugar a ningún trabajo que se pueda plasmar en algo físico, realizado, pero es totalmente crucial para el diseño de los distintos circuitos que necesitaremos, así como de la forma de control, de actuación y de otros detalles.

Acerca de las otras tareas diremos que en ciertos momentos se verán entorpecidas por problemas como averías de dispositivos, o ausencia de material necesario para continuar esa tarea, problemas totalmente típicos del trabajo con cuestiones de hardware.

En los siguientes capítulos, se describirán todas estas tareas con un grado mayor de detalle.

Capítulo 2: Fundamentos teóricos.

2.1.- El vehículo robótico en sus modelos cinemático y dinámico.

2.1.1.- Descripción.

En nuestro proyecto se dan varios elementos objeto de un estudio teórico, y la carretilla es uno de ellos. Para la realización de este estudio hemos de dejar de lado consideraciones técnicas propias de la carretilla, aunque sí tendremos en cuenta algunas características como la configuración de la carretilla, que tiene cuatro ruedas, tracción delantera, y la dirección proporcionada por el movimiento de las ruedas traseras. Centrándonos en este sistema, veremos ahora, y dada esta configuración, los distintos modelos que se usarán para la descripción del sistema, modelos cinemático y dinámico, y también consideraciones sobre alguna estrategia de seguimiento de caminos.

Consideraremos una serie de hipótesis básicas:

- El robot se mueve sobre una superficie plana.
- Los ejes de guiado son perpendiculares al suelo.
- Supondremos una situación ideal de rodadura, sin deslizamiento en un periodo de control.
- Sin partes flexibles en el robot.
- Durante el período suficientemente pequeño en el que se mantiene constante la consigna de dirección, el vehículo sigue en su movimiento hasta el siguiente punto a considerar un arco de circunferencia.
- El robot se comporta como un sólido rígido, de forma que si existen partes móviles, estas se situarán en la posición adecuada, mediante el sistema de control.

Podemos ver que en el caso de la carretilla, se cumplen todas estas hipótesis con bastante rigurosidad, desde luego, los suelos de las fábricas son, en general, planos, su gran peso no predispone a la carretilla para el deslizamiento, es rígida y sin partes

flexibles, y tiene una configuración tal que sigue un arco de circunferencia en períodos cortos de tiempo.

Pero la carretilla dispone, además, de otros dos ejes para su movimiento, los pertenecientes al mástil. Este mástil tiene una especie de configuración cilíndrica, con dos articulaciones, y el eje del cilindro, en el eje de giro del mástil, pero con poca libertad en su movimiento, unos pocos grados, y la coordenada radial la representa la altura de las palas. Aunque no parece necesario hablar mucho más sobre el sistema del mástil, debido a que realmente, no interesa llegar a un punto con una cierta inclinación, sino llegar de manera totalmente vertical, y al tener la carga depositada en las palas, inclinar el mástil al máximo hacia la cabina para que no se caiga.

Para las descripciones teóricas se utilizará la notación existente en el libro "Robótica, Manipuladores y Robots Móviles" (Aníbal Ollero, 2001). Para una mayor información sobre la notación y otras cuestiones de los modelos, se puede consultar en este libro.

2.1.2.- Restricciones cinemáticas.

Sea $p = [p_1 ... p_r]^T$ un vector de las variables necesarias para determinar la posición y orientación de un robot. Estas variables no son independientes en el movimiento del robot, algunas de estas variables están sujetas a restricciones que se han de cumplir debido a una fuerza que obligue a su cumplimiento, como pueden ser las fuerzas que mantienen unido el vehículo, así como restricciones de otros tipos.

Nos podríamos quedar con un número mínimo de variables independientes, que son las que describen el movimiento, ya que las otras van unidas a las primeras mediante restricciones, que pueden ser de dos tipos, holónomas y no holónomas.

En las restricciones holónomas, no intervienen las velocidades, así que tienen la forma:

$$G_k(p,t) = 0; \quad k = 1,...,s;$$

Las no holónomas, dependen de las velocidades, pero no sólo de manera sencilla, sino que no han de ser integrables, es decir, que no se deduzcan por derivación total de una restricción holónoma con respecto al tiempo. En los robots móviles existe este tipo de restricciones. Consideremos el movimiento de la rueda en la figura 2.1. La rueda, de radio r, se mueve en una dimensión, la variable del actuador podría ser el giro

θ, y la variable x, la que indica el espacio recorrido. Podemos establecer condiciones entre estas variables.

$$x' = c\theta'$$

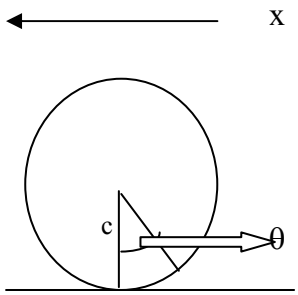


Figura 2.1: Movimiento de la rueda en una dirección.

Pero si observamos esta ecuación, se puede integrar, obteniéndose:

$$x - c\theta = constante$$

Lo cual nos hace deducir que la restricción es holónoma. Observando el mismo movimiento, pero desarrollándose en un plano, como en la figura 2.2:

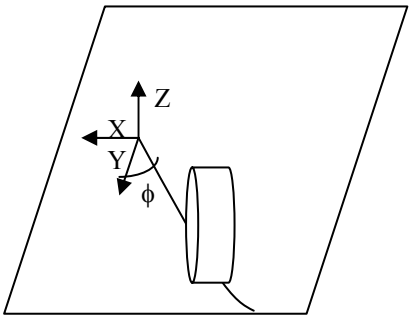


Figura 2.2: Movimiento de la rueda en el plano.

Podemos ver que en este caso aparecen restricciones no holónomas, si consideramos que el diámetro de la rueda correspondiente al punto de contacto con el suelo está siempre en posición vertical, tendremos cuatro coordenadas para expresar completamente la posición de la rueda, así como su orientación. Estas coordenadas son las del punto de contacto, (x,y); el ángulo θ entre la vertical y un radio de referencia, indicando así cuanto ha girado el disco, como se puede ver en la figura 2.1; y el ángulo ϕ de orientación de la rueda. Tenemos que tener en cuenta ciertas restricciones de las ruedas. La condición de rodadura sin deslizamiento impone que el espacio que el punto de contacto recorre sobre el borde de la rueda es igual al que recorre sobre el plano. Podemos, con todo esto, proyectar la velocidad del punto de contacto en los ejes del plano:

$$-x'\operatorname{sen}\phi + y'\cos\phi = c\theta'$$

$$x'\cos\phi + y'\operatorname{sen}\phi = 0$$

Estas dos restricciones no son integrables, el significado físico de esto es que si hacemos girar la rueda sobre su eje, sin deslizar, así como sobre la vertical, podemos llegar a cualquier valor de las cuatro variables, con lo que no encontramos ninguna relación funcional entre ellas.

Si hablamos de las velocidades, hemos de determinar sólo dos variables de las cuatro que tenemos, (x´,y´,θ´,ϕ´), siempre que la rueda no deslice, estas son ϕ´ y cualquiera de las otras tres variables. El sistema tiene en este caso dos grados de libertad. Podemos decir que en el caso de las restricciones no holónomas, en general, el número de grados de libertad es el número de variables, p, menos el número de restricciones, r.

Existen distintos tipos de ruedas, fijas, de direccionamiento, de castor y suecas.

2.1.3.- Modelo de la bicicleta.

Este modelo es, como resulta obvio, el de un vehículo de dos ruedas. Podemos ver su configuración en la figura 2.3.

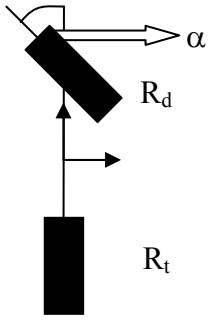


Figura 2.3: Modelo de la bicicleta.

Podemos considerar un sistema de referencia {G} y un sistema {L} con centro en el punto de guiado del vehículo, y con el eje $\hat{Y}_L$ en la dirección del eje longitudinal del vehículo. El vehículo tendrá un ángulo de orientación ϕ con el eje $\hat{Y}_G$ del sistema de referencia principal.

Podemos suponer, si el período de control es suficientemente pequeño, que el vehículo se desplaza en ese intervalo según un arco de circunferencia del denominado círculo osculador, lo podemos ver en la figura 2.4.

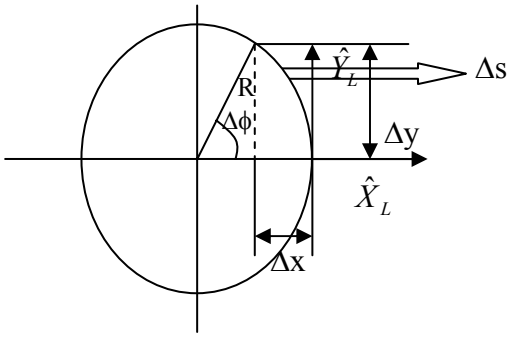


Figura 2.4: Círculo osculador.

Veamos como obtendríamos la velocidad lineal del vehículo:

$$v = \frac{\Delta s}{\Delta t}$$

La velocidad angular viene dada por la siguiente expresión:

$$\omega = \frac{\Delta \phi}{\Delta t}$$

Δs es el espacio recorrido por el punto de guiado del vehículo, Δϕ es el cambio de orientación en el intervalo de control, que es Δt.

Pero podemos relacionar las variables Δs y Δϕ de la siguiente manera:

$$\Delta s = R\Delta\phi$$

R es el radio del círculo osculador, es decir, el radio de la circunferencia que describe el punto de guiado. Este radio R se denomina radio de giro.

Ahora hallaremos la curvatura, que no es más que la inversa del radio de giro.

$$\gamma = \frac{1}{R} = \frac{\Delta\phi}{\Delta s}$$

Podemos escribir las ecuaciones de movimiento en el sistema {L} en la posición inicial.

$$^{L}(\Delta x) = -(R - R\cos(\Delta\phi))$$

$$^{L}(\Delta y) = R\,\mathrm{sen}(\Delta\phi)$$

Como ya vimos con anterioridad, la orientación inicial del vehículo respecto al sistema {G} es de ϕ. Si rotamos ϕ el sistema {L}, obtenemos el movimiento en el sistema {G}:

$$\Delta x = R[\cos(\Delta\phi) - 1]\cos(\phi) - R\,\mathrm{sen}(\Delta\phi)\,\mathrm{sen}(\phi)$$

$$\Delta y = R[\cos(\Delta\phi) - 1]\mathrm{sen}(\phi) + R\,\mathrm{sen}(\Delta\phi)\cos(\phi)$$

Si suponemos, como ya se dijo, que el intervalo de control es pequeño, el cambio de orientación también lo es, con esto, podemos decir que:

$$\cos(\Delta\phi) \cong 1$$

$$\text{sen}(\Delta\phi) \cong \Delta\phi$$

Podemos aplicar esta aproximación a las ecuaciones anteriores:

$$\Delta x = -R\Delta\phi\,\text{sen}(\phi)$$

$$\Delta y = R\Delta\phi\cos(\phi)$$

De la figura 2.4, se puede deducir que:

$$\Delta x = -\Delta s\,\text{sen}(\phi)$$

$$\Delta y = \Delta s\cos(\phi)$$

Si dividimos las ecuaciones anteriores entre Δt, además de hacer tender Δt a cero, podemos ver que:

$$x' = -v\,\text{sen}\,\phi$$

$$y' = v\cos\phi$$

Tenemos esas dos ecuaciones, además de la que se obtiene con la variación en el tiempo de la orientación ϕ:

$$\phi' = \omega$$

Con todos estos datos, se puede describir un modelo cinemático de la bicicleta para un robot.

2.1.4.- Modelo Jacobiano.

Se tiene un vector p, representando un punto en el espacio n de coordenadas generalizadas, y q el vector de m variables de actuación, n>m. Respetando la notación anterior, p´ y q´ son las derivadas de estos vectores. Se utilizará, a no ser que se diga lo contrario, el sistema de referencia global {G}. El modelo directo, es decir, el que relaciona las variables generalizadas con las actuaciones, es de la siguiente manera:

$$p' = J(p)q'$$

el término J(p) es el Jacobiano, que se puede escribir de la forma:

$$p' = f(q') + \sum_{i=1}^{m} g(p)_i\, q_i$$

con f y g, funciones vectoriales analíticas.

Pero veamos lo que conocemos hasta ahora, $p = \begin{bmatrix} x & y & \phi \end{bmatrix}^T$ es el vector de las coordenadas globales del punto de guía del vehículo y su orientación. Hacemos f(q´) = 0 y m = 2. Es fácil ver que:

$$p' = \begin{bmatrix} -\operatorname{sen}\phi \\ \cos\phi \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega$$

en estas ecuaciones, v es la velocidad lineal del vehículo y ω es la angular.

Podemos avanzar en las expresiones de las ecuaciones:

$$\begin{bmatrix} x' \\ y' \\ \phi' \end{bmatrix} = \begin{bmatrix} -\operatorname{sen}\phi & 0 \\ \cos\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

En la ecuación anterior, aparece el vector p´ de la posición y orientación, así como el vector $q' = \begin{bmatrix} v & \omega \end{bmatrix}^T$, que representa las variables de entrada.

Si combinamos las dos primeras ecuaciones, obtenemos una restricción independiente de v:

$$x' \cos\phi + y' \operatorname{sen}\phi = 0$$

Esta restricción es no holónoma, y está impuesta por el propio movimiento del vehículo robótico, el cual sólo puede moverse en cada instante en la dirección de su eje de simetría longitudinal. Operando con la relación anterior:

$$tan\phi = -\frac{x'}{y'}$$

Como se puede ver, la posición del vehículo y su orientación no son independientes.

Pero no sólo es interesante el modelo directo, también lo es el inverso, incluso se podría decir que es más interesante desde el punto de vista del control del robot móvil, ya que nos permite obtener las variables de actuación a partir de las variables generalizadas del robot.

El modelo inverso se puede obtener de manera relativamente sencilla a partir del modelo directo, invirtiendo el Jacobiano, o, en caso de que éste no sea cuadrado, con la pseudoinversa. Multiplicamos en los dos miembros por J^T y despejamos q´, para obtener:

$$q' = \left\{ \left[J(p) \right]^T \left[J(p) \right] \right\}^{-1} \left[J(p) \right]^T p'$$

Deseamos obtener esta ecuación para nuestro modelo, se obtiene:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} -\operatorname{sen}\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ \phi' \end{bmatrix}$$

Podemos ver, reflejada en este modelo, la ecuación:

$$v = -x' \operatorname{sen}\phi + y' \cos\phi$$

que ya se vio en el apartado anterior, concretamente al final del mismo.

2.1.5.- El modelo de la carretilla.

Hasta el momento se ha desarrollado la teoría sobre el modelo de la bicicleta, esto se justificará más adelante. Pero la carretilla objeto del proyecto no es un vehículo de dos ruedas, sino que lo es de cuatro. Podemos decir que la carretilla tiene una configuración en sus ruedas de tipo Ackerman, con la particularidad de que las ruedas móviles, con capacidad de guiado, son las traseras. Podemos ver un esquema de la configuración Ackerman en la figura 2.5.

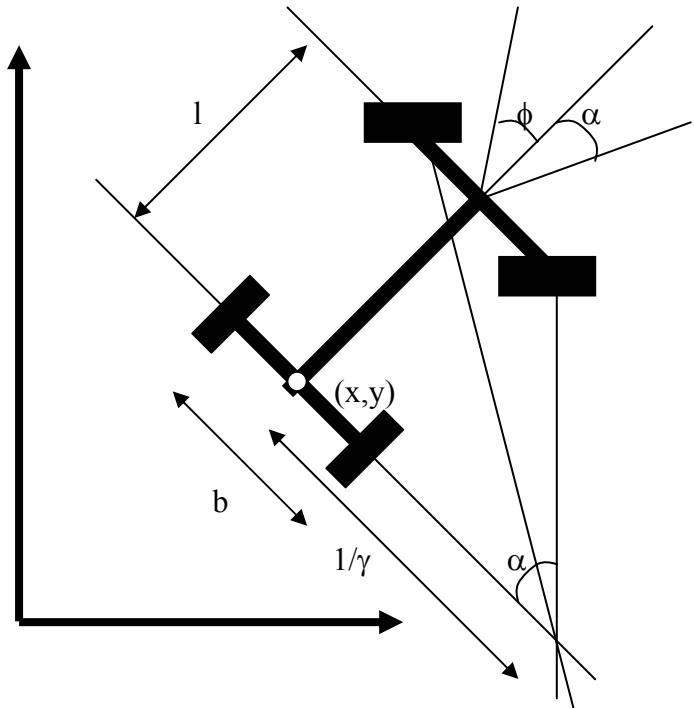


Figura 2.5: Configuración Ackerman.

Como se puede ver en la figura 2.5, el centro de guiado del vehículo, que coincide con el origen del sistema de referencia local {L} está situado en la mitad del eje de las ruedas de tracción. En el caso de la carretilla, éstas son las ruedas delanteras.

Pero el vehículo puede representarse por el modelo de la bicicleta, teniendo en cuenta que el ángulo ordenado por el volante es α, la velocidad lineal es v y la angular es ω, indicadas en las ecuaciones del modelo de la bicicleta. Esto explica que se haya puesto énfasis en este modelo. Es estas ecuaciones, se ha de tomar l como la longitud

del vehículo. Por regla general no existen ecuaciones para el modelo cinemático inverso de la configuración Ackerman, se ha de usar integración numérica para resolver esto.

La velocidad de cada una de las ruedas es diferente en este modelo, por eso el cálculo de la velocidad del centro de guiado a partir de la velocidad de una de las ruedas es erróneo. Además, como se puede ver en la figura 2.5, las ruedas de dirección al girar se mueven según un arco distinto. Este arco tiene diferente radio, así como una distinta orientación en las ruedas directrices, ya que sus ejes han de confluir con el eje de las ruedas delanteras (motrices) en un punto desde el que se mide la curvatura del movimiento. Estos efectos son propios del modelo, pero existen también efectos parásitos que producen más errores, como deformación de neumáticos, que dan lugar a orientación distinta de la deseada.

Podemos observar que el modelo descrito tiene dos grados de libertad, siempre referidos al movimiento, es decir, olvidándonos del mástil. Estos modelos se pueden describir de manera aproximada por el modelo de la bicicleta, como ya hemos apuntado con anterioridad.

2.1.6.- Estimación de la posición y orientación.

Para estimar la posición y orientación del robot, debemos integrar el modelo directo, lo que, es el caso del modelo de la bicicleta, da lugar a las ecuaciones:

$$\begin{bmatrix} x \\ y \\ \phi \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ \phi_0 \end{bmatrix} + \begin{bmatrix} \int_0^t -v \operatorname{sen} \phi d\tau \\ \int_0^t v \cos \phi d\tau \\ \int_0^t \omega d\tau \end{bmatrix}$$

Estas ecuaciones son integración directa del modelo, y el vector $p_0 = [x_0 \quad y_0 \quad \phi_0]^T$ es la posición y orientación inicial del vehículo. Es decir, que conociendo una posición inicial, y las consignas de control que se dan en un cierto período, podemos estimar la siguiente posición.

Con esto, podemos decir que queda descrito el modelo cinemático de nuestro vehículo robótico. Con posterioridad, podemos utilizar este modelo para aplicar estrategias de control al vehículo, así como estimar la siguiente posición del mismo, como hemos visto en este último apartado.

2.1.7.- Modelo dinámico de los actuadores del vehículo robótico.

En la cuestión de la dinámica en los vehículos robóticos entran en juego la dinámica del chasis del vehículo y la interacción del mismo con el terreno. Esto hace que esta cuestión sea compleja en su estudio, no sólo por la complejidad de los estudios dinámicos del chasis del vehículo, sino también porque no es sencillo establecer una interacción concreta con el terreno, sobre todo en vehículos de exteriores. En vehículos destinados a trabajar en entornos estructurados, como la carretilla, podría ser algo más fácil este estudio, ya que el terreno donde el robot realizará su trabajo es conocido.

Por todo esto, lo más común es aproximar los sistemas de dirección y tracción de los vehículos, que suelen estar bastante desacoplados entre sí, por modelos linealizados sencillos, de primer o segundo orden.

El sistema de direccionamiento se puede aproximar por un sistema de primer orden como este:

$$\gamma' = -\frac{\gamma}{\tau} + \frac{K}{\tau}u$$

En esta ecuación, γ es la curvatura del vehículo, u es la señal de control y τ es la constante de tiempo del sistema.

La tracción puede ser aproximada por un sistema lineal de primer orden:

$$v' = -\frac{v}{\tau_v} + \frac{K}{\tau_v}v_d$$

con v la velocidad del vehículo, v_d, la velocidad ordenada, y τ_v la constante de tiempo.

Se pueden aplicar modelos para estudiar la interacción con el terreno, como parece lógico, se puede aumentar la complejidad de los mismos cuanto se quiera, sólo con añadir efectos de mayor orden al modelo. Pero este estudio es complejo, y no entraremos en él.

2.2.- Sistema de localización absoluta.

2.2.1.- Descripción.

En el sistema se tiene un sensor láser de localización absoluta, dentro de un entorno estructurado. Cuando hablamos de entorno estructurado, nos referimos a que se conoce el lugar donde se realiza la tarea por parte del robot, y ese lugar está convenientemente balizado. El lugar se equipa con una serie de balizas que devuelven la señal que emite el sensor, y esa señal es percibida por el sensor. Este sensor es capaz de determinar tanto la distancia a la que se encuentran las balizas, como el ángulo en el que se encuentran. Este dispositivo es giratorio, lo hace en un plano paralelo al suelo del sensor, con lo que es capaz de asignar a cada una de las medidas que recibe el valor de la distancia a la baliza, así como el ángulo que ha girado el sensor en el plano respecto a un punto de referencia que es el ángulo cero, marcado en la carcasa del aparato.

En el caso de este proyecto, se ha utilizado un sensor comercial de marca Robosense. Este sensor viene equipado con un ordenador, que no sólo está encargado de la comunicación con el ordenador que hace las funciones de controlador, sino que también realiza un procesado de las medidas que recibe y, mediante un algoritmo de triangulación y optimización, es capaz de determinar la posición del sensor en un espacio de coordenadas determinado por la posición de las balizas en el entorno.

Este algoritmo está programado en el ordenador del sensor, por lo que no se conoce de manera exacta su funcionamiento, pero se puede ver algún algoritmo de los utilizados para esta cuestión en otros proyectos, de manera que ilustren algo el método que utiliza el sensor de posicionamiento global, casi con toda seguridad muy similar a los que ahora se exponen.

2.2.2.- Sistema con focalización por infrarrojos.

Como ya hemos visto, este sistema tiene la misión de proporcionar las coordenadas de la posición del robot en su navegación por el entorno, siguiendo una trayectoria programada por el usuario.

El sistema consta de balizas emisoras de impulsos codificados, y un receptor montado sobre una torre, pudiendo éste girar en planos horizontal y vertical. Un sistema de control focaliza la emisión de la baliza en el punto centro del receptor de infrarrojos.

Con este sistema, el receptor es capaz de localizar las balizas, dando una media del ángulo horizontal y vertical de las mismas, de cada una de ellas. Un telémetro láser solidario con el receptor de infrarrojos obtiene la medida de la distancia a la baliza enfocada.

Como vemos, este sistema se parece en algo al de la carretilla, con la diferencia de que nuestro sensor sólo se mueve en un plano, no en los dos, como el que nos ocupa.

Tras conocer como funciona el sistema, podemos utilizar una clara estrategia para el posicionamiento, esta estrategia no es otra que la colocación de una baliza a bordo del robot y un receptor en origen de coordenadas externo. Podemos ver un esquema de esta estrategia en la figura 2.6:

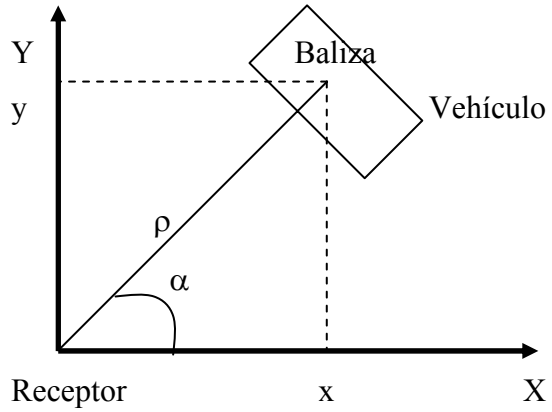


Figura 2.6: Posición con el receptor en el origen de coordenadas.

El sistema es da lugar a una fácil determinación de la posición en coordenadas cilíndricas, pasando a cartesianas:

$$x = \rho \cos \alpha$$

$$y = \rho \operatorname{sen} \alpha$$

Se pueden calcular las matrices de covarianza del sensor, con sus errores, D_ρ y D_α. En este sistema, el sistema de control de infrarrojos ha de hacer el seguimiento del robot con la baliza montada en él.

2.2.3.- Receptor a bordo y balizas en medio externo.

Para este sistema, se colocan dos balizas externas al vehículo, y se necesitan dos medidas de la distancia y el ángulo de esas dos balizas, con esto podemos determinar la posición del móvil.

Este sistema es similar al utilizado por el sensor Robosense, pero en el caso del Robosense, necesita al menos tres balizas. Incluso puede operar con más de tres. Pero

como ahora veremos, con dos es suficiente. La existencia de más balizas permite mejorar la medida, al optimizar ésta para reducir los errores que en ella se pudieran producir. Veamos como funciona en la figura 2.7:

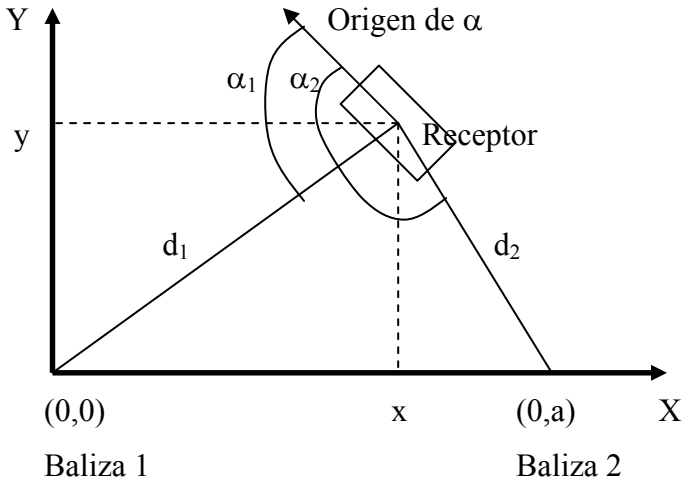


Figura 2.7: Medidas realizadas con el receptor a bordo del robot.

Podemos realizar cálculos trigonométricos, y obtener las coordenadas (x,y) del robot:

$$x = \frac{d_1}{a}\sqrt{a^2 - d_2^2 \,\mathrm{sen}^2(\alpha_2 - \alpha_1)}$$

$$y = \frac{d_1}{a} d_2 \,\mathrm{sen}(\alpha_2 - \alpha_1)$$

Si se tuvieran más balizas, habría que realizar cálculos más complejos, incluyendo la resolución del problema de optimización que el tener más ecuaciones que variables proporciona.

Capítulo 3: La carretilla elevadora.

3.1.- Descripción.

3.1.1.- El vehículo.

El vehículo que va a ser automatizado es una carretilla elevadora de marca Linde y modelo E 16, en su serie 335. Todo esto tiene una relativa importancia, ya que las carretillas de la marca Linde ofrecen distintas características, dependiendo del modelo. Algunas de estas características son indiferentes a la hora de controlar la carretilla, pero otras son importantes, y determinan las estrategias que se han usado en la automatización. Este vehículo está pensado para transportar y estibar cargas. Estas cargas han de ser las indicadas en el diagrama de capacidades de carga que se puede encontrar en el manual de servicio de la carretilla.

El manual de servicio de la carretilla acompaña a la carretilla cuando se adquiere. Este manual, y el catálogo de recambios es la mayor (y casi única) fuente de información de que se ha dispuesto. El manual de servicio ofrece información sobre el modo de empleo del vehículo, así como esquemas eléctricos e hidráulicos, destinados a un mantenimiento sencillo de la carretilla. La carretilla también se acompaña por un cargador para las baterías de la misma. Una imagen de la carretilla se puede ver en la figura 3.1.

Figura 3.1: La carretilla.

3.1.2.- Datos técnicos de interés.

Podemos ver en el manual de servicio una serie de datos que dan una idea de las prestaciones de la carretilla. Esto no es sólo importante de cara a pregonar las virtudes del vehículo, sino que también es interesante para ver las capacidades que el vehículo tiene, con el objeto de no superarlas, lo que, en ciertos casos, sería crítico para la carretilla o peligroso para el usuario. Primero debemos decir que la tracción se realiza mediante dos motores eléctricos, uno en cada rueda motriz, y que todo el sistema de dirección y elevación e inclinación del mástil es hidráulico.

Figura 3.2: Rueda delantera izquierda.

A continuación indicamos algunos de los datos técnicos de mayor interés:

El vehículo tiene cuatro ruedas, en las delanteras tiene la tracción, y en las traseras, la dirección (ver figuras 3.2 y 3.3). Las ruedas delanteras se mueven también cuando se mueve el mástil.

Este mástil está dotado de una horquilla con dos palas, la horquilla puede subir y bajar por el mástil, pero no puede moverse lateralmente, decimos esto porque hay carretillas que sí tienen esta posibilidad. El mástil puede subir la horquilla hasta una altura de 3.857 mm, ver figura 3.4. Estos valores dependen de algunos parámetros, si se tiene carga o no, y otros parámetros, pero este valor da una idea de lo que puede subir una carga.

La capacidad de carga es de 1,6 toneladas. Esta capacidad no es constante, depende de la elevación del mástil, existen gráficas en las que se puede ver esta dependencia. A mayor elevación, menor carga se puede levantar.

La distancia del centro de gravedad a la carga es de 500 mm, este parámetro es denominado ‘c’ en el manual.

La carretilla pesa 3.385 Kg sin carga. Este peso no será el del vehículo robótico, ya que al robotizarlo se introducen elementos que aumentan su peso. Se puede observar que es un vehículo muy pesado, lo cual introduce consideraciones de dificultad de manejo, encontrar un gato que levante ese peso y borriquetas que lo sostengan para

realizar pruebas con él en alto, así como la necesidad de dispositivos de desconexión rápida de la alimentación para su parada rápida, del estilo de setas, o similar.

Figura 3.3: Rueda trasera derecha.

Como ya se ha apuntado en anteriores apartados, el vehículo tiene tracción delantera.

La anchura entre ruedas del vehículo de 910 mm entre las delanteras, y 757 mm en las traseras.

La inclinación del mástil es de 4,5º hacia delante, y de 7,0º hacia detrás.

El radio de giro depende de la posición del mástil, pero con él en posición vertical es de 1615 mm.

En cuanto a la velocidad máxima que puede alcanzar, ésta es de 12,9 Km/h si está cargada, y de 15 Km/h si está descargada.

La carretilla dispone de dos motores de tracción de 4 Kw de potencia cada uno. Estos motores son independientes para cada rueda motriz. También dispone de un motor de elevación, es decir, una bomba que mueve el fluido hidráulico a través de las válvulas y hacia los cilindros de dirección y elevación. Su potencia es de 9 Kw.

Tanto los motores de tracción, como la bomba hidráulica son eléctricos. La potencia para su funcionamiento, así como para el funcionamiento de toda la electrónica que posee la carretilla, entendida esta como la que trae tras salir de fábrica, no la

adicional para su automatización, se da mediante una batería de 48 voltios. La batería tiene una capacidad de 600 Ah, lo que le permite gran autonomía.

Figura 3.4: El mástil.

3.1.3.- Aspecto y comportamiento externo.

En este apartado trataremos del manejo de la carretilla respecto de un operador externo, que se encuentre montado en la cabina de la misma, accionando los distintos mandos de esta cabina.

La tracción se maneja con dos pedales, uno para avanzar marcha delante, y otro para ir marcha atrás. Se puede observar que no pueden ser pisados a la vez, ya que tienen un mecanismo que los une, y que no deja que uno se hunda sin que el otro sobresalga, es decir, que si oprimimos uno de los pedales, el otro sube. También tiene un pedal de freno, que maneja un freno mecánico e hidráulico que para la máquina, así como un freno de mano en forma de palanca, como los usuales de coche. Posteriormente profundizaremos en el funcionamiento del sistema de tracción.

Para el sistema de dirección se usa, como en la mayoría de los vehículos, un volante (figura 3.5). El volante tiene acoplado un mango para girarlo con mayor comodidad. Este volante se acopla directamente a la válvula que mueve la dirección. La dirección puede girar de tal forma que las ruedas hacen que el centro de rotación del vehículo quede entre las dos ruedas delanteras del vehículo, en este punto, una de las ruedas de tracción se para, porque ha de estar parada en el giro de la carretilla, o bien ha de marchar en sentido contrario a la otra rueda, donde está la tracción.

Figura 3.5: Volante y detalle del salpicadero.

Para el manejo del mástil, se tiene una palanca con la posibilidad de moverse hacia delante y detrás, y hacia los lados. El movimiento hacia delante y detrás es para inclinar el mástil hacia la cabina o lejos de ella, y hacia los lados, sube y baja la horquilla. La sube hacia la derecha, y la baja hacia la izquierda. Esta palanca acciona dos válvulas, para elevación y para inclinación, de manera directa. Ver figura 3.7.

En el cuadro de mandos existen indicadores a modo de chivatos, como la luz de freno de mano accionado, una barra que indica la batería que le queda al vehículo y otros indicadores, como el de aviso de nivel bajo de aceite en el circuito hidráulico, o las horas que le quedan de funcionamiento en un display, ver figura 3.6. También tiene una llave de contacto, que corta la corriente a los motores, así como una seta de emergencia, que corta la energía a todo el vehículo, no sólo a los motores. En el cuadro hay también huecos para colocar interruptores, que pueden ser usados.

Figura 3.6: Cuadro de indicadores, seta de emergencia y llave de contacto.

Figura 3.7: Freno de mano y palanca de control del mástil.

Otros aspectos del funcionamiento de la carretilla son, por ejemplo, la existencia de un sensor que no deja que se mueva la carretilla a no ser que un tripulante esté sentado en el asiento de la cabina, otro interruptor que no permite que la carretilla se mueva con la cabina abierta, y existe una palanca que se usa para levantar la cabina para

acceder al interior de la carretilla y realizar trabajos de mantenimiento en ella, cargar sus baterías, o trabajos de otro tipo.

La carretilla tiene una batería de 48 voltios, que se ha de cargar cuando se quede sin energía. Para cargarla, se ha de levantar la cabina, como se ha explicado antes, con la palanca, y levantando un seguro que permite que se levante. Una vez abierta la cabina, se ha de desconectar la batería a la máquina, y conectarla al aparato cargador que se compró con la carretilla, y que proporciona una gran intensidad, se han comprobado intensidades de hasta 100 amperios en el display que mide la intensidad que da el cargador. Las baterías no deben descargarse más del 80 % de su capacidad total. Si esta descarga se produjera, traería como consecuencia la posible destrucción de las baterías, o el dañado de las mismas.

Para utilizar la carretilla se ha de levantar la seta de protección, usar la llave de contacto para encender la carretilla, y accionar los mandos para su movimiento.

3.1.4.- Funcionamiento interno.

La carretilla tiene dos sistemas diferenciados, un sistema de tracción eléctrico, y otro hidráulico para el movimiento de la dirección y de todo lo relacionado con el mástil.

En cuanto a la tracción, hemos de decir que se trata de una tracción eléctrica, proporcionada por dos motores, uno para cada rueda, y que tienen un variador para la transmisión de la potencia a la rueda. Estos motores están montados sobre un eje que posee reducción automática, porque en curva, las dos ruedas no van a la misma velocidad. Ver figura 3.8.

Figura 3.8: Eje delantero.

Como ya hemos explicado, cuando la curvatura de las ruedas es tal que el centro de rotación se traslada en el eje hasta la altura de una de las ruedas, o entre las dos ruedas, se tienen unos sensores que informan de esta circunstancia al controlador de la carretilla, que reacciona haciendo que la rueda que queda por el lado de dentro de la curva no reciba potencia, y dejándola libre, moviendo sólo la rueda exterior a la misma. Con esto se consigue que la rueda del interior de la curva no gire cuando se dé esta contingencia, o incluso vaya marcha atrás si el centro de la rotación es tal que se tiene que dar esto, es decir, cuando el centro de rotación cae entre las dos ruedas del vehículo. Todas estas características se han observado en la carretilla mientras se ha mantenido en alto. Para las pruebas a las que se vio sometida.

Cuando se suelta un pedal, se reduce la velocidad hasta el frenado total, la carretilla tiene un sistema conocido por frenado de corriente útil, en el que se aprovechan corrientes que quedan en el motor para frenar la carretilla.

En cuanto a la interacción con el tripulante, el controlador del que ya hemos hablado recibe la orden de velocidad mandada por el operador mediante los pedales, que están conectados, tanto el de marcha delante como el de marcha atrás, por un sistema a un potenciómetro “MBC Industries” de 3K74 ohmios, el cual está sujeto a una tensión de 14,9 voltios por parte del controlador, que lee las subidas y bajadas del mismo, por efecto de los pedales, mediante las subidas y bajadas de la tensión en el cursor del

potenciómetro. Las desviaciones del valor medio de esos 14,9 voltios, aproximadamente 7,5, son lo que interpreta el controlador para el sistema de tracción. Así, si se pisa el pedal de marcha adelante, sube la tensión en el cursor, el controlador lo lee, y la carretilla avanza. Por el contrario, si se pisa el de marcha atrás, la tensión en el cursor baja, y la carretilla retrocede. No se activa la tracción de manera inmediata al subir o bajar de 7,5 voltios, sino que hay una pequeña zona muerta de voltaje, en la que el controlador no activa los motores.

Figura 3.9: Con los pedales controlamos la tracción.

Se ha de destacar también, que al apretar cualquiera de los pedales de tracción, además de mover el potenciómetro, se pulsa también un contactor que da corriente al motor. Por lo tanto, las condiciones para que se active la tracción son, tener el cursor del potenciómetro a un voltaje superior o inferior a 7,5 voltios y tener pulsado el contactor.

Si se pulsa el contactor sin que esté el potenciómetro en las proximidades del centro, el controlador responde mal, y se obtiene un comportamiento erróneo, saltan los relés de los motores, pero conmutando constantemente, y la carretilla no se mueve. Cuando esto ocurre, se ha de resetear el controlador, porque ya no vuelve a funcionar, la manera más sencilla es pulsar la seta de emergencia y luego levantarla.

No se tiene mucha información del controlador, lo que se sabe por observación del esquema eléctrico de la carretilla, así como por las múltiples pruebas realizadas, es que, con el cambio de tensión, se activan los relés que ponen los motores de continua de

la tracción en directa o en reversa, consiguiendo movimiento en un sentido o en el otro. Con un cierto voltaje, saltan los relés a posición de directa, y con el aumento del mismo, se le da mayor velocidad al vehículo.

Este controlador, posee como una de sus prestaciones el control de aceleración, no por apretar a fondo el pedal se consigue mayor aceleración, ya que ésta está controlada.

Posteriormente se hablará de los esfuerzos encaminados a actuar en ese potenciómetro, para controlar la tracción de manera automática.

En cuanto a la dirección, la inclinación del mástil y la elevación de la horquilla, se sabe ya que se dispone de un sistema hidráulico para su gobierno. El sistema consta de una bomba hidráulica y tres válvulas principales, para dirección, inclinación del mástil y elevación de la horquilla, además de otras válvulas antirretorno y un tanque de fluido que completa el circuito.

La bomba hidráulica es la encargada de mover el fluido hidráulico, un aceite, por el circuito, pero solamente será activada cuando se necesite un esfuerzo por parte de esa bomba, es decir, cuando se accione una válvula. Al accionar una válvula, la bomba funciona, y mueve el fluido a través de esa válvula hacia el cilindro correspondiente a la parte que se quiere mover.

Las válvulas son manuales, el propio esfuerzo del operador al mover el volante, o al accionar la palanca que controla el mástil, hace que esas válvulas conmuten en su estado. Esta manualidad en las válvulas hace que no sean útiles para el control automático, porque lo más sencillo en cuestiones de control es disponer de dispositivos que respondan a señales eléctricas. Posteriormente veremos la solución adoptada en este aspecto. Tienen un comportamiento más o menos proporcional, cuanto más se acciona la palanca, más se abre la válvula.

Como ya hemos apuntado, la bomba se ha de mover cuando se requiera de su esfuerzo. Para realizar esto, el control del vehículo lee ciertos valores que dan las válvulas, y que permiten activar la bomba cuando se actúa sobre una válvula. Para la válvula de dirección, la señal que se lee es de cortocircuito cuando se actúa sobre el volante, y de circuito abierto cuando no se actúa. En las válvulas de inclinación y elevación, se da una señal proporcionada por lo que, probablemente, es un potenciómetro, alimentado con 15 voltios, y que lee un valor en torno a 7,5 voltios, similar al sistema de tracción, aunque luego veremos que con menos problemas que este para su emulación. De esta manera, si se sube de 7.5, se activa la bomba, pero cuanto

más se suba, más fuerza tiene la bomba, y si se baja, algo similar. Con estos voltajes, controlamos mediante dos parámetros la velocidad de accionamiento de la dirección y el mástil, la apertura de la válvula y el caudal que da la bomba. Esto será interpretado por el controlador, ya que las válvulas tienen unas curvas de rendimientos respecto al caudal y otros parámetros, que se pueden usar para mejorar el rendimiento de la carretilla en cuanto a empuje. Cuando se trata de bajar la horquilla, se ha de desalojar fluido del cilindro, sin intervención de la bomba.

En cuanto al interruptor que no permite el movimiento de la carretilla, si no es con alguien sentado en el asiento de la cabina, es sólo eso, un interruptor. También es un simple interruptor el que hace que no funcione la carretilla cuando está la cabina levantada. Estos interruptores dan un cortocircuito cuando hay un operador sentado y la cabina está cerrada, y un circuito abierto en caso contrario.

El freno de mano se acciona mediante cable, además de oprimir un interruptor que desconecta la tracción.

3.2.- Soluciones de actuación propuestas.

3.2.1.- Actuación sobre la tracción.

Para el movimiento automático de la carretilla se ha de poder actuar sobre los distintos sistemas que se han descrito en el apartado anterior. Pero esto es, en muchos casos, difícil.

En el caso de la tracción, se podría actuar sobre los pedales directamente con algún mecanismo o motor, pero el hecho de que se disponga detracción eléctrica lo desaconseja, ya que un de las ventajas de esta tracción es que puede ser controlada de manera más o menos sencilla con señales eléctricas. El problema de aplicar directamente estas señales a los motores es que se desconoce del todo su funcionamiento y, además, la existencia de dos motores puede complicar el modelo del robot móvil que pretendemos realizar, introduciendo elementos propios de una configuración diferencial, mezclada en cierto modo con la configuración Ackerman que

ya se ha presentado. Además, parece absurdo desaprovechar toda la electrónica de control de la que dispone la carretilla, que soluciona todos estos problemas.

Interesa también la intervención en un punto concreto de la carretilla, lo cual facilitará con posterioridad los mecanismos de conmutación de manual a automático. Es de destacar también el hecho de que el punto a intervenir para introducir las señales de control externas ha de estar accesible de manera sencilla al autor de la intervención, o al menos, sería deseable.

Por todo esto se decidió la actuación sobre el potenciómetro que recibe la acción del tripulante sobre los pedales y la transmite al controlador. En principio, está en un lugar muy accesible, al levantar la cabina, debajo del asiento, y tiene pocos cables para cortar, son tres, uno de cero voltios, otro de 15, 14,9 concretamente, y otro, el cursor del mismo, que es el que se mueve por acción de los pedales. Como no se tenía ninguna información del controlador, y la empresa que hizo la carretilla no proporcionaba esta información, lo cual parece lógico, se hubo de probar para captar el funcionamiento del sistema.

En este punto, se probó a sustituir este potenciómetro por un voltaje que variara, probando con fuentes de tensión aplicadas entre el cero y el cursor, pero esto no dio ningún resultado satisfactorio. Se midió la resistencia del potenciómetro, y se observó que había prácticamente la misma resistencia entre los extremos que entre uno de los extremos y el cursor, estando el potenciómetro en reposo, por supuesto. Estamos hablando de valores de 3K74 Ω entre extremos, y 3K66 Ω y 3K61 Ω entre el cursor y cada uno de los extremos. Esta circunstancia dio lugar a que se pensara que la resistencia entre la resistencia física y el cursor del potenciómetro era elevada, concretamente se observó que podía ser de 1K74 Ω. Se observó también que la resistencia que se podía leer entre extremos del potenciómetro o entre cursor y extremo cambiaba cuando el potenciómetro estaba conectado al controlador, sin duda por efecto de la resistencia de entrada del circuito electrónico que realiza la función controladora. Esa resistencia no se podía medir con un polímetro tras conectar el contacto de la carretilla, al menos, se hacía infinita, o se salía de rango.

Tras probar con una fuente, se probó a cargar esa fuente con una resistencia en paralelo y en serie. El objetivo de esto era probar si el control se realizaba por intensidad, y se trató de hacer una fuente de intensidad. Pero no se obtuvieron tampoco resultados satisfactorios.

Se realizaron medidas de la intensidad de cada una de las ramas del potenciómetro. Se pudo ver que cuando se bajaba el potenciómetro de 7,5, salía intensidad por el cursor, pero la mayoría de esa intensidad se iba por el cable del cero, quedándose el de 15 voltios con una intensidad similar, y al subir el cursor de 7,5 voltios, entraba intensidad por el cursor, intensidad que era dada por el cable de los 15 voltios, quedándose el cable de cero voltios con la intensidad que entraba en reposo, prácticamente. Esto lleva a pensar en que el control no sólo tiene en cuenta el voltaje del cursor, sino otras consideraciones como que el que baje el voltaje entre cursor y cero supone que suba entre el cable de 15 voltios y el cursor. Parece como sí con la tensión controlara el salto de los relés de marcha de los motores, y con la intensidad, la velocidad de los mismos.

Tras observar todo esto, se pensó en sustituir el potenciómetro por dos fuentes que estuvieran coordinadas y que, cuando subiera una, bajara la otra, para emular el comportamiento en voltaje del poenciómetro, aunque la coordinación entre ambas fuentes parecía complicada.

En ese momento se descubre la existencia de unos potenciómetros digitales, en los que el valor de la resistencia se puede controlar con señales digitales. Se tiene una matriz de resistencias, y con una serie de señales, se puede mover el cursor en ellas, el cursor va a saltos, pero son pequeños. Estos potenciómetros son caros, así que antes de comprarlos, se hacen pruebas con potenciómetros de distintos valores, a ver si se puede sustituir el potenciómetro así como así. Se prueban potenciómetros de distintos valores, y no se obtienen buenos resultados. El problema puede ser que los potenciómetros que existen en el mercado no son de valores próximos a 4 KΩ, que es el valor nominal del potenciómetro de la carretilla, así que se opta por sustituir el potenciómetro original por uno que sea lo más parecido posible, si puede ser igual. Para esto se realiza un montaje como el de la figura 3.10.

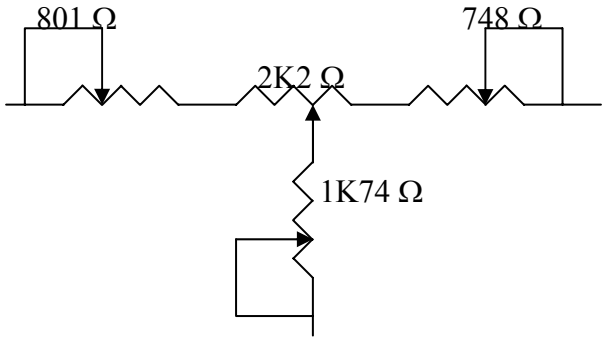


Figura 3.10: Montaje de emulación de potenciómetro.

Este montaje se produce debido a que no se pudieron encontrar potenciómetros de valores próximos a los 3K7 Ω del potenciómetro original. Con este montaje se probó si se podía sustituir un potenciómetro por otro, los resultados en cuanto a las resistencias se obtuvieron realizando un ajuste, mediante prueba error. La resistencia de 2K2 Ω es el potenciómetro que se usó, y las otras tres resistencias son potenciómetros con un extremo cortocircuitado al cursor, como se puede ver en el dibujo, y se ajustaron para que el conjunto resultara con la misma resistencia que el original en la carretilla.

Tras comprobar que se podía sustituir el potenciómetro por un montaje similar al anterior, se realizó ese mismo montaje con potenciómetros digitales. Se usaron cuatro, potenciómetros digitales modelo X9312 de Xicor. Son unos potenciómetros controlados digitalmente por tres señales digitales, con capacidad para almacenar la posición del cursor y de funcionamiento asíncrono. Tienen una matriz de 100 resistencias, con lo que, si son de 10 KΩ, la resistencia variará a saltos de 100 Ω. Estos son los potenciómetros digitales más sencillos que se encontraron en cuanto a su manejo. El problema que presentan estos dispositivos es que son de 10 KΩ entre sus extremos, y se optó por colocar cuatro en paralelo, con todas sus patas conectadas entre sí, ya que tenían las mismas señales de control, así como los mismos extremos y el mismo cursor en todos. Esto se hizo para que se tuviera un potenciómetro de 2K5 Ω más o menos, con el que se podía realizar un montaje como el expuesto anteriormente. En apartados posteriores se hablará de este dispositivo. Veámoslo en la figura 3.11.

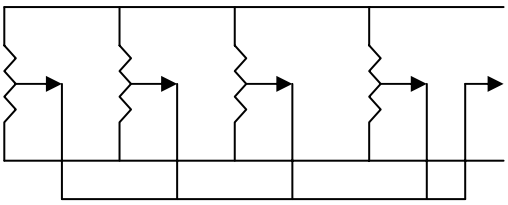


Figura 3.11: Montaje de cuatro potenciómetros en paralelo.

Pero con esto no se consiguió un funcionamiento del todo correcto. El problema venía dado por el contactor que se oprime cuando se pisa el pedal, ha de oprimirse en el momento adecuado, de acuerdo con el efecto que realizan los pedales en modo manual, en su funcionamiento normal. Para sustituir este contactor se usó un relé.

En cuanto a los dos interruptores de seguridad, que impiden el funcionamiento de la carretilla cuando no hay nadie sentado o cuando la cabina está subida, hubieron de ser puenteados. El interruptor del asiento se ha dejado así, para que la carretilla se mueva sin tripulante, pero el otro interruptor sólo estuvo puenteado mientras se realizaban pruebas en la carretilla. Porque una vez instalados estos sistemas, no se necesita que la carretilla funcione con la cabina levantada.

3.2.2.- Actuación sobre el sistema hidráulico.

El sistema hidráulico está basado en válvulas de control manual. No hay una manera de actuar sobre el sistema en forma eléctrica directa.

Se puede pensar en acoplar a los mandos algún tipo de motor que los mueva obedeciendo a las órdenes del control, así, se pueden mover tanto el volante como la palanca que controla el mástil. Pero esta solución ofrece problemas a la hora de conmutar de manual a automático, ya que sería un sistema que quedaría amarrado a los mandos. Ofrece también el problema derivado de las holguras en las cadenas o engranajes para llevar el movimiento del motor a los mandos. Sumado a todo esto, no hay mucho sitio en la cabina para introducir estos dispositivos en su interior.

Pero en los sistemas hidráulicos hay dispositivos denominados electroválvulas, con las que se pueden realizar conmutaciones en el circuito, así como control proporcional de caudal en una válvula, todo ello mediante señales eléctricas. Este método parece el mejor, pero se han de sustituir las válvulas manuales por otras eléctricas. Sin embargo, si se sustituyen las válvulas manuales, perdemos el control manual de la carretilla. Hay válvulas que son controladas de las dos maneras, manual y

eléctrica, pero son difíciles de acoplar al sistema de accionamiento desde cabina que posee la carretilla, además, con esas válvulas puede no ser fácil transmitir las señales al controlador para que se encienda la bomba. Por otro lado, la conmutación entre manual y automático con estas válvulas no resultaría del todo efectiva, ya que, en manual, se podrían controlar eléctricamente, y en automático, manualmente.

Finalmente, se optó por la duplicación del sistema hidráulico. La duplicación se realizó en los talleres de Aphisa, Aplicaciones Oleohidráulicas S.A.. Se realizó un nuevo circuito hidráulico al que se le acoplaron tres electroválvulas proporcionales para el control de la dirección y elevación e inclinación en el mástil, así como dos electroválvulas todo - nada, para la conmutación entre los dos circuitos, el manual, es decir, el antiguo, y el automático, el nuevo y eléctrico. Una de las todo - nada se usó para conmutar el circuito de dirección, y la otra para conmutar el circuito relacionado con el mástil.

Así, conseguíamos el propósito de actuar de manera eléctrica, manteniéndose la posibilidad de conmutar entre manual y automático.

En cuanto a la conmutación manual automática, hemos de decir que las válvulas todo – nada actúan en el momento en que están alimentadas. Se necesitan 24 V para hacerlo. Así, podemos colocar un interruptor que alimente estas válvulas, con lo que el circuito hidráulico queda conmutado. Estas válvulas todo – nada son del fabricante italiano Atos, concretamente de accionamiento directo, modelo E010.

Las válvulas de control son tres, son electroválvulas proporcionales, y actúan debido a que reciben el flujo del circuito hidráulico tras la conmutación del circuito manual original al circuito automático por medio de las válvulas todo – nada de las que acabamos de hablar. Están alimentadas a 24 voltios, y el rango de entrada de las señales que las controlan es de seis a 18 voltios, con doce voltios en su punto medio. El hecho de que sean proporcionales quiere decir que la apertura de la válvula es proporcional a estos voltajes, así, a un voltaje superior a seis, la válvula se abrirá más en uno de sus sentidos cuanto mayor sea ese voltaje, siempre sin superar los 18 voltios. Ocurre algo similar para los voltajes menores de seis, cuanto más pequeño sea ese voltaje, más apertura, pero en este caso en sentido contrario, siempre sin bajar de los seis voltios.

Si se aplica en las entradas de señal de la válvula un voltaje que no se encuentre en este rango, la válvula nos lo indica con una luz roja, en contraposición a la luz verde con la que nos muestra que todo está en orden. Esto no tendría mucha más importancia, pero cuando se enciende esa luz roja, la válvula queda bloqueada hasta que se apague y

encienda de nuevo, lo cual hace que sea de gran importancia el procurar que las señal de entrada esté en los valores esperados de la válvula entes de que ésta se encienda, porque si no, puede bloquearse, como ya hemos dicho.

Para atacar la válvula hemos de tener en cuenta consideraciones como la intensidad de señal que le ha de entrar, así como la adaptación de los voltajes de la tarjeta de salida de datos, con la que se controlará la válvula, para que estos voltajes sean los correctos para la válvula. Es importante también no atacar la válvula directamente con la tarjeta de control, a fin de aislar y proteger el ordenador de posibles sobretensiones y otras contingencias que pudieran dañar el ordenador de control.

Pero no sólo hay que mandar señales a las válvulas, también hemos de mandar señales al controlador interno a la carretilla para que proceda al encendido de la bomba hidráulica. En este sentido, hemos de diferenciar dos situaciones distintas, si se trata del sistema de dirección o si se trata del sistema de movimiento del mástil.

La dirección digamos que tiene que realizar un esfuerzo más o menos constante, y no excesivamente dependiente de la carga de la carretilla, mientras que el sistema del mástil ha de realizar esfuerzos distintos en función de los diferentes pesos de las cargas situadas sobre él. Por eso, la bomba hidráulica realiza un esfuerzo constante cuando se trata de mover las ruedas en el sistema de dirección, y un trabajo que no es ni mucho menos constante para levantar las cargas.

Cuando hablamos de la dirección, por tanto, debemos observar que la bomba es puesta en marcha con un movimiento constante mediante un interruptor que se pulsa al accionar los mandos de la misma, es decir, el volante. Este interruptor será sustituido por un relé que activará la bomba en el caso de que se necesite un movimiento por parte de la dirección. Deseamos llamar la atención sobre la velocidad de giro de la dirección, controlada en este caso por la apertura de la válvula.

En el sistema del mástil, las cosas son distintas, ya que se intenta que la apertura de la válvula que da entrada al fluido al cilindro, bien de elevación, bien de inclinación, del mástil vaya en consonancia con el caudal que la bomba introduce en el circuito. Esto es determinado por consideraciones de hidráulica en las que no entraremos, pero de lo que se trata es de conseguir la mejor relación entre el caudal en el circuito y la apertura de la válvula para cada uno de los pesos que se levantan, así como para la velocidad con la que se levantan. A la hora de controlar, lo que lee el controlador interno a la carretilla es un voltaje consecuencia de un potenciómetro que se ve sometido a quince voltios entre sus extremos, con lo que en el cursor se lee un voltaje en torno a los 7,5 voltios.

Pero, por suerte, en este caso sí que lo único que lee el controlador es un voltaje, no como en el caso de la tracción, y se puede actuar sobre la bomba del circuito hidráulico con un voltaje.

Así, se presenta un interesante problema de control mutivariable, con dos entradas para una salida deseada. Con el caudal de la bomba y la apertura de la válvula hemos de controlar el movimiento del sistema del mástil, de la inclinación o de la elevación. Este problema no se va a tratar aquí, por otra parte, parece que ya ha sido resuelto por los ingenieros de Linde, ya que la carretilla funciona de la manera que estamos explicando en modo manual.

En todo caso, se necesitarían señales analógicas, tanto para el control de las válvulas, una por cada válvula, como para el control de la bomba, en principio, dos, una para el control de la bomba para elevación y otra para inclinación del mástil. Hemos de decir que dos son las señales analógicas de las que se dispone en este momento.

Esto nos hace pensar en usar una única señal de las dos analógicas para la bomba, con métodos de control distintos para cada caso, pero el comportamiento del controlador no es igual en los dos casos, lo que no lo hace muy aconsejable.

La ausencia de suficientes señales analógicas para este control, al menos utilizando las tarjetas que se han usado, condiciona también el problema. Se puede usar una única señal analógica, que se multiplexe en el tiempo mediante un multiplexor analógico, tras el que iría un mantenedor de orden cero, es decir, un integrador que mantendría la señal durante todo el intervalo de muestreo, o bien se podrían usar más tarjetas. El hecho de que se necesite una multiplexión analógica, con el mantenedor de orden cero viene determinado porque, en muchos casos, no se pueden dejar señales de entrada al controlador al aire, es decir, que si se conmuta de manual a automático, y luego, mediante relés o un método similar, se lleva la señal de control al lugar que convenga, ya sea válvula o entrada para control de bomba, pueden darse efectos no deseados, como que la bomba funcione sin que nosotros queramos, por efecto de la desconexión, ya que el controlador lee un voltaje que hace que eche a andar la bomba, o bien que el hecho de conmutar una válvula puede bloquearla, bien por tener la señal de control al aire, bien porque ésta no está dentro del rango adecuado para la válvula.

Por todos estos problemas ya relatados, se optó por una solución muy sencilla, que simplifica el hardware y el software de control, pero que no podemos conocer si será la más conveniente en un futuro, con un mayor número de los medios disponibles aplicados a la caretilla. Esta solución consiste en echar a andar la bomba con el relé que

la activa en el caso de la dirección, y aplicar las dos señales analógicas de las que se dispone a las dos válvulas del mástil, o bien a la válvula de dirección, conmutando entre las distintas posibilidades mediante relés, de manera que no se produzca bloqueo alguno de las válvulas. También se controla mediante el ordenador la alimentación a las válvulas. Esto tiene el objeto de reiniciar la válvula en caso de bloqueo, sin tener que desconectar la alimentación, sólo con el ordenador, y se realiza con una señal digital y un circuito excitador de relé, relé éste que alimenta a las válvulas proporcionales.

Utilizando el relé en vez de una señal analógica no se consigue tanta velocidad en el movimiento del mástil. Se ve que la solución no es óptima en ese aspecto, pero sí mucho más sencilla.

Para la excitación mediante voltaje de las válvulas, se ha optado por utilizar amplificadores operacionales del modelo LM324, que poseen alimentación asimétrica. Se han alimentado éstos a 24 y cero voltios, y el montaje en el que están colocados es un amplificador no inversor de ganancia dos, montaje extremadamente sencillo, y que tiene como objeto el atacar estas válvulas con voltajes dentro del rango que se especifica, ya que la tarjeta controladora no envía voltajes mayores de diez voltios. Así, se programa la tarjeta para que envíe voltajes de cero a diez, y éstos son duplicados por este sencillo circuito. El voltaje que da la tarjeta no está entre cero y diez, sino que ha de estar limitado a los valores entre tres y nueve, con el valor medio en seis, para que los valores se adecuen a lo que demanda la válvula a su entrada. Esta limitación se hará vía software.

Se ha de procurar también que exista un valor coherente con lo que la válvula espera antes de su encendido, para evitar el bloqueo de la válvula. Todo esto se consigue programando el controlador de forma adecuada.

Capítulo 4: Trabajos realizados en la carretilla.

4.1.- Introducción.

4.1.1.- Necesidades.

Como comienzo de este capítulo evaluaremos brevemente las necesidades que la carretilla tiene para su automatización.

Aparte de las ya mencionadas antes, la incorporación de las electroválvulas, se han de realizar trabajos de cableado para llevar las señales desde el controlador hasta los distintos puntos de actuación, así como procurar una serie de mandos para la conmutación manual – automático y el encendido de los dispositivos. Montar el sensor de que se dispone en la carretilla y procurar su alimentación y conexión con el control de la carretilla es otra de las necesidades. También se ha de buscar ubicación para el ordenador controlador, así como para el circuito de adaptación de señales.

Se necesitan también baterías que alimenten todo esto y que su energía llegue a todos los dispositivos que la necesiten.

4.1.2.- Trabajos realizados.

Como respuesta a todas estas necesidades, se han realizado trabajos de cableado para llevar las señales y la alimentación a los lugares donde se necesita, así como trabajos de elaboración de circuitería para adaptación de señales y control de la carretilla. Se han instalado interruptores en el cuadro de mandos de la carretilla para conmutación manual automático y encendido de ordenador y sensor láser.

Se han colocado el ordenador y los circuitos en una caja, y se ha conectado la misma con los cables para llevar las señales a donde se requieran. Los cables han sido debidamente encauzados y marcados para un mejor orden.

El láser ha sido instalado y cableado convenientemente.

4.2.- Cableado.

4.2.1.- Cableado de los interruptores.

Se han descrito tres interruptores, para la conmutación de manual a automático, para el encendido del ordenador, y para el encendido del láser. Estos interruptores están situados en el cuadro de mandos, a la izquierda del volante, como se puede ver en le figura 4.1.

Figura 4.1: Interruptores en cabina.

Encima de los interruptores hay una etiqueta que marca la función de cada uno de ellos. Estos interruptores reciben dos cables cada uno, y estos cables van a parar al controlador, en el controlador activan relés que hacen que se conmute la señal o que se enciendan láser y ordenador. Así, reciben 24 voltios de las baterías, a través del circuito de control, y dan esos 24 voltios o no, dependiendo de su posición, para realizar estas funciones. Los cables llegan todos al controlador por el mismo sitio, con lo que se han de introducir en la carretilla, donde se unen a otros cables procedentes de válvulas y del

potenciómetro de control para llegar a la caja de control. Se puede ver como se esconden los cables en la carretilla en la figura 4.2.

Figura 4.2: Cables para interruptores.

Se pueden observar en la figura los cables de los interruptores, convenientemente embozados en una funda en espiral para que viajen juntos, de manera ordenada, así como las presillas de color blanco que hacen que los cables vayan en paralelo a los cables propios de la carretilla.

4.2.2.- Cableado de tracción.

Para la tracción se necesita que lleguen los cables al potenciómetro, situado en la cabina por debajo, y no sólo esto, ya que también han de llegar los cables del control. Esto es así porque se ha de poder conmutar entre el potenciómetro original y el potenciómetro automático para el cambio manual automático, por eso han de llegar los cables del potenciómetro manual. En el circuito de control se conmuta la señal que recibe el controlador interno a la carretilla, entre los dos potenciómetros, por eso han de llegar tanto los cables de la electrónica de la carretilla como los del potenciómetro. En la figura 4.3 podemos ver los empalmes realizados para el potenciómetro, así como el trazado de los cables bajo la cabina. La unión de los cables se realiza mediante fichas de empalme.

Figura 4.3: Cabina vista desde abajo.

En la figura, además de los empalmes que se pueden ver en el centro, se ven también los cables que vienen de los interruptores, que se unen con los cables relacionados con la tracción y acaban en la parte trasera de la cabina, por donde salen al exterior. Se trata, en total, de cinco cables asociados a la tracción, y que corresponden al cursor del potenciómetro, y al positivo del mismo, así como al lugar donde la carretilla espera la medida del cursor del potenciómetro, y el lugar por el que el controlador de la carretilla introduce los quince voltios que le permiten realizar una lectura del potenciómetro, el otro cable que queda es la tierra, común a los dos, tanto negativo del potenciómetro como del controlador interno a la carretilla. Todo este cableado se ha hecho para que se pueda abrir la carretilla con el objeto de cargar las baterías, o para revisar algo de su interior. También se puede observar el mecanismo que acciona el potenciómetro en modo manual, en la parte superior de la figura.

En referencia a la tracción, también hay que hablar de los dos cables del que van del interruptor que activa la tracción, son dos cables, que serán conectados con un relé en caso de que se tenga que mover la carretilla, lo que se hace es colocar un relé en paralelo con el interruptor del modo manual. Se ha de activar este relé a la vez que el potenciómetro para la tracción, como ya se ha explicado. Se conectan a los cables ya existentes con fichas de empalme.

4.2.3.- Cableado de las válvulas.

En el caso de las válvulas no se necesita cableado de ida y vuelta, solamente se necesitan cables que lleven la señal a la válvula.

Los cables que se necesitan son los de alimentación (24 voltios) y tierra en el caso de las válvulas E010, que son las todo – nada. Se tiene un tercer cable en estas válvulas, identificado como la tierra del solenoide, y que se ha conectado al cable de tierra.

En el caso de las válvulas proporcionales, se necesitan tres cables, uno de tierra, otro de alimentación, que como ya se ha dicho es a 24 voltios, y otro que lleva la señal que ataca a la válvula, y que ya sabemos que está en el rango de seis a 18 voltios. Esta señal se toma respecto al cable de tierra.

Con todo esto, tenemos cinco válvulas, con tres cables cada una, son quince cables que han de ser canalizados debidamente para permitir, como en los anteriores casos, el movimiento de la cabina.

La conexión de los cables a la válvula se realiza mediante conectores del tipo IP-65, contemplado en la norma DIN 43650, en concreto, del modelo SP-667. Las válvulas de tipo E010 vienen representadas en la figura 4.4.

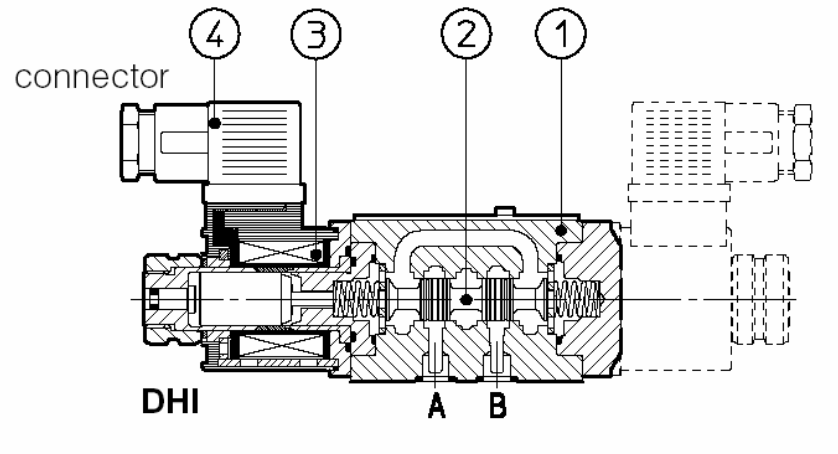


Figura 4.4: Válvula todo – nada E010.

Se puede observar el conector señalado por el número 4. En este conector se tienen numerados con 1 al positivo, con el 2 al negativo, y con el símbolo de tierra a la tierra del solenoide. Ya se ha dicho que la tierra del solenoide y el negativo se han conectado.

En el caso de la válvula proporcional, se tiene un conector de similares características. Podemos observar un esquema de la válvula en la figura 4.5.

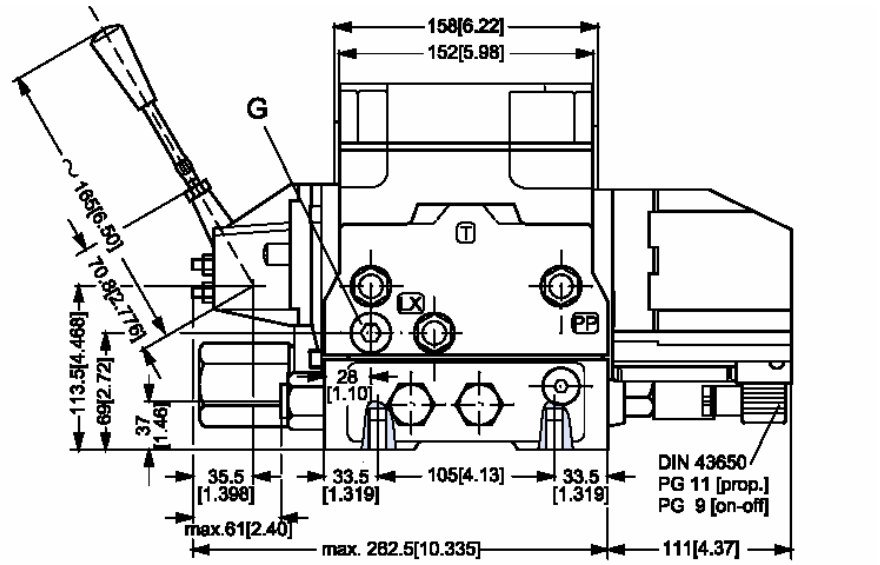


Figura 4.5: Válvula proporcional de tipo PVG120.

Se puede observar que el conector eléctrico es igual que el de la válvula E010, pero, en este caso, el conector 1 es alimentación, el 2 es la señal a la que es proporcional la apertura de la válvula, y el símbolo de tierra denota la tierra. El número 3 se utiliza en ciertos montajes de seguridad, que se pueden consultar en la documentación del fabricante, disponible en formato electrónico y en nuestro poder. En la figura 4.6 podemos ver las válvulas con los cables que de ellas salen.

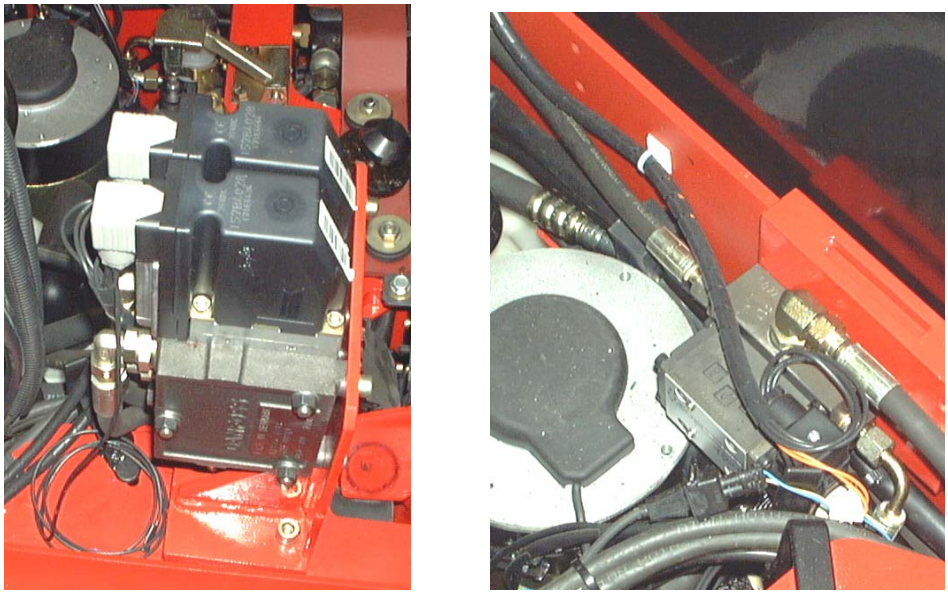


Figura 4.6: Detalle de las válvulas con los cables.

Se pueden observar los cables y las regletas de conexión. En la primera foto, se ven las válvulas PVG120, y en la segunda, una de las todo – nada E010. Posteriormente se verán las consideraciones de consumo de las válvulas, pero la PVG120 tiene dos modos de funcionamiento, proporcional de nivel alto, y proporcional de nivel medio. En

este caso, se tiene la válvula funcionando en modo proporcional de nivel alto, o PVEH, frente al PVEM, en este caso, está alimentada a 24 voltios, y la señal de entrada va de seis a 18 voltios, con el valor medio en doce voltios.

En el caso del circuito hidráulico, se tienen también dos cables más de cada válvula. Para la dirección, el relé que activa la bomba está en paralelo con el interruptor de la válvula, como en el caso de la tracción. Para las dos válvulas del mástil, dos cables que dan la señal de la bomba, con 7,5 voltios, no funciona la bomba, y con más o menos, sí, se tiene el cable de señal y el de tierra, la alimentación no es preciso sacarla.

Pero estos cables van a un control interno de la carretilla, que ha de ser conmutado del actuador manual de las válvulas al actuador automático, con lo que necesitamos dos cables más para el control interno de la carretilla, para inclinación y elevación del mástil, en total, cuatro cables más para la activación de la bomba en estos casos.

4.2.4.- Etiquetado de los cables.

Para la organización de los cables, aparte de agruparlos en el tubo helicoidal en el que van insertados, se ha realizado un etiquetado de los mismos con unos indicadores de distintos colores, con símbolos, números y letras impresos en estas etiquetas.

En primer lugar, diremos que, para cada caso, y si corresponde, se tiene una etiqueta con el signo + para la alimentación, - para el negativo, otra con el signo de tierra para la tierra, y otra con una ‘S’ para la señal, todas ellas amarillas, esta S también se usa cuando lo que se tiene es el cursor de un potenciómetro, o una señal referente a él, por ejemplo, la señal que el controlador interno de la carretilla espera del cursor. Así, pasamos a las etiquetas. En el caso de números mayores de nueve, se han usado dos números, ya que no existen etiquetas de números mayores de 9.

- Número 1, color marrón, válvula proporcional de elevación en sus tres cables.
- Número 2, color rojo, válvula proporcional de inclinación, también sus tres cables.
- Número 3, color naranja, válvula proporcional de dirección, los tres cables.
- Número 4, color amarillo, válvula todo - nada para el mástil, tres cables. Este caso es especial, ya que se tiene un negativo y una tierra del solenoide, con

lo que el negativo está marcado con un signo menos '-', pero se acaba uniendo en el circuito a la tierra del solenoide.

- Número 5, color verde, válvula todo – nada de la dirección, como en el caso anterior, tres cables, con positivo, negativo, y tierra, estos dos últimos, conectados.
- Número 6, color azul, relé que activa la bomba de dirección. Son dos cables.
- Número 7, color violeta, controlador de tracción, en principio, tres cables, pero pueden ser dos, ya que la tierra es común con el controlador interno a la carretilla.
- Número 8, color gris, dos cables del relé de tracción.
- Número 9, color blanco, tres cables del potenciómetro original (del control manual). Si son dos, es debido a que la tierra es común con el cable del controlador de tracción interno de la carretilla, con el número 7.
- Letra A, color amarillo, señales de la válvula para el control de la bomba en el caso de la elevación, dos cables.
- Letra E, color amarillo, cables que van al controlador interno a la carretilla para manejar la bomba en el caso de la elevación, dos cables.
- Letra L, color amarillo, señales de la válvula para la bomba en caso de la inclinación, dos cables.
- Letra N, control interno de la bomba para la inclinación, dos cables.
- Número 10, pulsador del cuadro de mandos que enciende y apaga el láser.
- Número 11, pulsador conmutador manual – automático.
- Número 12, interruptor del ordenador.

4.3.- Circuitería.

4.3.1.- Bloques de actuación.

En la parte de circuitería, podemos distinguir distintos bloques. Los circuitos empleados no son demasiado complejos, la complejidad está en encontrar soluciones

para la resolución de los distintos problemas, debido a la falta de información, como ya se ha mencionado en esta memoria.

Se puede distinguir así un bloque dedicado a la tracción, en el que destacan los potenciómetros digitales de Xicor x9312, también un bloque de actuación sobre el sistema hidráulico, en el que destacan los amplificadores operacionales de alimentación asimétrica LM324 de National Semiconductor. Existe otro bloque que podríamos denominar de activación y conmutación de señales, con relés, algunos activados por interruptor, y otros por circuitos excitadores de relés, mediante señales digitales. Hay un último bloque de conexiones, conexiones con los cables expuestos en el anterior capítulo, con el ordenador de control, y entre las dos placas que forman el circuito eléctrico.

El circuito se encuentra realizado en placas de prueba, con gran cantidad de cables soldados. Lo conveniente en un futuro será realizarlo en una placa de circuito impreso.

4.3.2.- Bloque de tracción.

En el caso de la tracción, ya se ha explicado el proceso de elección de los potenciómetros digitales como medio de actuación.

Para este bloque se utilizan cuatro potenciómetros x9312 de Xicor en paralelo, con lo que se consigue que, de los diez KΩ que tiene cada uno, se pase a unos 2.5 KΩ aproximadamente (ver figura 3.11). Las causas de hacerlo de esta forma ya se han explicado. Existían potenciómetros de menos valor en el mercado, pero eran más difíciles de utilizar y no tenían tampoco los valores adecuados. Para acercarnos a los valores del potenciómetro que el controlador interno de la carretilla espera, se han colocado potenciómetros con uno de sus extremos cortocircuitado con el cursor, a modo de resistencias variables, en los extremos y en el cursor de los potenciómetros digitales para adecuar los valores a los que la carretilla acepta (ver figura 3.10), pudiendo adaptar estos valores. Los valores de estos potenciómetros también están descritos en el apartado 3.2.1, dedicado a las soluciones de actuación propuestas sobre la tracción.

Pero pasemos a hablar de los potenciómetros x9312. Podemos decir que son potenciómetros digitales y su funcionamiento es relativamente sencillo. Este potenciómetro consta de una red de 99 resistencias en serie que se puede conectar al cursor mediante una serie de interruptores analógicos, la conexión se realiza en los

puntos entre las distintas resistencias, y en los extremos de dicha red de resistecias se encuentran los extremos del potenciómetro, así, podemos mover el cursor del mismo en saltos de, aproximadamente 10000Ω/100Ω, que son 100Ω. Para realizar esto se tiene un contador de siete bits, que tiene incremento y decremento, conectado con una memoria no volátil, que le permite almacenar la posición del cursor cuando no tiene alimentación. Este contador va a un decodificador de uno a cien que saca una salida digital a uno por una de sus salidas, activando uno de los interruptores analógicos que conectan el cursor a un punto concreto de la red de resistencias. Veamos el esquema en la figura 4.7.

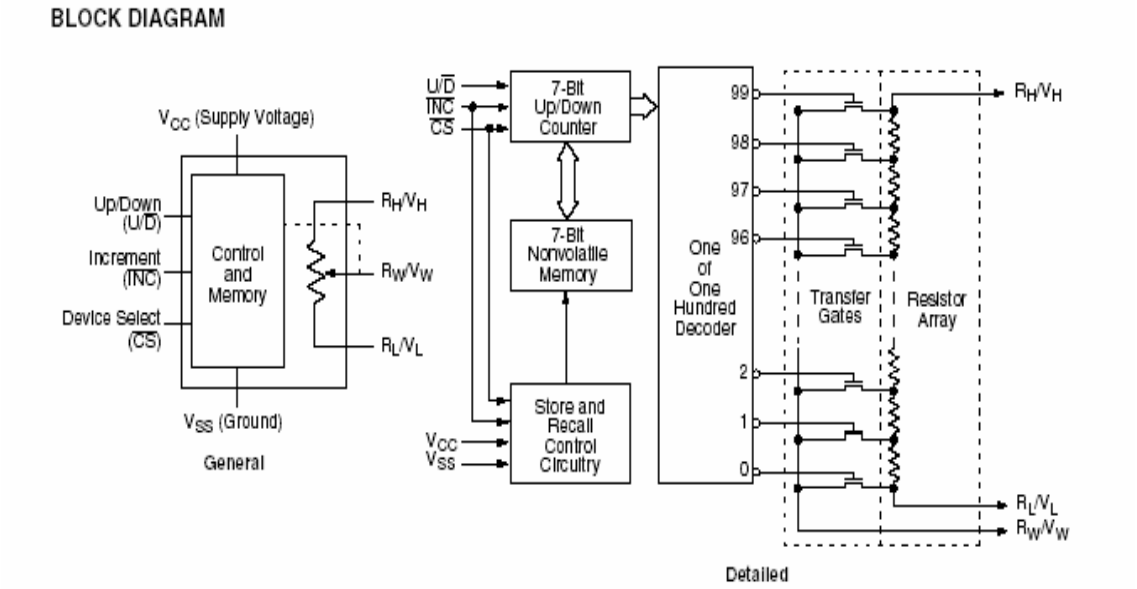


Figura 4.7: Esquema del potenciómetro digital x9312 de Xicor.

Se pueden observar las distintas patas, Up/Down, Increment, Device Select, alimentación, tierra y las tres patas del potenciómetro, la marcada con H ha de estar al voltaje positivo, la marcada con L, al negativo, en nuestro caso a tierra, y la marcada con W es el cursor. Estos potenciómetros tienen cierta limitación entre el voltaje mínimo del extremo negativo del cursor y la tierra, por eso no son simétricos como los potenciómetros convencionales, pero aguantan bien los quince voltios a los que se han de someter, pues esa es la alimentación que la carretilla da para el potenciómetro de tracción. En cuanto a la intensidad, también la soportan bien, teniendo en cuenta que se divide entre los cuatro que hemos montado en paralelo.

Externamente, el potenciómetro tiene el patillaje indicado en la figura 4.8.

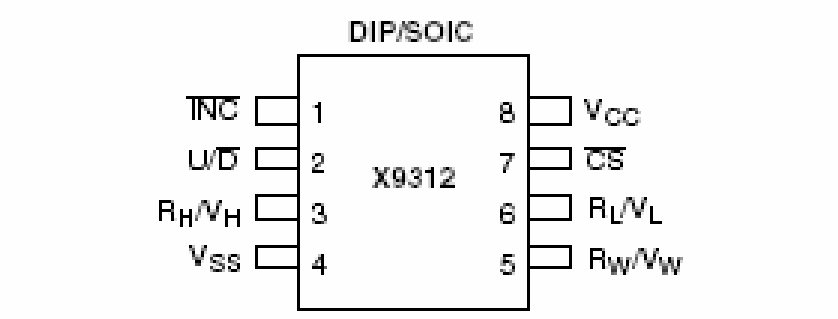


Figura 4.8: Patillaje del potenciómetro digital x9312 de Xicor.

Se puede ver que tiene ocho patas, V_{CC} es alimentación, V_{SS} es tierra. De la alimentación diremos que es de cinco voltios, y que se obtiene de las baterías de control de la carretilla, a través de la fuente de continua que alimenta al ordenador controlador. Esta fuente, como toda fuente de ordenador, tiene salidas de cinco voltios que no sólo son útiles para la alimentación del ordenador, sino también para estos potenciómetros. La alimentación llega a todos los chips, recordemos que hay cuatro, y la única consideración adicional a esto es la colocación de un pequeño condensador entre alimentación y tierra, físicamente cerca de los chips, que compensa las pequeñas caídas en la línea, y que mejora el funcionamiento de los potenciómetros y los hace más fiables.

En cuanto a R_H/V_H, R_L/V_L y R_W/V_W, sólo decir que son los tres terminales del potenciómetro, y que tienen el condicionante de voltaje indicado anteriormente.

Faltan las patas INC (activa a nivel bajo), U/D (incremento a nivel alto y decremento a nivel bajo, y CS (activa a nivel bajo), cuyo funcionamiento podemos ver en la figura 4.9.

$\overline{CS}$	$\overline{INC}$	U/$\overline{D}$	Mode
L	↓	H	Wiper up
L	↓	L	Wiper down
↑	H	X	Store wiper position
H	X	X	Standby current
↑	L	X	No store, return to standby

Figura 4.9: Funcionamiento de las señales del potenciómetro digital x9312.

Podemos ver que el CS es un chip select activo a nivel bajo, con él activo, podemos tener U/D en dos posiciones, si está a nivel alto, queremos decir que vamos a incrementar el potenciómetro, pero si está a nivel bajo, indicamos que queremos bajar el

potenciómetro. Así, con estas dos señales puestas adecuadamente, podemos subir o bajar el potenciómetro tantas veces como flancos de bajada se tengan en la señal INC. Un ejemplo: para bajar el potenciómetro siete pasos, pondremos CS a cero, U/D también a cero, y subiremos y bajaremos INC siete veces, de manera que haya siete flancos de bajada, con lo que bajaremos el potenciómetro. Si hacemos lo mismo, pero con U/D a nivel alto, tendremos que el potenciómetro sube siete pasos, y se coloca en la posición anterior.

Anteriormente hemos hablado de una memoria no volátil que el potenciómetro posee, para almacenar la posición en la memoria, basta con poner a nivel alto INC antes de que se deseleccione el chip, poniendo CS a nivel alto, ya que será en ese flanco en el que se almacenará el contenido del potenciómetro. Sin embargo, si se deselecciona el chip con la señal INC a nivel bajo, no se guarda la posición en memoria.

Los potenciómetros vienen en forma de chip DIP de ocho patas, y se conectaron a la placa en la que realizan su función mediante zócalos DIP soldados a la misma.

Existen más consideraciones de carácter técnico, como máximas frecuencias de funcionamiento, o tiempos de permanencia de las señales, así como impedancias de entrada de las patas, consideraciones de potencia que el dispositivo puede disipar, rangos de funcionamiento, tanto en voltaje o intensidad como en temperatura, tamaños. Todas estas consideraciones y otras se pueden encontrar en la documentación proporcionada por el fabricante que se incluye al final de esta memoria.

4.3.3.- Actuación sobre el sistema hidráulico.

Para la actuación sobre el sistema hidráulico, tenemos que convertir la señal de la tarjeta de actuación del ordenador, que va de cero a diez voltios, a una seña de seis a 18 voltios, que es la que las válvulas aceptan. Para esto se ha propuesto restringir el rango de salida de la tarjeta a tres a nueve voltios, y multiplicarlo por dos, con lo que tenemos los seis a 18 voltios requeridos, se pierde rango dinámico respecto a otras posibilidades, pero se obtiene a cambio gran sencillez en el circuito a realizar, ya que se necesitaban resultados visibles rápidos en el proyecto.

Para la multiplicación por dos, se optó por un sencillo montaje con amplificadores LM324, en montaje amplificador no inversor, con dos resistencias iguales que hacen que la multiplicación sea por dos, aproximadamente. Veamos la figura 4.10.

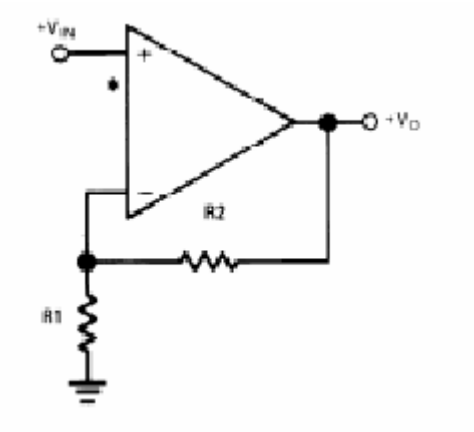


Figura 4.10: Montaje amplificador no inversor.

Como se puede ver en la figura, tenemos la siguiente relación entre V_O y V_{IN}:

$$V_O = \frac{V_{IN}}{R_1}(R_1 + R_2)$$

Si $R_1 = R_2$, como ya hemos dicho, $V_O = 2V_{IN}$, con esto, amplificamos por dos. Pero se han usado resistencias discretas sin paridad alguna, con lo que esto no es real, ya que R_1 nunca es igual a R_2 y V_O nunca es igual a $2V_{IN}$. En principio, y tratándose de un bucle cerrado, no nos debería preocupar demasiado, porque estas pequeñas diferencias se compensarían con el controlador, pero si se quiere una mayor precisión, se propone usar potenciómetros a modo de resistencia variable en vez de las resistencias discretas, como ya se han usado para otras aplicaciones en este proyecto, y realizar un ajuste más preciso hasta conseguir ganancia dos.

Existen otras consideraciones, como el hecho de que no haya alimentación estabilizada para los operacionales, importantes en el rechazo al modo común de los mismos, pero se han probado estos montajes y funcionan con la alimentación sin estabilizar, así que no hace falta estabilización, aunque, por otro lado, es aconsejable.

El amplificador utilizado es el LM324 de National Semiconductors, es un chip en el que se reúnen cuatro amplificadores operacionales de alimentación asimétrica, es decir, no se necesita $V_+=+V_{CC}$ y $V_-=-V_{CC}$, sino que se puede utilizar una alimentación en la que $V_-=0$, conectarla a tierra. Esto se hace interesante por varios motivos, como son que el rango de tensiones a la salida es sólo positivo, como que no se necesitan tensiones negativas y el amplificador puede saturar en el cero, que este hecho de tener sólo estas tensiones en una alimentación simétrica limitaría el rango de salida del amplificador, y obligaría a una diferencia entre V_+ y V_- de más de 36 voltios, y cuestiones meramente prácticas, como que se tienen 24 voltios de alimentación y se

complicaría el circuito al tener que sacar una alimentación simétrica y una tierra virtual para alimentar un amplificador asimétrico.

Se tienen dos de estos chips, ya que se usan cinco operacionales, con lo que un circuito tiene los cuatro operacionales usados, y el otro uno. ¿Por qué cinco?, porque no sólo se necesitan señales para las tres válvulas, sino que también se necesitan dos señales para excitar la bomba del circuito, aunque estas señales no se usan, usándose el relé de la dirección, como ya dijimos, se tienen disponibles por si acaso son útiles en una futura ampliación. Se probó una conmutación de las señales de la tarjeta, que son sólo dos, para atacar a los operacionales mediante relés, pero esto no funcionó, porque las válvulas se bloqueaban, para realizar esto, se proponen circuitos multiplexores analógicos, con un sample and hold detrás para mantener la señal a la salida y que no se bloqueen las válvulas. De todas formas, necesitamos tres señales, una para cada válvula, se usa una para la dirección, y la otra se conmuta con relés entre las dos válvulas del mástil, y así parece que funciona. Una descripción más detallada de las señales se hará en los apartados de conmutación y de conexiones. El patillaje del LM324 se puede ver en la figura 4.11.

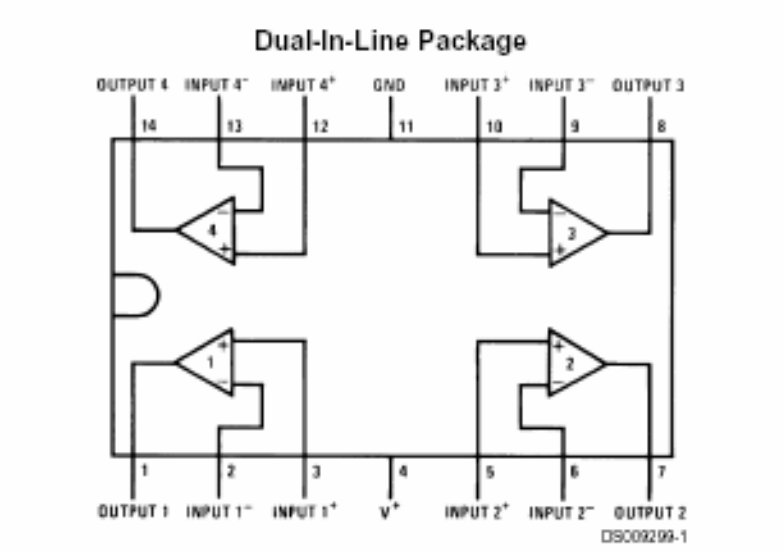


Figura 4.11: Patillaje del LM324 de National.

Este circuito se ha adquirido en su forma DIL, con sus zócalos de conexión debidamente soldados a la placa de adaptación de señales. Posteriormente se identificarán las entradas y salidas de la placa que corresponden a cada señal, tanto en la actuación como en la recepción de datos del ordenador. Información sobre este circuito integrado, rangos de funcionamiento, formas de presentarlo, circuitos de la misma

familia, patillaje, etcétera, así como sobre circuitos en los que se puede usar, se puede encontrar en el documento adjunto a la memoria, al final de la misma.

4.3.4.- Bloque de activación y conmutación de señales.

En este bloque se realiza la activación de las señales que son necesarias para el movimiento de la carretilla, así como la conmutación de las distintas señales, dependiendo de los modos de funcionamiento (manual y automático) y de la válvula a atacar.

Así, se tienen relés que conmutan de manual a automático. Todos los relés que se usan son dobles. Esto viene bien en ciertos casos en los que se usa una misma señal para conmutar dos o más señales.

En el caso del potenciómetro de tracción, se tienen dos señales que son el cursor y el extremo de quince voltios de las señales que el controlador interno de la carretilla espera, y que son conmutadas mediante relé entre el potenciómetro original y el potenciómetro digital que ya se ha descrito. Como el relé es doble, con uno solo de estos dispositivos realizamos la función. El relé está excitado por 24 voltios que provienen del conmutador manual – automático que está en el cuadro de mandos marcado con M/A.

Se necesita también la alimentación de las válvulas todo – nada, para la conmutación manual – automático. Esta conmutación se realiza por medio de otro relé doble, excitado con el mismo conmutador del cuadro de mandos a 24 voltios. Pero no sólo se alimentan estas válvulas al conmutar, sino que también se alimentan las válvulas proporcionales de control de la dirección y el mástil, en su inclinación y elevación.

Se usan también relés que son activados por señales digitales. Uno de ellos es para la señal que ha de ser activada antes de que se mueva la carretilla en el sistema de tracción, otro para la señal que echa a andar la bomba del sistema hidráulico, necesaria para el funcionamiento de la dirección, y otro para alimentar las válvulas con el objeto de que tengan antes de su encendido una señal coherente con lo que la válvula espera, evitando así su bloqueo, así, para que una válvula esté alimentada, se ha de tener la carretilla en estado automático, mientras que la señal de alimentación a las válvulas ha de estar a uno. Posteriormente mencionaremos las señales digitales encargadas de estas activaciones. Para excitar los relés con esas señales digitales se necesitan circuitos excitadores de relé como el de la figura 4.12.

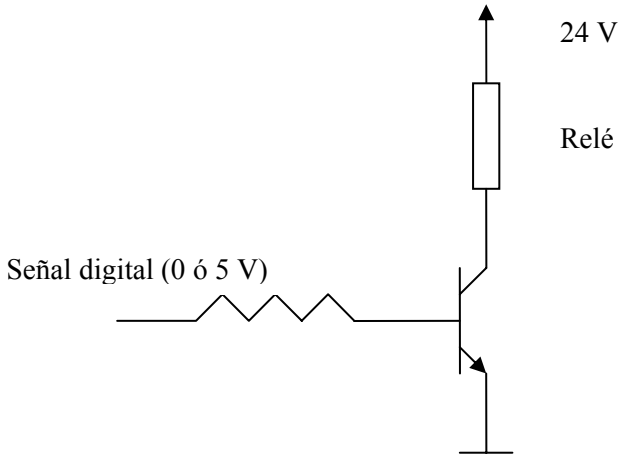


Figura 4.12: Circuito excitador de relé.

En este circuito se utiliza una señal digital en la base del transistor, para así cerrar o no el circuito, produciendo una excitación en el relé dependiente de esa señal digital. Con un uno en la base del transistor (cinco voltios), el relé se cierra y se puede controlar su estado. Con un cero en la base, el relé se desconecta.

Se tiene también un juego de relés que encamina mediante activación de señales digitales las dos señales analógicas de la tarjeta a cada uno de los puntos donde se necesitan, las tres válvulas de dirección y mástil y las dos señales que emulan los potenciómetros de activación de la bomba del circuito hidráulico, pero como esto no se usa y será, probablemente, sustituido por otro sistema basado en más señales analógicas de control o en un multiplexor de señales analógicas, no hablaremos más de ello, aunque se tienen, como ahora veremos, las conexiones para implementarlo.

4.3.5.- Bloque de conexiones.

En este bloque se habla de las distintas conexiones que se tienen en la tarjeta de control. Tiene como objeto nombrar las distintas conexiones, así como relacionarlas con el lugar físico marcado en la placa de esas conexiones.

Para conectar el ordenador a la placa de control, se ha utilizado un cable plano de 30 pines que ya describiremos.

El lugar de una conexión es una regleta en la que hay escritas letras como abreviaturas de la señal que va conectada o bien símbolos o colores que diferencian esa señal de las demás. Así, podemos nombrar e identificar todas las señales que entran y

salen de las placas. Este apartado está fuertemente relacionado con el apartado 4.2.4 en el que se habla de los distintos cables y de cómo están identificados, ya que muchos de estos cables van a parar a estas conexiones, siguiendo esto, se deben poder asignar cables a conexiones. Poco más hay que decir, sólo enumerar esas conexiones, y describirlas.

- O es la alimentación para el ordenador, son 24 voltios de las baterías que van a la fuente de alimentación de continua del ordenador, convenientemente conectados y que se activan cuando se enciende el pulsador marcado con la O en el cuadro de mandos. Esta protegida por fusible.
- V es la alimentación de las válvulas, son tres salidas de 24 voltios, activadas al conmutar de manual a automático, y a las que va esa alimentación de las válvulas.
- W_C es la señal que el control interno de la carretilla espera del cursor, se conmutará mediante relé y dependiendo del modo de funcionamiento (manual o automático) entre el potenciómetro original y el digital incorporado para la automatización.
- H_C es la señal que el controlador interno de la carretilla espera para el nivel alto del potenciómetro, está a unos quince voltios, y es lo que alimenta al potenciómetro, se conmuta de manera idéntica al caso anterior.
- W_P es el cursor del potenciómetro original, conmutado con el potenciómetro digital.
- H_P es el extremo a quince voltios del potenciómetro original, conmutado con el potenciómetro digital, por supuesto.
- 5V, entrada de cinco voltios, se usa para recibir cinco voltios de la fuente de alimentación del ordenador y con ella alimentar los potenciómetros digitales.
- 24V son cinco entradas de 24 voltios procedentes de las baterías, y que distribuyen la alimentación por el sistema con sus correspondientes fusibles de protección para láser, ordenador y válvulas y circuito. Son cinco para poder absorber más intensidad, puesto que son la alimentación.
- C es la señal que viene del conmutador manual – automático situado en el cuadro de mandos de la carretilla, y que lleva 24 voltios para excitar los relés correspondientes que conmutan las distintas señales y alimentan las válvulas en el modo automático.

- D'_2 es la señal de activación del relé de tracción. Viene de la señal digital D_5 de la tarjeta de entrada y salida, y es consecuencia de la necesidad de usar dos placas, ya que no cabía todo en una, conecta la placa donde están la mayoría de los circuitos con la segunda placa, en la que se encuentra el conector de cable plano con la tarjeta de entrada – salida y algunos relés.
- D'_1 es la señal de activación del relé de dirección, corresponde a la señal digital D_4 de la tarjeta, y existe por los motivos explicados en el anterior punto.
- A^2_1 y A^1_1 son las señales del relé de tracción, entre las que se conmuta. La primera es el positivo, y la segunda, tierra.
- A^2_2 y A^1_2 son las señales del relé de dirección, funcionan como en el caso anterior.
- Las conexiones de la regleta marcadas en color negro son la tierra, común en todo el circuito. Son varias, como en alimentación, para absorber toda la intensidad sin problemas.
- L es la alimentación del láser, 24 voltios convenientemente protegidos por fusible, que se activan al pulsar el interruptor marcado con la L en el cuadro de mandos.
- D_1, D_2 y D_3 son las señales digitales procedentes del ordenador que equivalen a esas mismas señales en la tarjeta de entrada – salida, y que son para las señales INC, U/D y CS del potenciómetro digital. Estas entradas son propiciadas por la existencia de dos placas, como ya se ha dicho, para la conexión entre ellas.
- E^2_V es la señal del potenciómetro de la válvula de elevación. Sólo se necesita esta señal, porque aplicando un voltaje entre este punto y la tierra, convenientemente conectada al negativo de este potenciómetro, podemos controlar con un voltaje el caudal que la válvula introduce. Está conmutada con la señal de la válvula original. No está conectada en este momento, por las razones que ya se expusieron, pero está disponible.
- E^1_V es una señal como la anterior, pero para el potenciómetro de la válvula de inclinación. Tampoco está conectada, estando estos cables conectados directamente con el controlador interno de la carretilla, como lo estuvieron en la carretilla original.

- E_4 es la salida del ordenador destinada a emular al potenciómetro de la válvula de elevación, se conmuta mediante relés para lograr controlar el flujo que introduce la bomba. Esta señal viene de la tarjeta de control, conectada mediante cable plano a la tarjeta de adaptación, y convenientemente conmutada.
- S_4 es la salida que va al controlador interno a la carretilla, y que tiene como objeto el controlar el flujo de la bomba, emulando el potenciómetro como una tensión variable. Corresponde a la señal E_4, que viene del ordenador controlador externo. Esto se ha montado así debido a la existencia de dos placas, con lo que se tiene que disponer de entradas como E_4 para comunicar la placa en la que llegan las señales del ordenador, con la otra placa, en la que están la mayoría de los relés que conmutan estas señales.
- E_5 es como E_4, pero para el potenciómetro de la válvula de inclinación. Se aplican los mismos condicionantes que en el caso anterior.
- S_5 es como S_4, pero para el potenciómetro de la válvula de inclinación.
- E_3 es la salida el ordenador que va a la válvula de dirección, convenientemente conmutada por relés, es consecuencia, como antes, de la existencia de dos placas.
- S_3 es la salida que va a la válvula de dirección para controlar su apertura. Ya se ha hablado de los rangos de voltaje de la misma.
- E_2 es como E_3, pero para la válvula de elevación, con las mismas características que E_3.
- S_2 es como S_3, pero para la válvula de elevación.
- E_1 es la salida procedente del ordenador para la válvula de inclinación.
- S_1 es la salida para la válvula de inclinación.

Muchas de estas señales pueden desaparecer cuando se realice un montaje mejor, no tan provisional, y en una placa de circuito impreso, no en una placa de prueba, ya que esto permitiría integrar todo el circuito en una sola placa, con la reducción de señales que ello conlleva, por ejemplo, las señales de comunicación entre las dos placas.

4.3.6.- Situación física del circuito.

El circuito se encuentra atornillado a una caja de PC IBM PS/2, que se ha vaciado para este fin, y a la que se le han hecho agujeros para la colocación de las dos

placas del circuito, que también han sido taladradas para este fin. En la misma caja se encuentra el ordenador de control, también atornillado a la caja. En un futuro, se ha de buscar una caja adecuada, tanto para el ordenador como para los circuitos, tratando de que ocupen un espacio pequeño, por cuestiones prácticas, ya que el espacio en la carretilla es reducido, y por cuestiones estéticas. De todas maneras, y teniendo en cuenta que el montaje es provisional, el hecho de integrar en una sola caja de PC todo el hardware necesario parece un esfuerzo relativamente válido en pos de esta miniaturización del hardware. El circuito se puede ver en la figura 4.13.

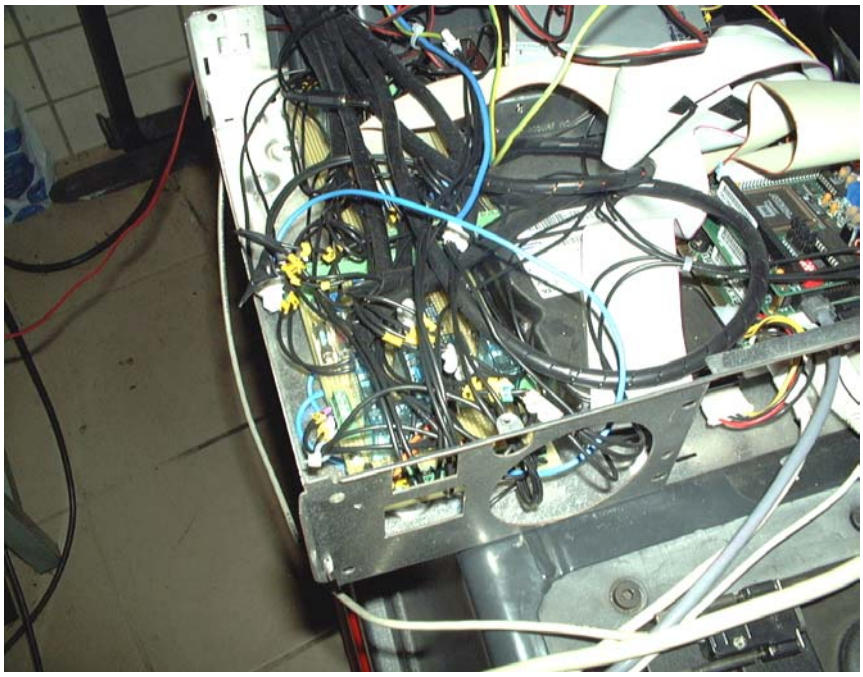


Figura 4.13: Circuito de conexiones.

Se puede ver la placa en la que está la mayoría del circuito de conexiones, y en la parte superior de la foto se encuentra la placa a la que llega el cable plano del ordenador. En la derecha tenemos el ordenador de control. Pero veámoslo más ampliado en la figura 4.14.

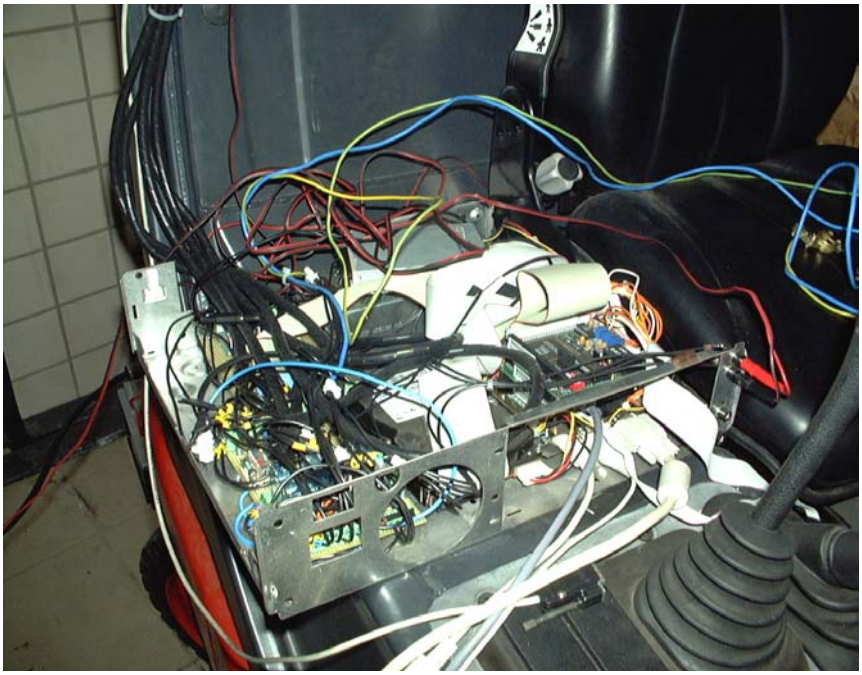


Figura 4.14: Vista ampliada del circuito.

En esta figura se pueden ver las dos placas del circuito, tapadas por los cables que vienen de la carretilla, identificados por las distintas etiquetas, como ya se ha dicho.

4.4.- Otros trabajos.

4.4.1.- Colocación de sensores.

En el caso de la carretilla, sólo disponemos por el momento de un sensor, el sensor láser Robosense de Siman. Este sensor ha de estar colocado en la parte alta de la carretilla, para evitar así que algo interfiera entre las balizas y él. Se ha colocado en el techo, atornillado a una superficie de madera que está, a su vez, atornillada al techo de la cabina. Así colocamos el sensor en una zona en la que no se molesta demasiado su visión, excepto por el mástil, pero eso es inevitable. No sólo se ha de colocar el sensor, sino que se ha de procurar su alimentación a 24 voltios y la conexión del mismo con el ordenador, mediante un cable RS-232 al puerto serie del mismo. Se han realizado estas tareas.

4.4.2.- Consideraciones de alimentación.

La alimentación del sistema se realiza mediante dos baterías de doce voltios cada una, colocadas en serie, y que satisfacen las exigencias de consumo que el sistema necesita. La carretilla tiene baterías de 48 voltios, pero no se usan para el sistema de control automático, sólo realizan su función primitiva de alimentar los sistemas propios de la carretilla, como la tracción, la electrónica del vehículo o la bomba del circuito hidráulico. No se han usado para la alimentación de circuito automático porque éste funciona a 24 voltios y se necesitaría un conversor de 48 a 24 voltios que diera potencia suficiente para alimentar ese circuito, y no se ha encontrado. Además, el hecho de necesitar la potencia que esas baterías suministran para el funcionamiento de la carretilla habría supuesto bajadas de tensión en la alimentación, perniciosas para el funcionamiento de los circuitos introducidos, aun estando la tensión de salida estabilizada por el conversor de 48 a 24 voltios.

Las baterías introducidas cumplen las consideraciones de consumo que se relatan a continuación.

Las válvulas, a 24 voltios, necesitan 3.75 amperios, con lo que consumen 90 vatios.

La electrónica y los relés no consumen más de 10 vatios, curándose en salud.

El láser necesita 0.8 amperios a 24 voltios, consumiendo así 19.2 vatios.

El ordenador consume unos 23 vatios, según las especificaciones del fabricante.

En total se consumen unos 132 vatios, con lo que se asegura el funcionamiento con estas baterías.

En la figura 4.15 se pueden ver estas baterías, marca Bosch modelo Silver mientras se están cargando con los cargadores de baterías destinados a tal cometido.

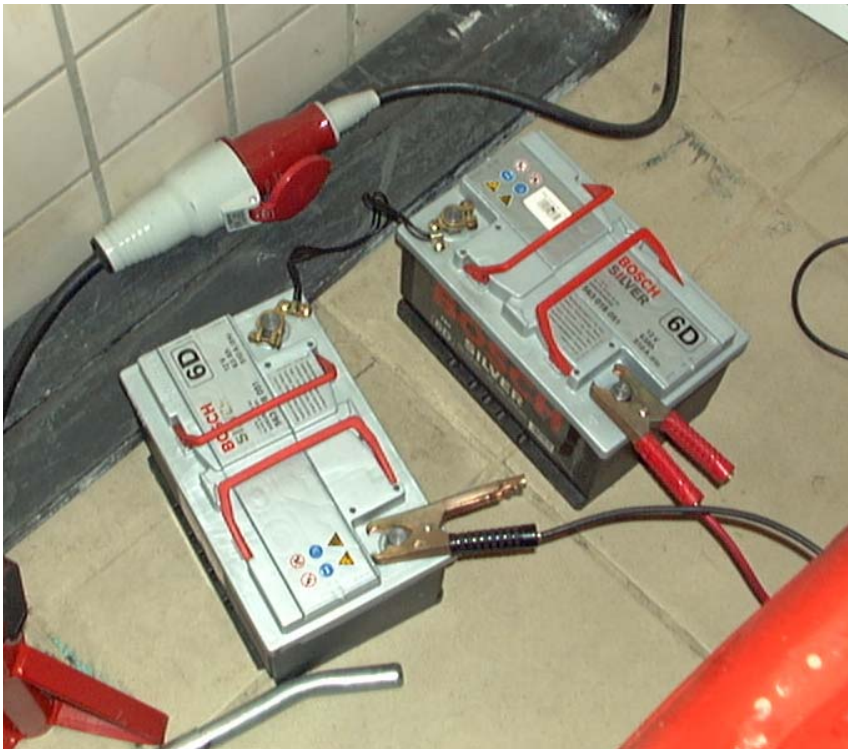


Figura 4.15: Baterías en carga a la vez, colocadas en serie.

Estas baterías se colocan en el lado derecho de la cabina, de manera provisional, y sobre ellas se coloca la caja del ordenador y los circuitos. Se han de bajar de la cabina cuando se cargue la batería de la carretilla, puesto que se ha de inclinar la cabina y no es conveniente inclinar las baterías, por la posible pérdida de fluido de su interior (ácido).

Son baterías sin mantenimiento y tienen una duración suficiente para la realización de experimentos. Se ha comprobado que la carretilla puede funcionar durante unas dos horas sin problemas con estas baterías y con los sistemas que tiene montados, lo cual no asegura tanto tiempo si se montan más sensores, como es obvio. De hecho, es posible que duren más, pero, como ocurre con las baterías de la carretilla, no es conveniente su descarga a más del 80 % de su capacidad, conllevando la posible rotura de la batería, por lo que no se ha probado más de esas dos horas.

Capítulo 5: El sensor Robosense.

5.1.- Descripción externa.

5.1.1.- Generalidades.

El sensor Robosense es un sensor electro – óptico de tipo A (no es necesaria la protección para los ojos) diseñado para la orientación de vehículos autónomos en un entorno balizado, fabricado por la empresa israelí SIMAN.

Está pensado para robots móviles o AGVs y preparado para ser utilizado en condiciones industriales de trabajo, permitiendo funcionar a los vehículos con un cierto grado de autonomía.

Incorpora siete funciones principales: Láser pulsado, escaneo circular, unidad de auto – calibración, módulo de procesado de señal, ordenador de navegación, módulo de comunicación y fuente de potencia. Como se puede ver en la figura 5.1.

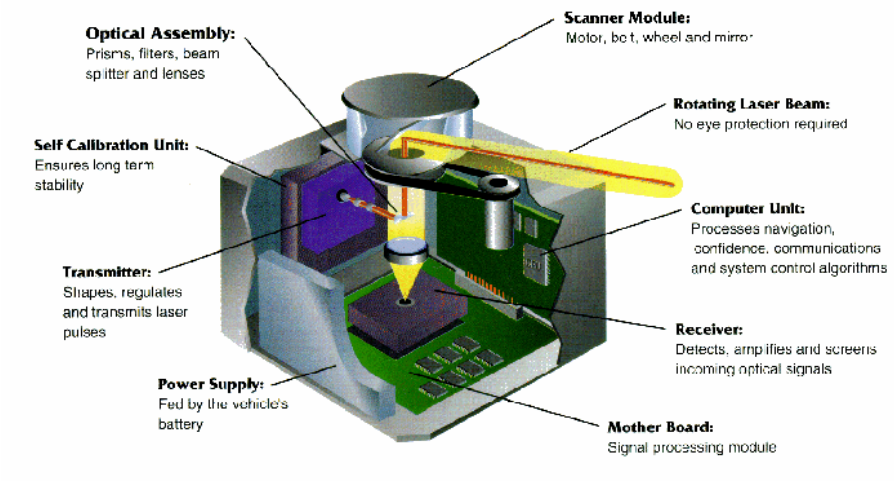


Figura 5.1: Esquema de funcionamiento del láser Robosense.

Podemos observar en el esquema que se tiene una torreta con un motor que hace girar un espejo, con lo que se escanea en el plano de giro del espejo de manera circular con el láser, producido y encaminado hacia el espejo en el interior del sensor. Se pueden

ver también las placas de procesado de señal y de control del giro del espejo, así como la placa de navegación, comunicaciones y algoritmos de control del sistema.

Funciona con 24 voltios, y no es necesario que estén estabilizados, pues pueden venir directamente de las baterías. Consume, como ya se ha dicho, unos 0.8 amperios.

Su peso es de 4.2 kilogramos, esto, unido a su reducido tamaño, hace que sea fácil de instalar en un vehículo.

Con el sensor se proporcionan también seis blancos o balizas, que usa para la navegación, y que tienen forma cilíndrica, con once centímetros de diámetro y 53 centímetros de altura. Están recubiertos de un material reflectante similar al que se usa en las balizas de señalización de tráfico, y que se puede comprar en láminas. Si algún sensor se pierde o deteriora, se pueden fabricar con un tubo de plástico de desagüe de un canalón común de un tejado, recubriéndolo con esas planchas reflectantes, que son adhesivas. Basta con cortar el tubo a las dimensiones del alto del cilindro. Veamos una en la figura 5.2.

Figura 5.2: Blanco para la navegación del sensor Robosense.

El ordenador de navegación es un 586 a 133 megahertzios, montado en una placa palm sized 5066 SBC (single board computer) de Octagon Systems, es decir, en una pequeña placa que cabe en la palma de la mano. Esta placa incorpora también el módulo de comunicaciones, consistente en un protocolo RS232 full duplex, como el de los puertos COM1 o COM2 del ordenador.

Este ordenador incorpora software de control, comunicaciones, auto – calibrado, etcétera.

Para la prueba del sensor, se distribuye también un programa bajo MS – DOS denominado Robodemo, en el que se pueden probar todos o la mayoría de los comandos del sensor, así como un programa Transfer que permite cargar ficheros en el láser, o descargárselos de él.

Para el manejo se entrega también un programa llamado Sitemap, en su versión 2.00, con este programa se pueden definir distintos entornos balizados en los que se puede navegar, y permite así, pasar de unos entornos a otros. Con esto lo que se pretende es navegar en lugares divididos, por ejemplo, en estancias en las que, estando en una, no se ven los blancos de la otra y viceversa. También permitiría pasar puertas. Es un programa en MS – DOS, que pasa información de los distintos entornos al controlador del sensor para permitirle la navegación. No se ha utilizado mucho en este proyecto, sólo para probarlo y sin muchos resultados. Exige un modelado del plano donde se desea la navegación automática, así como la posición de los blancos.

Para una mayor información, se puede consultar el manual que también se adjunta con el sensor.

5.1.2.- Instalación.

El sistema se instala en el techo del vehículo, ya que se ha de procurar que esté paralelo al suelo, de manera que el plano que el sensor barre en su escaneo sea paralelo al suelo. Esto ha de ser de esta forma porque debe recibir la luz láser rebotada de las blancos, con lo que para que rebote y llegue otra vez al sensor, ha de incidir en ellos perpendicularmente.

Se ha de procurar que la visión de los blancos que tiene el sensor sea interrumpida lo menos posible, aunque está permitida una interrupción limitada en el tiempo y parcial, es decir, de algún blanco, no de todos a la vez.

En cuanto a la instalación de los blancos, aparte de que estén visibles desde el sensor, han de ser instalados de manera vertical, para que reflejen el láser del sensor. Siguiendo este método, lo más apropiado es colgarlo de la pared que delimite el entorno, manteniendo el blanco totalmente vertical. Se ha de colgar, como es obvio, a la altura que permita que sea “visto” por el sensor, y es aconsejable que el láser impacte aproximadamente en el centro del blanco.

Para mejorar el resultado de los blancos, se pueden fabricar los llamados blancos omega, en los que se recubre la pared en la zona en la que está colgado el blanco de un material no reflectante, una cartulina negra, por ejemplo. Es importante darse cuenta de que la poca potencia del láser, así como el alto poder de reflexión de los blancos da lugar a una menor existencia de falsos blancos, que pueden ser muy perjudiciales, sobre todo en el momento del mapeo, como ahora veremos.

Para que los blancos sean visibles por el sensor, el suelo ha de tener inclinaciones de $\pm 1^{\circ}$. Se pueden colocar dos blancos, verticalmente uno encima del otro, para rangos esperados del sensor mayores de 25 metros.

En cuanto a la alimentación, es de 24 voltios con un consumo de intensidad de 0.8 amperios, como ya se ha dicho. Se conecta mediante un receptáculo tipo Burndy UTG 010-4S hembra de cuatro pines. Ha de ser protegido con un fusible de 2.5 amperios. Cuando está encendido se indica con un LED verde.

Para conectarlo al ordenador de control, que denominaremos host, se necesita un cable de MODEM nulo RS232 de tres hilos (transmisión, recepción y tierra), con un conector estándar de nueve pines de tipo D. El protocolo RS232 es con ocho bits de datos, un bit de stop y sin paridad, a 19200 bits por segundo, y así hay que configurar el host para realizar la conexión.

5.1.3.- Funcionamiento.

El láser realiza un escaneo circular emitiendo en un plano paralelo al suelo, por efecto de un motor eléctrico que hace que la fuente láser gire, y va encontrando los distintos blancos. Al encontrar un blanco, el láser rebota y vuelve al sensor, con lo que se tiene una medida en el sensor de la distancia al blanco, así como del ángulo que ha girado el motor, referido a un punto que representa el ángulo cero, marcado con un línea en la carcasa del láser, al lado de la "torreta" de la que sale la luz láser.

Entre las distintas funciones que tiene el dispositivo, hay una de radar en la que recibe simplemente la distancia y posición de los blancos y la manda al host que la imprime por pantalla, así como otras muchas entre las que destaca la posibilidad de realizar un mapa del entorno en cuanto a la posición relativa del vehículo a los blancos.

Se ha de disponer de cuatro o más blancos instalados, y tras recibir información de su distancia y posición, se realiza el mapeo mediante un algoritmo similar al explicado en el capítulo dos, cuando se hablaba de la navegación. Se supone que es

similar, puesto que, en realidad, no se conoce nada del algoritmo. Tras esto, se manda al host la posición del vehículo en el nuevo mapa. Del mapa sólo se conoce como lo construye, porque es necesario en ciertos criterios a la hora de navegar. Desde el cero de ángulos, si giramos en el sentido contrario a las agujas del reloj nos acabaremos encontrando un blanco, este blanco representa el punto origen de coordenadas (0,0). Al continuar el giro, se ha de encontrar otro blanco, este blanco marcará junto con el origen el eje X. Así, ya el eje Y es perpendicular al eje X por el origen, y si pedimos al sensor la posición de un blanco, nos la dará referida a este mapa. Se desea llamar la atención sobre que el mapa creado es de dos dimensiones, como el escaneo. En la figura 5.3 vemos un esquema del mapeo.

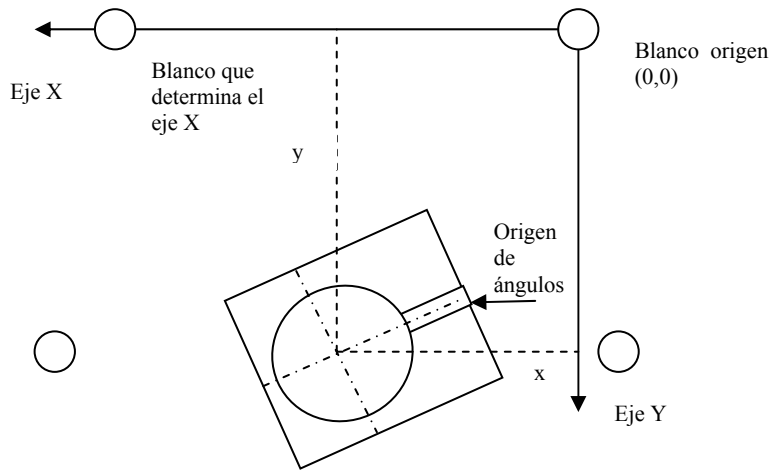


Figura 5.3: Mapeo visto desde arriba.

Tras este mapeo, se puede realizar la navegación dentro del mapa. Cada cierto tiempo se dará la posición dentro del mapa creado, y se enviará al host. Esta posición consiste en la coordenada X en el mapa, la coordenada Y y el ángulo del vehículo con el eje X, así como el nivel de confianza, que es una medida en porcentaje de lo que se puede confiar en la medida, ya que puede ser peor de lo normal por pérdida de blancos o excesiva proximidad a alguno de ellos.

Todas estas órdenes se mandan en primera instancia utilizando el programa Robodemo, que permite también otras opciones, como recibir el mapa en el host, con las coordenadas de los blancos, dar orden de una recolocación ciega de la posición del vehículo, para cuando se ha movido en el mapa sin que se esté realizando la navegación y sin actualizar la posición, con lo que se pierde esta posición y se ha de recuperar, y

otras muchas funciones que serán descritas al hablar del protocolo de comunicación, puesto que todas estas funciones se identifican con órdenes que el host manda al láser y con mensajes que el láser devuelve al host.

5.2.- La placa de control.

5.2.1.- Descripción general.

Para el control de las funciones de láser, éste posee una placa de control que proporciona funciones de comunicación con el host, así como funciones de tratamiento de los datos, ya que es capaz de interpretar la información proveniente del láser para tratarla y conseguir así mandarla al host con sentido, es decir, que es capaz de realizar cálculos para determinar un mapa, o para inferir la posición del vehículo dentro de un mapa. Estos datos son enviados al host tras ser procesados, con lo que en el ordenador de control del vehículo tenemos la información que necesitamos y en el formato en el que la necesitamos. Para realizar esto, la placa tiene un programa en su memoria no volátil encargado de todas estas tareas, se trata del programa Robosense.exe, que se proporciona con el software del sensor. Este programa no está destinado a ejecutarse en el host, sino en el Robosense. Se puede cargar en el sensor mediante la utilidad Transfer.exe. Para su correcto funcionamiento, también se poseen ficheros de configuración y de calibración, como el fichero kiul.dat, que sirven para compensar el efecto de la temperatura, muy importante como luego veremos. Se pueden transferir otros ficheros, como el mapstore.dat, el targets.dat, el popstore.dat, el frames.dat o el zones.dat, relacionados con el mapa, con la posición de los blancos y con las zonas de funcionamiento proporcionadas por el programa sitemap.exe, del que ya hemos hablado.

Podemos ver la placa en la figura 5.4.

Figura 5.4: Placa SBC 5066 de Octagon Systems, utilizada en el Robosense.

5.2.2.- Funcionalidades de la placa.

La placa es la SBC 5066 de Octagon. Es una placa de tipo palm sized integrada en el sensor, dentro de la carcasa, y que ofrece las siguientes funcionalidades:

- Placa 586 a 133 Mhz, equivalente en velocidad a un Pentium 75.
- DOS 6.22 en ROM de la placa.
- Compatible con Windows, Windows NT y QNX.
- BIOS Phoenix con extensiones industriales.
- Administración avanzada de consumo.
- Disco flash, con programador, es decir, que puede ser usado para cargar datos (hace las veces de disco duro).
- Posibilidad de más de 33 Megabytes de DRAM, aunque esta placa posee sólo un mega.
- Dos puertos serie (COM1 y COM2), con chip 16C550.
- Un puerto paralelo multifunción.
- Función watchdog.
- Software de diagnóstico.
- Sistema de ficheros flash.
- Interrupciones aisladas ópticamente.

- Funciona con cinco voltios, proporcionados por la fuente de alimentación del Robosense, que recibe 24 voltios y los convierte en cinco para la placa, aparte de alimentar con el voltaje que corresponda los otros dispositivos del sensor.

Con todas estas características, la placa puede ser programada, así como hacer que arranque de manera directa con el programa cargado, como es por otra parte, lógico en aplicaciones de automática.

Se puede seleccionar, mediante un jumper, el modo de arranque, si se arranca desde la EPROM (denominada SSD0) que tiene el sistema operativo DOS 6.22 en ROM, o bien si se arranca desde el programa que se encuentra cargado en la flash (denominada SSD1), y que es un programa de usuario cargado mediante la utilidad transfer, por ejemplo, siempre con mecanismos que permiten arrancar el sistema desde la flash y cargar y ejecutar el programa de usuario.

En el caso del láser, se tiene en flash el sistema operativo, para que arranque la placa, en lo que tarda unos 25 segundos, y además el programa de usuario, que se carga tras arrancar poniéndolo en el autoexec.bat.

También se puede seleccionar con un jumper si se arranca en modo terminal serie o no. La placa es capaz de proporcionar un terminal serie por el puerto COM1, y con cualquier programa de emulación de terminal, como el Hyperterminal, tenemos una consola en la que podemos manejar la placa, pasar ficheros de EPROM a flash, ejecutar comandos, etcétera. Para el paso de ficheros se tienen utilidades especiales. Existen utilidades de formateo de unidades flash, e incluso una utilidad para rearrancar el sistema. Existe la posibilidad de configurar el puerto para la comunicación a distintas tasas binarias, pero siempre con ocho bits de datos, uno de stop y sin paridad. Parece aconsejable usar 9600 bits por segundo, la que viene por defecto, y con la que funciona bien.

5.2.3.- Problemas encontrados.

No es un accidente común, pero puede ocurrir que se borre o modifique la EPROM del sistema. En esta EPROM se encuentra tanto la BIOS como el sistema operativo. En nuestro caso, este accidente ocurrió, propiciando un mayor conocimiento de la placa en cuestión, ya que hasta el accidente no se había necesitado conocer nada sobre ella. No se conocen las causas del accidente, pero es probable que viniera por un

fallo de alimentación producido porque se soltó uno de los cables de alimentación de su borne en el conector, dando lugar a algún cortocircuito, sobretensión repentina o un fallo en la lectura de la EPROM.

El problema se solucionó cuando se recibió una segunda EPROM en buen estado mandada por el fabricante del sensor. La ayuda proporcionada por la división española de Octagon Systems sirvió para aprender más sobre el funcionamiento de la placa, así como para las cuestiones de carga del programa. Sustituyendo la EPROM y cargando el sistema operativo y el programa de usuario de nuevo en la flash, así como procurando que la placa arranque desde flash y no desde EPROM se solucionó el problema.

Aunque mandar físicamente otro chip no habría sido necesario. Hubiera bastado con mandar vía e-mail el contenido de la EPROM y reprogramarla. Esto se comprobó sacando el contenido de la nueva EPROM y grabándolo en la antigua, con lo que la placa funcionó correctamente con la EPROM antigua ya arreglada.

El chip EPROM que se utiliza es un chip común de 512 kbytes y se dispone de medios en el laboratorio para su programación, un programador de EPROMs normal, así como el software que reconoce la EPROM. Pero el hecho de que sea de 16 bits dio lugar a una peculiaridad en su programación, ya que se tenía que hacer en dos veces, programando primero una mitad y luego programando la otra. Hay medios en la aplicación que hay en el laboratorio para programar EPROMs para cargar una imagen binaria (.bin) como la que se tiene para esta memoria, ya que se puede cargar a partir de un punto de la memoria y también a partir de un punto de la imagen. Con esto, sólo hay que cargar la primera mitad de la imagen en la primera mitad de la memoria, y la segunda mitad de la imagen en la segunda mitad de la memoria.

De todas formas, si el accidente volviera a ocurrir, se tiene el chip antiguo de la EPROM con la BIOS y el sistema operativo cargado, sólo habría que sustituirlo.

Pero nos podemos preguntar por qué falló realmente el sistema, si no arrancaba desde esta EPROM, sino desde la flash, donde estaba el programa de usuario, y el problema venía porque, como ya hemos dicho, en la EPROM estaba también la BIOS, con lo que la placa no sabía arrancar, no es que al arrancar no encontrara el sistema.

En el ordenador de la carretilla hay mucha información del funcionamiento de la placa, así como los programas que pueden ayudar a cargar aplicaciones en la flash y el sistema operativo que se encuentra cargado en la flash, con el autoexec.bat que permite arrancar la aplicación robosense.exe al arrancar la 5066. También se tiene en el

ordenador la imagen binaria de la EPROM. Lástima que mucha de esta información sea en forma de e-mails con Octagon Systems de España y no sean archivos más accesibles. De todas maneras, si no se repite el accidente, no es necesario usar más que la utilidad transfer.exe, proporcionada por SIMAN, que es una utilidad común de transferencia de ficheros por el puerto serie, basta con configurarla para 19200 bits por segundo, ocho bits sin paridad y un bit de stop, y decirle el puerto serie del host en el que se transmite. Hay más información sobre esta y otras formas de transferencia en el manual del Robosense. Se possee documentación sobre esta placa.

5.2.4.- Comunicaciones de la placa.

La 5066 utiliza su puerto COM1 para el terminal serie, así que la comunicación del programa de usuario, robosense.exe, con el host, se realiza por el puerto COM2. Este puerto ha de estar configurado a 19200 bits por segundo, con ocho bits de datos, uno de stop y sin paridad. Se utiliza un protocolo de caracteres que ahora veremos para comunicar el host con el láser.

El puerto paralelo de que dispone se utiliza en esta aplicación para la comunicación de la 5066 con el controlador del láser, que es un microcontrolador del que no se conoce mucho, salvo que realiza el procesado de la señal que viene del láser, la interpreta y la pone en un formato digital para realizar operaciones con ella en la 5066.

5.3.- El protocolo de comunicación.

5.3.1.- Consideraciones generales.

Como ya hemos visto, el Robosense se comunica con el host mediante un protocolo de caracteres, que viaja mediante una conexión serie RS232 en un cable de MODEM nulo de tres hilos a 19200 bits por segundo, con ocho bits de datos, un bit de stop y sin bit de paridad.

Se establecen una serie de mensajes para la comunicación con el sensor. Aunque se tiene un programa en MS – DOS que permite esta comunicación, es un programa hecho, ejecutable, sin el código para poderlo integrar en otro programa, por eso y porque no se desea un programa bajo MS – DOS sino bajo Linux, se ha de conocer este protocolo para realizar programas de comunicación que corren en el host y permiten mandar órdenes al sensor y recibir y tratar datos de él.

Aparte del programa de automatización de la carretilla, en el que está metido el programa de comunicación con el láser, se ha programado también una aplicación muy similar al robodemo.exe que proporciona el fabricante, quizá no tiene todas las funciones que tiene el programa original, pero funciona mejor y se cuelga menos que el proporcionado, quizá por estar programado bajo Linux, que es multitarea, no como el caso de MS – DOS, que necesita de otros procedimientos para lograr la multitarea que no parecen llevarse demasiado bien con el sistema operativo.

El protocolo está basado en el principio maestro – esclavo. El ordenador del vehículo, hasta ahora llamado host, es el maestro, y el Robosense es el esclavo. Cada mensaje por parte del maestro es contestado por el esclavo dentro de un periodo de tiempo predeterminado. Si no se manda en ese periodo, es he de tomar una determinación sobre el mensaje. Lo que aquí se hace es reenviar el mensaje en el maestro.

Para la transferencia de archivos, el protocolo es distinto. Como ya se ha dicho, se utiliza el programa de MS – DOS transfer.exe, proporcionado por el fabricante.

5.3.2.- Formato de los mensajes.

Un mensaje es una secuencia de hasta de 256 bytes. El primer byte de la secuencia es el HANDSHAKE, y vale 0x0D en hexadecimal, utilizando la notación que usa el C para estos números, que se usará a partir de ahora. Este carácter está definido como un carácter especial y no será utilizado dentro de un mensaje, habrá de ser sustituido por otros caracteres, como ya veremos posteriormente. No se considera parte del mensaje, y, por lo tanto, no entra en el checksum ni cuenta para el tamaño en bytes del mensaje.

El segundo byte de la secuencia es la longitud del mensaje. Es el número de bytes del mismo, menos uno, debido a que el HANDSHAKE no se cuenta.

El tercer byte es la identificación del mensaje, y su valor es el número que identifica la instrucción como un código. Por ejemplo, el código 18 en el programa robodemo.exe significa poner el Robosense en modo radar, el mensaje que se le manda para que realice esto tiene en su byte de identificación un 18.

Tras estos bytes vienen los bytes de datos del mensaje, que dependen del tipo de mensaje, hay, incluso, mensajes sin datos.

El último byte de un mensaje es el checksum, es el complemento a uno de la suma de todos los bytes del mensaje, módulo 256. Luego veremos esto con más detenimiento.

5.3.3.- Caracteres especiales.

Dos son los caracteres especiales, el 0x0D, que ya hemos visto, y el 0x1B.

El primero es el carácter de HANDSHAKE, que podríamos traducir como protocolo o puesta de acuerdo. Inicia todos y cada uno de los mensajes de la comunicación y nunca aparece en el cuerpo del mensaje.

Pero, ¿qué ocurre si uno de los datos del mensaje es un valor de 0x0D? Aquí es donde aparece el segundo de los caracteres especiales del mensaje, que llamamos ESC y que es el 0x1B. Lo que se hace es sustituir el carácter 0x0D por dos caracteres, el primero es el 0x1B y el segundo es el complemento bit a bit del 0x0D, es decir, el 0xF2.

Esta contingencia obliga a sustituir también el carácter especial 0x1B, si aparece en el cuerpo del mensaje, por dos caracteres, que no son otros que el propio carácter ESC 0x1B, y su complemento bit a bit, el 0xD8.

Así, en la otra parte, se ha de deshacer el cambio para que el mensaje sea legible, y tengamos los datos que se pretendían enviar en el receptor. Es decir, que cualquier par de caracteres encabezado por el carácter ESC se sustituye por el complemento bit a bit del carácter que sigue a ESC.

Veamos un ejemplo:

El mensaje a enviar es:

- Byte 1: 0x06 Longitud del mensaje.
- Byte 2: 0x0D Identificación del mensaje.
- Byte 3: 0xF2 Primer byte de datos.
- Byte 4: 0x1B Segundo byte de datos.
- Byte 5: 0xAB Tercer byte de datos.

- Byte 6: 0x35 Byte de checksum.

El mensaje se convierte antes de la transmisión en:

- Byte 1: 0x0D Byte de HANDSHAKE añadido a todo mensaje.
- Byte 2: 0x06 La longitud no cambia a pesar de los caracteres especiales y del HANDSHAKE, como ya dijimos.
- Byte 3: 0x1B Debido a la conversión de 0x0D en 0x1B y 0xF2.
- Byte 4: 0xF2.
- Byte 5: 0xF2 Es como el anterior, pero como no tiene 0x1B delante, se lee tal cual es.
- Byte 6: 0x1B Debido a la conversión de 0x1B en 0x1B y 0xD8.
- Byte 7: 0xD8.
- Byte 8: 0xAB.
- Byte 9: 0x35 Tampoco afecta al cheksum esta conversión.

Como se puede observar, el checksum se ha de hacer en la transmisión antes de recodificar el mensaje por efecto de los caracteres especiales y ya no se toca a pesar de que estos caracteres se transmitan. Por esto, en la recepción, se ha de transformar estos caracteres especiales en los originales para comprobar después el checksum del mensaje original.

Algo muy similar podemos aplicar a la longitud del mensaje, que es la original, y en la que no intervienen los caracteres especiales.

5.3.4.- Creación y comprobación del checksum.

Como ya hemos dicho, el checksum es el complemento a uno de la suma de los bytes del mensaje módulo 256, y sin contar los bytes de HANDSHAKE y de checksum.

El mensaje tiene esta forma general:

$$M = \{B(k)\} \rightarrow |1 \le k \le B(1) = n - 1$$

Que no es más que decir que cada byte tiene un cierto valor, y que hay tantos bytes como el valor del primero de estos octetos, que es el que indica el número de bytes del mensaje.

Así el checksum es:

$$B(n) = 256 - \left[\left(\sum_{1}^{n-1} B(k)\right)\right] módulo 256$$

Esto quiere decir que el valor del mensaje en n, que es el octeto de checksum, es el resultado de restarle a 256 el resto de la división entera (módulo) de la suma de los valores de cada uno de los octetos anteriores entre 256.

Para comprobar si el mensaje es correcto en la recepción, sólo hay que sumar los bytes recibidos tras eliminar los caracteres especiales, como ya se ha dicho, incluido el octeto de checksum, y el resultado ha de ser divisible por 256. Es decir, que el resto de la división entera entre 256 ha de ser cero, que el módulo 256 ha de ser cero.

Veámoslo:

$$\left[\sum_{1}^{n} B(k)\right] módulo 256 = 0$$

Esto es así, porque en la transmisión no hacemos más que añadirle al mensaje un byte en el que ponemos el valor que le falta a la suma de los valores de los octetos para ser divisible entre 256, por eso, si le hacemos el módulo a la suma de los valores de los octetos y no es cero, el mensaje ha llegado mal y ha de ser descartado.

Se desea recalcar que en recepción sí que se ha de sumar el octeto de checksum en la comprobación.

Esta medida es sólo de prevención de errores, no los corrige y no es infalible, pero sirve para disminuir en gran medida la probabilidad de error. Como no los corrige, la única solución es el descarte del mensaje y su retransmisión.

Veamos como sería en el ejemplo anterior:

- Byte 1: 0x06 Longitud del mensaje. 6
- Byte 2: 0x0D Identificación del mensaje. 13
- Byte 3: 0xF2 Primer byte de datos. 242
- Byte 4: 0x1B Segundo byte de datos. 27
- Byte 5: 0xAB Tercer byte de datos. 171

Resultado de la suma: 459

$$459 módulo 256 = 203 \qquad 256 - 203 = 53 = 0x35$$

- Byte 6: 0x35 Byte de checksum 53

Ahora se debería someter al mensaje a los cambios debidos a la existencia en el cuerpo del mensaje de caracteres especiales, además de añadirle el octeto de HANDSHAKE, sin que ello suponga ningún cambio del checksum calculado.

Veamos cómo se comprobaría el checksum en la recepción, siempre deshaciendo antes los cambios provocados por los caracteres especiales:

- Byte 1: 0x06 Longitud del mensaje. 6
- Byte 2: 0x0D Identificación del mensaje. 13
- Byte 3: 0xF2 Primer byte de datos. 242
- Byte 4: 0x1B Segundo byte de datos. 27
- Byte 5: 0xAB Tercer byte de datos. 171
- Byte 6: 0x35 Byte de checksum 53

Resultado de la suma: 512

$$512 módulo 256 = 0$$

Y vemos, de este modo, que el mensaje es correcto o, al menos, tiene una alta probabilidad de serlo. Lo que sí que sabríamos, en todo caso, es que un mensaje con la comprobación de checksum incorrecta ha sufrido alguna modificación en el tránsito por el medio de transmisión, puede, aun así, que el mensaje sea correcto y que el error esté en el byte de checksum, pero esa posibilidad es remota, y hemos de rechazar estos mensajes. En la práctica, aceptamos los mensajes de checksum correcto, y descartamos los de checksum incorrecto.

5.3.5.- Protocolo.

Cuando llega al Robosense o al ordenador del vehículo un octeto de HANDSHAKE, comienza a recibir un nuevo mensaje. Cualquier mensaje anterior incompleto es ignorado. El comportamiento del maestro y del esclavo a la recepción de un mensaje es distinto.

Si el mensaje es recibido por el esclavo, éste responde con un mensaje de aprobación, que ha de ser transmitido antes de 250 milisegundos. En la práctica, no podemos saber en el maestro si el mensaje se ha transmitido en el esclavo antes de ese tiempo, por lo que despreciaremos el tiempo de propagación del mensaje, y consideraremos que se ha excedido ese tiempo si han transcurrido 250 milisegundos desde que se mandó el anterior mensaje por parte del maestro, que es el mensaje al que el esclavo ha de contestar. Las causas de pérdida del mensaje se deben a defectos del medio de transmisión o a ruidos, y puede ser que el mensaje que se haya perdido sea el que manda el maestro, o bien la respuesta del esclavo. Incluso puede no haber respuesta por indisposición del esclavo. Cuando no se recibe el mensaje de aprobación en menos de esos 250 milisegundos, se han de tomar medidas, una de ellas puede ser retransmitir el mensaje en el maestro, para esperar una nueva respuesta.

El mensaje de aprobación es de la forma:

- Byte 1: Número de bytes. 4
- Byte 2: Identificación del mensaje. 1
- Byte 3: 0 si el mensaje del maestro se recibió bien, o el checksum del mensaje recibido en el esclavo, si se ha detectado un error mediante este procedimiento.
- Byte 4: Checksum, que es 0xFB si el mensaje se ha recibido bien en el esclavo.

Como podemos ver, los errores de comunicación entre maestro y esclavo se reflejan en el mensaje de aprobación, concretamente en el tercer byte. Esto da lugar a dos causas de retransmisión del mensaje en el maestro, una es por no recibir el mensaje de aprobación del esclavo en menos de 250 milisegundos, y la otra es por recibirlo, pero indicando un error, es decir, con el checksum del mensaje recibido en el esclavo en el tercer byte, en vez del cero en ese octeto que se ha de recibir en caso de mensaje correcto.

El esclavo no mandará un mensaje de aprobación si el mensaje recibido es incompleto. Esto lo conoce cuando el valor del octeto número uno es mayor que el número de bytes recibidos. Si ocurre lo contrario, es decir, se reciben más octetos de los que indica el primer byte, los octetos de más son ignorados. Lo más probable en este caso es que se detecte un error de checksum, y así se indicará al maestro en el mensaje de aprobación de la manera que ya hemos explicado.

5.3.6.- Mensajes del maestro al esclavo.

Estos mensajes son respondidos por el esclavo con el mensaje de aprobación, además de por la realización de la acción que el mensaje le manda. En el maestro han de ser formados estos mensajes, para ser enviados al esclavo, siempre con las condiciones que ya se han explicado respecto a los caracteres especiales y de la cabecera.

En negrita se indican el nombre y el código del mensaje, tras ellos una explicación de lo que hacen, y luego la forma del mensaje. El código del mensaje coincide con el número que se ha de escribir en el programa robodemo.exe, que proporciona el fabricante, para que se realice la función que el código implica. En el programa bajo Linux que se ha escrito imitando al robodemo.exe de DOS también se utilizan los mismos códigos, por lo que funciona de manera muy parecida al programa

original, aunque en este programa faltan algunos de los comandos que se pueden utilizar en el proporcionado por el fabricante. En el programa de control de la carretilla se usan muy pocos comandos de todos los que dispone el Robosense, a lo sumo tres o cuatro, y alguno más que se utiliza antes de ejecutar el programa de control para preparar el sensor para la navegación, esto se realiza con el programa que imita en Linux al robodemo.exe de MS – DOS.

Start Mapping (11): Con este mensaje el espejo de escaneo comienza a rotar.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x0B Código de operación de Start Mapping.
- Byte 3: 0xF2 Checksum.

Stop Mapping (12): El espejo deja de rotar, si hay un mapa en la memoria del Robosense, éste se almacena en un fichero llamado mapstore.dat.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x0C Código de operación de Stop Mapping.
- Byte 3: 0xF1 Checksum.

Learn Map (14): El escáner comienza a rotar y el Robosense ejecuta el procedimiento de creación del mapa. Cuando el mapa está completo, envía al maestro la posición del vehículo en el nuevo mapa creado.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x0E Código de operación de Learn Map.
- Byte 3: 0xEF Checksum.

Report Map (17): Se almacena el mapa actual en mapstore.dat y se envían las coordenadas de los blancos del mapa actual a una tasa de 30 blancos por segundo.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x11 Código de operación de Report Map.
- Byte 3: 0xEC Checksum.

Radar (18): El escáner comienza a rotar y se envían al maestro las coordenadas medidas de cada blanco de una forma polar, es decir, distancia y ángulo como si el origen de coordenadas polares fuera el Robosense.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x12 Código de operación de Radar.
- Byte 3: 0xEC Checksum.

Send Target (19): El esclavo envía las coordenadas del blanco que se solicite dentro del mapa ya creado, si no está en el mapa, envía las coordenadas del origen (0,0). Un blanco solicitado no está en el mapa cuando se pide un número mayor que el número de blancos. Si en el parámetro Target_id se pone un número mayor que el número de blancos, ocurre esto. Los blancos se numeran según son encontrados en el mapeo, que ya se explicó.

- Byte 1: 0x04 Cuatro bytes en el mensaje.
- Byte 2: 0x13 Código de operación de Send Target.
- Byte 3: 0x** Target_id, número que identifica al blanco en cuestión.
- Byte 4: 0x** Checksum, depende del tercer byte.

Start Position (21): El espejo comienza a rotar y se hacen los preparativos necesarios para el posicionamiento.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x15 Código de operación de Start Position.
- Byte 3: 0xE8 Checksum.

Stop Position (22): El escáner deja de rotar y la última posición se guarda en el fichero postore.dat.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x16 Código de operación de Stop Position.
- Byte 3: 0xE7 Checksum.

Blind Relocation by Frame (211): Los datos que se transfieren en esta instrucción dependen de si Robosense opera en modo Single Frame o en modo RollFrame. Si el modo es desconocido para el maestro, puede conocerlo enviando la

instrucción RollFrame Mode (45) antes de intentar la recolocación de la posición. Robosense reaccionará enviando su modo, RollFrame mode = 0 indica Single Frame y RollFrame mode = 1 significa RollFrame. Estos modos tienen que ver con la aportación del programa Sitemap al Robosense, en principio, como apenas se ha utilizado este programa, no se ha usado este modo RollFrame, por eso no se hablará mucho más de él y no se ha programado la orden para pedir el modo en el que se encuentra el esclavo. Debemos decir que el modo normal es el Single Frame, con un solo frame definido, que es en el que está el mapa, y el RollFrame da lugar a la posibilidad de moverse entre distintos frames con distintos mapas por la aportación del programa Sitemap que da la opción de definirlos y de cambiar en la navegación de frame. También existen órdenes para cambiar de modo. El Robosense, ante esta orden, envía la posición recolocada.

- Byte 1: 0x04 Cuatro bytes en el mensaje.
- Byte 2: 0xD3 Código de operación de Relocate Position.
- Byte 3: 0x** Identificación de frame, si se está en RollFrame, si está en Single Frame, no importa.
- Byte 4: 0x** Checksum, depende de la identificación de frame.

Blind Relocation by Position (212): En este caso, la recolocación de la posición se realiza teniendo en cuenta la última posición de la que se tiene constancia en el maestro. Se puede dar el hecho de que el vehículo se mueva sin estarse realizando la actualización de la posición, con lo que se pierde la posición y no puede ser actualizada, se puede, con esta orden, conocer de nuevo la posición dándole la última que se conocía en el maestro. Si se da una posición próxima, tardará poco en recolocarse en el mapa, pero si no está tan próxima, puede tardar lo mismo o más que en hacer de nuevo el mapa, lo cual puede se conveniente en ese caso, aunque conlleva la pérdida del mapa anterior, ya que, el hacer un nuevo mapa puede dar lugar a que no sea el mismo que antes se tenía, no sólo por cuestiones de precisión, sino también por efecto de la distinta orientación del vehículo, que puede dar lugar a que en el mapeo se elija un origen distinto al que tenía el mapa primitivo. Esto puede originar que un camino trazado en el anterior mapa no sea válido, por ejemplo. De todas formas, esta forma de actuación se ha supuesto, porque no viene en el manual mucha información sobre ella. El esclavo manda la posición recolocada.

- Byte 1: 0x09 Nueve bytes en el mensaje.

- Byte 2: 0xD4 Código de operación de Blind Relocation by Position.
- Byte 3: 0x** Byte más significativo de la X de la posición anterior conocida en centímetros.
- Byte 4: 0x** Byte menos significativo de la X de la posición anterior conocida en centímetros.
- Byte 5: 0x** Byte más significativo de la Y de la posición anterior conocida en centímetros.
- Byte 6: 0x** Byte menos significativo de la Y de la posición anterior conocida en centímetros.
- Byte 7: 0x00 Reservado.
- Byte 8: 0x00 Reservado.
- Byte 9: 0x** Checksum. Depende de los octetos anteriores.

Update Position (23): La instrucción Start Positioning debe ser enviada al Robosense antes de enviar esta instrucción. El esclavo manda las coordenadas y la orientación de la posición de manera repetida. En el manual no figura la frecuencia de envío, pero se han medido frecuencias de unas cuatro a cinco veces por segundo.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x17 Código de operación de Update Position.
- Byte 3: 0xE6 Checksum.

Change Frequency (24): Como en el caso anterior, esta orden debe ser enviada al Robosense después de haber enviado la instrucción Start Positioning. La tasa de actualizaciones de la posición se cambia a la nueva frecuencia, y las coordenadas de la posición y de la orientación son enviadas a esta tasa al maestro.

- Byte 1: 0x04 Cuatro bytes en el mensaje.
- Byte 2: 0x18 Código de operación de Change Frequency.
- Byte 3: 0x** Nueva frecuencia.
- Byte 4: 0x** Checksum. Depende de los octetos anteriores.

Get Confidence Level (25): Envía el nivel de confianza, que, recordemos, es un valor de cero a cien que mide lo buena que es una posición del vehículo. Puede menguar por la pérdida de un blanco, o por excesiva proximidad a él.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x19 Código de operación de Get Confidence Level.
- Byte 3: 0xE4 Checksum.

Report Last Position (27): El esclavo envía su última posición conocida. Puede usarse para la actualización ciega de la posición a partir de esta última posición conocida por el esclavo.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x1B Código de operación de Report Last Position.
- Byte 3: 0xE2 Checksum.

Store Last Position (28): El esclavo almacena su última posición conocida en el fichero postore.dat.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x1C Código de operación de Store Last Position.
- Byte 3: 0xE1 Checksum.

Recover Position (29): El Robosense recupera la posición almacenada en postore.dat y la envía al maestro.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x1D Código de operación de Recover Position.
- Byte 3: 0xE0 Checksum.

Report Rolling Frames Mode (45): El esclavo informa al vehículo sobre su modo de funcionamiento, si es Single Frame oRollFrame. En el primer caso, manda un cero, y en el segundo un uno.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x2D Código de operación de Report Rolling Frames Mode.
- Byte 3: 0xD0 Checksum.

Download Location (113): Se descarga una posición y se almacena en el esclavo en postore.dat. También se supone que hace esto, pues no se tiene mucha información en el manual. De todas formas, no se ha usado mucho. La orientación se

mide determinando el ángulo del cero de ángulos del Robosense respecto al eje X del mapa creado.

- Byte 1: 0x09 Nueve bytes en el mensaje.
- Byte 2: 0x71 Código de operación de Download Location.
- Byte 3: 0x** Byte más significativo de la coordenada X de la posición en centímetros (se supone también que es en centímetros, como en el caso anterior).
- Byte 4: 0x** Byte menos significativo de la coordenada X de la posición en centímetros.
- Byte 5: 0x** Byte más significativo de la coordenada Y de la posición en centímetros.
- Byte 6: 0x** Byte menos significativo de la coordenada Y de la posición en centímetros.
- Byte 7: 0x** Byte más significativo de la orientación en milirradianes.
- Byte 8: 0x** Byte menos significativo de la orientación en milirradianes.
- Byte 9: 0x** Checksum. Depende de los octetos anteriores.

Additional Map (114): Arranca un procedimiento de mapeo y añade los nuevos blancos mapeados al fichero targets.dat. También envía las coordenadas de los blancos del mapa al maestro, utilizando el mensaje Send Map, que después veremos, y lo envía tantas veces como blancos hay en el mapa. Sirve para el modo de operación RollFrame.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x72 Código de operación de Additional Map.
- Byte 3: 0x8B Checksum.

Position From Target (210): El esclavo envía las coordenadas polares del blanco que se requiere en el sistema de coordenadas del Robosense.

- Byte 1: 0x04 Cuatro bytes en el mensaje.
- Byte 2: 0xD2 Código de operación de Position From Target.
- Byte 3: 0x** Identificación del blanco, Target_id.
- Byte 4: 0x** Checksum. Depende del octeto anterior.

Reset Sensor (54): Se borran todos los ficheros de datos en el Robosense, que arranca de nuevo en modo Single Frame.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x36 Código de operación de Reset Sensor.
- Byte 3: 0xC7 Checksum.

Select Confidence Report Mode (20): Si el modo seleccionado en el tercer byte del mensaje es el 0x00, el esclavo manda la confianza en el número del frame y el número de blancos que ve, si no, la confianza se manda como hasta ahora se ha hecho, en porcentajes. Este segundo supuesto es el supuesto por defecto.

- Byte 1: 0x04 Cuatro bytes en el mensaje.
- Byte 2: 0x14 Código de operación de Select Confidence Report Mode.
- Byte 3: 0x** Modo de devolución de la confianza.
- Byte 4: 0x** Checksum. Depende del byte anterior.

5.3.7.- Mensajes del esclavo al maestro.

En este caso, los mensajes los forma el Robosense, y los manda para que el maestro los interprete. Son respuestas a peticiones por parte del maestro, y corresponde al programa de control del vehículo en el ordenador de control del mismo, que no es otro que el host, que ahora llamamos maestro, el interpretarlos, esperando la cabecera, rehaciendo el mensaje si tiene caracteres especiales y sacando los datos que traen los mensajes para utilizarlos, ya que, en algunos casos, se necesita rehacer la información recibida, colocando el byte más significativo y el menos significativo de cierto valor enviado en su lugar para que puedan formar entre los dos bytes una magnitud coherente y aprovechable por la máquina host de control del vehículo. En este caso no se incluye el código del mensaje, puesto que no se puede utilizar en el programa robodemo.exe ni en su versión para Linux.

Pero ya hemos visto un mensaje del esclavo al maestro, no es otro que el asentimiento o, como lo hemos llamado anteriormente, mensaje de aprobación. El esclavo manda este mensaje si ha recibido un mensaje completo del maestro, y si el cheksum del mensaje es correcto, después del asentimiento, mandará los mensajes correspondientes al comando que el maestro le haya pedido que realice.

Position While Mapping: Se envía al maestro tras un mapeo con éxito. Para que la información sea coherente, se han de montar los bytes de la posición y la orientación para formar tres palabras de 16 bits en las que están la posición y la orientación del vehículo.

- Byte 1: 0x09 Nueve bytes en el mensaje.
- Byte 2: 0x71 Código de operación de Position While Mapping.
- Byte 3: 0x** Byte más significativo de la coordenada X de la posición.
- Byte 4: 0x** Byte menos significativo de la coordenada X de la posición.
- Byte 5: 0x** Byte más significativo de la coordenada Y de la posición.
- Byte 6: 0x** Byte menos significativo de la coordenada Y de la posición.
- Byte 7: 0x** Byte más significativo de la orientación.
- Byte 8: 0x** Byte menos significativo de la orientación.
- Byte 9: 0x** Checksum. Depende de los octetos anteriores.

Send Target: Se envía al maestro en respuesta al mensaje que éste manda denominado Send Target.

- Byte 1: 0x09 Nueve bytes en el mensaje.
- Byte 2: 0x6F Código de operación de Send Target.
- Byte 3: 0x** Número de blancos en el mapa.
- Byte 4: 0x** Identificación del blanco (Target_id).
- Byte 5: 0x** Byte más significativo de la coordenada X del blanco.
- Byte 6: 0x** Byte menos significativo de la coordenada X del blanco.
- Byte 7: 0x** Byte más significativo de la coordenada Y del blanco.
- Byte 8: 0x** Byte menos significativo de la coordenada Y del blanco.
- Byte 9: 0x** Checksum. Depende de los octetos anteriores.

Send Map: Se envía al maestro tantas veces como blancos en el frame actual, esto se indica en el cuarto byte. La tasa de transmisión es de unas 30 veces por segundo.

- Byte 1: 0x09 Nueve bytes en el mensaje.
- Byte 2: 0x0F Código de operación de Send Map.
- Byte 3: 0x00 Reservado.
- Byte 4: 0x** Número de blancos en el mapa.

- Byte 5: 0x** Identificación del blanco (Target_id).
- Byte 6: 0x** Byte más significativo de la coordenada X de la posición.
- Byte 7: 0x** Byte menos significativo de la coordenada X de la posición.
- Byte 8: 0x** Byte más significativo de la coordenada Y de la posición.
- Byte 9: 0x** Byte menos significativo de la coordenada Y de la posición.
- Byte 10: 0x00 Reservado.
- Byte 11: 0x** Checksum. Depende de los octetos anteriores.

Update Position: Se envía al maestro de manera continua durante el posicionamiento, o actualización de la posición en el mapa. La tasa de envío que se ha medido está entre unas cuatro o cinco veces por segundo.

- Byte 1: 0x09 Nueve bytes en el mensaje.
- Byte 2: 0x17 Código de operación de Update Position.
- Byte 3: 0x** Byte más significativo de la coordenada X de la posición.
- Byte 4: 0x** Byte menos significativo de la coordenada X de la posición.
- Byte 5: 0x** Byte más significativo de la coordenada Y de la posición.
- Byte 6: 0x** Byte menos significativo de la coordenada Y de la posición.
- Byte 7: 0x** Byte más significativo de la orientación.
- Byte 8: 0x** Byte menos significativo de la orientación.
- Byte 9: 0x** Nivel de confianza.
- Byte 10: 0x** Checksum. Depende de los octetos anteriores.

Send Confidence Level: Se envía al maestro para responder al mensaje Get Confidence Level, con el que se pide al esclavo el nivel de confianza.

- Byte 1: 0x04 Cuatro bytes en el mensaje.
- Byte 2: 0x19 Código de operación de Send Confidence Level.
- Byte 3: 0x** Nivel de confianza.
- Byte 4: 0x** Checksum. Depende de los octetos anteriores.

Recover Position: Se envía al maestro en respuesta al mensaje del vehículo Recover Position. Es casi igual a otros mensajes mandados por el esclavo en los que se da una posición, como el Update Position, del que se diferencia sólo en el código de operación.

- Byte 1: 0x09 Nueve bytes en el mensaje.
- Byte 2: 0x1D Código de operación de Recover Position.
- Byte 3: 0x** Byte más significativo de la coordenada X de la posición.
- Byte 4: 0x** Byte menos significativo de la coordenada X de la posición.
- Byte 5: 0x** Byte más significativo de la coordenada Y de la posición.
- Byte 6: 0x** Byte menos significativo de la coordenada Y de la posición.
- Byte 7: 0x** Byte más significativo de la orientación.
- Byte 8: 0x** Byte menos significativo de la orientación.
- Byte 9: 0x** Checksum. Depende de los octetos anteriores.

Position From Target: Se envía al maestro de manera continua en modo radar, como respuesta al mensaje Radar. La tasa de transmisión depende de los blancos que ve el Robosense, y de su separación angular. Como se ha comprobado, si están demasiado juntos, puede que no le dé tiempo a transmitirlos todos hacia el maestro. Recordemos que, en modo radar, la posición de los blancos se da en coordenadas polares (distancia, ángulo) respecto del Robosense.

- Byte 1: 0x08 Ocho bytes en el mensaje.
- Byte 2: 0xD2 Código de operación de Position From Target.
- Byte 3: 0x** Identificación del blanco (Target_id).
- Byte 4: 0x** Byte más significativo de la distancia al blanco.
- Byte 5: 0x** Byte menos significativo de la distancia al blanco.
- Byte 6: 0x** Byte más significativo del ángulo del blanco.
- Byte 7: 0x** Byte menos significativo del ángulo del blanco.
- Byte 8: 0x** Checksum. Depende de los octetos anteriores.

RollFrame Mode: Mensaje que el esclavo envía como respuesta a Report Rolling Frames Mode. En su byte tercero manda un cero si el sensor está en modo Single Frame, y un uno si está en modo RollFrame.

- Byte 1: 0x04 Cuatro bytes en el mensaje.
- Byte 2: 0x2D Código de operación de RollFrame Mode.
- Byte 3: 0x** RollFrame Mode.
- Byte 4: 0x** Checksum. Depende de los octetos anteriores.

5.3.8.- Transferencia de ficheros.

Para realizar la transferencia de ficheros se tiene un programa bajo MS – DOS que es el ya mencionado transfer.exe. También se pueden transferir ficheros mediante comandos del robodemo.exe. Estos comandos no han sido implementados en el programa bajo Linux, pero se mencionarán en previsión de una posible ampliación del programa.

El maestro no debería lanzar el programa transfer.exe antes de que se reciba el mensaje de aprobación del esclavo, convenientemente revisado y con el checksum correcto.

El proceso de transferencia de ficheros comienza con una orden del maestro enviada al Robosense. Éste responde con el habitual mensaje de aprobación y lanza el programa transfer.exe que tiene en la placa de control 5066, esperando a que el host lance su programa transfer.exe. Cuando se ha completado la transferencia, el esclavo se resetea.

El programa transfer.exe ha de correr en los dos ordenadores, maestro y esclavo. Una forma de que lo haga es mediante los comandos del robodemo.exe, que el 5066 interpreta mediante su programa robosense.exe, para luego lanzar el transfer.exe y que se realice la transferencia. Otra forma es lanzar el transfer.exe en los dos ordenadores, cosa que se puede hacer conectando la 5066 por el puerto COM2 al host y utilizando el puerto COM1 de esta placa como terminal serie, así, podemos lanzar los dos transfer.exe, uno en cada ordenador y realizar la transferencia.

El programa transfer.exe utiliza el protocolo XMODEM para la transferencia de ficheros. Este programa funciona de la siguiente manera para transmitir el fichero nombre_de_fichero:

TRANSFER {opciones} nombre_de_fichero

Las opciones son:

/S	Enviar un fichero.
/R	Recibir un fichero.
/B#	Establecer la tasa binaria de transmisión, por defecto 9600 en los puertos serie. El carácter ‘#’ indica el número de la tasa de transmisión.

/COM# Puerto serie para la transferencia (COM1, por defecto, ó COM2). Se deberá poner el número en vez del carácter '#'.

/V Uso con tarjeta de video, representa Rs y Ts para cada bloque. Podemos ver así la evolución de la transferencia.

Un ejemplo de utilización del programa:

C:>TRANSFER /r /b19200 /com1 B:\TEST.EXE

Se recibe el programa TEST.EXE que está en la unidad B a 19200 bits por segundo por el puerto COM1.

Así, se utilizará el comando correspondiente en robodemo.exe y luego se ejecutará transfer.exe para la realización de la transferencia con el Robosense.

Los comandos son los siguientes:

Receive MAPSTORE.DAT & TARGETS.DAT (60): El Robosense envía al host el fichero MAPSTORE.DAT y el TARGETS.DAT.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x3C Código de operación de Receive Mapstore &Targets.
- Byte 3: 0xC1 Checksum.

Y las líneas de comando de MS – DOS son:

TRANSFER /R /B19200 /COMn MAPSTORE.DAT

TRANSFER /R /B19200 /COMn TARGETS.DAT

En COMn se ha de poner COM1 ó COM2, según corresponda al host, es decir, según esté conectado al esclavo por un puerto u otro, se seguirá usando esta nomenclatura para los siguientes comandos.

Send MAPSTORE.DAT & TARGETS.DAT (61): El Robosense recibe el fichero MAPSTORE.DAT y el TARGETS.DAT.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x3D Código de operación de Send Mapstore y Targets.
- Byte 3: 0xC0 Checksum.

Las líneas de comando en DOS:

TRANSFER /S /B19200 /COMn MAPSTORE.DAT

TRANSFER /S /B19200 /COMn TARGETS.DAT

Receive POSTORE.DAT (62): El Robosense envía el fichero POSTORE.DAT.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x3E Código de operación de Receive Postore.
- Byte 3: 0xBF Checksum.

La línea de comando:

TRANSFER /R /B19200 /COMn POSTORE.DAT

Send POSTORE.DAT (63): El Robosense recibe el fichero POSTORE.DAT.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x3F Código de operación de Send Postore.
- Byte 3: 0xBE Checksum.

La línea de comando del DOS:

TRANSFER /S /B19200 /COMn POSTORE.DAT

Receive FRAMES.DAT & ZONES.DAT (64): El Robosense envía los ficheros FRAMES.DAT y ZONES.DAT.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x40 Código de operación de Receive Frames y Zones.
- Byte 3: 0xC7 Checksum.

Con las líneas de comando:

TRANSFER /R /B19200 /COMn FRAMES.DAT

TRANSFER /R /B19200 /COMn ZONES.DAT

Send FRAMES.DAT & ZONES.DAT (65): El Robosense recibe los ficheros FRAMES.DAT y ZONES.DAT.

- Byte 1: 0x03 Tres bytes en el mensaje.
- Byte 2: 0x41 Código de operación de Send Frames y Zones.
- Byte 3: 0xC7 Checksum.

Las líneas de comando del DOS son:

TRANSFER /S /B19200 /COMn FRAMES.DAT

TRANSFER /S /B19200 /COMn ZONES.DAT

5.4.- Pruebas realizadas.

5.4.1.- Motivación de las pruebas.

Las pruebas que se realizaron con el Robosense tienen un motivo claro, y es la observación de la variación de la precisión de las medidas conforme el dispositivo está conectado. Se puede apreciar que la precisión aumenta con el tiempo que el láser está conectado. Esto se puede achacar al calentamiento del dispositivo tras un tiempo de operación. Para evitar este problema, el fabricante dispone de un fichero de calibración cuyo objetivo es minimizar el problema.

Con el Robosense se tiene un fichero de este tipo, denominado KIUL.DAT para realizar correcciones a la medida por esta contingencia. Pero el fichero del que se disponía en un principio no solucionaba este problema. Esto obligó a ponerse en contacto con los fabricantes del láser, SIMAN, para que dieran solución al problema. Tras pedir una serie de ficheros internos al dispositivo, que les fueron enviados, fabricaron y mandaron, tras mucho insistir por nuestra parte, un nuevo fichero de calibración que solucionó el problema de forma parcial, ya que el hecho de mejorar la precisión en los primeros instantes de funcionamiento la empeoró levemente cuando el calentamiento ya se había producido. En el instante en que el dispositivo estaba caliente, los resultados eran excelentes con el primer fichero de calibración de que se disponía.

La falta de precisión era sólo aplicable a la posición, es decir, a los valores de X e Y dentro de un mapa, puesto que la orientación era mucho más precisa y constante en su error durante todo el periodo de funcionamiento del dispositivo.

De todas maneras, por la construcción del aparato, era posible intuir esa variación de funcionamiento, pensando, como ya se ha dicho, que se debía al calentamiento, y conociendo la fuerte dependencia en su comportamiento que tienen los dispositivos láser con la temperatura. Aun así, la existencia de una unidad de auto – calibración en el sensor, hacía pensar que este problema habría sido resuelto, explicando así que el láser poseyera esta unidad.

5.4.2.- Descripción de las pruebas.

Para la prueba del dispositivo, el fabricante marca unas pautas a seguir. Se coloca el dispositivo en una superficie plana, en nuestro caso, el suelo. El Robosense ha de estar convenientemente alimentado y conectado al ordenador que actuará de host. Los blancos se colocan sobre el suelo alrededor del láser y totalmente visibles por él.

Posteriormente, utilizando el programa robodemo.exe, del que se debe disponer en el host, se realiza un mapeo con la orden 14, Learn Map. Esto debe tardar un minuto o minuto y medio. Tras esto, debe dar la posición con una precisión de ± 8 centímetros en cuanto a la X y a la Y, para cada blanco.

Tras el mapeo, el esclavo nos da la posición en el nuevo mapa, la precisión ha de ser de ± 2.5 centímetros en la X y en la Y y de ± 3 milirradianes en la orientación, con un nivel de confianza del 100 % si cuatro o más blancos son encontrados.

El reposicionamiento ciego (Blind Relocation) ha de tardar menos de un minuto, y tener una precisión mejor de ± 10 centímetros y ± 5 milirradianes.

Si se realiza una actualización de la posición, con las órdenes 21, Start Position y 23, Update Position, podemos mover y rotar el dispositivo por el mapa y debemos obtener precisiones de ± 2 centímetros y de ± 2 milirradianes.

Para que el sensor pueda realizar la actualización, no debe haber entre cada medida distancias de más de 80 centímetros o de 80 milirradianes en la orientación. Si las hay, el Robosense se perderá y mandará un mensaje de error.

Si no se obtienen esos parámetros, no tenemos el láser en buenas condiciones de funcionamiento.

5.4.3.- Resultados de las pruebas.

Se realizaron dos tipos de pruebas con el fichero de calibración primitivo. En la primera, en modo radar, se realizó una medida de la distancia a un blanco, alineado en la coordenada X, para observar la evolución del error en la distancia conforme avanzaba el tiempo de funcionamiento del dispositivo. El Robosense se montó en el suelo sobre dos cartones que permitían alinear su cero de ángulos con la junta de las baldosas del suelo del laboratorio, sobre la que se colocó un blanco. Los cartones permitían alinear el sensor con las juntas, así como medir ángulos, pues uno de ellos era un semicírculo graduado en ángulos, con escalas en grados y radianes, que se fabricó para este cometido.

Colocado de esta forma, el ángulo debía ser casi nulo, como así era, y el rango devuelto por el Robosense determinaba la distancia al blanco.

La prueba se realizó durante dos horas, con medidas cada cinco minutos, manteniendo siempre alimentado el sensor, sin cortar su alimentación. Se medía, se paraban cinco minutos y se volvía a medir. Como es en modo radar, la medida es continua, se mandaba medir, luego se paraba tras unos instantes y se cogía como medida la media de los últimos cinco resultados obtenidos en la pantalla del host. Con esto puede que se filtrara algo el error, pero dio conclusiones interesantes. El blanco se encontraba a una distancia de 10636 milímetros según medía el metro, que equivale a 26 baldosas del laboratorio. Se obtuvieron los siguientes resultados (tabla 5.1):

Tiempo transcurrido en minutos.	Distancia medida en milímetros.	Error en valor absoluto en milímetros.	Error relativo en porcentaje.
0	10156	530	4.96
5	10285	401	3.75
10	10254	432	4.04
15	10335	351	3.28
20	10399	287	2.69
25	10468	218	2.04
30	10518	168	1.57
35	10480	206	1.93
40	10608	78	0.73
45	10543	143	1.34
50	10589	97	0.91
55	10588	98	0.92
60	10575	111	1.04
65	10601	85	0.80
70	10538	148	1.38
75	10635	51	0.48
80	10590	96	0.90
85	10687	1	0.01
90	10580	106	0.99
95	10616	70	0.66

Tiempo transcurrido en minutos.	Distancia medida en milímetros.	Error en valor absoluto en milímetros.	Error relativo en porcentaje.
100	10602	84	0.79
105	10576	110	1.03
110	10638	48	0.45
115	10637	49	0.46
120	10608	78	0.73

Tabla 5.1: Experimento de medida de distancia a un blanco con el Robosense.

Se puede observar que el error tiene una tendencia negativa con el tiempo, pero que el hecho de ser una sola medida sin procesamiento alguno ni técnicas de reducción de errores, hace que no decrezca de manera monótona. En el siguiente ejemplo veremos como esa tendencia tiene un mayor componente de monotonía. Veamos en la figura 5.5 la evolución del error:

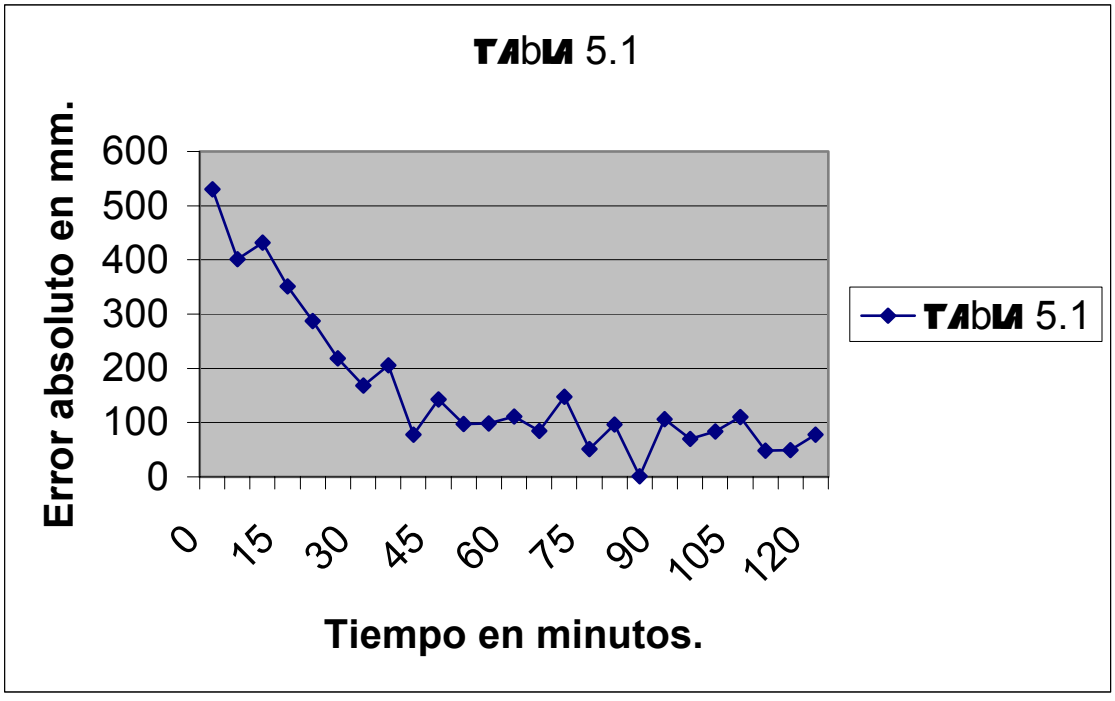


Figura 5.5: Evolución del error de la tabla 5.1.

Se puede observar la tendencia decreciente del error, pero, cuando parece estabilizarse, más o menos tras pasar una hora encendido, se tiene mucha variación.

La segunda prueba que vamos a describir consistió en observar la evolución del error con el sensor en un mapa. Veremos el error de la X y de la Y. En principio el experimento debía durar dos horas como el anterior, pero veremos que a la hora y cuarto

el error decreció a cero y sin cambios, por lo que no se siguió. En el mapa que se creó, el Robosense debía estar situado en el punto X = 288 centímetros, Y = 206 centímetros, ángulo = π. Las medidas se hicieron con el comando Update Position, cogiendo la media de los cinco últimos valores en la pantalla del host al parar la actualización de posición. Se observó muy poco error en el ángulo desde el principio, así que no se consideró en el estudio. De esta manera, se encendió el aparato y se fueron realizando pruebas con los siguientes resultados:

Tiempo (min.)	X (cm)	Y(cm)	Ángulo (mRad)	Error X (cm) en valor absoluto	Error Y (cm) en valor absoluto	Error relativo X (%)	Error relativo Y (%)
0	253	180	3129	35	26	12.15	12.62
5	265	189	3130	23	17	7.99	8.25
10	271	193	3130	17	13	5.90	6.31
15	275	196	3130	13	10	4.51	4.85
20	278	199	3130	10	7	3.47	3.40
25	280	201	3131	8	5	2.78	2.43
30	283	202	3131	5	4	1.74	1.94
35	285	204	3131	3	2	1.04	0.97
40	286	205	3131	2	1	0.70	0.49
45	287	205	3131	1	1	0.35	0.49
50	287	205	3133	1	1	0.35	0.49
55	287	205	3132	1	1	0.35	0.49
60	289	207	3131	1	1	0.35	0.49
65	288	206	3131	0	0	0	0
70	288	206	3132	0	0	0	0
75	288	206	3132	0	0	0	0

Tabla 5.2: Experimento de error de posición en un mapa con el Robosense.

Vemos que, tanto el error de la X como el de la Y decrecen de manera monótona, y que el error relativo es un poco mayor en la Y porque la posición real es un poco menor que la X. Aparte de un mejor comportamiento en la monotonía, vemos que también tiene mejor comportamiento en la precisión, con errores menores de un centímetro. Se puede observar también que la variación del ángulo es de apenas tres

milirradianes, lo cual justifica el no incluirlo en el estudio. Se supone que esa pequeña variación es debida a que no se usa demasiado la medida del láser para determinar el ángulo, por lo que no influye tanto la temperatura ni el paso del tiempo. Veámoslo en una gráfica (figura 5.6):

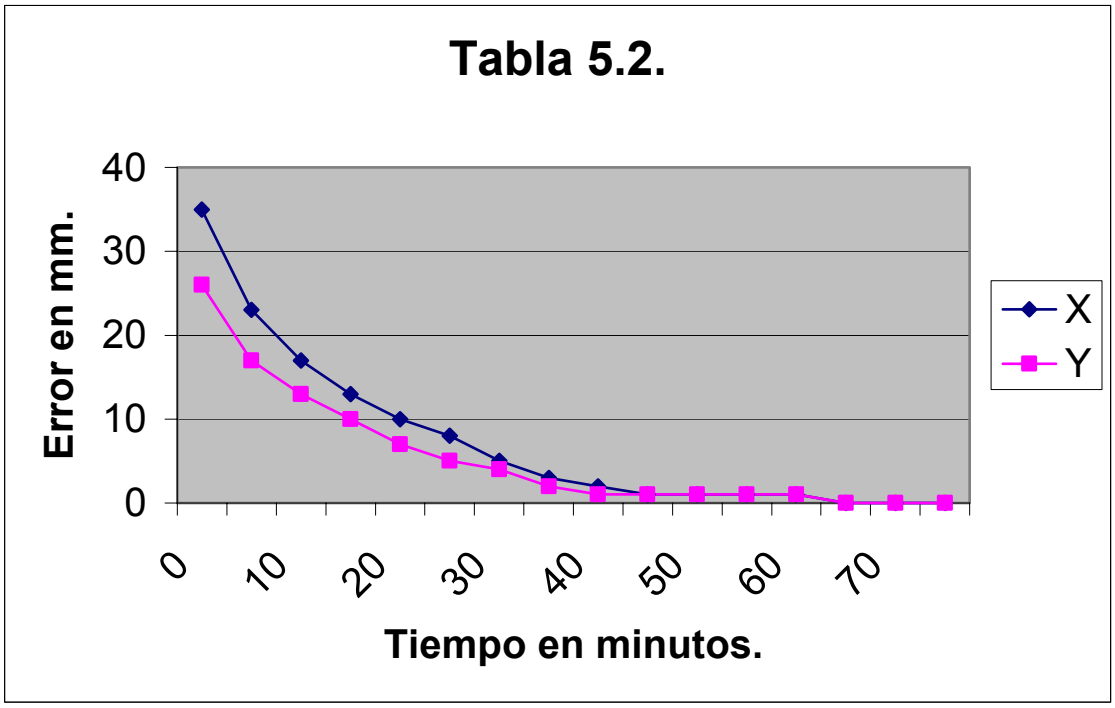


Figura 5.6: Evolución de los errores de la tabla 5.2.

Se puede observar el comportamiento monótono decreciente del error, como son errores absolutos, el error de la Y es algo menor, porque la coordenada Y es menor, pero si hubiéramos representado el error relativo, sale algo mayor, como se observa en la tabla 5.2.

Podemos identificar la curva que se obtiene con una exponencial decreciente, de hecho, se intentó hacer coincidir con una exponencial, para hallar el error y poder corregirlo, dependiendo del tiempo, pero se repitió el experimento, ahora con medidas cada diez minutos en vez de cinco, y se vio que no se correspondían con la curva identificada, obtenida mediante mínimos cuadrados. Esto puede explicarse porque el hecho de repetir el experimento cada diez minutos, en vez de cada cinco, supuso una pauta distinta de calentamiento, ya que el motor en su funcionamiento se calienta y calienta todo el dispositivo. También pudo influir la temperatura ambiente, pues se hizo en días distintos. Con esto se despreció la posibilidad de tener una tabla de corrección del error dependiente del tiempo, debiendo ser dependiente de la temperatura, pero

como no se tienen medidas de la temperatura interior a la carcasa del dispositivo, no se pudo intentar una corrección basada en ese parámetro.

Los que sí parece que pueden acceder a medidas de la temperatura interna a la carcasa son los fabricantes, por eso mandaron otro fichero de calibración que mejora los resultados.

Se pensaron otras soluciones para este problema, como calentar de alguna forma el dispositivo para que funcionara correctamente desde el principio, pero no se realizaron.

Como conclusión a todos estos experimentos, podemos decir que el láser se comporta mejor con balizas situadas de forma regular, como en un cuadrado, que si colocamos los blancos de forma irregular. También podemos ver que el error es relativamente constante con la distancia, así, al menos, el error relativo mejora con la distancia, pues a un error similar, como el valor real es mayor, el valor relativo es menor.

Capítulo 6: El ordenador de control (hardware).

6.1.- Descripción.

6.1.1.- Características.

El ordenador de control se presenta en formato PC 104. Es un Axiom SBC8251, con procesador Pentium a 200 MHz. Se le proporcionaron dos SIMMs de memoria RAM de 64 Mb de capacidad. En la placa base tiene integrada una tarjeta de vídeo de chip Chips & Tecnologies 65550 de 1 Mb de memoria. También tiene integrado en al placa un disco de estado sólido Disk on Chip de 2 Mb de capacidad, orientado a un funcionamiento de la placa sin un disco duro externo, sólo con este pequeño disco integrado en la placa, lo que reduce su tamaño.

Pero nosotros necesitamos mayor capacidad de almacenamiento, así que se dispone de un disco duro de 20 Gb. Además, se le acopló una unidad de disco flexible de tres pulgadas y media y una unidad de CD - ROM.

La placa tiene un bus IDE para discos duros, conexión para unidad de disco flexible, dos puertos serie con chip 16C550 y un puerto paralelo. Tiene salidas para el monitor y para el teclado.

Pero lo integrado en la placa no es todo, a través del bus PC 104, se le pueden conectar varias tarjetas de las que disponemos en el laboratorio, como son una tarjeta de red ethernet, tarjeta de dos puertos serie RS 422/485, tarjeta PCMCIA, tarjeta de entradas y salidas digitales, tarjeta de entrada y salida general (analógicas y digitales), tarjeta de contadores y timers, y dos tarjetas de control de motores.

Todo esto se ha tenido que montar y alimentar con una fuente. Inicialmente, se disponía de una fuente de alimentación de alterna, común a la de los ordenadores de sobremesa, aunque con una forma algo más alargada. Para la realización de pruebas, se sustituyó esta fuente por una de continua a 24 voltios, que se extraen directamente de la batería del vehículo, tras pasa por los mecanismos de encendido del ordenador, y

protegida con un fusible, pero sin estabilización de ningún tipo. Cuando no se están realizando pruebas, los 24 voltios que necesita esta fuente se sacan de una fuente de continua, que se alimenta con la red y saca 24 voltios de continua.

6.1.2.- Función.

En el ordenador de control se han de desarrollar y ejecutar los programas de control del vehículo, ha de realizar el control a más bajo nivel, es decir, la actuación sobre los ejes de movimiento del vehículo, así como otras funciones de más alto nivel, como la ejecución del algoritmo de control de la trayectoria del móvil, o funciones de comunicación para teleoperación, así como funciones de monitorización del comportamiento del robot, en forma de datos de su posición, curvatura, velocidad, etc.

Pero el ordenador también ha sido el ordenador de trabajo del proyectando, con lo que ha realizado funciones de ordenador de sobremesa, para búsqueda de información en Internet, para redacción con procesadores de texto, para recepción de correo electrónico, y alguna otra función.

6.1.3.- El formato PC 104.

Este formato se desarrolla por un consorcio internacional de distintas empresas que ofrecen soluciones de este tipo, con la esperanza de que se imponga en el mercado. Lo que ofrece es un tamaño reducido de los ordenadores, que puede ser muy apreciado en múltiples aplicaciones como la que nos ocupa.

El PC 104 es, en esencia, un PC AT con bus ISA. El PC 104 tiene dos tipos de buses, correspondientes a los buses PC y PC/AT, de ocho bits y de 16 bits, respectivamente. Esto corresponde a cada una de las dos filas de la conexión del bus PC 104, si se tiene sólo un bus de ocho bits, se tiene una de las dos filas de conexiones existentes, de 64 pines en dos hileras de 32, y si se tiene el bus PC/AT de 16 bits, se tiene la otra conexión adicional, de 40 pines en dos hileras de 20. Esto da lugar a los 104 pines del nombre del formato. Se puede encontrar bastante información sobre el consorcio PC 104 en su página web www.pc104.org, así como en las páginas web de los distintos fabricantes que proveen este tipo de productos para la industria. También se adjunta documentación de las especificaciones al final del proyecto.

La existencia de dos tipos de módulos PC 104, de ocho y 16 bits, da lugar a que no se puedan conectar de manera aleatoria en el bus, ya que la conexión de un módulo de ocho bits se ha de realizar sabiendo que no va a tener módulos de 16 bits sobre él porque si no se hace así, se elimina la posibilidad de que se conecten los cuarenta pines del bus PC/AT a todo el bus. Esto se produce porque la conexión de un módulo en la pila del bus ofrece conexión a los módulos que van por encima de él en la pila. Esto da lugar a la posibilidad de conectar más módulos o tarjetas que en un bus ISA normal, ya que el número de tarjetas ISA a conectar está restringido al número de slots libres en el bus, mientras que, en este caso, la restricción viene dada por condicionantes eléctricos, y se pueden conectar del orden de ocho o más tarjetas funcionando bien. Hay pocas placas base ISA con ocho slots para el bus. En la figura 6.1 vemos la forma de apilar las tarjetas o módulos.

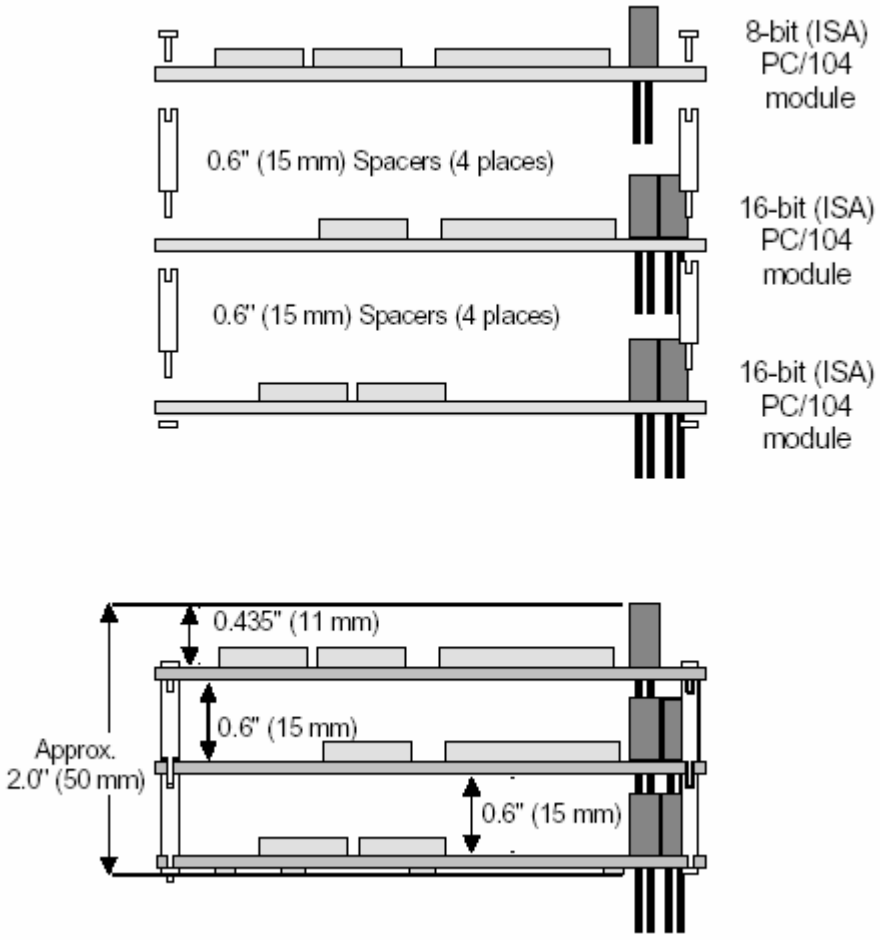


Figura 6.1: Forma de colocar los módulos PC 104 en la pila del bus.

Podemos ver que los módulos de ocho bits van encima de todos los de 16, y que se conectan mediante tornillos que permiten enanchar un módulo con el que está bajo él y, a su vez, conectar los módulos que van encima con este módulo, tras encajarse los

pines del bus. Podemos ver, también, que los pines salen hacia abajo en las tarjetas, y que los conectores hembra de las mismas están en la parte de arriba.

En la figura 6.2, se puede observar una vista desde arriba de los módulos, en la que vemos la distribución de los pines.

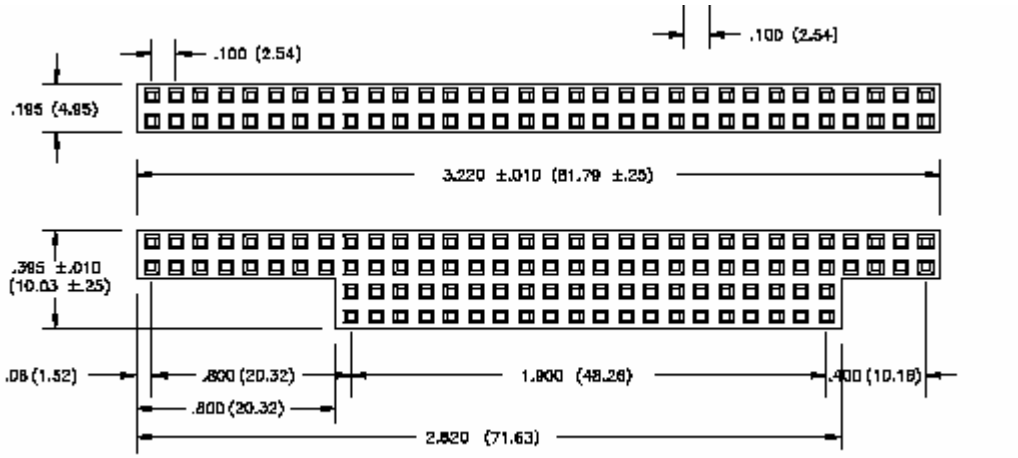


Figura 6.2: Distribución de los pines del conector PC 104 de ocho y 16 bits.

En cuanto a lo que cada uno de estos pines representa, podemos decir que se corresponden con las señales propias del bus ISA, y se pueden ver en las especificaciones anexadas al final del proyecto.

Existe un formato de bus denominado PC 104 plus, que incorpora a este tipo de tarjetas las funcionalidades del bus PCI, pero no se han utilizado en el proyecto tarjetas de este tipo.

El ordenador se montó en una caja y se conectó a la tarjeta de adaptación de señales, como ya se ha contado, se puede ver en la figura 6.3.

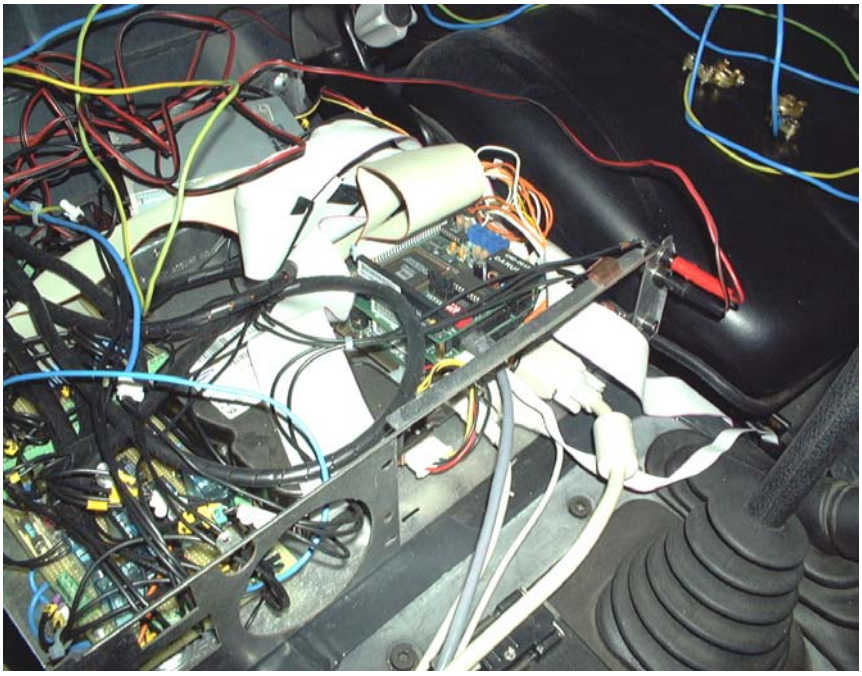


Figura 6.3: Detalle del ordenador PC 104 montado en la caja en la carretilla.

6.1.4.- Módulos disponibles.

La placa base es una SBC8251 de Axiom, como ya se ha dicho, pero se tienen varias tarjetas para la realización de distintas tareas dentro del ordenador de control, aunque en el proyecto sólo se ha programado la tarjeta AX1040, por lo que será explicada con más detenimiento. También se ha usado la tarjeta ethernet, pero ésta no se ha programado, sino que se ha utilizado con software existente para conexión a internet y para conexión de equipos usando tecnología IP.

Así, se tiene una tarjeta AX10410 de Axiom de adquisición de datos de alta velocidad y actuación, analógica y digital, que posteriormente veremos mejor.

Una tarjeta AX10425 de Axiom que posee funciones de contador – timer que no se ha utilizado, pero que puede servir para tareas de temporización y relojes para dispositivos externos, así como para cuenta de entradas, por ejemplo, de encoders.

Se dispone de una tarjeta de entradas y salidas digitales de 48 bits, la AX10420 de Axiom, tampoco se ha utilizado.

La tarjeta ethernet es la AX10465 de Axiom, esta tarjeta se encuentra conectada y funcionando, dando servicio de conexión a internet al ordenador.

Una tarjeta de dos puertos serie con estándar RS - 422/485 que incorpora funcionalidades de conexión en bus y transmisión balanceada al conocido estándar RS –

232, aunque se pueden configurar mediante jumpers para que funcionen como un RS – 232 común. Incorporan un chip 16C550 que provee esta funcionalidad, pero tampoco se utilizaron por ser suficientes, de momento, los dos puertos serie que posee la placa. No se logró que funcionara correctamente con los dos puertos serie ya existentes en la placa base, pero era reconocida por el ordenador.

También se pensó en la posibilidad de conectar una tarjeta PCMCIA, se dispone de la AX10440 de Axiom, que permite conectar dos módulos PCMCIA, para expansión de memoria, tarjetas de disco duro, utilización de cámaras que se conectan con este formato, y otras muchas utilidades, pero no se utiliza de momento.

Para el control de motores se dispone de dos tarjetas de control, que permitirían controlar dos ejes de la carretilla. Estas tarjetas, de ocho bits, incorporan funcionalidades de realimentación mediante encoders de la posición del eje, así como la posibilidad de programar los parámetros del controlador interno a la tarjeta, es decir, sintonizar el PID interno que poseen. Son dos tarjetas 5928 de Tech80. No se han utilizado porque no se disponía de drivers adecuados al sistema operativo para ellas, y se exigían resultados más rápidos que los que hacer esos drivers habría conllevado, pero estas tarjetas habrán de ser utilizadas con casi total seguridad en un futuro, aparte de porque están fabricadas para esto e integran de manera fácil las medidas de los encoders, porque descargan al ordenador del cálculo de la siguiente señal de control a aplicar, no necesitándose, como en el actual diseño, el cálculo dentro del programa de control. Además, este cálculo se puede realizar ahora, porque es para un solo eje, pero cuando se tengan más ejes y que leer medidas de más sensores, puede que no sea posible un tiempo de control satisfactorio con la máquina de que se dispone. La tarjeta posee salida analógica de doce bits, 18 entradas / salidas de propósito general, muy adecuadas para activar ciertos relés de los que ya se ha hablado, entrada de encoder de un MHz y entrada auxiliar para encoders, registros de posición, velocidad y aceleración de 32 bits y posibilidad de filtro PID de control ajustable. Se trataría, entonces, de ajustar el PID en la programación al eje a controlar, y que el programa de control, mediante un algoritmo concreto de control, diera la referencia a la tarjeta. En la figura 6.4 se puede observar.

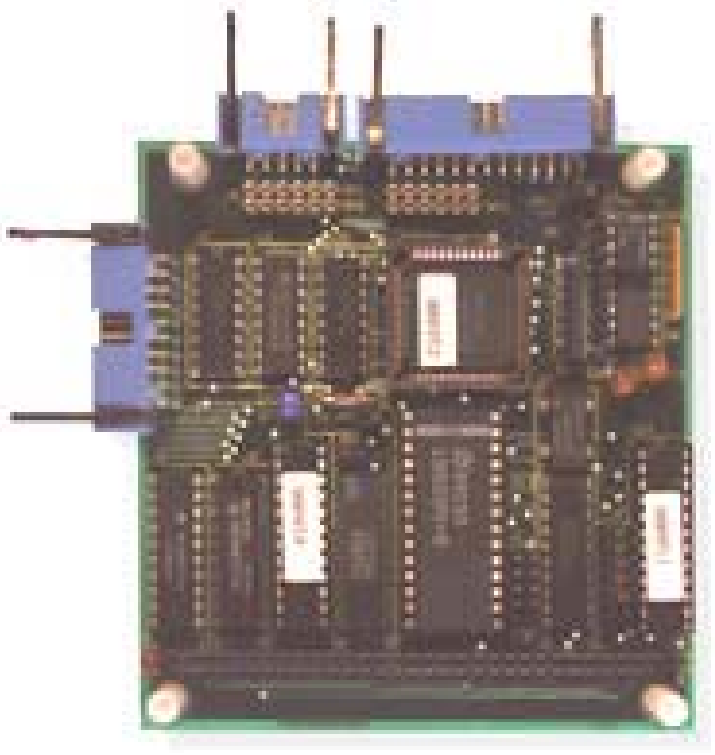


Figura 6.4: Tarjeta 5928 de Tech 80 de control de motores.

Todas las tarjetas disponen de manuales en el laboratorio, así como de software de control para MS – DOS. Se han probado todas, resultando que funcionan correctamente, pero se desea llamar la atención sobre los recursos de la placa, estos, son limitados en cuanto a los vectores de interrupción y las direcciones base de que se dispone, puede que, en un principio, viendo las direcciones base de los dispositivos instalados, se piense que funcionan, pero al probar, podemos darnos cuenta de que no es así, y no se pueden colocar todas las tarjetas, de hecho, sólo con la colocación de la AX10410, hay una interferencia que no se ha podido explicar con la tarjeta ethernet, y el hecho de utilizar el puerto serie que viene en la placa interfiere en el comportamiento de la AX10410, provocando un comportamiento catastrófico del vehículo, aunque esto se pudo solucionar.

Para la prueba, se hubo de cambiar los distintos valores de los vectores de interrupción, así como de las direcciones de entrada / salida de los distintos dispositivos para “encajarlos” en el mapa de entrada / salida del ordenador. Aunque los valores provenientes de fábrica de estos parámetros, que en otros casos, con tarjetas muy similares, habían funcionado, no lo hicieron en este, incluso con sólo esa tarjeta conectada, y sin ninguna injerencia aparente de ninguno de los distintos dispositivos de

la placa base que quedaban conectados. Se daban situaciones en las que el ordenador, con los parámetros de fábrica de una tarjeta, y sin ninguna otra conectada, ni siquiera arrancaba.

6.2.- La tarjeta AX10410 de Axiom.

6.2.1.- Introducción.

El módulo AX10410 es miembro de la familia de Axiom PC104, está preparado para instalarse directamente en una placa base PC104, convirtiéndose así en un sistema de adquisición de datos y de control de alta velocidad. Ofrece cinco de las más deseadas funciones en un PC104, que son entradas analógicas, salidas analógicas, entradas digitales, salidas digitales y contador / timer.

La tarjeta ofrece 8 entradas analógicas diferenciales o 16 sencillas (single ended), de doce bits de resolución, con una tasa máxima de muestreo de 100 KHz. Posee un amplificador programable, con valores de ganancia de uno, dos, cuatro y ocho. Los rangos de entrada de las señales son de 10, 5, 2.5 y 1.25 voltios para señales unipolares (cero y positivas), y de ± 10, ± 5, ± 2.5 y ± 1.25 voltios para señales bipolares (negativas y positivas). La transferencia de datos puede ser procesada a tres velocidades de conversión analógica / digital. La velocidad más baja está iniciada por un trigger software. La velocidad media está iniciada por interrupción, y es superior a los 3 KHz. La velocidad alta está lanzada por un timer interno o un reloj externo, utilizando DMA para la transferencia de datos. En el proyecto no se usa la conversión analógica digital, por lo que este tema no se ha tratado con profundidad.

El módulo ofrece dos canales de doce bits de salidas analógicas, el conversor digital / analógico puede operar con un voltaje de referencia de ± 5 voltios en bipolar, o de 10 voltios en unipolar, será este último modo de operación el que se usará, amplificado externamente al ordenador por dos, como ya se ha dicho.

Disponibles en la tarjeta están 16 bits para entradas / salidas digitales, concretamente, ocho bits son para entradas y otros ocho para salidas, accesibles desde el conector externo a la tarjeta de 50 pines, que es el que se conecta al circuito externo de

adaptación y activación de señales, que ya hemos visto, mediante cable plano de 50 hilos, por supuesto. Este cable lleva también las señales anteriores (analógicas). Los niveles lógicos de las entradas / salidas son compatibles TTL / DTL.

La tarjeta posee gran cantidad de módulos adicionales, de los que no se dispone, destinados a acondicionamiento de señal y ha monitorización de señales, están destinados a aplicaciones concretas, como termopares o medidas de bajo nivel, amplificación de bajo ruido, etcétera.

En el manual están las especificaciones más detalladas, aunque no entraremos en ello, puesto que se considera suficiente con lo que ya se ha narrado. De todas formas, el manual está disponible en el laboratorio para su consulta en caso de duda.

En la figura 6.5 se puede observar la tarjeta, en una versión de 16 bits, la que se posee en el laboratorio es muy similar, pero de ocho bits, con lo que no tiene la segunda fila de conexiones en el bus PC 104. Puede, sin embarque, que la tarjeta sea igual, y que las cuarenta conexiones adicionales están para poder conectar más módulos en la pila, sin influir en la tarjeta AX10410.

Figura 6.5: Tarjeta AX10410.

6.2.2.- Configuración del módulo.

Esta configuración es lo primero que se ha de hacer tras insertar la tarjeta en el ordenador, puesto que una mala configuración puede hacer que el módulo no sea reconocido por el ordenador, e, incluso, puede hacer que algún elemento ya instalado en el mismo no funcione bien, llegando hasta a no poder arrancar el ordenador.

De manera básica, se ha de configurar la dirección base en los puertos de entrada – salida y el número de solicitud de interrupción (IRQ). Estos dos parámetros no pueden ser compartidos, y son, el primero, el medio de acceder en lectura y escritura a la tarjeta desde el programa de usuario, y el segundo, el número que se va identificar para la gestión de una interrupción producida por esta tarjeta.

Para las pruebas de configuración, se usó Windows 98, y se hicieron antes de la instalación de Linux en el sistema.

Dirección base.

En cuanto a la dirección base hemos de decir que la tarjeta ocupa 16 posiciones consecutivas en el espacio de direcciones de entrada – salida. La primera de estas direcciones, o dirección base se puede seleccionar en la tarjeta mediante un juego de seis interruptores DIP, de color rojo, nombrado por el fabricante S1. Estos interruptores ponen a uno o a cero los bits de la dirección base, dando lugar a determinarla. Se ha de tener cuidado con esto, porque no puede haber módulos con la misma dirección base, como ya hemos dicho, pero tampoco pueden coincidir posiciones del espacio de direcciones para módulos distintos. Se pueden consultar los administradores de dispositivos que vienen con los distintos sistemas operativos para conocer los rangos e IRQ´s que están ocupados en el ordenador, a fin de que no existan coincidencias entre los ya ocupados por otros dispositivos y la tarjeta AX10410. Lo mismo puede ser aplicado al resto de tarjetas.

Los interruptores deben moverse con el ordenador apagado.

La dirección base se asigna en un rango del 200 hexadecimal al 3F0 hexadecimal, debiéndose buscar un espacio de 16 direcciones libres para insertar la tarjeta. Veamos como es el S1 en la figura 6.6:

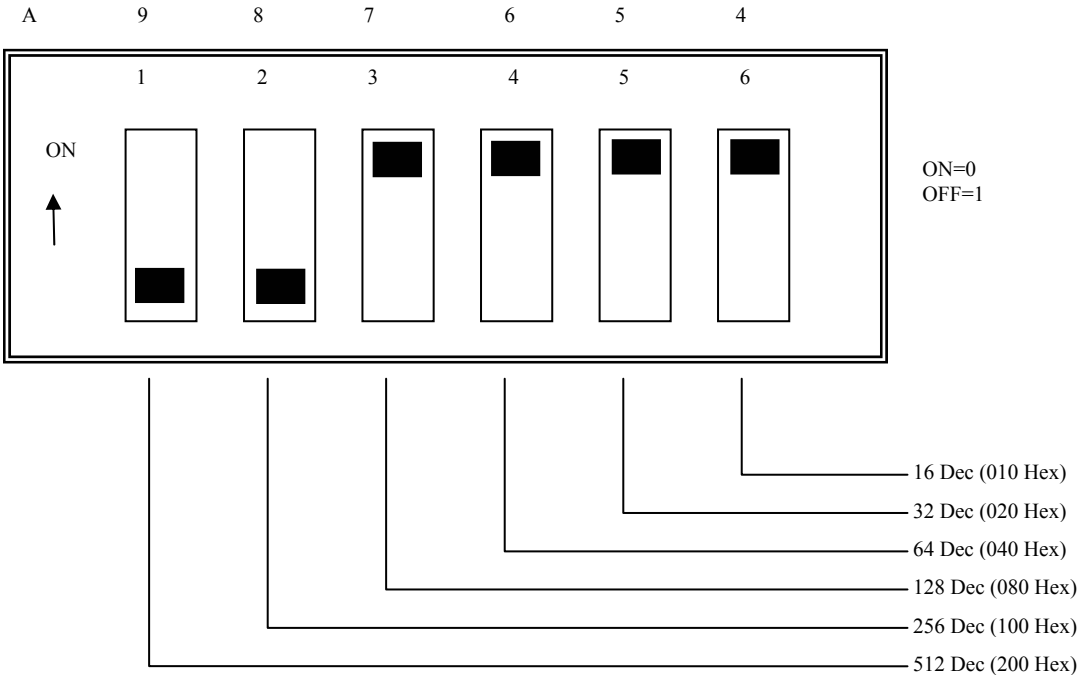


Figura 6.6: Interruptores S1 de configuración de la dirección base en AX10410.

Para asignar la dirección correcta hemos de colocar los interruptores en su posición ON u OFF. En la figura 6.6, tal y como están colocados los interruptores, tenemos la posición 300 en hexadecimal, ya que tenemos los interruptores numerados con 1 y 2 en OFF. Esto significa que, como el tenerlo en OFF es un uno, la dirección base es de 200 Hex + 100 Hex, es decir, 300 Hex, 768 en decimal, que es la dirección base proporcionada por el fabricante, que aparentemente debe funcionar, pues no está ocupada por ningún dispositivo, pero que no permite ni que arranque el ordenador, en la práctica.

Así, podemos ir moviendo los interruptores para asignar una dirección base. Hemos de tener en cuenta que el espacio de memoria que requiere la tarjeta es de 16 posiciones, por lo que, por ejemplo, si la dirección base es la 300 Hex, la tarjeta ocupará de la 300 Hex a la 30F Hex, y leyendo o escribiendo en esos puertos podremos acceder a la tarjeta. Posteriormente veremos la asignación de cada puerto con los distintos registros internos de la tarjeta.

Nivel de interrupción.

En cuanto al nivel de interrupción, en este caso, a diferencia de otras tarjetas, no se programa mediante interruptores, sino que se puede programar vía software en uno de los registros de la tarjeta. También se verá con posterioridad.

Selección entre single – ended y diferencial.

Pero no es sólo la dirección base lo que hay que cambiar, también se tiene que determinar el modo de funcionamiento de las entradas analógicas de la tarjeta. Esto se hace mediante el interruptor SW4 y el jumper JP3 de la tarjeta. Se tienen dos modos de funcionamiento, el single – ended y el diferencial. En el primero, todos los canales del módulo están referenciados a una tierra común, lo que significa que la fuente de señal y el amplificador están referenciados a la tierra. En este modo, la AX10410 tiene 16 entradas analógicas. El segundo de estos modos es el modo diferencial, que consiste en que dos amplificadores están conectados ambos a la fuente de señal, con polaridad + y polaridad -. Sirve para rechazar el ruido del modo común mediante este amplificador diferencial, pero la señal que se envía ha de tener esta forma. Existen sensores que mandan esta señal en este modo. La desventaja del modo diferencial es que sólo se dispone de ocho entradas analógicas en vez de 16. En realidad, no hemos usado entradas analógicas en el proyecto, por lo que actualmente, no tiene demasiada importancia esta configuración, pero para las pruebas, se usó el modo single – ended, que es más sencillo de probar. Veamos como se selecciona en la figura 6.7.

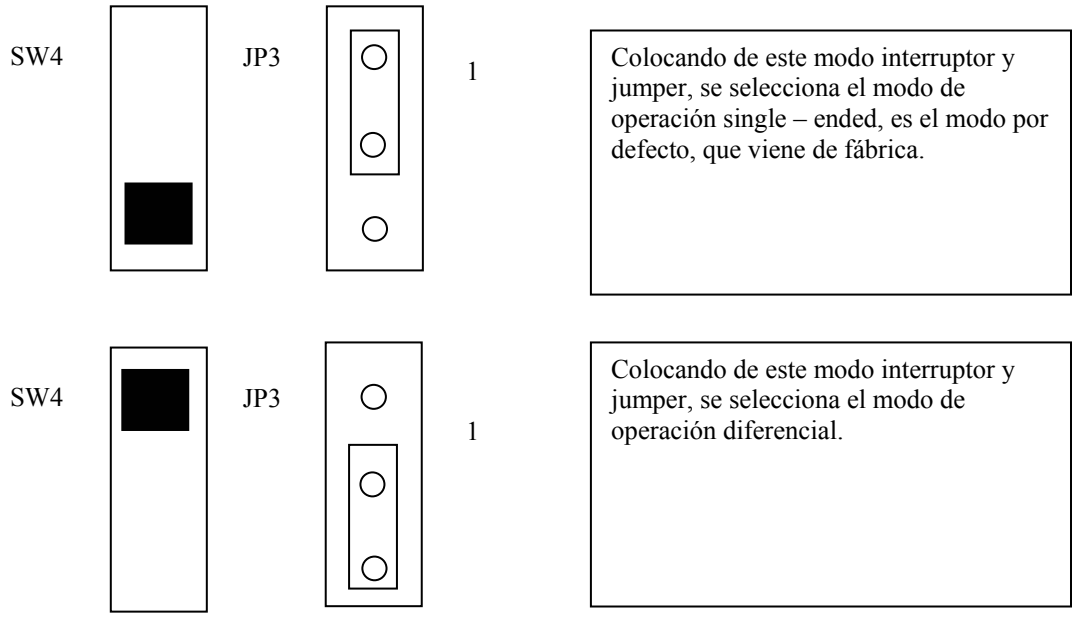


Figura 6.7: Selección del modo single – ended o diferencial.

Selección entre convertidor analógico / digital unipolar y bipolar.

El interruptor SW1 y el jumper JP2 sirven para configurar los convertidores analógico / digital en modo unipolar o bipolar. En modo unipolar, sólo sirven señales de

valor positivo, pero en modo bipolar, las señales podrán ser de valores tanto positivos como negativos. Podemos ver el modo de configurarlo en la figura 6.8.

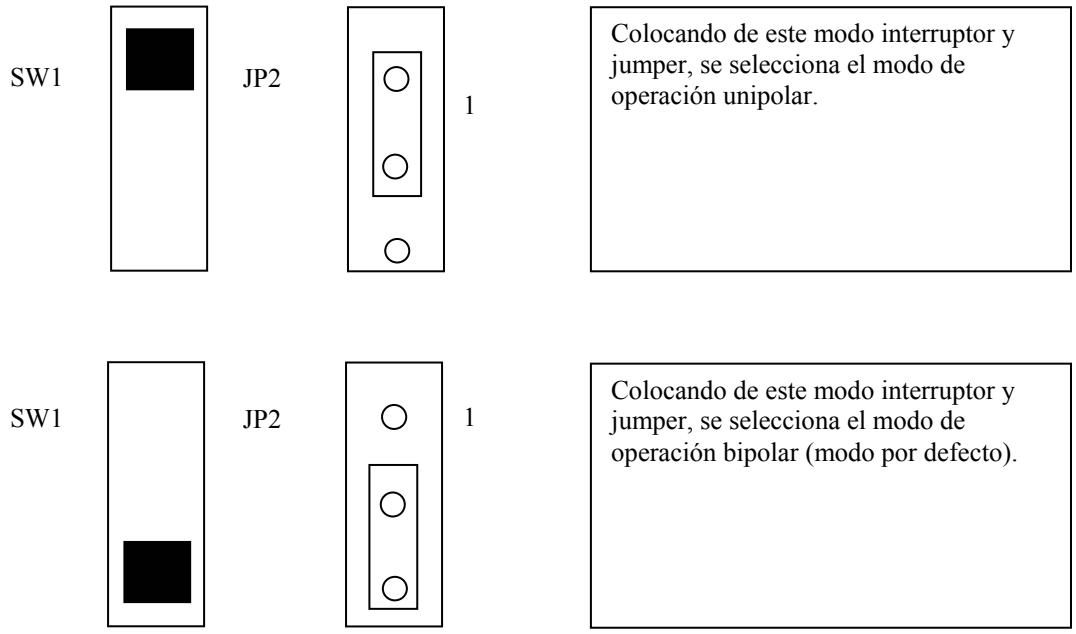


Figura 6.8: Configuración en modo unipolar o bipolar del convertidor A/D.

Selección entre convertidor digital / analógico unipolar y bipolar.

Se tienen dos convertidores digital / analógico. Estos sí son importantes, pues se actúa con ellos, como ya hemos visto. Pueden ser configurados de manera independiente, no como en el caso anterior, en el que, o bien todos eran unipolares, o bien todos bipolares. Dos interruptores, el SW2 y el SW3 son los encargados de esta configuración. El interruptor SW3 se asocia al canal 0 del convertidor digital / analógico, y el SW2 se asocia al canal 1. En nuestro caso, los configuramos de modo unipolar, ya que sólo vamos a actuar con señales positivas. Podemos verlo en la figura 6.9.

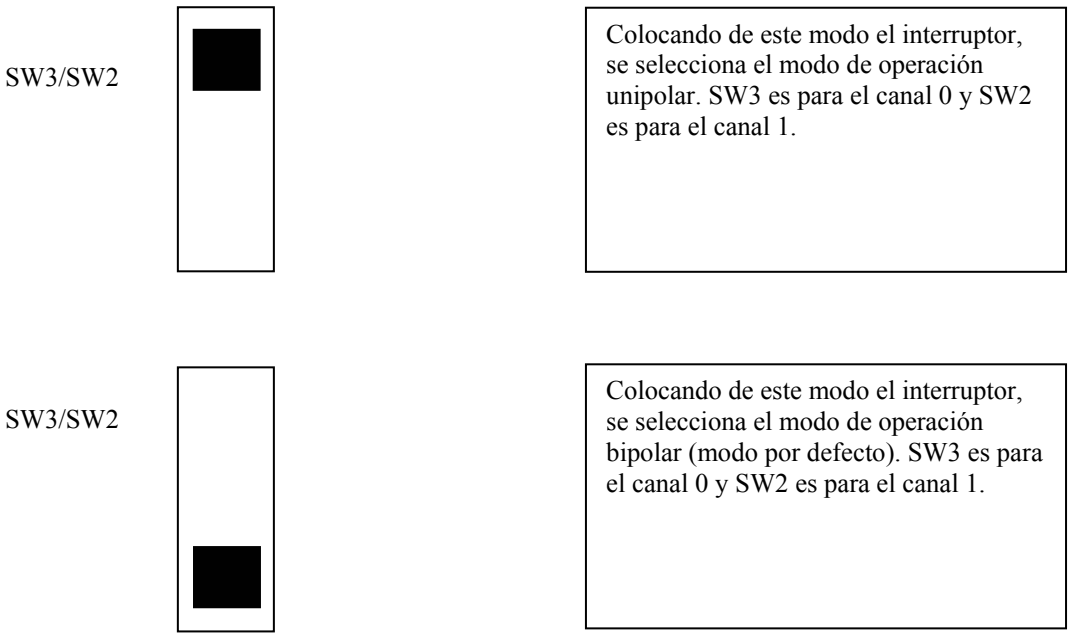


Figura 6.9: Configuración en modo unipolar o bipolar del convertidor D/A.

Selección del DMA.

El DMA es el acceso directo a memoria, que permite una transferencia de datos desde el dispositivo a mayor velocidad. Existen dos canales DMA disponibles, el DRQ1 y el DRQ3. Como en todo lo demás, se ha de procurar que no haya coincidencia de estos canales con canales usados por otros dispositivos. En nuestro caso, no usamos DMA, por lo que se han dejado los provenientes de fábrica por defecto. Los jumpers JP4 y JP5 se usan para esta selección, de la forma que podemos ver en la figura 6.10.

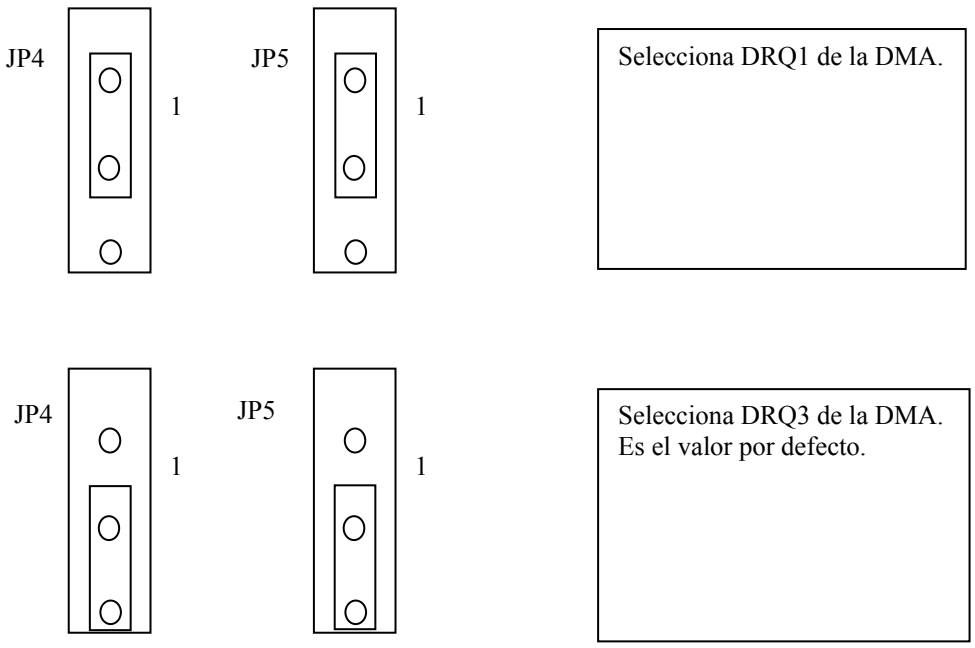


Figura 6.10: Configuración para la selección del canal DMA.

Selección del contador 0 del contador 8254.

Ya se ha dicho que la tarjeta posee un contador interno 8254, con el jumper JP6 se puede seleccionar una fuente externa del contador 0 del mismo, o una fuente interna de 100 KHz. No se ha usado en el proyecto, así que se dejó en la configuración por defecto.

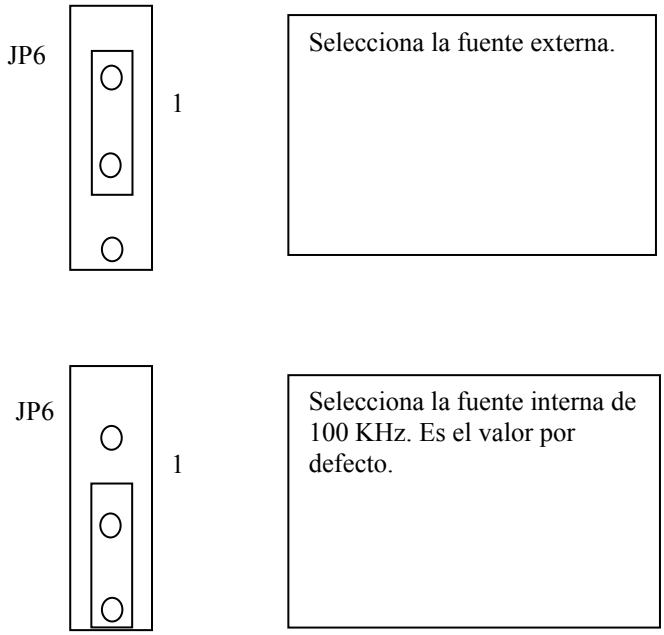


Figura 6.11: Selección de la fuente para el contador 8254.

Selección del temporizador del conversor analógico / digital.

La fuente de temporización para la conversión analógica / digital puede configurarse a 1 MHz o a 10 MHz, esto se puede realizar con el jumper JP7.

La frecuencia base (Fbase), de 1 MHZ o de 10 MHz, se conecta al chip programable contador / temporizador, es decir, el 8254. Concretamente, se conecta a la entrada de reloj 1. El contador 1 y el contador 2 se conectan internamente en cascada para generar una señal fija periódica, que sirve de fuente para lanzar la conversión analógica / digital. La tasa de temporización se puede calcular con la fórmula siguiente:

$$Timer\_rate = \frac{Fbase}{Divisor1 \bullet Divisor2}$$

Divisor1 y Divisor2 están asociados al valor establecido en los contadores 1 y 2 del chip 8254. Veamos la configuración del jumper en la figura 6.12:

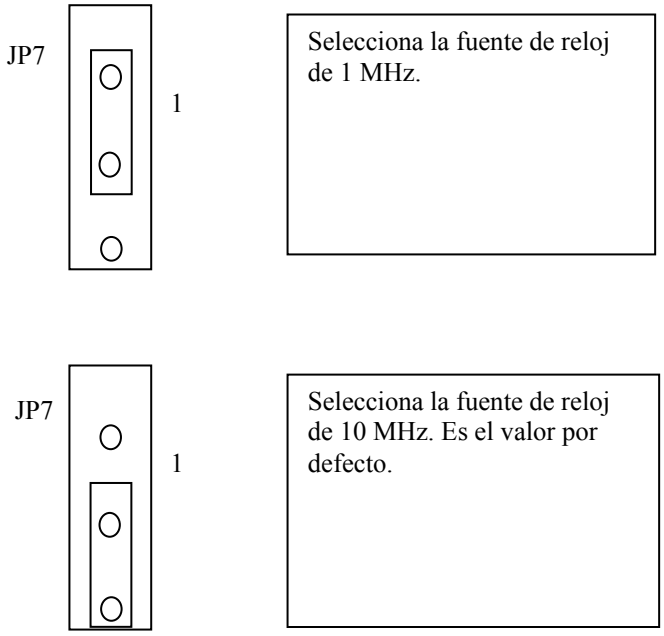


Figura 6.12: Configuración de la temporización del conversor A/D.

Tampoco se usa la temporización en nuestro diseño, en primer lugar, porque no existen conversiones analógicas / digitales, y en segundo lugar, porque el método escogido en el driver del dispositivo para esa conversión es mediante software, es decir, el programa principal controla toda la temporización y es el que lanza la conversión. La tarjeta avisa mediante interrupción de que la conversión ha sido completada. Pero, insistimos, esta característica no se ha usado, y no se ha utilizado porque no se reciben en este primer montaje datos analógicos del exterior, de hecho, dependiendo de cómo se

continúe el montaje veremos si se ha de usar esta conversión o si no hará falta, como otras muchas de las funcionalidades que proporciona esta tarjeta.

6.2.3.-Asignación de los pines de la conexión.

La tarjeta AX10410 actúa y recibe señales del exterior mediante un conector de cable plano de 50 pines. Este cable va a la tarjeta de adaptación de señales, con lo que se llevan las señales a los distintos lugares de la carretilla para que realicen su función. Veamos como es el conector en la figura 6.13.

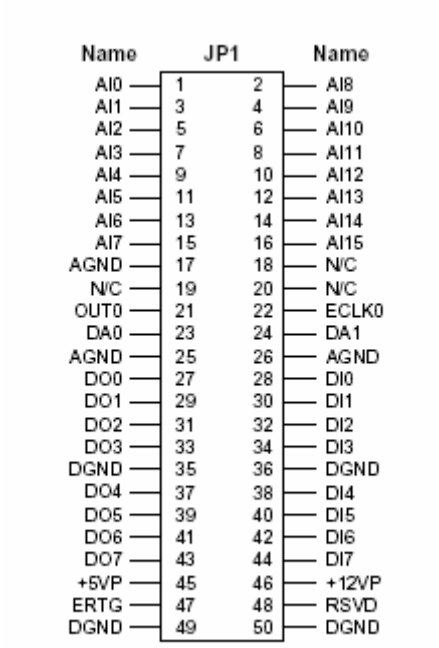


Figura 6.13: Asignación de los pines del conector del módulo AX10410.

El conector recibe en la tarjeta el nombre de JP1. En el cable plano, el pin 1 está marcado, y el cable de los 50 que corresponde a ese pin es de color rojo.

Veamos como quedan los pines en la tabla 6.1.

PIN	NOMBRE	I/O	DESCRIPCIÓN
1	AI0	Input	Entrada analógica (single ended: canal 0, diferencial: canal 0 alto)
2	AI8	Input	Entrada analógica (single ended: canal 8, diferencial: canal 0 bajo)
3	AI1	Input	Entrada analógica (single ended: canal 1, diferencial: canal 1 alto)
4	AI9	Input	Entrada analógica (single ended: canal 9, diferencial: canal 1 bajo)
5	AI2	Input	Entrada analógica (single ended: canal 2, diferencial: canal 2 alto)
6	AI10	Input	Entrada analógica (single ended: canal 10, diferencial: canal 2 bajo)

PIN	NOMBRE	I/O	DESCRIPCIÓN
7	AI3	Input	Entrada analógica (single ended: canal 3, diferencial: canal 3 alto)
8	AI11	Input	Entrada analógica (single ended: canal 11, diferencial: canal 3 bajo)
9	AI4	Input	Entrada analógica (single ended: canal 4, diferencial: canal 4 alto)
10	AI12	Input	Entrada analógica (single ended: canal 12, diferencial: canal 4 bajo)
11	AI5	Input	Entrada analógica (single ended: canal 5, diferencial: canal 5 alto)
12	AI13	Input	Entrada analógica (single ended: canal 13, diferencial: canal 5 bajo)
13	AI6	Input	Entrada analógica (single ended: canal 6, diferencial: canal 6 alto)
14	AI14	Input	Entrada analógica (single ended: canal 14, diferencial: canal 6 bajo)
15	AI7	Input	Entrada analógica (single ended: canal 7, diferencial: canal 7 alto)
16	AI15	Input	Entrada analógica (single ended: canal 15, diferencial: canal 7 bajo)
17	AGND	Ground	Tierra analógica.
18 a 20	N/C	-	No conectado.
21	OUT0	Output	Salida del contador 8254.
22	ECLK0	Input	Reloj externo para el contador 8254.
23	DA0	Output	Salida del conversor D/A 0.
24	DA1	Output	Salida del conversor D/A 1.
25 y 26	AGND	Ground	Tierra analógica.
27	DO0	Output	Salida digital canal 0.
28	DI0	Input	Entrada digital canal 0.
29	DO1	Output	Salida digital canal 1.
30	DI1	Input	Entrada digital canal 1.
31	DO2	Output	Salida digital canal 2.
32	DI2	Input	Entrada digital canal 2.
33	DO3	Output	Salida digital canal 3.
34	DI3	Input	Entrada digital canal 3.
35 y 36	DGND	Ground	Tierra digital.
37	DO4	Output	Salida digital canal 4.
38	DI4	Input	Entrada digital canal 4.
39	DO5	Output	Salida digital canal 5.
40	DI5	Input	Entrada digital canal 5.
41	DO6	Output	Salida digital canal 6.
42	DI6	Input	Entrada digital canal 6.
43	DO7	Output	Salida digital canal 7.
44	DI7	Input	Entrada digital canal 7.
45	+5VP	Source	5 voltios de alimentación del PC.
46	+12VP	Source	12 voltios de alimentación del PC.
47	ETRG	Input	Trigger externo.
48	RSVD	Output	Reservado.

PIN	NOMBRE	I/O	DESCRIPCIÓN
49 y 50	DGND	Ground	Tierra digital.

Tabla 6.1: Pines del conector del módulo AX10410.

Podemos ver como los pines de las entradas digitales realizan funciones diferentes, dependiendo de si se está en modo single ended o diferencial. También podemos ver que hay dos tierras aisladas, la analógica y la digital. En el montaje que se realizó en la placa de adaptación de señales, estas tierras fueron conectadas.

6.2.4.- Registros del módulo AX10410.

La AX10410 utiliza 16 direcciones consecutivas en el espacio de entrada / salida del ordenador. Así, para acceder a los distintos registros, sólo hemos de acceder a la dirección base más el número del registro que corresponda. En la tabla 6.2 podemos ver esos registros.

Situación.	Función.	Tipo
Base + 0	Byte bajo del conversor A/D. Iniciar A/D.	Lectura. Escritura.
+ 1	Byte alto del conversor A/D.	Lectura.
+ 2	Control de escaneo multiplexado.	Lectura / Escritura.
+ 3	Entrada digital. Salida digital.	Lectura. Escritura.
+ 4	Byte bajo del conversor D/A 0.	Escritura.
+ 5	Byte alto del conversor D/A 0.	Escritura.
+ 6	Byte bajo del conversor D/A 1.	Escritura.
+ 7	Byte alto del conversor D/A 1.	Escritura.
+ 8	Registro de status. Borrar interrupción.	Lectura. Escritura.
+ 9	Registro de control.	Lectura / Escritura.
+ 10	Registro de control del timer.	Escritura.
+ 11	Control de ganancia.	Lectura / Escritura.
+ 12	Contador 0 del 8254.	Lectura / Escritura.
+ 13	Contador 1 del 8254.	Lectura / Escritura.
+ 14	Contador 2 del 8254.	Lectura / Escritura.

Situación.	Función.	Tipo.
+ 15	Control del contador 8254.	Escritura.

Tabla 6.2: Registros del módulo AX10410.

Veamos ahora la descripción de los registros.

A/D Data Register (base + 0 y base + 1, lectura).

Estos son los dos registros destinados a los datos del conversor A/D.

	7	6	5	4	3	2	1	0
Base + 0	D3	D2	D1	D0	C3	C2	C1	C0

LSB

Bits 0 a 3, de C0 a C3. Se utiliza para especificar uno de los 16 canales (del cero al 15) como el canal de entrada para las conversiones A/D.

Bits 4 a 7, de D0 a D3. Son los bits menos significativos de los datos del conversor A/D.

	7	6	5	4	3	2	1	0
Base + 1	D11	D10	D9	D8	D7	D6	D5	D4

MSB

Bits 0 a 7, de D11 a D4. Son los bits más significativos del conversor A/D. Se han de unir a los anteriores para tener el valor de la conversión, respetando siempre su posición en la palabra de doce bits. Esto se ha de realizar en el programa de usuario, para que el dato convertido sea válido, y acorde con un valor coherente de la conversión.

Software Start (base + 0, escritura).

Una escritura en el registro base + 0 actúa como trigger de una conversión A/D, lanza la conversión.

Multiplexed Scan Register (base + 2, lectura / escritura).

	7	6	5	4	3	2	1	0
Base + 2	UC3	UC2	UC1	UC0	LC3	LC2	LC1	LC0

Los bits 4 a 7 son el canal final (end channel), y los bits 0 a 3 son el canal inicial (start channel). Este registro es de lectura / escritura, y controla los límites de escaneo de canales para la conversión A/D multiplexada. Aproximadamente 1.5 microsegundos después del comienzo de una conversión, mientras el simple / hola está manteniendo el canal anterior, el canal multiplexado se incrementa, preparado para la siguiente conversión. Cuando se finaliza la conversión del canal en el ending channel, se repite el ciclo, empezándose por el starting channel. Cuando se escribe en este registro, se inicializa el contador de manera automática, empezando por el starting channel.

Si lo que se quiere realizar es una conversión en un canal único, deben coincidir el starting y el ending channel en un número, el del canal deseado para la conversión.

Digital Input Register (base + 3, lectura).

En este registro están las entradas digitales, como son ocho, cada bit equivale a una entrada.

Leyendo en el registro, podemos saber el valor de la entrada digital en cada momento.

	7	6	5	4	3	2	1	0
Base + 3	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0

Digital Output Register (base + 3, escritura).

En este registro están las entradas digitales, como son ocho, cada bit equivale a una entrada.

	7	6	5	4	3	2	1	0
Base + 3	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

Escribiendo en el registro, podemos poner el valor de la salida digital en cada momento.

D/A0 Output Registers (base + 4 y + 5, escritura).

Escribiendo un valor en estos registros, teniendo en cuenta la posición de los bits, sacamos un valor analógico por el canal 0 equivalente al valor escrito. Se han de desplazar los bits necesarios para que el valor cuadre en los doce bits del conversor.

	7	6	5	4	3	2	1	0
Base + 4	B7	B6	B5	B4	B3	B2	B1	B0

	7	6	5	4	3	2	1	0
Base + 5	X	X	X	X	B11	B10	B9	B8

Vemos que los bits más significativos del registro base + 5 no se usan, pues, como ya se ha dicho, la conversión D/A es de doce bits.

D/A1 Output Registers (base + 6 y + 7, escritura).

Escribiendo un valor en estos registros, teniendo en cuenta la posición de los bits, sacamos un valor analógico por el canal 1 equivalente al valor escrito. Se han de desplazar los bits necesarios para que el valor cuadre en los doce bits del conversor.

	7	6	5	4	3	2	1	0
Base + 6	B7	B6	B5	B4	B3	B2	B1	B0

	7	6	5	4	3	2	1	0
Base + 7	X	X	X	X	B11	B10	B9	B8

Vemos que los bits más significativos del registro base + 5 no se usan, pues, como ya se ha dicho, la conversión D/A es de doce bits.

Para escribir en estos registros (tanto D/A0 como D/A1), se ha de hacer en el orden lógico, es decir, primero el byte menos significativo y luego el más significativo. La salida analógica no es modificada hasta que no se escribe el segundo byte, el alto. Esto se debe realizar así porque los registros de conversión D/A son de doble buffer.

Status Register (base + 8, lectura).

	7	6	5	4	3	2	1	0
Base + 8	EOC	U/B	MUX	INTP	NC3	NC2	NC1	NC0

Los bits del 0 al 3, NC0 a NC3 son bits de status, y dan información de la configuración de la siguiente conversión. Son el número del siguiente canal para la conversión si EOC es 0. El canal cambia poco después de que EOC se ponga en alto. Cuando EOC es 1, estos bits son indeterminados.

El bit 4 es INTP, que es la señal de interrupción que va dirigida a un nivel del 2 al 7 por el control register. Si no hay interrupción, INTP es 0. Tras la generación de una interrupción, INTP se pone a 1 y se mantendrá en alto hasta que se resetee escribiendo en este mismo registro. Por lo tanto, se ha de escribir en este registro en algún punto de la rutina de interrupción para que se readmitan las interrupciones de este módulo.

El bit 5 es MUX, e indica si la AX10410 está en modo single – ended (S) o diferencial (D), que recordemos que se determinaba mediante el interruptor externo SW4. Si vale 0, está en modo diferencial con ocho canales, si vale 1, en modo single – ended con 16.

El bit 6 es U/B, e indica si el conversor A/D está en modo unipolar (U) o bipolar (B), elegido mediante el interruptor SW1. Si vale 0, está en modo bipolar, y si vale 1, en modo unipolar.

El bit 7 es EOC, sirve para indicar el fin de un ciclo de conversión A/D. Si vale 0, la conversión está completa y el dato es válido, si vale 1, la conversión no se ha acabado, y el dato no es válido. El hecho de empezar una conversión A/D pone este bit a 1.

Clear Interrupt Register (base + 8, escritura).

Una escritura en este registro limpia el bit interrupt request INTP, permitiéndose una nueva interrupción. Como ya se ha dicho, esa escritura tiene que ser programada en la rutina de interrupción.

Control Register (base + 9, lectura / escritura).

	7	6	5	4	3	2	1	0
Base + 9	INTE	I2	I1	I0	X	DMAE	T1	T0

Los bits 0 y 1, que son los T0 y T1 controlan la fuente de los pulsos que lanzan la conversión A/D. Veámoslo en la tabla 6.3.

T1	T0	Acción
0	X	Inicio sólo por software.
1	0	Inicio con la subida de una señal externa (extrig).
1	1	Inicio con una señal del timer interno.
X significa que su valor no importa.		

Tabla 6.3: Posibles valores de los bits T0 y T1 en el registro de control.

El bit 2 es el bit DMAE, sirve para habilitar el acceso directo a memoria, cuando DMAE es 1, y para deshabilitarlo cuando DMAE es 0.

El bit 3 no tiene importancia.

Los bits 4 a 6, que son de I0 a I2 seleccionan el nivel deseado de interrupción, que ya dijimos que se hacía vía software, así como que no debe coincidir con el de otros módulos instalados. En la tabla 6.4 vemos los posibles valores.

I2	I1	I0	Nivel de Interrupción.
0	0	0	Inválido.
0	0	1	Inválido.
0	1	0	Nivel 2.
0	1	1	Nivel 3.
1	0	0	Nivel 4.
1	0	1	Nivel 5.
1	1	0	Nivel 6.
1	1	1	Nivel 7.

Tabla 6.4: Posibles valores de los bits I0 a I2 en el registro de control.

El bit 7 es el bit INTE, y sirve para habilitar o deshabilitar las interrupciones generadas por la AX10410.

Si INTE es 1 y DMAE es 0, una interrupción se genera cuando se acaba cada conversión A/D, con el nuevo dato disponible. Si INTE es 1 y DMAE es 1, una interrupción se genera cuando un final de cuenta (T/C) se recibe del controlador DMA 8237 para indicar el final de una transferencia DMA.

Timer Trigger Control Register (base + 10, escritura).

	7	6	5	4	3	2	1	0
Base + 10	X	X	X	X	X	X	X	TRGE

El bit 0 es TRGE, el trigger externo ETRG que es el pin 50 del conector externo de 50 pines y este habilitador de trigger (TRGE) están relacionados. Cuando TRGE está bajo (0), el timer está habilitado, y no puede estar controlado por ETRG. Si TRGE está en alto (1), el timer puede ser controlado por ETRG. El timer es del chip AMD 8254, que ya se ha explicado, con distintas frecuencias de reloj seleccionadas por el jumper JP7. En la tabla 6.5 vemos las relaciones entre TRGE y ETRG.

TRGE	ETRG	Timer.
0	0	Habilitado
0	1	Habilitado
1	0	Deshabilitado

TRGE	ETRG	Timer.
1	1	Deshabilitado

Tabla 6.5: Relación entre TRGE y ETRG para el trigger en la AX10410.

Gain Control Register (base + 11, lectura / escritura).

	7	6	5	4	3	2	1	0
Base + 11	X	X	X	X	R3	R2	R1	R0

Los bits 0 a 3, que corresponden a los bits R0 a R3 especifican la ganancia del conversor D/A, según la tabla 6.6.

Los bits 4 a 7 no tienen importancia alguna.

Poalridad		Código de ganancia				Ganancia	Rango
UNI	BIP	R3	R2	R1	R0		
X	0	0	0	0	0	1	± 10V
X	0	0	0	0	1	2	± 5V
X	0	0	0	1	0	4	± 2.5V
X	0	0	0	1	1	8	± 1.25V
0	X	0	0	0	0	1	0V a 10V
0	X	0	0	0	1	2	0V a 5V
0	X	0	0	1	0	4	0V a 2.5V
0	X	0	0	1	1	8	0V a 1.25V
Otros no son válidos.							
Nota: 0 significa seleccionado este modo, X es no seleccionado.							

Tabla 6.6: Posibles valores de la ganancia para el registro Gain Control.

8253 Programmable Internal Timer (de base + 12 a base + 15).

Son los registros que quedan, permiten programar el timer 8253 interno de la AX10410. No se han utilizado, así que sólo los nombraremos, pero se puede obtener información sobre ellos en el manual de la tarjeta, del que se dispone en el laboratorio, concretamente, en el apéndice G del mismo.

Base + 12, counter 0 register, es de lectura / escritura.

Base + 13, counter 1 register, es también de lectura / escritura.

Base + 14, counter 2 register, como los dos anteriores, de lectura / escritura.

Base + 15 es el registro de control del 8254, es de escritura.

Capítulo 7: El ordenador de control (software).

7.1.- El sistema operativo.

7.1.1.- ¿Cuál se ha usado?

Dependiendo de la tarea que se quisiera realizar, hemos utilizado un sistema operativo u otro. Para la prueba de las tarjetas se ha usado Windows 98, debido a que los drivers y programas de utilización que presenta el fabricante vienen para el sistema operativo DOS, más o menos compatible con ese sistema operativo, no haciéndose necesaria la programación de un driver en otro sistema operativo para la simple prueba de una tarjeta. En esto también ha influido la familiarización que el proyectando tenía con el sistema operativo en cuestión, así com la facilidad de ver los componentes instalados anteriormente a la inclusión de un nuevo módulo, evitando así coincidencias de IRQ y de dirección base.

Para la prueba y calibración del láser Robosense también se utilizó Windows, por la misma razón.

Sin embargo, el funcionamiento de la carretilla se ha realizado bajo el sistema operativo GNU/Linux, al que se puede acceder en forma de distintas distribuciones, iguales en lo básico, pero distintas en algunos aspectos. La distribución que se ha utilzado es la Debian en su versión del kernel 2.2.19pre17, en su distribución Potato, que era la distribución con la condición de estable en el comienzo del proyecto.

7.1.2.- ¿Por qué Linux?

Cabe la pregunta de por qué si los módulos que se han utilizado vienen con sus drivers para DOS, se ha utilizado este otro sistema operativo, que implica la creación de nuevos drivers y la familiarización con un nuevo sistema operativo. La respuesta tiene varias causas.

En primer lugar, podemos ver las ventajas de Linux en sí, como que es un sistema basado en UNIX, con una manera similar de gestionar los recursos del ordenador, que se ha manifestado en la mayoría de los casos superior a las formas que tienen los sistemas operativos basados en DOS, la multitarea real, la forma de tratar los archivos, la estabilidad del kernel (núcleo del sistema operativo) 2.2.19, hace que sea un sistema ideal para aplicaciones como la que nos ocupa, en la que un fallo o que el ordenador se cuelgue pueden tener consecuencias graves, no querríamos tener una carretilla de 3500 Kg con unas palas de más de un metro andando y girando sin control sólo porque el sistema operativo nos ha mostrado una pantalla azul. Esta forma de tratar los dispositivos hace al Linux muy interesante, ya que el hecho de programar drivers puede dar lugar a accesos a lugares de memoria que no deberían ser cambiados, en estos casos, la existencia de un kernel "por debajo" de toda aplicación que se ejecuta da lugar a no poder acceder a lugares prohibidos, porque simplemente el núcleo no lo permite, con lo que, desde luego, el ordenador no se cuelga, o se cuelga menos que con otros sistemas operativos.

Pero Linux tiene otras ventajas, como su gratuidad, que lo hace ideal en aplicaciones de investigación como ésta, en las que parece que debe primar el conocimiento sobre el ánimo de lucro. Linux se distribuye mediante licencia GPL (General Public License), que permite la gratuita distribución, e incluso la fomenta, así como la posibilidad de cambiar el software distribuido con esta licencia, siempre que el producto creado así conserve esta licencia GPL. El texto de esta licencia se puede leer en la página de GNU, www.gnu.org. Esta gratuidad da lugar a que sea muy sencilla, y también gratuita, la actualización del software, ya que se pueden descargar los paquetes de internet y tenerlos instalados de manera inmediata, sobre todo con conexiones de banda ancha como las que se poseen en el laboratorio.

También se ha utilizado este sistema operativo, así como esta distribución y no otra (Mandrake, Sussex, BSD, Red Hat, etcétera) debido a que la forma de descarga e instalación de los paquetes es muy sencilla a la hora de actualizar el sistema. Así, siguiendo este criterio de facilidad, también ha sido fundamental que el software de otro vehículo robótico del laboratorio, el Romeo 4R, esté programado bajo este sistema operativo, lo que ha permitido que se pueda utilizar, con ciertos cambios para adaptar el software a nuestro vehículo, gran parte de la aplicación programada para el Romeo 4R, como son algún driver, con leves cambios, la estructura, ciertas clases con modificaciones, etcétera. Para una mayor información sobre todas estas cuestiones,

como el Linux, los drivers para Linux, la licencia GPL, la estructura del programa y otras muchas cuestiones, se puede consultar el proyecto de fin de carrera de Rafael Martín de Agar (año 2002), que es el creador de la práctica totalidad del software del Romeo 4R, y que se extiende en todas estas cuestiones con una profundidad muy superior a la que nosotros llegaremos, pues estas y sólo estas consideraciones eran el objeto de su proyecto.

Por último, se ha utilizado Linux también por cuestiones de futuro y posible ampliación del software, ya que se puede actualizar el núcleo con un parche que le confiere características de sistema operativo en tiempo real, y esto es muy apreciado en aplicaciones robóticas. La posibilidad aún no implantada en el software existente de utilizar temporizaciones independientes para cada hilo del programa, hace que esta opción de tiempo real sea muy interesante para el control de la carretilla, que, al fin y al cabo, es un sistema totalmente de tiempo real.

Como ya se ha dicho, Linux es un sistema operativo estilo UNIX, con sus usuarios y sus distintos privilegios. Existe un usuario administrador del sistema o root, es necesario conocer la clave de acceso para entrar en el sistema como un usuario cualquiera, también como root, que es el usuario que tiene todos los privilegios, como el de insertar módulos (drivers), apagar el sistema, reiniciarlo, ejecutar todos los programas y comandos, así como crear o quitar usuarios y cambiar sus privilegios. La clave de acceso del superusuario o root para nuestro sistema es “carretilla”. Deberemos entrar como root cuando el sistema nos pida el usuario con la palabra login, y luego, teclear este password o clave de accesso cuando nos sea requerido. Se aconseja ejecutar el programa de control en modo superusuario, evitando así problemas de permisos por ser algunos archivos propiedad del autor del proyecto y otros, propiedad del administrador del sistema, que en este caso son la misma persona, aunque no sea así para el sistema operativo.

7.2.- El software de control.

7.2.1.- Breve descripción.

El software de control se ha desarrollado en C++, utilizando el compilador g++ que proporciona el Linux. Así, compilando una serie de archivos, podemos obtener el programa completo de control de la carretilla. Pero no sólo se han de compilar los archivos de los distintos módulos del programa, también se han de compilar los drivers de los distintos dispositivos de los que disponemos en el ordenador. En nuestro caso, sólo necesitamos compilar el driver de la tarjeta AX10410, puesto que usamos ese dispositivo y el puerto serie, que ya tiene su driver.

7.2.2.- ¿Por qué C++?

En un principio, sobre todo a los que saben programar y han programado anteriormente en C, pueden surgir reticencias sobre la utilización del C++ en vez de este otro lenguaje de programación. Estas reticencias son, en nuestra opinión, producto del desconocimiento del C++, y son superadas en el momento en el que se tiene un conocimiento algo más profundo de este lenguaje.

Y es que las propiedades que el C++ posee le hacen más adecuado para programas grandes en los que la modularidad y la buena organización de las distintas partes del programa son aspectos de una importancia crucial. Así, el C++, como lenguaje orientado a objetos que es, nos permite modificar unos objetos sin influir en los otros, ni en su correcto funcionamiento. Las características de herencia entre clases del C++ son idóneas para la concepción en niveles que tiene el programa, y que ahora veremos. Conceptos como el encapsulado son ideales cuando no es un único programador el que va a utilizar el código, sino que hay mucha gente que puede utilizar las funciones que le proporcionan las clases programadas por un programador, llamémosle principal, para realizar distintas aplicaciones con ese código.

Tras esto, que son cuestiones que afectan al programa en cuestión, C++ también proporciona comodidades generales, digamos que independientes de la aplicación a programar, como un trato más cómodo de la entrada / salida, o de los ficheros.

Otra de las causas, y no diremos que es de las menos importantes, porque sería mentir, es que el programa del Romeo 4R está escrito en C++, y ya hemos hablado de

aprovechar lo ya programado para este vehículo para, con cambios para adaptarlo a nuestro vehículo.

Aun así, en caso de que alguno de los programadores que van a utilizar las distintas clases programadas en C++ para una aplicación concreta sea un partidario, digamos, irreductible de C++, es conocido que la compatibilidad entre C y C++ es total o prácticamente total, y en el caso del compilador gcc o el g++, que son los que Linux proporciona, es total, desde luego.

7.2.3.- Drivers en Linux.

Para ver cómo trata Linux los dispositivos, se recomienda el libro Linux Device Drivers, o bien echar un vistazo al proyecto ya mencionado de Rafael Martín de Agar, de 2002, por lo que a nosotros respecta, diremos que en Linux se tratan los dispositivos como ficheros, y que hemos de leer o de escribir en esos archivos de dispositivo para leer o escribir en el dispositivo propiamente dicho.

Para compilar un módulo o driver de Linux, ser ha de tener el kernel preparado para esa compilación. Un núcleo se prepara para eso compilándolo con una serie de opciones que permitan la creación de módulos. Las compilaciones de un módulo son válidas para ese kernel concreto, en otra versión del núcleo, puede que no funcionen o no funcionen del todo bien. En nuestro caso, no tenemos el kernel compilado con las opciones correctas para poder crear módulos, así que el driver de la AX10410 ha de ser compilado en el ordenador de control del Romeo 4R, que sí está preparado para ello y tiene el mismo núcleo que el de la carretilla.

Pero no sólo hay que crear el driver, también hay que crear un enlace al dispositivo, es decir, el archivo especial al que se accederá cuando se quiera acceder al dispositivo. Este archivo está en el directorio que Linux tiene para los dispositivos, "/dev". Tras esto, se ha de insertar el módulo, que es un archivo ".o" con la orden insmod.

Del driver diremos sólo que lo que hace es intercambiar datos entre un espacio de usuario, y un espacio que pertenece al kernel, restringido al usuario, y al que pertenecen los dispositivos. Con esto evitamos que cualquier usuario pueda escribir en un dispositivo sin pasar por el driver, pues el núcleo lo impediría, así, programando de manera correcta el driver, sabemos que el resto del programa no nos va a modificar nada distinto del dispositivo al que quiere acceder, previniendo de posibles cuelgues.

También hemos de decir que ha sido programado en C, lenguaje más apropiado que el C++ para estas aplicaciones de tan bajo nivel.

De todas maneras, el driver ya está realizado, y sólo hay que insertarlo, para que esto se haga correctamente, pues es necesario un acceso al mismo tras su inserción en el sistema, se puede ejecutar el script "driver_con_es" o driver_sin_es" en el directorio "/home/eduardo/carretilla/E16", que es el directorio que contiene todo el software de la carretilla. Este script inserta el driver y accede a él con una sola orden, permitiendo que todo funcione correctamente.

7.2.4.- Estructura del programa.

El programa está dividido en tres capas, esto es consecuencia de la concepción original del mismo de Rafael Martín de Agar. Se trata de que las distintas capas se ocupen de una cosa distinta, así, una primera capa está orientada al dispositivo, la segunda está pensada para enganchar esa capa con la capa tercera, y la tercera capa es la que provee las distintas funciones al usuario final del programa. Esta división se ve promocionada por la utilización del lenguaje C++, y las capas se traducen en herencia de distintas clases.

En la capa de dispositivo, se tienen los drivers y las distintas clases que se encargan de acceder a ellos y realizar una cierta preparación de los datos de los mismos. En esta capa destacamos el driver de la AX10410, así como la clase Puerto_serie, encargada de configurar el puerto serie y de acceder a él a través del driver que se proporciona con el sistema operativo.

En la capa segunda, se tienen clases que utilizan los dispositivos y que transforman sus datos para que se acceda a ellos de manera útil. En este nivel están la clase AX y la clase Robosense, dedicadas a la AX10410 y al sensor láser de Siman. La clase AX utiliza el driver de la AX10410 y la clase Robosense hereda de la clase Puerto_serie, pues, como ya se ha dicho, se accede a este dispositivo mediante el puerto serie.

En la capa tercera, se tiene la clase que hereda de todas las demás, la clase Romeo, pues es la misma clase que se tiene en el Romeo 4R. Esta clase hereda de todas las clases que se tengan referentes a los distintos dispositivos, en nuestro caso, hereda de dos clases, la clase AX y la clase Robosense. Si se ve el código de esta clase, se puede ver que hereda de más clases, esto es consecuencia de que quitar las clases

referentes a los dispositivos del Romeo era costoso, por eso se hereda de ellas, aunque luego en el programa no se accede a ellas, ya que en ese caso, se produciría un error, pues estaríamos intentando acceder a dispositivos que no existen en nuestro ordenador. En esta clase se realiza la combinación de los distintos datos que se obtienen, y se tienen los procedimientos que serán usados por el usuario final para el control de la carretilla mediante un algoritmo que se ejecutará en el main del programa, utilizando las funciones de la clase Romeo, de la que se instancia un objeto global en el archivo en el que está el main, el objeto romeo. El objeto es global por la existencia de hilos en el programa, es necesario que lo sea pues si no, no hay forma de acceder a los procedimientos del objeto desde otro hilo del programa.

En la figura 7.1, podemos ver la estructura del programa.

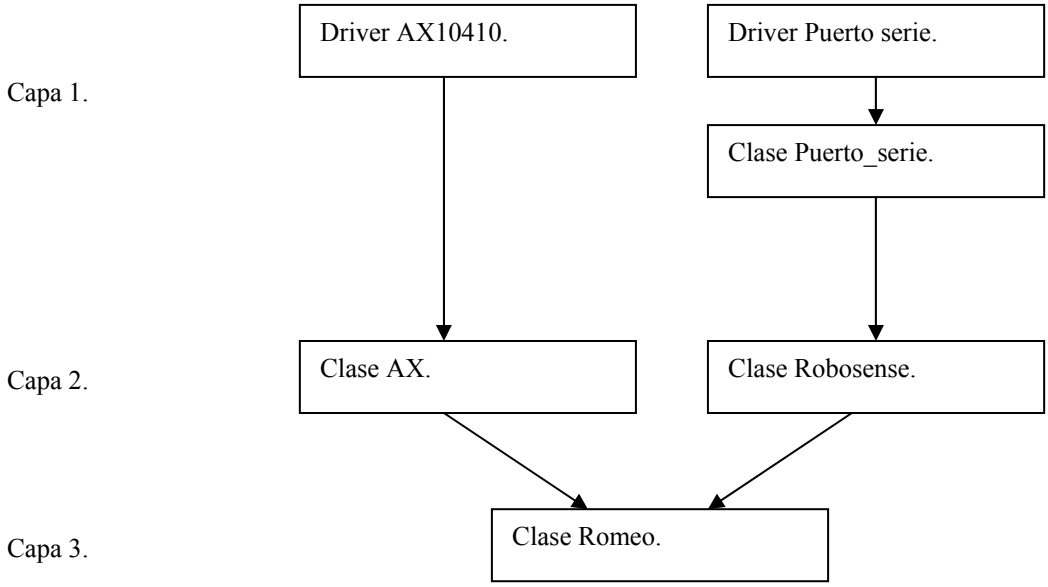


Figura 7.1: Estructura del programa de control de la carretilla.

Se pueden ver las distintas clases y los niveles, representándose la herencia mediante flechas.

Pero no sólo se tiene una división en capas del programa, también se puede dividir en el tiempo de ejecución porque no sólo se ejecuta un main en el programa principal, sino que se necesita un procesamiento paralelo. Esto se ha resuelto mediante hilos, así, se tienen varios hilos corriendo en el programa. Un hilo es un proceso que se crea a partir de otro y que, desde el momento de su creación, corre en paralelo con él.

Tiene una serie de atributos y prioridades, así como formas de traspasar datos de un hilo a otro, o al main.

En nuestro programa se identifican hilos con dispositivos, pero no sólo eso. Así, si existe un dispositivo del que se han de leer datos, utilizaremos un hilo que los vaya leyendo, para no tener que parar el programa principal a la espera de un nuevo dato de un dispositivo. Este parar el programa se haría crítico con muchos dispositivos conectados, pero también puede darse que el hecho de tener que dividir el tiempo de proceso entre muchos hilos por le existencia de muchos dispositivos puede acabar ralentizando el programa y que no funcione correctamente. Sobre este supuesto hemos de decir que se posee un ordenador suficientemente rápido para que esa circunstancia no se produzca con incluso muchos más dispositivos de los que realmente se tienen. El sistema operativo provee de mecanismos para la creación, el lanzamiento y la comunicación entre hilos, que hemos de decir que está muy bien resuelta en Linux y por el lenguaje C++.

Así, tenemos tres hilos relacionados con nuestro proyecto, aparte de los antiguos, que provienen del Romeo 4R, pero que no se lanzan. Estos tres hilos representan lo que se quiere que se ejecute en paralelo con el programa principal, compartiendo tiempo de proceso con él.

En primer lugar, podemos ver que, en nuestro proyecto, sólo hay un dispositivo que recoge datos del exterior y los manda al ordenador de control. Ese dispositivo es el Robosense, por lo que tendremos un hilo que se encargue de actualizar el los nuevos datos del Robosense, utilizando el procedimiento “update_robosense” de la clase Romeo.

En segundo lugar, el programa posee un procedimiento en la clase Romeo encargado de estimar ciertas variables que determinan la posición, orientación y velocidad de nuestro vehículo. Esta función es “update_estimación”, y está pensada para mezclar los datos de las estimaciones de distintos dispositivos, en caso de que hubiera varios, quedándose con la que se considera que es la más fiel a la situación real del robot. En nuestro caso sólo se tiene un dispositivo, pero se ha mantenido este procedimiento para no variar la estructura original. Lo que sí hace este procedimiento es llamar a funciones que calculan otros parámetros a partir de las medidas enviadas por el Robosense, así como realizar un filtrado de los mismos, para eliminar datos espurios comparándolos con estimaciones odométricas de la próxima posición.

En tercer lugar, se tiene un hilo para el cálculo de la señal de control a aplicar a la dirección de la carretilla, se implementa un PID. En el caso del Romeo se han utilizado tarjetas de control de motores, que pueden ser programadas y sintonizadas como un PID, y en las que lo que hay que actualizar es solamente la referencia, cerrando ellas por sí solas el bucle de control. Estas tarjetas no se han usado, entre otras cosas porque no se disponía de encoders con los que realimentar la posición de las ruedas, y estas tarjetas aceptan sólo realimentación por encoders, por lo que el bucle de control PID se hubo de cerrar dentro del programa de control, y para no pararlo, se utilizó un hilo.

Por último, se tiene el "main", que corre en paralelo con estos otros hilos, en él se ejecuta el algoritmo de control deseado para el vehículo, por ejemplo, un algoritmo de persecución pura. Este algoritmo fijará la referencia de la posición de las ruedas de la carretilla para que el robot siga una trayectoria deseada de manera lo más aproximada posible. También puede consultar datos de la posición y orientación instantánea del robot, datos necesarios para calcular la próxima referencia a aplicar.

Por último diremos que existen otros hilos programados, que llaman a otros procedimientos de la clase Romeo, pero se refieren a dispositivos del Romeo 4R y no son lanzados en nuestro programa.

7.2.5.- Estructura de directorios.

Para encontrar todos estos programas, hay que buscar en el directorio de usuario del autor del proyecto en el ordenador de control. Este directorio es el "/home/eduardo".

Dentro de este directorio hay otros muchos, pero el que nos interesa es el directorio referente al proyecto, que es el "/home/eduardo/carretilla".

A su vez, en este directorio existen varios directorios que contienen el programa antiguo para el Romeo 4R, así como un directorio en el que se puede ver un programa también realizado por el autor del proyecto, que es similar al programa robodemo.exe que se proporciona por el fabricante con el sensor láser Robosense.

Pero el directorio que nos interesa es el "/home/eduardo/carretilla/E16", en el que están todos los archivos del programa de control de la carretilla, así como otros archivos, como directorios para archivos de log (registro de datos de las pruebas), archivos para obtener caminos mediante el procesado de esos archivos de log, el makefile del programa, para modificar sólo el código que se desee cambiar, y luego, con

introducir la orden make, el programa se compila (sólo la parte que se cambia) y se enlaza.

Dentro de este directorio están los directorios de los dispositivos, del Robosense, del puerto serie, del módulo AX10410 (con los directorios del driver dentro), y directorios de dispositivos del Romeo 4R. Existe un directorio para los hilos, otro para los distintos algoritmos, que no tienen por qué ser válidos todos para la carretilla, pues algunos se apoyan en métodos de guiado por imágenes o por sónares, y, por ahora, la carretilla no posee ni cámaras ni sensores de este tipo, también un directorio del programa de grabar trayectorias a partir de los archivos de log, un directorio de caminos a seguir para el algoritmo de persecución pura y, como ya se ha dicho, un directorio para los archivos de registro, de log.

7.3.- Elementos programados.

7.3.1.- Driver de la AX10410.

El driver de la AX10410 tiene dos versiones, la diferencia entre una versión y otra, es que en una se escribe por la pantalla, se usa la salida estándar del sistema, aunque no sea como producto de la salida estándar en sí, sino como producto de mensajes propios del núcleo, y en la otra versión no se escribe. La versión que escribe es útil en el proceso de depuración, pues podemos ver cuando se escribe algo en el dispositivo, pero luego no es aconsejable, pues es conocido que la entrada / salida retrasa mucho los programas y hace que el programa entero no funcione bien, por eso se tiene la otra versión. Los archivos en C se llaman ax10410.c y ax10410.h.

En el driver, escrito en C, de lo que se trata es de dar acceso a los usuarios a los registros de la tarjeta AX10410, permitiendo utilizarla y configurarla. En realidad, este driver es una minúscula modificación del driver implementado para la tarjeta AX5411, que va montada en el vehículo Romeo 4R, ya que esta tarjeta es una versión muy similar a la que se posee para la carretilla, de hecho, aparte de la diferencia principal, que es que es una tarjeta ISA y no PC104, diferencia que no tiene ninguna importancia a la hora de programarla, sólo se diferencian estas dos tarjetas en un par de registros, con

lo que nos hemos limitado a copiar el driver, cambiando la dirección base de la tarjeta, pues no tienen la misma dirección base las dos tarjetas en cada ordenador, y cambiar los registros necesarios, en sus direcciones base y en la forma, pero son cambios muy leves. De hecho, remitiremos al lector al proyecto de Rafael Martín de Agar (año 2002) para conocer más datos sobre este driver.

La dirección base de la AX10410 es la 2C0 hexadecimal, y el nombre del dispositivo se cambia a ax10410. Respecto a los registros, el registro gain control pasa del número 1 en la AX5411 al número 11 en la AX10410. Cambian también los registros de los convertidores digital / analógico. En la AX5411, se tiene el primer registro correspondiente a cada convertidor con los cuatro bits menos significativos del valor digital a convertir en el nibble (cuatro bits, igual que un byte son ocho) más significativo de este registro, y los ocho bits más significativos en el segundo registro, mientras que en la AX10410 tiene el primer byte del valor digital en el primer registro, y los cuatro bits que quedan están en el nibble menos significativo del segundo registro. Se habrán, pues, de realizar las distintas transformaciones de bits para adaptar ese valor a los registros del conversor. Las entradas / salidas digitales en la AX5411 están en los registros 10 y 11, pues son 16 en vez de las ocho de la AX10410, ubicadas en el registro 3. En un anexo se puede ver el código del driver.

7.3.2.- La clase AX.

Esta clase provee del conjunto de procedimientos que permiten utilizar la AX10410, y se encuentra en los archivos ax10410.cpp y ax10410.hpp.

Se tienen funciones para abrir la tarjeta y para cerrarla, esto es necesario para poder utilizar un dispositivo, pues ya hemos dicho que Linux trata los dispositivos como ficheros.

Se tienen también funciones para incrementar y decrementar el potenciómetro digital, escribiendo en las salidas digitales de manera correcta, mandando pulsos, en las que la señal CS del potenciómetro se corresponde con la señal digital de salida de la tarjeta D3, la señal INC, con la D1, y la señal U/D, con la D2. Seleccionamos el chip, luego incrementar o decrementar, y luego, por D1, sacamos un tren de pulsos que en cada flanco de bajada incrementa o decrementa el potenciómetro.

Hay funciones para arrancar y parar la carretilla, en ellas se incrementa o decrementa el potenciómetro, según convenga, pero eso no es todo, ya que se ha de

activar la señal del relé que activa la tracción, esta señal es la salida digital D4. En esta activación se ha de tener cuidado, pues es crucial para que el controlador interno de la carretilla actúe de manera correcta la manera de activar este relé. En el caso de arrancar la carretilla, se activa este relé, y luego se incrementa el potenciómetro con los pulsos que se deseen según la velocidad a la que se quiere poner la carretilla, con la señal de incremento activada, para que suba el potenciómetro, siempre con el chip select del dispositivo en nivel bajo, que es como se selecciona el dispositivo. Se ha de guardar este valor de incremento del potenciómetro al llamar a este procedimiento, pues es la única manera que tenemos de saber cuánto se ha incrementado, para luego, a la hora de parar, decrementar el potenciómetro el mismo número de pulsos que para incrementarlo. Si lo que deseamos es decrementarlo, la secuencia es algo distinta, y es distinta porque haciendo lo que parece lógico, que es realizar lo anteriormente narrado, pero con la señal de incremento / decremento en nivel bajo, no funciona. Así, lo que se hace imita un poco más, si cabe, el funcionamiento de la carretilla cuando se pisa el pedal del acelerador, así, se decrementa el potenciómetro tres pulsos, luego se activa la señal de D4, y luego se incrementan los pulsos necesarios hasta la velocidad que se quiere alcanzar.

Para parar marcha adelante y marcha atrás, sólo hay que hacer el proceso inverso al de arrancar, en cada uno de los casos, marcha adelante, primero se decrementa tantos pulsos como antes se incrementó, y luego se deselecciona la señal D4. En el caso de parar marcha atrás, hay que incrementar los pulsos antes decrementados menos tres, luego se deselecciona D4, y luego se incrementan los tres pulsos que quedan. Al lector puede llamarle la atención que se diga que se incrementan tres pulsos, viendo luego que en el procedimiento aparece un 6, se incrementan la mitad, pues el 6 sirve para subir y bajar el potenciómetro. Por último, respecto a esto, decir que el número que marca la velocidad ha de ser impar, si no, por razones totalmente desconocidas, pero que tienen que ver con la utilización del puerto serie a la vez que el dispositivo, no funciona bien, y no sabemos explicar el por qué.

Existe una función que centra el potenciómetro. Esto se ha de hacer cada vez que se apague la alimentación del potenciómetro, y siempre con la carretilla apagada, si no, no obedecerá los dictados del dispositivo. El procedimiento de centrado es sencillo, se baja 100 (con un 200, por el motivo anterior) pasos el potenciómetro, que esté donde esté se quedará en su punto más bajo, y luego se sube 47 (con un 94), porque es así como se queda en el medio, debería ser en 50, pero como depende del controlador

interno a la carretilla, puede ser que pequeños problemas de tolerancias influyan. Si se prueba y falla, puede ser esta la causa, se puede tocar ese valor de 47 (94) a un valor del entorno hasta que funcione.

Esta clase proporciona una función para girar las ruedas, en ella se activan dos señales digitales, la D5, que activa el relé que enciende la bomba del circuito hidráulico, y la D6, que activa el relé que proporciona alimentación a las válvulas, de esta forma, si no se usan las válvulas, se apagan, se ahorra batería y se permite un reseteo de las mismas, ya que se pueden quedar bloqueadas por entradas no previstas y, hasta que no se apagan y se encienden de nuevo, no se desbloquean. Para actuar sobre la entrada de la válvula de dirección, se usa la salida analógica 1, que saca un voltaje que se multiplica por dos en el circuito de adaptación de señales, para que la válvula acepte este voltaje, ha de estar en el rango entre 6 y 18 voltios, con 12 voltios en el medio, yantes de encender las válvulas ha de haber un valor en ese rango en la entrada de las mismas, por lo tanto, si multiplicamos el voltaje por dos, tenemos que en el centro, con la válvula inmóvil, hemos de sacar 6 voltios de la tarjeta, y moveremos a derecha o izquierda moviendo el posible valor de 6 voltios, siempre sin pasarnos de 9, y superando siempre los 3 voltios. Esto se asegura en la función, si el valor que le damos no está en ese rango, no actúa y devuelve un error, pero de otra forma, ya que lo que se hace es mandarle a la función un valor entre -3 y 3, y a eso se le suman 6 antes de actuar, lo que se comprueba es que el valor que se le ha mandado esté en ese rango. Con una entrada negativa gira a la derecha, y con una positiva, a la izquierda. Tras esto, se realiza la conversión a un valor en todo el rango del conversor, y se escribe en los registros correspondientes.

Para subir y bajar las palas se tienen dos funciones, funcionan como las de girar rueda, primero se pone un valor de 6 en la salida analógica 0, luego se activan D5 y D6, y posteriormente se actúa con un valor distinto de 6, convertido al rango del conversor D/A. También lo podremos ver en un anexo.

7.3.3.- La clase Robosense.

Esta clase se puede encontrar en los archivos robosense.cpp y robosense.h.

En esta clase se proporcionan los elementos necesarios para obtener datos del sensor Robosense, hereda de la clase Puerto_serie, para poder acceder al puerto serie. Esta clase se ha programado basándose en la clase que el Romeo 4R posee para

comunicarse con un dispositivo de posicionamiento global (GPS), pues, aparte del paralelismo entre la función del GPS en un entorno global y la del Robosense en un entorno balizado, comparten también similitud en el modo de acceso, mediante el puerto serie.

Se tiene una función pública de inicio del Robosense, que inicia el puerto serie con las condiciones necesarias para establecer la comunicación con el láser, es decir, 19200 bits por segundo, un bit de stop y sin paridad.

Una función que manda comandos al Robosense que crea los paquetes ya explicados en el capítulo dedicado al Robosense, teniendo en cuenta caracteres especiales y demás consideraciones como el checksum ya apuntadas en esta memoria. Esta función utiliza la función para enviar una cadena al Robosense, que mapea la función de la clase Puerto_serie para enviar cadenas por el puerto, servidumbres de la estructura de capas que tan buen resultado ha dado a la organización del programa.

Necesitamos una función que lea datos del puerto serie procedentes del sensor, esta función existe, y llama a otra que calcula una marca de tiempo para cada mensaje recibido, esto es de capital importancia para asignarle tiempos a los datos recibidos (calc_time_mark_robosense). Esa función se apoya en otra que lee del puerto serie carácter a carácter, ese carácter es procesado, pues puede ser un carácter especial, una cabecera o un checksum que se ha de comprobar, y si el mensaje es válido, se procesa.

El procesado del que hablamos no es otro que la obtención de resultados del láser, mediante otra función que interpreta el mensaje recibido según el campo que identifica cada mensaje, y trata los datos según corresponda, realizando las conversiones necesarias para que los datos signifiquen algo, pues pueden venir en varios octetos que han de ser ordenados para que representen un número entero. Aquí las operaciones de desplazamiento de bits tienen un papel crucial, así como transformaciones de unidades, de milirradianes a radianes, o de milímetros a metros, así como las conversiones de tipo entero a flotante, que tantos problemas suelen dar y aquí no fueron menos.

Por último, hablaremos de funciones que son consecuencia de la posibilidad de encapsular los datos que tiene el C++, estamos hablando de funciones necesarias para obtener los datos que el sensor proporciona, y que no pueden obtenerse en procedimientos que no pertenecen a la clase Robosense, pues estos datos son privados. Nos referimos a funciones para conocer la marca de tiempo (get_time_mark_robosense), la coordenada x en el mapa hecho por el Robosense (get_x_robosense), la coordenada y (get_y_robosense), el ángulo con el eje x del mapa

construido, llamado en el programa teta (get_teta_robosense), y el nivel de confianza en el dato recibido que el sensor proporciona (get_confianza_robosense).

Podremos ver esta clase en el anexo dedicado al código del programa.

7.3.4.- La clase Romeo.

La clase Romeo se encuentra definida en los archivos romeo.cpp y romeo.hpp.

En primer lugar, y tras definir las variables que necesitaremos, se realiza la inicialización del vehículo en el procedimiento init_romeo, entendida ésta como la inicialización de los distintos dispositivos conectados al vehículo. Para nuestro caso, sólo son dos, la AX10410 y el Robosense. Se pueden ver más inicializaciones en el código, pero éstas pertenecen a dispositivos del Romeo 4R, no de la carretilla. Las inicializaciones se realizan llamando a las funciones de inicialización de las distintas clases dedicadas a cada dispositivo, de las que hereda la clase Romeo.

Se tiene un procedimiento llamado end_romeo, que fija las referencias de velocidad y curvatura a cero, para detener el vehículo, y que cierra los dispositivos susceptibles de ser cerrados, como la AX10410, también usando funciones miembro de las clases de las que hereda.

Posteriormente, la clase se divide en funciones dedicadas a cada uno de los dispositivos, huelga decir que sólo nos dedicaremos a las de los dispositivos pertenecientes a la carretilla, así como a otras funciones no dependientes de dispositivo que se utilizan y que están definidas en esta clase.

Para el Robosense, se tienen varias funciones privadas, encargadas de rellenar la estructura de datos de tipo ROBOSENSE_DATA, en la que podemos encontrar todos los datos medidos en un cierto instante de tiempo por el sensor Robosense, como x e y en el mapa, orientación y nivel de confianza, así como otros parámetros que se pueden inferir de estos, o realizando operaciones con los datos anteriores y estos, así, tenemos también la velocidad y la curvatura en cada instante medido y variables auxiliares para conseguirlas, como la distancia recorrida desde la llegada del último dato, y la variación de la orientación experimentada desde el último dato, todo ello acompañado por una marca de tiempo. Estos valores se guardan en una cola circular de cinco (aunque el tamaño puede variar) datos tipo ROBOSENSE_DATA, adecuada para recuperar datos de instantes pasados, necesarios para ciertos cálculos.

En cuanto a las funciones públicas del Robosense, tenemos una función para iniciarlo, que realiza las tareas de inicialización necesarias, procurando que el Robosense conteste a una orden de parada, y si no, da error. También tenemos la función update_robosense, que rellena la estructura de datos anterior utilizando las funciones privadas explicadas anteriormente, una función para obtener los datos privados del Robosense denominada get_robosense_data, que los coloca en el lugar adecuado de la cola circular, y una función que actualiza una variable privada bandera de nuevos datos en el sensor (new_robosense_data_flag), que se llama new_robosense_data.

Respecto a la AX10410, tenemos una función llamada set_ax_speed, que arranca la carretilla marcha adelante o marcha atrás, dependiendo del valor de velocidad, si es positivo o negativo, así como detiene la carretilla, si está en movimiento y la velocidad que se le manda es nula. Esto lo puede hacer porque guarda la velocidad que antes se le ha aplicado a la carretilla, para mover el potenciómetro en sentido contrario los mismos pulsos que se movió para arrancar el robot. También se tiene una función para determinar la curvatura a aplicar a la carretilla, denominada set_axt_curv, que sólo mapea la función miembro pública de la clase AX girar_rueda, para mantener la estructura del programa. Esto se acompaña por un par de funciones para obtener la referencia de velocidad y curvatura de la carretilla, get_ref_ax_speed y get_ref_ax_curv.

Pero no sólo se tienen funciones dedicadas a los distintos dispositivos, sino que también se tienen otras funciones destinadas a procesar los datos recibidos, así como a utilizarlos para el control de la dirección de la carretilla. De este modo, la función update_estimacion calcula el tiempo actual al realizar la nueva iteración de esta función mediante otro procedimiento de la clase, la función get_relative_time, que hereda de clases dedicadas a temporización, luego, comprobamos si hay nuevos datos de los dispositivos conectados, en nuestro caso, sólo del Robosense. Esto lo sabemos si está activada la bandera de nuevos datos del Robosense, de la que ya hemos hablado. Así, actualizamos los datos de la iteración anterior en la estructura de datos de tipo ESTIMATED con los nuevos datos del sensor, esta estructura tiene campos de x, y, orientación, curvatura, velocidad, confianza y marca de tiempo, siempre que sean coherentes, es decir, siempre que no superen un cierto umbral predicho por la función de odometría del programa, en la que se usa esta técnica para conocer los próximos valores de los parámetros de situación, ángulo, curvatura y velocidad de la carretilla, conociendo los antiguos. Si los nuevos datos superan a los predichos en un cierto

umbral, estos no se actualizan. De hecho, el bucle de la estimación es mucho más rápido (concretamente, de cuatro a cinco veces) que la actualización que realiza el láser, con lo que hay muchas iteraciones de este bucle en las que no se tienen datos nuevos del láser y se han de obtener mediante odometría. De hecho, en el caso de la curvatra, la actualización de la estructura de tipo ESTIMATED no se hace de manera inmediata al superar el umbral, sino que ésta se sustituye si no supera en un cierto umbral a la media de las cinco últimas estimaciones de curvatura, si lo hace, se actualiza el valor con esa curvatura media, si no, se actualiza con la media de las cuatro anteriores. Esto se puede conseguir porque se tiene una cola de cinco valores de tipo ESTIMATED. Con esto, filtramos los valores de curvatura obtenidos, pues, al ser inferidos de datos medidos por el Robosense, pueden darse datos realmente dispares, muy contraproducentes para el control de la curvatura de las ruedas mediante el PID. Esto hace que la curvatura se actualice muy lentamente, con lo que no se puede realizar un guiado en bucle cerrado de la carretilla nada más que para movimientos de una grandísima sencillez, como conseguir que la carretilla se mueva con una curvatura, partiendo de otra, de seguir trayectorias en algoritmos de persecución pura no podemos ni hablar. Cabe pensar en usar directamente la estructura de datos del sensor para esto, pero utilizando estas otras funciones y estructuras de datos, abrimos la posibilidad de incorporar nuevos dispositivos a la estimación de posición, y elegir entre las mejores medidas de ellos, o bien actualizar los datos con unos menos precisos pero más rápidos (con más medidas) con los datos más fiables de otros sensores más lentos, que es, a grosso modo, lo que se hace en el Romeo 4R. Cabe destacar, que para poder estimar la curvatura, la carretilla se ha de mover, si no, el denominador de la división entre la variación del ángulo y la distancia recorrida es nulo, y se producen indeterminaciones o infinitos como consecuencia de obtener la curvatura de manera indirecta.

En cuanto a las funciones que utilizan estos datos, destacamos la función del PID de la dirección, update_PID_direccion, en la que se recibe una referencia del algoritmo de control de la carretilla, y se intenta alcanzar mediante un algoritmo PID comparándola con el valor de curvatura medida por el sensor láser. Con el valor que se obtiene, se actúa en la dirección. Se para la función para conseguir un periodo de actuación relativamente constante de cinco milisegundos, y luego, se ejecuta el algoritmo con constantes para el PID de $k_r = 0.11$, $t_i = 15$ y $t_d = 0.3$. Estas constantes se han ajustado siempre en pruebas de campo, no en el laboratorio, pues necesitamos que la carretilla se mueva para conocer la curvatura.

Se puede encontrar más información sobre esta forma de programar en el proyecto del software de control del Romeo 4R, así como el código del programa en un anexo a este proyecto.

7.3.5.- Los hilos.

En nuestro caso, se tienen tres hilos en el programa, se puede ver su código en los archivos hilo_robosense.cpp, hilo_robosense.hpp, hilo_pid_dirección.cpp, hilo_pid_dirección.hpp, hilo_estimacion.cpp e hilo_estimacion.hpp.

Los tres hilos realizan más o menos lo mismo, y es repetir en bucle las funciones update_robosense, update_PID_direccion y update_estimacion, respectivamente. Ya hemos visto lo que hacen estas funciones, así que, de esta forma, separamos la actualización de los valores del sensor, del valor de la estimación y del valor generado por el PID para la dirección de la ejecución del programa principal. En cada caso, se tienen tres funciones, una para crear el hilo, otra para destruirlo y otra que incluye el código que se ejecuta al lanzarse el hilo, que no es más que un bucle infinito que repite cada una de las funciones anteriores desde el lanzamiento a la destrucción del hilo correspondiente.

Los tres hilos se lanzan en el main en el orden correcto que ahora veremos, y luego, al finalizar el programa, se eliminan, también en el orden correcto.

Podemos encontrar mucha más información de los hilos en los mismos lugares que venimos recomendando en este capítulo, y el código, en el anexo.

7.3.6.- El main.

Está definido en el archivo main.cpp del directorio E16.

Se comienza con una función manejadora de la señal SIGINT, equivalente a pulsar control + C, que lo que hace es interrumpir el programa, parar los hilos lanzados y parar el vehículo y los sensores activos.

Posteriormente se incluye una función de monitorización de la variable estimated, que es, prácticamente como monitorizar los datos obtenidos por el Robosense, estos datos se van guardando en un fichero de extensión .log que se gurda en el directorio log destinado a eso, con nombre numérico relacionado con la fecha y hora del inicio de la prueba.

Después se lanza el hilo del Robosense, para poder iniciar el flujo de datos con el sensor, mediante las órdenes que se mandan al láser 21 y 23, para que navegue en el mapa anteriormente creado. El hilo ha de estar activo, pues si no, no se pueden recibir los datos del sensor.

Tras esto, se arranca la carretilla, con un número impar, como ya se ha dicho, usando la función de la clase Romeo que arranca. Así, estamos en condiciones de lanzar el hilo del PID de la dirección y el hilo de la estimación, que leerá los datos del Robosense y los actualizará.

Pero debemos obtener la referencia de curvatura para el PID, esto se realiza mediante un algoritmo, por ejemplo, el de persecución pura, aunque esto no se puede realizar por la excesiva lentitud en la realimentación de la curvatura, de la que ya hemos hablado. Así que la referencia queda fija en un número, y podemos ver que el programa realizado funciona, podemos ver como, al cabo del tiempo y tras partir con una cierta curvatura aleatoria, el robot es capaz de ponerse recto, con curvatura cero, pues es esta curvatura la que se le ha indicado de manera continua al PID como referencia.

Se lanza en este momento la función de monitorización, una vez por iteración del bucle de control.

Se tienen mecanismos software de protección, para que no se puedan poner referencias de curvatura mayores de un cierto valor.

Tras estos bloques, y, bien porque se ha llegado al final del algoritmo, bien porque se pulsó control + C, se para el programa. Para parar el programa, se manda parar a la carretilla, mandándole velocidad cero, se manda parar al Robosense, mandándole la orden 12 y se destruyen los hilos de estimación, Robosense y PID de dirección, en este orden. Luego, se usa end_romeo para terminar el romeo y cerrar los dispositivos abiertos de manera correcta. Para más información, consultar el proyecto del Romeo 4R.

7.3.7.- Otros programas realizados.

Ahora hablaremos del programa que se realizó para imitar al proporcionado por los fabricantes del Robosense, en él se pueden mandar comandos y recibir resultados por pantalla de los datos mandados por el sensor. Se parece mucho en su interfaz al programa robodemo.exe, en el que se mandan órdenes mediante comandos numéricos tecleados, correspondientes a los códigos de cada orden. No tiene todas las órdenes

implementadas, pero sí las más importantes. En realidad, es una pequeña variación del código empleado para el programa de control, en el que hay un hilo para recibir los datos y asentimientos del Robosense, mientras que se permite mandar comandos al sensor. El programa se para, esperando un nuevo comando, mientras el hilo espera la respuesta del láser.

Este programa es necesario en la ejecución de las pruebas, pues es con él con el que se le ordena a la carretilla que haga el mapa en el que luego navegará, mediante la orden 14. Luego, podemos salirnos de este programa y el mapa queda guardado en el ordenador del sensor. Arrancamos el programa principal y podemos recibir datos del sensor para navegar en el mapa creado. Siempre que nos movamos en el mapa sin que el sensor esté en modo navegación, hemos de volver a crear el mapa, pues la recolocación ciega de posición no está implementada.

Hemos de añadir que, a pesar de no tener todas las órdenes operativas, funciona mejor que el robodemo.exe (se cuelga menos), probablemente por estar programado bajo Linux, que es realmente multitarea y soporta los hilos de manera real.

Capítulo 8: Conclusiones y líneas de desarrollo futuras.

8.1.- Conclusiones.

8.1.1.- Pruebas realizadas.

Para evaluar el comportamiento de la carretilla, nos remitiremos a las pruebas que hemos realizado con ella. Muchos de estos experimentos estaban encaminados a la calibración y prueba de funcionamiento de los distintos elementos del programa, ya que había muchos elementos, como el Robosense y el programa de control, que no podían probarse en la total plenitud de sus funciones en el laboratorio.

Para realizar las pruebas se ha de balizar el entorno en el que se van a realizar. Lo ideal es una estructura regular de balizas, pero el patio en el que se realizaron no permite una estructuración regular de las balizas. Para su colocación, se adhirieron a la pared mediante cinta aislante. Se ha de destacar la poca durabilidad de esta forma de sujeción, pero así se pueden mover y no se tiene que perforar la pared. En un futuro se aconseja clavar cuatro alcayatas por baliza, las dos de abajo, mirando hacia arriba, y las dos de arriba, mirando hacia abajo, para no sólo sujetar cada blanco, sino también para que éste no se mueva por efecto del viento. Otra opción es pegarlo a la pared con una cinta adhesiva a doble cara para moqueta o para cristal, las dos tienen un pegamento fuerte, más la de cristal, pero puede que no se pudieran despegar de manera sencilla, y la lluvia podría derribar las balizas. Además, no es muy aconsejable dejar las balizas en el lugar de pruebas, pues es un lugar de público acceso y una baliza queda muy bien decorando una estantería. Por eso, podríamos probar a sujetar las balizas mediante un sistema que permitiera dejar la "instalación" fija en el lugar de prueba, llevándose las balizas. Proponemos la prueba de cinta con velcro, poniéndose un tira en la baliza, y otra en la pared, de esta forma, sólo habría que llegar, pegar la baliza y, al acabar, despegarla e irse, dejando la tira en la pared hasta la hora de un nuevo experimento. Este método solucionaría la gran cantidad de tiempo que se ha de emplear en la preparación

del entorno para las pruebas, y la necesidad de más de una persona, pues las balizas se han de poner a una altura considerable. Se desea llamar la atención sobre este aspecto en concreto, que creemos hay que mejorar, y menos mal que las baterías duran bastante y luego las pruebas son largas, si no, habría que tardar entre media y una hora en preparar una prueba de cuarenta minutos. A todo esto hay que unir la dificultad para sacar la carretilla del laboratorio, pues la portada de acceso suele estar bloqueada por coches estacionados delante.

Pero esto no es todo. El lugar elegido para las pruebas no es el más apropiado, aunque creemos que no hay otro con posibilidad de colgar las balizas y sin mucho tránsito de personas que molesten. El caso es que en ese lugar existen tres depósitos cilíndricos plateados que ejercen de descomunales blancos para nuestro sensor. Esto no tendría mucha importancia si siempre fueran detectados, pero esto no ocurre siempre, dando lugar a medidas o mapeos defectuosos. Además, la existencia de estos blancos puede cambiar el mapa, despistando al probador de la carretilla, puede que se haya realizado un mapeo correcto teniendo en cuenta estos blancos espurios, pero las medidas no le parecen coherentes al autor de las pruebas porque no cuenta con ellos. Para evitarlo se propone forrar los depósitos de negro en una tira lo suficientemente ancha a la altura del Robosense.

Para la realización de las pruebas, se ha de seguir un sencillo protocolo. Con la carretilla apagada y el Robosense encendido, se ejecuta el programa similar al robodemo.exe que se posee en el directorio dedicado a este programa en /home/eduardo/carretilla. Se le da al sensor la orden 14 de aprender el mapa. Cuando acaba, se ejecuta el programa de control. Cuando se indica que el potenciómetro está centrado y la carretilla inicializada, y sólo entonces, se enciende el vehículo. Así, el vehículo estará preparado para moverse. Se para con control + C o cuando ha llegado al final del camino, al acabar el algoritmo, como esto último no se da, pues no es capaz de seguir caminos, hemos de pararlo con control + C. Si esta orden no funciona, se aconseja parar la carretilla con la seta de emergencia, usar control + 4, que sale del programa pero no lo termina, luego, volver a ejecutar el programa de control, que hace que todo se vuelva a parar, poniendo el potenciómetro en su posición centrada. Tras esto, ejecutar el programa como el robodemo.exe y mandar un 12 para que se pare. Hay que estar atento a la ejecución del programa, pues en ciertos casos requiere el pulsado de una tecla para continuar, esto es consecuencia de la necesidad de tiempo para recibir datos del Robosense, de la necesidad de tiempo para encender la carretilla tras el

centrado del potenciómetro y de que nos hallamos, no hay que olvidarlo, ante un programa de prueba.

Para las pruebas se utilizó en primera instancia un terminal serie por uno de los puertos RS – 232 del ordenador de control, facilidad que da Linux en uno de sus programas de configuración y herencia del UNIX, que permitía acceder al ordenador desde terminales serie "tontos", con un ordenador portátil en el que corría un programa emulador de terminal por el puerto serie. Pero esta forma interfería en el funcionamiento de los potenciómetros, produciendo descontrol en la carretilla. Luego se probó a acceder al ordenador de la carretilla desde el portátil, vía telnet, con el programa putty y el asunto mejoró, pero no se logró que funcionara bien hasta que no se usaron números impares para la velocidad, como ya se ha expuesto. El motivo, el Robosense también usa el puerto serie, y seguía interfiriendo de manera inexplicable en el funcionamiento del potenciómetro. Con esto y la colocación de un condensador topológicamente cercano a los chips de los potenciómetros, entre alimentación y tierra se desterró ese problema. Pero la conexión vía telnet mejoró también las posibilidades de manejo, al poder usar sin problemas las teclas control + C y tener un juego de caracteres y colores similar al que el Linux presenta en una pantalla CRT normal, aumentándose la velocidad de reacción a las pulsaciones de teclas.

Pero antes de todo esto, se ha de cargar el driver de la AX10410 para poderla usar. No se aconseja usar la orden insmod, pues si se usa y luego no se accede de alguna forma al driver, también de manera inexplicable se pierde la tarjeta de red, con lo que difícilmente podremos acceder al ordenador vía telnet. Por eso se hizo el script Linux driver_sin_es, en /home/eduardo/carretilla/E16, que ejecutado carga el driver y accede a él de una única vez, logrando que no se pierda la conexión telnet. La conexión con un ordenador portátil solucionó la inexistencia de pantalla para el ordenador de control, desde la que lanzar los programas y ver los resultados. El telnet carga algo al procesador, pero no demasiado y permite correr bien el programa de control. La conexión entre los dos ordenadores se realizó de manera física de dos formas, una, directa, con un cable de red cruzado entre los dos ordenadores, y la otra, mediante un hub inalámbrico del que se dispone en el laboratorio, y que nos permite usar el ordenador de control sin estar montados físicamente en la carretilla, como se puede ver en el vídeo demostrativo.

En las pruebas, se intentó calibrar el PID de control de la dirección, así como mejorar los datos del Robosense, para que fueran uniformes. Tras muchas pruebas se

logró el sistema de filtrado que ya se ha explicado en capítulos anteriores, que funciona correctamente, pero es muy lento y no permite la ejecución de un algoritmo de seguimiento de caminos tipo persecución pura. Lo más que se obtuvo es que el sistema de dirección controlara, lográndose que la carretilla centrara las ruedas tras partir con ellas con una cierta curvatura, pues era esa la referencia que se nos había dado. Esto puede parecer poco, pero con los sensores de los que se disponía, parece suficiente y demuestra que el hardware diseñado y el software programado funcionan correctamente. Todo esto, siempre con la carretilla en movimiento, pues si no, no se pueden realizar medidas de curvatura con el Robosense para realimentarlas y controlar la dirección.

Como vídeo de demostración, se hizo un guiado en bucle abierto del vehículo, ya que en bucle cerrado no se pudo, muy similar en concepto al que se había filmado tiempo antes para la presentación ante la comisión de control del proyecto, pero con una mayor fiabilidad que en aquella prueba (funcionó de milagro y no fue repetible), y con todos los elementos de cableado en su sitio. Se observó que en el caso de no utilizar conexiones con los puertos serie (en este caso no eran necesarias, pues no se usaba el Robosense y el acceso al ordenador era vía telnet), los números de pulsos para la velocidad con el potenciómetro habían de ser pares para que funcionara el vehículo. Inexplicable, pero cierto.

8.1.2.- Balance de lo realizado.

Podemos ver en esta memoria que el trabajo realizado ha sido muy amplio y con muchas dificultades, pero esto era de esperar, pues uno de los propósitos del mismo era introducirse en el funcionamiento interno de un vehículo industrial, y es conocida la prácticamente nula disposición que tienen los fabricantes de estos vehículos a sacar a la luz los entresijos del funcionamiento de sus equipos. Han existido también problemas debido a la dificultad de manejo de un vehículo tan pesado como el que nos ocupa, que se han traducido en demoras en el tiempo de ejecución de las tareas. Otros muchos problemas como fallos, funcionamientos inadecuados y averías han dado lugar a mayores retrasos, pero como contrapartida nos han permitido obtener un mayor conocimiento de los dispositivos averiados, preparándonos para resolver esos problemas de manera autónoma si se vuelven a producir. Estos fallos también han permitido internarse en problemas que, en principio, no estaban previstos, permitiendo al

proyectando, de esta manera, aprender interesantes aspectos del funcionamiento de muchos dispositivos que, en la actualidad, están muy arraigados en los entornos de la robótica y la informática industrial.

Pero, a pesar de las dificultades, se ha logrado realizar un trabajo con unos resultados claros que se espera que puedan ser continuados. Porque no hemos de engañarnos, a pesar de todo lo hecho, este proyecto no acaba más que de comenzar. Esperamos que no se pare, siguiendo las líneas que en este capítulo se trazan u otras distintas. También esperamos que esta memoria sirva en alguno de sus aspectos a los que seguirán con el trabajo para aclararles puntos del funcionamiento del vehículo y para ahorrarles trabajo.

Se tiene, además, la esperanza de que una gran parte del trabajo desarrollado en la carretilla, en los circuitos, en el software del ordenador y en campos no tan tangibles como la documentación obtenida (se ha realizado un vasto trabajo de documentación, reflejado en multitud de documentos sobre los elementos de que se dispone actualmente en la carretilla y los que próximamente se obtendrán, como los sensores, documentos en forma de papel y en formato electrónico), de la que se puede ver una parte en el anexo segundo de la memoria, sean útiles y no haya necesidad de sustituir mucho de lo ya montado.

En resumen, se ha empezado un camino y esperamos que sea continuado para que la carretilla pueda llegar a ser un proyecto importante dentro del grupo de Robótica, Visión y Control.

A título personal, el proyectando ha tenido contacto con conocimientos de robótica, por supuesto, pero también de electrónica, sistema operativo Linux, programación en C++, y otros conocimientos que, de seguro, le serán muy útiles en el futuro. También ha tenido la posibilidad de realizar un trabajo de forma similar a como lo haría en un entorno laboral típico, con un horario y cobrando, así como la posibilidad de entablar contactos con proveedores, con miembros de la asistencia técnica, y otros conocimientos que no se estudian pero que son igualmente importantes.

8.1.3.- Agradecimientos.

Hasta este momento he procurado usar el impersonal o la primera persona del plural en el proyecto, pero ahora voy a usar el singular. Y lo voy a usar para agradecerle

a todos los que han hecho posible la finalización de este proyecto que esto haya sido posible.

En primer lugar, quiero darles las gracias a mis padres, porque todo lo que soy se lo debo ellos en mayor o menor medida. Creo que nunca os lo he dicho, y ya es hora, gracias. También porque dicen que es de bien nacido ser agradecido, y si no se lo agradeciera, les haría un feo doble.

En segundo lugar, a mi tutor, D. Guillermo Heredia Benot, por toda la ayuda y los consejos que me ha dado, así como a D. Aníbal Ollero Baturone, por haber hecho posible la realización de este proyecto.

En tercer lugar, a mis hermanos, familiares y amigos, porque siempre han tenido tiempo para preguntarme cómo lo llevaba y encima esperaban hasta que acabara de contárselo, así como para darme consejos con toda su buena voluntad y sin que yo se los pidiera. Alfonso, qué bien va tu ordenador. Juanju, perdona por no dejarte dormir la siesta.

Ahora voy con los compañeros del laboratorio y del grupo. A todos, gracias por vuestra ayuda. En especial a Rafa "Linux" Martín de Agar, por introducirme en el maravilloso mundo de ese sistema operativo tan bueno, bonito y barato, y estar ahí siempre para arreglar lo que yo estropeaba. También a José Víctor Acevedo, el "Jefe de Canaletas", por toda la ayuda prestada con la electrónica, aunque finalmente no consiguiera convencerme para realizar el circuito impreso, y por ser el cámara en "Carretilla Productions". Por último, a Marcos "Alarkus" del Toro, por todo su "apoyo moral y logístico" y por envidar a grande con tres pitos. Antes les di las gracias a los amigos, estáis en la intersección, junto con otros muchos, entre el conjunto "amigos" y el conjunto "compañeros". De entre los miembros del grupo, quiero destacar a D. Joaquín Ferruz Melero, por saberlo todo, hasta de El Señor de los Anillos y La Guerra de las Galaxias. Sauron y Yoda siguen entre nosotros. Nombraré también a Francisco López Pichaco, el técnico de laboratorio, por acertar siempre y a Alberto González Cantos, por aburrirse con el PID.

Gracias a todas las personas encargadas del soporte técnico de todos los elementos que he utilizado, por aguantar mis preguntas en e – mail y por acceder a mis peticiones de información, en especial a Magda Andrés, de Octagon Systems España, no nos conocemos, pero sin ella no hubiera podido arreglar el láser. Y lo más curioso de la cuestión es que todavía no encuentro motivos para que me ayudara tanto.

Por último, gracias a Dios, por todo lo anterior, y por todo lo demás, gracias.

8.2.- Líneas de desarrollo futuras.

8.2.1.- Cuáles son estas líneas.

Las líneas de las que vamos a hablar corresponden a tres campos principales.

Una es las posibles mejoras de lo ya realizado, aunque funcione bien, siempre es susceptible de mejorar.

Otra será la integración de nuevos sensores en la carretilla, que en este momento, por falta de tiempo (y de sensores) no se ha podido realizar por desgracia.

Por último estableceremos líneas de desarrollo no futuras, sino muy futuras, y orientadas a lo que se piensa que puede ser una aplicación válida y útil en el mundo industrial, que es el mundo en el que realizan su labor estos vehículos.

8.2.2.- Mejoras de lo hecho.

En este apartado veremos que lo hecho se puede mejorar. Cabe una pregunta obvia, si se puede mejorar, ¿por qué no se ha hecho de manera inicial de esta forma? Esta pregunta tiene una respuesta en exigencias de tiempo a la hora de entregar resultados en un plazo fijado, esto obliga a tomar decisiones que pueden resultar más sencillas y rápidas a corto plazo, pero que pueden resultar mejorables en un plazo más largo. Otra consideración importante para explicar la necesidad de mejora de lo ya realizado es la de que la carretilla se encuentra en período de prueba y en estado de prototipo, es decir, no se tiene todavía una concepción entera y total de lo que va a ser el proyecto terminado en el futuro, por lo que hay cosas en un estado de provisionalidad que hacen que pueda resultar una pérdida de tiempo realizarlas de manera definitiva.

Como ejemplo de todo esto, podemos nombrar un caso evidente, como es el hecho de la placa de adaptación de señales, que está realizada en una placa provisional, bien soldada y duradera, pero que no debe ser el formato último de las placas electrónicas, ya que es sabido que son más fiables si se implementan en una placa de circuito impreso, siempre hablando de un montaje definitivo, pues si está sujeto a cambios, es una pérdida de tiempo y dinero hacer una placa de circuito impreso para luego tener que hacer otra totalmente distinta.

Pero, ¿por qué el montaje es provisional? Lo es porque no se han integrado todavía las tarjetas de control de motores, que pueden suponer un cambio total en la

forma de actuar sobre la carretilla. Teniendo en cuenta que se pueden usar estas dos salidas analógicas para el control, puede cambiar la forma de controlarlo, siempre introduciendo nuevos sensores. De todas formas, puede ser necesario, a pesar de introducir estas dos tarjetas, el tener nuevas salidas analógicas, pues podemos actuar sobre el caudal de la bomba del circuito hidráulico de manera analógica en los casos de la elevación y la inclinación del mástil. En este momento, se conmutan las señales mediante relés, pero no es un mecanismo muy acertado por causas que ya se han explicado, así, se aconseja utilizar otra tarjeta más del tipo AX10410, lo que daría dos señales más, o bien utilizar un multiplexor analógico con sample and hold que distribuya las señales por los distintos destinos que las necesiten. Esto implica tener más señales digitales para el control de estos multiplexores, existentes, por otro lado, en el mercado. Pero la existencia de salidas digitales en las tarjetas de control de motores podría solucionar el problema sin necesidad de integrar más tarjetas (como la que se tiene de entradas / salidas digitales), además de eliminar el bucle del PID de control que está programado. Esto es importante, pues integrar una tarjeta no es sólo conectarla en la pila del PC104, es también hacer un driver bajo Linux para la tarjeta, y no está claro que se pudieran conectar muchas más tarjetas juntas al PC104 sin que se interfirieran en su funcionamiento.

También podemos hablar de los potenciómetros digitales, los que se eligieron son, probablemente, los más sencillos en su funcionamiento. Esto es, sin duda, positivo, pero puede que si la elección hubiera sido de potenciómetros más complejos en su funcionamiento, como ciertos dispositivos síncronos, que existen en el mercado, puede que fueran más estables en su funcionamiento, aunque más complejos de utilizar, pues tienen que recibir un reloj, ese reloj hay que generarlo y no se pueden mandar los datos en cualquier momento, sino de manera síncrona, lo cual da lugar a la necesidad de un cierto protocolo que complica la comunicación. También hay potenciómetros digitales que funcionan enviándoles datos en formato BCD, pero esto exige la conversión y el mandar los datos por un número de líneas digitales de las que puede que no se disponga.

El caso es que, debido entre otras cosas a la premura por ciertos resultados, se han elegido estos sencillos potenciómetros. Contraponiéndose a su sencillez, los potenciómetros elegidos no tienen una estabilidad adecuada y varían en su funcionamiento a veces. El problema de esta forma de actuar es que no se actúa de manera directa, sino a través de los potenciómetros, es decir, no se pone un voltaje analógico del que nos podemos fiar, pues es generado por el ordenador, y el sistema de

tracción reacciona moviendo la carretilla a una velocidad más o menos proporcional a esa señal. Lo que hacemos es decirle al potenciómetro que se suba o se baje, de eso nos podemos fiar, pero no podemos fiarnos de que el potenciómetro se suba o se baje correctamente, normalmente lo hace, pero en algunos casos no, y esto se agrava si se producen cambios en la velocidad distintos de arrancar o parar, es decir aceleraciones, que en algoritmos como el de persecución pura no se producen, pues se pone una velocidad fija y se realiza el guiado.

Para contrarrestar esto, alguien podría pensar en realimentar la velocidad del vehículo y mover el potenciómetro en consecuencia, pero esto no es válido, pues se tiene una zona muerta de varios pulsos en los que la carretilla no se mueve, es decir, podemos pensar que la carretilla tiene el potenciómetro en su valor central porque no se mueve, y que esto no sea cierto, que esté quieta con un valor distinto del central.

Lo peor de todos estos errores es que sobre el potenciómetro se actúa de manera incremental, por eso hay que guardar el valor con el que se ha actuado para luego aplicarlo en sentido contrario al parar, y esa manera de actuar incremental acumula errores, por ejemplo, si nos creemos que el potenciómetro está en el centro, pero está más alto del centro, y le ordenamos bajar un poco, para una velocidad pequeña marcha atrás, puede que ni se mueva, y no sólo eso, sino que el sistema de tracción quedará bloqueado y se deberá apagar y encender la carretilla.

Así, para solucionarlo, la realimentación que antes se ha propuesto, se podría hacer, pero teniendo en cuenta que es una realimentación de una señal analógica, que no es otra que el voltaje que se puede medir en el potenciómetro, así sí que se podría conocer exactamente la posición del potenciómetro en cada momento, y centrarlo. Para esto tendríamos que usar una de las señales de entrada analógicas y un bucle de control de lo que sube o baja el potenciómetro, implementado en un hilo distinto del bucle principal.

La otra opción para el potenciómetro es fiarse de él, al fin y al cabo, falla a veces, pero no son tantos fallos, quizá mejorando un poco el programa de actuación se corregiría totalmente el problema. Por otra parte, en principio, e imaginando la carretilla trabajando en un ambiente industrial, parece que no necesitaría cambiar de velocidad, sino solamente pararse y arrancar y esto funciona con mucha corrección.

Otras mejoras de lo hecho se refieren a la carcasa y el montaje de los distintos elementos. La caja provisional en la que se encuentran el ordenador y los circuitos de adaptación de señal debería ser sustituida por una caja para el ordenador y otra para los

circuitos, con la posibilidad de conectar las dos de manera sencilla, pues es sabido que hay intercambio de muchas señales entre ambos elementos. Aunque esta cuestión es puramente estética, no deja de ser importante, pero aconsejamos que se acometa cuando se tenga el circuito montado de manera definitiva. La posición de las baterías, su aislamiento y la posición del ordenador y los circuitos es también una reforma que se debería acometer, teniendo en cuenta que la cabina se levanta y las baterías no se pueden inclinar, pues pueden verter ácido. Se aconseja buscar otra situación o acondicionar la cabina para que se pueda levantar sin que se inclinen las baterías.

En cuanto al programa de control, se han de realizar los drivers de las nuevas tarjetas a incorporar, así como mejorar en la legibilidad del programa y eliminar todos los elementos que no pertenecen a la carretilla y son herencia del Romeo 4R, que se ha preferido conservar sin ejecutarlos por si pueden servir de inspiración al futuro programador ante el buen funcionamiento que ya han demostrado de sobra en las pruebas del Romeo 4R. Por supuesto se han de crear las clases correspondientes a los distintos nuevos dispositivos para continuar con la estructura dispuesta en el programa.

Pero en el campo del software se puede mejorar también introduciendo el parche que permite convertir nuestro sistema Linux en un sistema de tiempo real, y cambiando el programa para aprovechar todas las buenas cualidades que los sistemas operativos en tiempo real tienen para este tipo de aplicaciones.

El Robosense también puede ser objeto de mejora, en primer lugar, implementando comandos que no lo están, y en segundo lugar, aprovechando la facilidad que tiene para navegar en distintos frames, moviéndose por diferentes estancias, lo que representa entornos distintos. Para esto podemos usar el programa sitemap.exe, que se ejecuta bajo MS – DOS, además de implementar algunas órdenes más del protocolo para poder cargar y utilizar los distintos mapas en el programa de control bajo Linux.

8.2.3.- Integración de nuevos sensores.

Aparte de sensores de nivel superior en el procesamiento, como las cámaras que necesitarían casi con toda seguridad el concurso de otro ordenador para el desentrañado de la información de la imagen, o los sónares para evitar obstáculos, que necesitarían de entradas adecuadas a la forma de presentar sus medidas, bien entradas analógicas en el

caso de este tipo de sónares, bien digitales para los sónares todo – nada, a los sensores a los que nos referimos en este apartado es a los encoders encargados de determinar y realimentar la curvatura de las ruedas y la velocidad del vehículo, y hablamos de encoders y no de potenciómetros, por ejemplo, para aprovechar las dos tarjetas de control de motores que se poseen, y que permiten la realimentación por este tipo de sensores.

El problema es que no sólo se tienen esos dos ejes en el movimiento de la carretilla, sino que es necesario controlar otros dos ejes del movimiento de la misma, como son el de la inclinación del mástil y la elevación de las palas. Puede que en el caso de la inclinación se pudiera dejar su control en bucle abierto, realizando un control todo – nada, con sólo dos posiciones, inclinado del todo hacia la cabina para llevar la carga sin que se caiga y colocado totalmente perpendicular al suelo para recoger la carga, cómo conocer que el mástil está perpendicular al suelo, con un fin de carrera podría bastar, así, podemos inclinar el mástil alejándolo de la cabina hasta que salte el fin de carrera, y ahí parar, sabiendo así que tenemos el mástil en perpendicular con el suelo.

Pero esto no es posible con la elevación, pues no se imagina un control en bucle abierto de la altura de las palas, al menos, esto limitaría mucho las posibilidades de la carretilla, por lo que necesitamos otro encoder y otra tarjeta de control de motores para este eje, si usamos las otras dos para curvatura y velocidad, aunque podemos no medir la velocidad con un encoder infiriéndola de la posición del potenciómetro y realizando pruebas que relacionen el voltaje del potenciómetro con la velocidad según una ley o tabla, pero entonces hemos de realimentar el voltaje en el potenciómetro de tracción, como antes dijimos, o fiarnos del potenciómetro como antes nombramos también. Existe otra posibilidad, que es realimentar una señal analógica que tiene la carretilla de velocidad, pero esa señal es la velocidad de la rueda derecha y no es fiable en ciertos casos como velocidades pequeñas o mucha curvatura. La última de las opciones es dejarlo como está ahora, y sacar la velocidad del Robosense, pero no se ha comprobado si esas medidas de velocidad son reales.

Otra opción podría ser la de medir los pulsos del encoder mediante una tarjeta como la que posee contadores, pero entonces habríamos de programar el driver de la misma, así como programar los mecanismos para contar los pulsos y el sentido de los mismos, pues esto no está implementado de manera directa en la tarjeta de timers – contadores como lo está en la de control de motores. El otro inconveniente que esto presenta, es el de que se ha de cerrar el bucle de control del eje en cuestión en el

programa principal, ya que de esta forma sólo podemos leer el valor del encoder con una tarjeta, y actuar con otra tarjeta distinta (con salida analógica) según el valor calculado por un PID que se ejecuta en un hilo del programa principal.

Pero si los encoders son relativamente complejos de leer con un ordenador, lo que no es tan complejo es un sensor de potenciómetro, quizá tiene menos precisión y también obliga a cerrar el bucle en el programa principal, además de un cierto tratamiento de la señal, pero luego es mucho más sencillo de leer por la AX10410, que tiene 16 entradas analógicas y tendríamos el dato directamente, sin ningún tratamiento adicional. Desde luego, y debido a que no es una señal analógica la que se usa para actuar sino un tren de pulsos que sube o baja el potenciómetro, el máximo candidato a utilizar este sensor de potenciómetro es el sistema de tracción, pues las tarjetas de control de motores no están preparadas para esta forma de actuar mandando trenes de pulsos para acelerar y frenar, sino mediante una tensión.

Por último, una consideración importante, y es que los sensores de potenciómetro dan una medida absoluta de la posición en cada momento, y los sensores de encoder, en su gran mayoría, dan el desplazamiento, respecto al punto en el que iniciaron su trabajo, es decir, una medida relativa, y esto último obliga a realizar procesos de centrado nada más encenderlos, llevando cada eje a su extremo para contar los pulsos recibidos. En ciertos casos, esto es importante, porque se fuerzan los motores de giro de la dirección, por ejemplo, pero en este caso, creemos que no es de tanta importancia, pues el sistema de movimiento hidráulico está preparado para mover la carretilla mientras está parada con facilidad, consecuencia de su facilidad de maniobra en espacios pequeños que estos vehículos poseen.

En todo caso, y tras ver las diferentes posibilidades que se presentan, hablaremos un poco sobre los encoders o sensores de potenciómetro que podemos colocar en la carretilla, obviando si se tienen medios en el ordenador de control para su funcionamiento. Sobre sensores de posicionamiento de este estilo se tiene bastante información en forma de distintos documentos y catálogos en el departamento. En ellos se pueden ver las diferentes prestaciones de varios modelos de diversos fabricantes.

En primer lugar, para medir la velocidad, podríamos colocar un sensor rotatorio de posición y velocidad del tipo de los anteriormente descritos. Este sensor giraría solidariamente con una de las ruedas, al menos la pieza móvil del mismo, proporcionando una medida de la velocidad de giro, que puede servir para determinar la velocidad. El problema que puede tener es fijar la parte móvil del sensor a la rueda y la fija al resto de la carretilla, conllevando trabajo de taller. Veamos cómo en la figura 8.1.

Figura 8.1: Una posibilidad de colocar un sensor de velocidad.

Existen otras posibilidades, como un sensor doppler o algún dispositivo similar, pero estos serían más complejos. En la figura podemos observar el esquema de lo que sería un sensor fijo en el guardabarros de la carretilla, y con la parte móvil enganchada en la rueda.

En el caso de la curvatura, podemos colocar un sensor de hilo fijo en algún lugar de la carretilla, y un hilo a una parte no móvil de la rueda (entiéndase no móvil como que no gira, sólo permite que la rueda cambie de dirección). Pero esta solución, muy válida en otros casos, en este no lo es tanto, pues la carretilla es capaz de realizar giros de muy pequeño radio, con lo que los soportes de las ruedas giran tanto que pueden hacer que el cable se doble, imposibilitando esta forma de medida. Lo que aconsejamos es medir de alguna forma el desplazamiento del cilindro que empuja los soportes de las ruedas para cambiar su dirección. Este es el cilindro que es empujado por el fluido hidráulico en el caso de cambio de dirección en la carretilla. Así, podemos colocar un

sensor de hilo amarrado a la carcasa, y medir la variación de la distancia a uno de los extremos del cilindro, que se mueve en línea recta, empujando y tirando de los soportes de las ruedas. Otra opción es usar un sensor de carril, en los que se tiene una parte fija, y el movimiento de la parte móvil por un carril nos permite ver el desplazamiento. En los dos casos hemos de convertir luego la medida a curvatura. Los dos casos tienen el inconveniente de necesitar su montaje en un taller, pues la carretilla no es susceptible de ser perforada para fijar los sensores con las herramientas de que disponemos en el laboratorio. También tienen el inconveniente de la poca accesibilidad del lugar para colocar el sensor, como podemos ver en la figura 8.2.

Figura 8.2: Detalle de la zona de las ruedas de dirección.

Se puede observar el reducido espacio del que se dispone para este cometido, y si utilizamos la solución del sensor de carril, el asunto es aún más complicado, pero en nuestra opinión es más conveniente.

Para el caso del mástil, tenemos las posibles medidas de inclinación del mismo y la elevación de las palas.

La inclinación podría ser medida con un sensor de hilo colocado en la cabina, que es fija, y enganchando el extremo del hilo al mástil, que tiraría del hilo en su movimiento, dando la medida de la inclinación. Esto tiene un inconveniente claro, y es que se ha de desenganchar el hilo en el momento en que la cabina se levante para cargar las baterías u otra tarea de mantenimiento. Luego habría que realizar las convenientes conversiones de estiramiento del hilo a ángulo de giro. Veámoslo en la figura 8.3.

Figura 8.3: Posición del sensor para la inclinación.

La elevación sería medida por un sensor de hilo colocado lo más próximo posible al suelo, pero procurando que sea una parte fija de la carcasa, o aún mejor si se pudiera colocar en el mástil, ya que así haríamos independiente la medida obtenida de la inclinación del mástil. El problema es encontrar un sitio que no interfiera el movimiento

de las palas en el mástil, y que, como en los casos anteriores, exige con casi total seguridad su montaje en un taller, aparte de tener los hilos de los sensores muy expuestos, pudiendo engancharse en cualquier lugar. Veámoslo en la figura 8.4.

Figura 8.4: Posición del sensor para la elevación.

Si el sensor es lo suficientemente pequeño, podríamos fijarlo ahí, pero este lugar es peligroso, pues se ha de tener cuidado de que el eje al inclinarse no destroce el sensor, además hay que ver si con las palas colocadas en el suelo, este método es válido. Por encima se hace difícil colocarlo, porque la cadena lo impide.

Como resumen diremos que en este caso se exige medir los tamaños, elegir los sensores, y colocarlos con cuidado de que no interfieran el movimiento de las partes móviles de la carretilla. Lo anterior es sólo una sugerencia, pero puede haber otras soluciones.

8.2.4.- Aplicaciones más lejanas en el tiempo.

En este caso, podemos hablar de la integración de otros sensores más complejos que los anteriormente citados, como los sónares.

También podemos incluir cámaras de vídeo para realizar el guiado de la carretilla en un entorno, o bien para encontrar e identificar un palé requerido entre los distintos palés de una pila. Las cámaras se pueden utilizar también para encontrar los

agujeros en los que se introducen las palas en la base del palé para recogerlo, enfilando las palas para recoger la carga de manera adecuada. En este caso surge el problema de dónde colocar las cámaras para que no sean dañadas por las palas en su movimiento, o por la carga al ser recogida, una ayuda para resolver este problema puede ser el reducido tamaño que pueden alcanzar las cámaras en la actualidad, teniendo en cuenta que para esta aplicación no se necesitarían, probablemente, cámaras muy precisas. Pero el hecho de introducir cámaras supone procesamiento de imagen, y eso es costoso computacionalmente, tan costoso que es muy probable que no se pueda realizar en el mismo ordenador de control de la carretilla, aunque con la tarjeta PCMCIA que posee admita la conexión de cámaras. Tampoco está claro que existan drivers fiables de dispositivos capturadotes de imagen bajo Linux. De todos modos, existe en el laboratorio otro ordenador en formato PC104 plus (con bus equivalente al bus ISA y con otro bus equivalente al PCI, más avanzado), que está preparado para el tratamiento de imágenes, y que cumple con las condiciones de tamaño que este proyecto necesita, pues, como hemos dicho, es PC104. Además, al existir dos ordenadores distintos, se puede montar Windows en el que trata las imágenes, de manera muy similar a como está dispuesto en el Romeo 4R, en el que un sistema Windows trata las imágenes y se comunica mediante sockets con el sistema Linux de control del vehículo.

Pero para encontrar los palés en la pila no tiene por qué ser necesario usar cámaras, se podría identificar totalmente la posición del palé en orientación, x e y, y, si se tienen sensores de posición suficientemente precisos, recoger el palé teniendo siempre los palés a una altura fija, por ejemplo, siempre a un metro, a dos metros y a tres metros en la pila. Para identificar los palés en este caso, podríamos usar sensores de códigos de barras similares a los que podemos ver en las cajas de los supermercados para pasar el precio a la caja registradora. Así, la carretilla ejecutaría órdenes del tipo "busca el palé número 25 que está en la pila de x = 7 metros, y = 5 metros, orientación = = 1.2 radianes entre los tres que hay, que están exactamente a 0 metros, 1 metro y 2 metros", y la carretilla se colocaría en esa posición, moviendo las palas con un sensor de códigos de barras hasta que encontrara el palé, por ejemplo, a 2 metros, y sólo tendría que poner las ruedas rectas en la dirección indicada y entrar una cierta distancia. Esa distancia puede ser fija, o bien se pueden poner sensores, por ejemplo, de fin de carrera que determinen si el palé allegado ya al tope de las palas para no empujar la pila con la carretilla y derribarla. Todo este proceso exige un entorno muy estructurado, todo tiene que estar en su sitio previsto. Puede que no sea realizable, y, desde luego, esto situaría a

la carretilla más dentro del ámbito de los AGVs que en el de los robots, pero es una solución.

Anteriormente se ha hablado de órdenes a la carretilla, esa puede ser otra línea muy interesante de desarrollo futuro, que permitiría su completa implantación en una factoría industrial, que, además, entronca con una de las líneas de investigación que va a acometer, si no lo está haciendo ya, el grupo de Robótica, Visión y Control. De este modo, podríamos tener un ordenador central encargado de planificar tareas para las carretillas, y serían enviadas a éstas vía radio – módem. Se exigiría tener varias carretillas automatizadas trabajando de manera cooperativa, y la planificación del ordenador central podría tener varios grados de profundidad, desde planificar sólo tareas, hasta planificar también las trayectorias de los robots, evitando colisiones entre ellos. Las carretillas deberían devolver al planificador su posición cada cierto tiempo, y éste decidiría la asignación de tareas y trayectorias por criterios de proximidad u otros criterios. Así, tendríamos una flotilla de carretillas elevadoras actuando como AGVs, pero con sus trayectorias determinadas de manera dinámica, a diferencia de las trayectorias estáticas marcadas de alguna forma en el entorno de los AGVs. Esta línea plantea problemas de comunicaciones (el entorno industrial es muy ruidoso), redes inalámbricas, planificación de tareas mediante distintos criterios, coordinación de distintos vehículos, etcétera. En principio, parece caro, pero la posibilidad de no tener caminos prefijados hace que, probablemente, no fueran necesarios tantos vehículos como en el caso de un sistema de AGVs para realizar la misma tarea con la misma rapidez.

Esto es sólo una línea muy futura, en nuestra opinión, primero hay que tener un solo vehículo funcionando totalmente, y luego, ya veremos. Pero sería bonito, ¿no?

Anexo I: Código del programa de control de la carretilla.

El driver, ax10410.c

```
/*********************************************************************
*
* Diver para tarjeta AX10410 de Axiom
* AUTOR: Rafael Martín de Agar Tirado
* ADAPTADO POR: Eduardo Gomez- Elegido Pedraza
*
* ULTIMA MODIFICACION: 21-VIII-2001
*
*********************************************************************/

/* Estandar en los módulos del kernel */
#include <linux/kernel.h>   /* Estamos trabajando con el kernel */
#include <linux/module.h>   /* Y específicamente con un módulo */

/* Esto no sé si es o no necesario */
/* Deal with CONFIG_MODVERSIONS */
#if CONFIG_MODVERSIONS==1
#define MODVERSIONS
#include <linux/modversions.h>
#endif

/* Para dispositivos de caracteres */
#include <linux/fs.h>      /* Definiciones del dispositivo de caracteres */
#include <linux/wrapper.h>  /* Para compatibilidad con el futuro */

/*********************************************************************
* Especifico de nuestro driver para la AX10410
*********************************************************************/
#include <asm/io.h>        /* para outb(), inb(),... */
#include <linux/ioport.h>  /* para trabajar con los puerto de I/O */
#include <asm/ioctl.h>     /* para las macros _IOR, _IOW, _IOWR */
#include <linux/errno.h>   /* para códigos de error */
#include <linux/delay.h>   /* para udelay() */

/* Interrupciones */
#include <linux/sched.h>
#include <linux/tqueue.h>
#include <linux/interrupt.h>

#include "ax10410.h"

/* Prototipo de las funciones implementadas */
static int write_one(unsigned int port, unsigned int val, int size);
static int read_one(unsigned int port, int size);
static int ax_probe(unsigned int port, unsigned int range);
static int ax_detect(unsigned int port, unsigned int range);
static int ax_init(void);
static int kernel_to_user(byte *kernel_pointer, byte *user_pointer, int size);
static int user_to_kernel(byte *kernel_pointer, byte *user_pointer, int size);

/* Interrupciones */
void irq_handler(int irq,  /* Manejador de interrupciones */
                 void *dev_id,
                 struct pt_regs *regs);
static struct wait_queue *ax_queue = NULL; /* Para interruptible_sleep_on
                                              y wake_up_interruptible */

/*********************************************************************/

/* En 2.2.3 /usr/include/linux/version.h incluye un macro para esto
   pero la 2.0.35 no, así que lo incluimos por si es necesario */
#ifndef KERNEL_VERSION
#define KERNEL_VERSION(a,b,c) ((a)*65536+(b)*256+(c))
#endif

/* Compilación condicional. LINUX_VERSION_CODE es el código de esta versión
   del núcleo */
```

```
#if LINUX_VERSION_CODE > KERNEL_VERSION(2,2,0)
#include <asm/uaccess.h>  /* para put_user */
#endif


/************* Declaraciones del dispositivo ************************/

/* Para evitar accesos concurrentes al dispositivo */
static int Device_Open = 0;

#define SUCCESS 0


/************* Funciones implementadas ******************************/

/*************************************************************************
* ax_probe()
* Comprueba que la tarjeta AX está presente, escribiendo en algunos registros
* y comprobando posteriormente lo que se ha escrito mediante lecturas de los
* mismos.
*************************************************************************/
static int ax_probe(unsigned int port, unsigned int range)
{
/* TEST_DATA_1 --> DRW_MUX_CTRL */
  write_one(AX_PORT+DRW_MUX_CTRL,TEST_DATA_1,BYTE);
  if (read_one(AX_PORT+DRW_MUX_CTRL,BYTE) != TEST_DATA_1)
  {
    //printk("ax_probe(): error recibiendo eco de la tarjeta");
    return -ENODEV;
  }
/* TEST_DATA_2 --> DRW_CTRL */
  write_one(AX_PORT+DRW_CTRL,TEST_DATA_2,BYTE);
  if (read_one(AX_PORT+DRW_CTRL,BYTE) != TEST_DATA_2)
  {
    //printk("ax_probe(): error recibiendo eco de la tarjeta");
    return -ENODEV;
  }
/* Sin error */
  return SUCCESS;
}

/*********************************************************************
* ax_detect()
* Comprueba que la tarjeta AX está presente (mediante una llamada a
* ax_probe()) y comprueba la disponibilidad del espacio de I/O para
* operar con la tarjeta, con llamadas a check_region() y
* request_region().
* Devuelve un número negativo en caso de error o SUCCESS en caso
* contrario.
*********************************************************************/
static int ax_detect(unsigned int port, unsigned int range)
{
  int err;
  if((err = check_region(port,range)) < 0) return err;  /* busy */
  if (ax_probe(port,range) != 0) return -ENODEV;        /* not found */

  request_region(port,range,"ax10410");
  return SUCCESS;
}

/*************************************************************************
* ax_init
* Inicializa la tarjeta AX:
* - Interrupciones
* - Canales inicial / final para conversión A/D
* - Ganancia
* - Registro de control
* Es llamada desde ax_open().
* Devuelve un valor negativo si hay algún error o SUCCESS en caso contrario.
*************************************************************************/
static int ax_init(void)
{
/* Solicitamos la irq AX_IRQ_NUMBER para la AX y le asignamos el manejador de
   interrupciones irq_handler */
//printk("Antes de request_irq\n");
  if(request_irq(AX_IRQ_NUMBER,irq_handler,SA_INTERRUPT,"ax10410",NULL) != 0)
  {
```

```
    //printk("ax_open(): error registrando la IRQ\n");
    return -1;
  }
/* Inicializamos todos los registros a los valores que debería tener el
   dispositivo por defecto al arrancar, por si acaso... */

/* Canales inicial y final para la conversión A/D */
  write_one(AX_PORT+DRW_MUX_CTRL, V_INIT_MUX, BYTE);

/* Ganancia */
  write_one(AX_PORT+DW_GAIN_CTRL, V_INIT_GAIN, BYTE);

/* Control de la 10410: Trigger, Interrupciones, ... */
  write_one(AX_PORT+DRW_CTRL, V_INIT_CTRL, BYTE);

/* Ponemos las salidas digitales y analógicas a sus valores iniciales*/
  write_one(AX_PORT+DW_DIG_OUT, V_INIT_OUT, BYTE);
  write_one(AX_PORT+DW0_DA_LOW, V_INIT_DA_LOW, BYTE);
  write_one(AX_PORT+DW0_DA_HI, V_INIT_DA_HI, BYTE);
  write_one(AX_PORT+DW1_DA_LOW, V_INIT_DA_LOW, BYTE);
  write_one(AX_PORT+DW1_DA_HI, V_INIT_DA_HI, BYTE);

/* Limpia cualquier posible interrupción espúrea que existiera activa.*/
  write_one(AX_PORT+DW_NO_INT,0x00,BYTE);

  return SUCCESS;
}

/*********************************************************************
* write_one()
* Escribe el dato "val" de tamaño "size" en el puerto de I/O "port"
* Según el tamaño se usa un comando u otro (outl, outw o outb).
* Devuelve siempre SUCCESS.
*********************************************************************/
static int write_one(unsigned int port, unsigned int val, int size)
{
  if (size == 4)
    outl(val, port);
  else if (size == 2)
    outw(val&0xffff, port);
  else
    outb(val&0xff, port);
  return SUCCESS;
}

/*********************************************************************
* read_one()
* Lee un dato de tamaño "size" del puerto de I/O "port".
* Devuelve el dato leido.
*********************************************************************/
static int read_one(unsigned int port,int size)
{
  if (size == 4)
    return inl(port);
  else if (size == 2)
    return inw(port);
  else
    return inb(port);
  return 0;
}

/*********************************************************************
* kernel_to_user()
* Función que transfiere datos del kernel-space, apuntados por
* "kernel_pointer"
* al user-space, a la posición apuntada por "user_pointer". La
* cantidad de bytes viene indicada por "size".
* Devuelve el número de bytes transferidos.
*********************************************************************/
static int kernel_to_user(byte *kernel_pointer, byte *user_pointer, int size)
{
  int bytes_transferred = 0;
  for(bytes_transferred =0; bytes_transferred < size; bytes_transferred++)
  {
    put_user(*(kernel_pointer++),user_pointer++);
  }
  return bytes_transferred;
```

```
}

/*************************************************************************
* user_to_kernel()
* Función que transfiere datos del user-space, apuntados por "user_pointer"
* al kernel-space, a la posición apuntada por "kernel_pointer". La
* cantidad de bytes viene indicada por "size".
* Devuelve el número de bytes transferidos.
*************************************************************************/
static int user_to_kernel(byte *kernel_pointer, byte *user_pointer, int size)
{
  int bytes_transferred = 0;
  for (bytes_transferred = 0; bytes_transferred < size; bytes_transferred++)
  {
    get_user(*(kernel_pointer++),user_pointer++);
  }
  return bytes_transferred;
}

/*************************************************************************
* ax_open ()
* Función llamada cuando se abre el archivo asociado a nuestro
* dispositivo.
* Además de impedir accesos concurrentes (Device_Open++) y de evitar
* que se descarge el módulo (con rmmod) cuando aún está abierto el archivo
* (MOD_INC_USE_COUNT), realiza la INICIALIZACION del dispositivo,
* llamando a la función ax_init.
* Devuelve un número negativo en caso de error o SUCCESS en caso
* contrario.
*************************************************************************/
static int ax_open(struct inode *inode,
                   struct file *file)
{

#ifdef DEBUG
  //printk ("ax_open(%p,%p)\n", inode, file);
#endif

/* Inicialización del dispositivo */
  if(ax_init() != 0)
  {
    //printk("ax_open(): error en la inicialización del dispositivo");
    return -EPERM;
  }

/* No queremos accesos concurrentes */
  if (Device_Open)
    return -EBUSY;

  Device_Open++;

/* No queremos que se descargue el módulo con el archivo abierto */
  MOD_INC_USE_COUNT;

  return SUCCESS;
}

/*************************************************************************
* ax_release()
* Función llamada cuando se cierra el archivo asociado al dispositivo.
* Se encarga de:
* - Liberar la interrupción asignada
* - Device_Open--
* - MOD_DEC_USE_COUNT
* Devuelve SUCCESS.
*************************************************************************/
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,2,0)
static int ax_release(struct inode *inode,
                      struct file *file)
#else
static void ax_release(struct inode *inode,
                       struct file *file)
#endif
{
#ifdef DEBUG
  //printk ("ax_release(%p,%p)\n", inode, file);
#endif
```

```
/* Liberamos la irq AX_IRQ_NUMBER */
//printk("Antes de free_irq()\n");
  free_irq(AX_IRQ_NUMBER, NULL);

/* Permitimos que se pueda volver a abrir el archivo */
  Device_Open --;

/* Decrementamos el uso del módulo */
  MOD_DEC_USE_COUNT;

#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,2,0)
  return SUCCESS;
#endif
}

/*************************************************************************
* ax_read()
* Esta función es llamada cuando un proceso realiza una operación de
* lectura sobre el dispositivo.
* Lee todas las entradas digitales (8) y aquéllas analógicas que hayan
* sido programadas previamente (1-16). Dicha programación se hace
* escribiendo en el registro DRW_MUX_CTRL.
* Una vez leidas todas las entradas, se llena el buffer "buffer" con
* los datos obtenidos, y se devuelve el número de bytes leidos.
*************************************************************************/
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,2,0)
static ssize_t ax_read(struct file *file,
    char *buffer,    /* The buffer to fill with data */
    size_t length,   /* The length of the buffer */
    loff_t *offset)  /* Our offset in the file */
#else
static int ax_read(struct inode *inode,
                       struct file *file,
    char *buffer,   /* The buffer to fill with
                     * the data */
    int length)     /* The length of the buffer
                     * (mustn't write beyond that!) */
#endif
{
  int bytes_read = 0;
  struct ax_10410_lectura lectura = { {0} , {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}};
  byte *plectura = (byte *)&lectura;
  byte i = 0, canal_superior = 0, canal_inferior = 0;
  byte multiplexer_scan_register = 0;
  unsigned short int tmp = 0;
/* N.P.I. de por qué, pero el sizeof nos devuelve 1 byte más de la cuenta */
  short int size_of_lectura = sizeof(struct ax_10410_lectura);


/* Realizamos la lectura de los registros correspondientes */

/* Entradas digitales */
  lectura.digital[0]=read_one(AX_PORT+DR_DIG_IN,BYTE);
  //lectura.digital[1]=read_one(AX_PORT+DR_DIG_IN_LOW,BYTE);
  //lectura.digital[2]=read_one(AX_PORT+DR_DIG_IN_HI,BYTE);

/* Entradas analogicas */
  multiplexer_scan_register = read_one(AX_PORT+DRW_MUX_CTRL, BYTE);
  canal_inferior = multiplexer_scan_register & 0x0f;
  canal_superior = (multiplexer_scan_register & 0xf0) >> 4;
  for (i=canal_inferior; i<=canal_superior; i++)
  {
/* Forzamos una conversión A/D (en el caso de trigger por software) */
/* Quizá deberíamos asegurarnos de que el trigger está configurado por SW
   leyendo del registro de estado */
    cli();
    write_one(AX_PORT+DW_START_AD,i,BYTE);

/* Leemos el resultado de la conversión */
/* Nos dormimos hasta que se produzca la interrupción indicando que se ha
   finalizado la conversión A/D */

  interruptible_sleep_on(&ax_queue);
  sti();

/* Finalizada la conversión, leemos los datos convertidos */
```

```
    tmp = read_one(AX_PORT+DR_AD_HI,BYTE);
    tmp = tmp << 8;
    tmp += read_one(AX_PORT+DR_AD_LOW,BYTE);

/* Los 4 LSb hay que descartarlos */
    tmp = tmp >> 4;

/* Y ya tenemos el valor "preparado" */
    lectura.analogico[i]=tmp;


//printk("ax_read: lectura.analogica[%d]= %x\n",i,lectura.analogico[i]);
  }

/* Pasamos del kernel-space al user-space */
/* Comprobamos qué es menor: length ó size_of_lectura y se lo pasamos a
   kernel_to_user() */
  length = length < size_of_lectura ? length : size_of_lectura;

//printk("ax_read: antes de kernel_to_user\n");

  bytes_read = kernel_to_user(plectura,buffer,length);

//printk("ax_read: bytes_read = %d",bytes_read);

/* Esperamos 50 useg. para empezar una nueva conversión*/
//  udelay(50);

#ifdef DEBUG
   //printk ("Read %d bytes, %d left\n",
     bytes_read, length);
#endif

/* Devolvemos el número de bytes leidos */
  return bytes_read;
}

/*****************************************************************
* ax_write()
* Esta función es llamada cuando un proceso intenta escribir en el
* archivo del dispositivo.
* Escribe en todas las salidas digitales (8) y en todas las analógicas
* (2).
* Devuelve el número de bytes escritos.
*****************************************************************/
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,2,0)
static ssize_t ax_write(struct file *file,
    const char *buffer,    /* The buffer */
    size_t length,   /* The length of the buffer */
    loff_t *offset)  /* Our offset in the file */
#else
static int ax_write(struct inode *inode,
                    struct file *file,
                    const char *buffer,
                    int length)
#endif
{
  struct ax_10410_escritura escritura = { {0} , {2458,2458} };
/*Lo dejamos a cero para la carretilla, con eso al inicio, no se debe mover*/
  byte *pescritura = (byte *) &escritura;
/* N.P.I. de porqué, pero sizeof nos da un byte más de la cuenta */
  short int size_of_escritura = sizeof(struct ax_10410_escritura);
  int bytes_written = 0;
  int i=0;
  unsigned short int tmp = 0;

/* Como "buffer" está declarado como constante, si se lo pasamos a
   user_to_kernel() el compilador se queja.
   Haciendo lo que viene a continuación y pasándole a user_to_kernel()
   la variable "pbuffer" ya no hay problemas */
  byte * pbuffer = (byte *) buffer;

/* Pasamos los datos que tenemos que escribir del user-space al kernel-space*/
/* Previamente calculamos el menor valor de entre length y size_of_escritura,
   para pasárselo a user_to_kernel() */
  length = length < size_of_escritura ? length : size_of_escritura;
```

```
  bytes_written = user_to_kernel(pescritura, pbuffer, length);

/* Ya tenemos los datos en el kernel-space. */
/* Procedemos a escribirlos en los registros correspondientes */

/* Salidas digitales */

  //for(i=0;i<1;i++)
  //{
    //printk("ax_write: escritura.digital = %d\n",escritura.digital);
  //}
  for(i=0;i<2;i++)
  {
    //printk("ax_write: escritura.analogica[%d] = %d\n",i, escritura.analogico[i]);
  }

  write_one(AX_PORT+DW_DIG_OUT,escritura.digital, BYTE);
  //write_one(AX_PORT+DW_DIG_OUT_LOW,escritura.digital[1], BYTE);
  //write_one(AX_PORT+DW_DIG_OUT_HI,escritura.digital[2], BYTE);

/* Para el caso de las salidas analógicas, tenemos q escribir 12 bits:
   4 en los MSB del registro DW0_DA_LOW y los restantes 8 en DW0_DA_HI.
   Para el canal 1, lo mismo pero con DW1. */

/* Canal 0 */
  tmp = escritura.analogico[0] & 0x00ff;
  //tmp = tmp << 4;
  write_one(AX_PORT+DW0_DA_LOW, tmp, BYTE);
  tmp = escritura.analogico[0] & 0x0f00;
  tmp = tmp >> 8;
  write_one(AX_PORT+DW0_DA_HI, tmp, BYTE);

/* Canal 1 */
  tmp = escritura.analogico[1] & 0x00ff;
  //tmp = tmp << 4;
  write_one(AX_PORT+DW1_DA_LOW, tmp, BYTE);
  tmp = escritura.analogico[1] & 0x0f00;
  tmp= tmp >> 8;
  write_one(AX_PORT+DW1_DA_HI, tmp, BYTE);

  return bytes_written;
}




/*****************************************************************************
* ax_ioctl()
* Esta función es llamada cuando un proceso intenta realizar una
* operación "ioctl" sobre nuestro dispositivo.
* Toma como parámetros, además de los habituales (inode y file) el
* número de la operación ioctl (que indica el comando que queremos ejecutar)
* y el parámetro para el comando anterior.
* Si la operación es de escritura o de lectura/escritura, la llamada
* al sistema ioctl devuelve la salida de esta función (que será algún valor leido
* de la tarjeta). Es decir, que están intercambiados los conceptos de
* lectura y escritura.
* Devuelve un valor negativo si el comando no es conocido.
*****************************************************************************/
int ax_ioctl(
    struct inode *inode,
    struct file *file,
    unsigned int ioctl_num,/* The number of the ioctl */
    unsigned long ioctl_param) /* The parameter to it */
{
  int i = 0;
  int tmp = 0;
  int ioctl_array[6] = {0, 0, 0, 0, 0, 0};
  byte *pioctl_array = (byte *)ioctl_array;
  byte *pioctl_param = (byte *)ioctl_param;
  unsigned int canal_inicial = 0; /* Para SET_CH */
  unsigned int canal_final = 0;  /* Para SET_CH */
  unsigned int ganancia = 0; /* Para SET_GAIN */
  int arg = 0;  /* Para las funciones implementadas para compatibilidad */
```

```
/* Lo que hacemos es pasarle a IOCTL un puntero a un array de 6 int SIEMPRE,
   independientemente de la operación que queramos hacer.
   Según la operación, tendrán relevancia más o menos elementos de dicho
   array.
*/
/* Por lo anterior, pasamos el contenido de dicho array del user-space al
   kernel-space
*/
  user_to_kernel(pioctl_array, pioctl_param, 6*sizeof(int));
/*
  for (i=0; i<6;i++)
  {
//printk("ax_ioctl(): ioctl_array[%d] = %d\n",i,ioctl_array[i]);
  }
*/

/* "Switch" según el número de la operación ioctl (comando) */
/* Se han implementado las funciones que son necesarias siguiendo el manual
   de la tarjeta. Para saber los parámetros que hay que pasar en cada caso
   y los valores que devuelve cada función, mirar el manual (pág ?-?). */
  switch (ioctl_num)
  {
    case SET_CH:
/* Cambia el rango de canales para la conversión A/D */
      canal_inicial = ioctl_array[0];
      canal_final = ioctl_array[1];

//printk("ax_ioctl(): canal ini/fin: %d %d\n", canal_inicial, canal_final);

      if (canal_inicial > canal_final || \
          canal_inicial < 0 || \
          canal_final > 15)
      {
        //printk("ax_ioctl(): error en la asignación canal inicial / final\n");
        return -1;
      }
      write_one(AX_PORT+DRW_MUX_CTRL,canal_inicial + (canal_final << 4), BYTE);
      break;
    case SET_GAIN:
/* Cambia la ganancia para las entradas analógicas */
      ganancia = ioctl_array[0];
      switch(ganancia)
      {
        case 1:
          ganancia = 0x0;
          break;
        case 2:
          ganancia = 0x1;
          break;
        case 4:
          ganancia = 0x2;
          break;
        case 8:
          ganancia = 0x3;
          break;
        case 16:
          ganancia = 0x4;
          break;
        default:
          //printk("ax_ioctl(): error en el valor de la ganancia\n");
          return -1;
      }
      write_one(AX_PORT+DW_GAIN_CTRL, ganancia, BYTE);
      break;
    case SFT_TRG:
/* Realiza una única conversión A/D */
/* Qué hacemos si no está configurado el trigger x SW?
   1) Damos un error, ó
   2) Lo configuramos por SW ahora
   3) Suponemos que está activado y punto <- elegida!!!
*/

/* Lanzamos el disparo */
  cli();
  write_one(AX_PORT+DW_START_AD,0,BYTE);
```

```
/* Esperamos a que se produzca la interrupción, indicando el fin de la
   conversion A/D, y a que entonces se despierte el proceso */
  interruptible_sleep_on(&ax_queue);
  sti();

// printk("ax_ioctl: despierto...");

/* Leemos el resultado y lo convertimos */
      ioctl_array[0] = read_one(AX_PORT+DR_AD_HI,BYTE);

//printk("ioctl(): HIGH = %x\n",ioctl_array[0]);

      ioctl_array[0] <<= 4;
      tmp = read_one(AX_PORT+DR_AD_LOW,BYTE);

//printk("ioctl(): LOW = %x\n",tmp);

      ioctl_array[0] += ((tmp & 0x00f0) >> 4);
      ioctl_array[1] = tmp & 0x000f;

/* Falta pasar los datos de vuelta al user-space */
      kernel_to_user((byte*)ioctl_array, (byte*)ioctl_param, 6*sizeof(int));

      break;

    case NADC_ARY:
/* Esta función no es necesario implementarla */
      break;

    case INTR:
/* Esta función no es necesario implementarla */
      break;

    case DMA:
/* Esta función no es necesario implementarla */
      break;

    case DIS_ID:
/* Deshabilita las interrupciones */
/* Para ello escribe un 0 en el bit 8 del registro de control (DRW_CTRL) */

/* Entonces, lo que hacemos es leer la configuración actual */
      tmp = read_one(AX_PORT+DRW_CTRL,BYTE);
/* Modificamos el bit 8 sin cambiar el resto */
      tmp &= 0x7f;
/* Y lo escribimos */
      write_one(AX_PORT+DRW_CTRL, tmp, BYTE);
      break;

    case READ_ST:
/* Esta función no es necesario implementarla */
      break;

    case TRAN_DATA:
/* Esta función no es necesario implementarla */
      break;

    case SET_TIMER:
/* Especifica los divisores de los contadores 1 y 2 para el timer */
/* PENDIENTE DE REVISAR !!!! */
      if(ioctl_array[0] < 2 || ioctl_array[1] < 2)
      {
        //printk("ax_ioctl(): divisores demasiado pequeños");
        return -1;
      }
      write_one(AX_PORT+DRW_CONT1, ioctl_array[1], BYTE);
      write_one(AX_PORT+DRW_CONT2, ioctl_array[0], BYTE);
      write_one(AX_PORT+DW_CONT_CTRL, 0x40 ,BYTE);
      interruptible_sleep_on(&ax_queue);
      //printk("SET_TIMER: Interrupción producida\n");
      break;

    case TRG_SLOPE:
/* Esta función no es necesario implementarla */
      break;

    case RD_CH:
```

```
/* Esta función no es necesario implementarla */
      break;

    case DO_BYTE:
/* Escribe un byte en la salida digital de un byte */
/* Para ello escribe el byte en el registro DW_DIG_OUT */
      write_one(AX_PORT+DW_DIG_OUT, ioctl_array[0], BYTE);
      break;

    case DI_BYTE:
/* Lee un byte de la entrada digital de un byte */
/* Para ello lee el byte del registro DR_DIG_IN */
      ioctl_array[0] = read_one(AX_PORT+DR_DIG_IN, BYTE);

/* Lo pasamos al user-space */
      kernel_to_user((byte *)ioctl_array,(byte *)ioctl_param,6*sizeof(int));
      break;

    case DAC_ONE:
/* Realiza una conversion D/A de un solo canal */
      switch(ioctl_array[0])
      {
        case 0:
          write_one(AX_PORT+DW0_DA_LOW,(ioctl_array[1] & 0x000f) << 4, BYTE);
          write_one(AX_PORT+DW0_DA_HI, (ioctl_array[1] & 0x0ff0) >> 4, BYTE);
        case 1:
          write_one(AX_PORT+DW1_DA_LOW,(ioctl_array[1] & 0x000f) << 4, BYTE);
          write_one(AX_PORT+DW1_DA_HI, (ioctl_array[1] & 0x0ff0) >> 4, BYTE);
      }
      break;

    case DAC_TWO:
/* Realiza una conversión D/A de ambos canales */
/* Canal 0 */
      write_one(AX_PORT+DW0_DA_LOW,(ioctl_array[0] & 0x000f) << 4, BYTE);
      write_one(AX_PORT+DW0_DA_HI, (ioctl_array[0] & 0x0ff0) >> 4, BYTE);
/* Canal 1 */
      write_one(AX_PORT+DW1_DA_LOW,(ioctl_array[1] & 0x000f) << 4, BYTE);
      write_one(AX_PORT+DW1_DA_HI, (ioctl_array[1] & 0x0ff0) >> 4, BYTE);
      break;

    case DAAD_INT:
      break;

/*    case DO_WORD:
* Escribe una palabra ("word") en la salida digital de un "word" de tamaño *
* Para ello escribe el "word" en los registros DW_DIG_OUT_LOW y DW_DIG_OUT_HI*
      write_one(AX_PORT+DW_DIG_OUT_LOW, ioctl_array[0] & 0x00ff, BYTE);
      write_one(AX_PORT+DW_DIG_OUT_HI, (ioctl_array[0] & 0xff00)>>8, BYTE);
      break;

    case DI_WORD:
* Lee un "word" de la entrada digital de un "word" de tamaño *
* Para ello lee el "word" de los registros DR_DIG_IN_LOW y DR_DIG_IN_HI *
      ioctl_array[0] = read_one(AX_PORT+DR_DIG_IN_HI, BYTE);
      tmp = read_one(AX_PORT+DR_DIG_IN_LOW, BYTE);
      ioctl_array[0] = (ioctl_array[0] << 8) + tmp;

* Lo pasamos al user-space *
      kernel_to_user((byte *)ioctl_array,(byte *)ioctl_param,6*sizeof(int));
      break;
*/
/* Funciones implementadas para compatibilidad con programas basados en el
   driver anterior */

    case CANAL_INICIAL:
      arg = ioctl_array[0];
/* Comprobamos que no se sale del rango */
      if( (arg < 0) || (arg > 15) )
      {
       //printk("ax_ioctl: error poniendo canal inicial\n");
       return -1;
      }
/* Leemos lo que hay en el registro y comprobamos que no es mayor que el
   canal final */
      tmp = read_one(AX_PORT+DRW_MUX_CTRL, BYTE);
      canal_final = (tmp & 0xf0) >> 4;
```

```
//printk("arg = %d canal_final = %d", arg, canal_final);
      if (arg > canal_final)
      {
       //printk("ax_ioctl: error: canal inicial es mayor que canal final\n");
       return -1;
      }
/* Construimos el dato para escribirlo */
      tmp = (tmp & 0xf0) | (arg & 0x0f);
/* Escribimos en el registro adecuado */
      write_one(AX_PORT+DRW_MUX_CTRL, tmp, BYTE);
      break;

    case CANAL_FINAL:
      arg = ioctl_array[0];
/* Comprobamos que no se sale del rango */
      if( (arg < 0) || (arg > 15) )
      {
        //printk("ax_ioctl: error poniendo canal final\n");
        return -1;
      }
/* Leemos lo que hay en el registro y comprobamos que el canal final no es
   menor que el canal inicial */
      tmp = read_one(AX_PORT+DRW_MUX_CTRL, BYTE);
      canal_inicial = tmp & 0x0f;
      if (arg < canal_inicial)
      {
       //printk("ax_ioctl: error: canal final es menor que canal inicial\n");
       return -1;
      }
/* Construimos el dato */
      tmp = (tmp & 0x0f) | ((arg << 4) & 0xf0);

/* Escribimos en el registro adecuado */
      write_one(AX_PORT+DRW_MUX_CTRL, tmp, BYTE);
      break;

    case GANANCIA:
      ganancia = ioctl_array[0];
//printk("ax_ioctl: ganancia = %d",ganancia);
      switch(ganancia)
      {
        case 1:
          ganancia = 0x0;
          break;
        case 2:
          ganancia = 0x1;
          break;
        case 4:
          ganancia = 0x2;
          break;
        case 8:
          ganancia = 0x3;
          break;
        case 16:
          ganancia = 0x4;
          break;
        default:
          //printk("ax_ioctl(): error en el valor de la ganancia\n");
          return -1;
      }
      write_one(AX_PORT+DW_GAIN_CTRL, ganancia, BYTE);
      break;

    case SALIDA_ANALOGICA:
/* No entiendo lo que se pretende con esta función ni cómo se le pasa el
   valor que se quiere poner en la salida especificada */

    case LECTURA:
      arg = ioctl_array[0];
      if ( (arg < 0) || (arg > 23) )
      {
        //printk("ax_ioctl: error: fuera de rango de canales digitales\n");
        return -1;
      }
      else if (arg < 8)
      {
        tmp = read_one(AX_PORT+DR_DIG_IN, BYTE);
```

```
        ioctl_array[0] = tmp & (0x01 << arg);
       }
       /*else if (arg < 16)
       {
        tmp = read_one(AX_PORT+DR_DIG_IN_LOW, BYTE);
        ioctl_array[0] = tmp & (0x01 << (arg-8));
       }
       else if (arg < 24)
       {
        tmp = read_one(AX_PORT+DR_DIG_IN_HI, BYTE);
        ioctl_array[0] = tmp & (0x01 << (arg-16));
       }*/
       ioctl_array[0] = ioctl_array[0] ? 1 : 0;
       kernel_to_user((byte *)ioctl_array,(byte *)ioctl_param, sizeof(int));
       break;

     case PONER_1:
       arg = ioctl_array[0];
       if ( (arg < 0) || (arg > 23) )
       {
         //printk("ax_ioctl: error: fuera de rango de canales digitales\n");
         return -1;
       }
       else if (arg < 8)
       {
/* Me temo que así no se puede implementar esta función porque estamos leyendo
el valor digital que hay de entrada, le estamos poniendo el bit adecuado a 1 y
lo estamos poniendo nuevamente de salida. Pero lo que queremos es modificar la
salida que hubiera anteriormente cambiando el bit adecuado a 1. Y para esto
habría que mantener una estructura que llevara el control en todo momento de
los valores digitales que están puestos a la salida. Así que esto falta... */
         tmp = read_one(AX_PORT+DW_DIG_OUT, BYTE);
         ioctl_array[0] = tmp | (0x01 << arg);
         write_one(AX_PORT+DW_DIG_OUT, ioctl_array[0], BYTE);
       }
       /*else if (arg < 16)
       {
         tmp = read_one(AX_PORT+DW_DIG_OUT_LOW, BYTE);
         ioctl_array[0] = tmp | (0x01 << (arg-8));
         write_one(AX_PORT+DW_DIG_OUT_LOW, ioctl_array[0], BYTE);
       }
       else if (arg < 24)
       {
         tmp = read_one(AX_PORT+DW_DIG_OUT_HI, BYTE);
         ioctl_array[0] = tmp | (0x01 << (arg-16));
        write_one(AX_PORT+DW_DIG_OUT_LOW, ioctl_array[0], BYTE);
       }*/
       break;

     case PONER_0:
       arg = ioctl_array[0];
       if ( (arg < 0) || (arg > 23) )
       {
         //printk("ax_ioctl: error: fuera de rango de canales digitales\n");
         return -1;
       }
       else if (arg < 8)
       {
         tmp = read_one(AX_PORT+DW_DIG_OUT, BYTE);
         ioctl_array[0] = ~( (~tmp) | (0x01 << arg) );
         write_one(AX_PORT+DW_DIG_OUT, ioctl_array[0], BYTE);
//printk("ioctl: %x", ioctl_array[0]);
       }
       /*else if (arg < 16)
       {
         tmp = read_one(AX_PORT+DW_DIG_OUT_LOW, BYTE);
         ioctl_array[0] = ~( (~tmp) | (0x01 << (arg-8)) );
         write_one(AX_PORT+DW_DIG_OUT_LOW, ioctl_array[0], BYTE);
       }
       else if (arg < 24)
       {
         tmp = read_one(AX_PORT+DW_DIG_OUT_HI, BYTE);
         ioctl_array[0] = ~( (~tmp) | (0x01 << (arg-16)) );
         write_one(AX_PORT+DW_DIG_OUT_HI, ioctl_array[0], BYTE);
       }*/
       break;
     default:
```

```
/* Error: "comando desconocido" */
      return -1;
  }
/* Para que no proteste el compilador */
  return SUCCESS;
}

/************** Declaraciones del módulo ****************************/

/* El "major number" para el dispositivo. Es global porque tiene que ser
   accesible para el registro y la liberación del dispositivo */
static int Major;

/* Esta estructura mantiene las funciones que serán llamadas cuando un
   proceso haga algo a nuestro dispositivo.
   Como se mantiene un puntero a esta estructura en la tabla de dispositivos,
   no puede ser local a init_module.
   NULL es para las funciones no implementadas.
*/

struct file_operations Fops =
{
  NULL,    /* seek */
  ax_read, /* read */
  ax_write,/* write */
  NULL,    /* readdir */
  NULL,    /* select */
  ax_ioctl,    /* ioctl */
  NULL,    /* mmap */
  ax_open,
#if LINUX_VERSION_CODE >= KERNEL_VERSION(2,2,0)
  NULL,   /* flush */
#endif
  ax_release  /* a.k.a. close */
};

/*************************************************************************
* init_module()
* Inicializa el módulo. Registra el dispositivo de caracteres.
*************************************************************************/
int init_module()
{
/* Debemos registrar el rango de puertos I/O del dispositivo */
  if (ax_detect(AX_PORT, AX_IO_SPACE) != 0)
  {
    //printk("init_module: error registrando la zona de I/O \n");
    return -1;
  }

/* Registra el dispositivo de caracteres, asignando a un archivo con "major"
   igual a AX_MAJOR las funciones registradas en la estructura Fops.
   El dispositivo queda registrado en /proc/devices con el nombre DEVICE_NAME.
*/
  Major = module_register_chrdev(AX_MAJOR, DEVICE_NAME, &Fops);

/* Un valor negativo indica un error */
  if (Major < 0)
  {
    //printk("init_module: error registrando el dispositivo AX10410\n");
    return Major;
  }

  printk("Modulo para el dispositivo AX10410 cargado\n");
  printk("By Rafael Martin de Agar Tirado\nModificado por Eduardo Gomez- Elegido
Pedraza\n");

  return 0;
}
/*************************************************************************
* cleanup_module()
* Limpia el módulo.
*************************************************************************/
void cleanup_module()
{
  int ret;

/* Libera el dispositivo */
```

```
  ret = module_unregister_chrdev(AX_MAJOR, DEVICE_NAME);

/* Si hay algún error, lo mostramos */
  if (ret < 0)
    //printk("cleanup_module: error en unregister_chrdev: %d\n", ret);

/* Liberamos la zona de I/O que teníamos reservada */
  release_region(AX_PORT, AX_IO_SPACE);

}

/************************************************************************
* BOTTON HALF
* Esta rutina se encarga de procesar la información recibida tras una IRQ
* De momento, en pruebas
************************************************************************/
static void info_recibida(void *info)
{
  //printk("Se ha producido una interrupción: %d\n",*((int*) info));
}

/************************************************************************
* MANEJADOR DE INTERRUPCIONES
* Rutina llamada cuando se produce una interrupción de nuestro dispositivo
************************************************************************/
void irq_handler(int irq,
                 void *dev_id,
                 struct pt_regs *regs)
{
//  printk("irq_handler(): ...\n");
  wake_up_interruptible(&ax_queue);
/* Indicamos a la AX que hemos procesado la interrupción */
  write_one(AX_PORT+DW_NO_INT,0x00,BYTE);

}
```

El driver, ax10410.h

```
/*************************************************************************
* AX10410.H
* En este archivo, se incluyen todas las declaraciones específicas de
* la la tarjeta AX10410 de AXIOM.
*
*************************************************************************/

/* El nombre de nuestro dispositivo para /proc/devices */
#define DEVICE_NAME "ax10410"

/* Longitud máxima de un mensaje generado por el dispositivo */
#define BUF_LEN 80

/* "Major number" del dispositivo */
#define AX_MAJOR 101

/* Espacio de puertos de I/O que ocupa el dispositivo */
#define AX_IO_SPACE 16

/* Dirección base del espacio de puertos de I/O */
#define AX_PORT 0x2c0

/* Posición del jumper JP1: Selección de la escala completa para conversión A/D */
#define AD_ESCALA_COMPLETA 10

/* Posición del jumper JP2: Selección de la escala completa para conversión D/A */
#define DA_ESCALA_COMPLETA 10

#define byte unsigned char

/* Inicialización del dispositivo */

#define V_INIT_GAIN 0x00  /* Ganancia inicial -> 1 */
#define V_INIT_MUX 0x00   /* Canal inicial y final de conversión A/D -> 0 */
#define V_INIT_OUT 0x00   /* Salidas digitales -> 0 */
#define V_INIT_CTRL 0xe8  /* Disparo por SW para la conversión A/D,*/
#define V_INIT_TIM 0x01   /* no external TRGE, IRQ=6, interrupción habilitada */
#define V_INIT_DA_LOW 6/*0x00*/ /*valor que se considera medio por la carretilla*/
#define V_INIT_DA_HI 102/*0x06*/    /*el byte alto de ese valor tan especial*/


/* Para la llamada a write_one() y read_one() */
#define BYTE 1
#define WORD 2
#define DWORD long int

#define ERROR -1
#define SUCCESS 0

/* Para las comprobaciones en ax_probe() */
#define TEST_DATA_1 0x55
#define TEST_DATA_2 0x5f

/* Comandos IOCTLS */
/* Han sido implementados siguiendo el User's Manual de la tarjeta */
/* El 2° parámetro que aparece es el número de función según el User's Manual */

#define SET_CH _IOR(AX_MAJOR, 3, int *)
#define SET_GAIN _IOR(AX_MAJOR, 4, int *)
#define SFT_TRG _IOWR(AX_MAJOR, 5, int *)
#define NADC_ARY _IOWR(AX_MAJOR, 6, int *)
#define INTR _IOWR(AX_MAJOR, 7, int *)
#define DMA _IOWR(AX_MAJOR, 8, int *)
#define DIS_ID _IOR(AX_MAJOR, 9, int *)
#define READ_ST _IOW(AX_MAJOR, 10, int *)
#define TRAN_DATA _IOWR(AX_MAJOR, 11, int *)
#define SET_TIMER _IOR(AX_MAJOR, 12, int *)
```

```
#define TRG_SLOPE _IOR(AX_MAJOR, 13, int *)
#define RD_CH _IOW(AX_MAJOR, 14, int *)
#define DO_BYTE _IOR(AX_MAJOR, 15, int *)
#define DI_BYTE _IOW(AX_MAJOR, 16, int *)
#define DAC_ONE _IOR(AX_MAJOR, 17, int *)
#define DAC_TWO _IOR(AX_MAJOR, 18, int *)
#define DAAD_INT _IOWR(AX_MAJOR, 19, int *)
//#define DO_WORD _IOR(AX_MAJOR, 20, int *)
//#define DI_WORD _IOR(AX_MAJOR, 21, int *)

/* Para mantener la compatibilidad con programas basados en el driver anterior
implementamos las mismas funciones que existían */

#define CANAL_INICIAL _IOR(AX_MAJOR, 30, int *)
#define CANAL_FINAL _IOR(AX_MAJOR, 31, int *)
#define GANANCIA _IOR(AX_MAJOR, 32, int *)
#define SALIDA_ANALOGICA _IOR(AX_MAJOR, 33, int *)
#define LECTURA _IOW(AX_MAJOR, 34, int *)
#define PONER_1 _IOR(AX_MAJOR, 35, int *)
#define PONER_0 _IOR(AX_MAJOR, 36, int *)

/* Interrupciones */
#define AX_IRQ_NUMBER 6    /* La IRQ es programable por SW, es la de la */
                          /*disquetera, pero bueno*/

// #include "axinfo.h"

/* Para hacer lecturas del dispositivo */
struct ax_10410_lectura {
  unsigned char digital[1];
  unsigned short int analogico[16];
};

/* Para hacer escrituras en el dispositivo */
struct ax_10410_escritura {
  unsigned char digital;
  unsigned short int analogico[2];
};

/* Registros del dispositivo, indicando si es de lectura (R), escritura (W)
   o ambos (RW) */
#define DR_AD_LOW       0
#define DR_AD_HI        1
#define DW_START_AD     0
#define DW_GAIN_CTRL    11
#define DRW_MUX_CTRL    2
#define DR_DIG_IN       3
#define DW_DIG_OUT      3
#define DW0_DA_LOW      4
#define DW0_DA_HI       5
#define DW1_DA_LOW      6
#define DW1_DA_HI       7
#define DR_ST           8
#define DW_NO_INT       8
#define DRW_CTRL        9
//#define DR_DIG_IN_LOW  10
//#define DR_DIG_IN_HI   11
#define DW_TIM_CTRL    10
//#define DW_DIG_OUT_HI  11
#define DRW_CONT0      12
#define DRW_CONT1      13
#define DRW_CONT2      14
#define DW_CONT_CTRL  15

/* Estado del dispositivo */
struct ax_10410_status
{
  struct ax_10410_lectura lectura;
  struct ax_10410_escritura escritura;
  byte canal_inicial;
  byte canal_final;
  byte ganancia;
};

/* Definiciones varias (que no se usan) */
#define R_NUM_CANAL_AD  0x0F
#define R_INT           0x10
```

```
#define R_CONV_EN_CURSO 0x80

#define W_HAY_DMA              0x04
#define W_START_SOFTWARE       0xFD
#define W_START_EXTERNO_AND    0xFE
#define W_START_EXTERNO_OR     0x02
#define W_START_INTERNO        0x03
#define W_TRGE                 0x08
#define W_DESP_NIVEL_INT       0x04
#define W_SI_INT               0x80

#define SWAIT_SIGS_IGNORE      -1
#define SWAIT_SIGS_DELIVER      0
#define SWAIT_SIGS_ABORT        1

#define CANAL0            0
#define CANAL1            1
#define CANALES           2

/*
#define CANAL_INICIAL     1
#define CANAL_FINAL       2
#define GANANCIA          3
#define SALIDA_ANALOGICA  4

#define LECTURA           5
#define PONER_1           6
#define PONER_0           7
*/

struct axstatics {
  int V_dig_in;
  int V_dig_in_low;
  int V_dig_in_hi;
  int V_dig_out;
  int V_dig_out_low;
  int V_dig_out_hi;
  int V_port;
  byte V_irq;
  byte V_dmae;
  byte V_mux;
  byte V_inic_mux;
  byte V_fin_mux;
  byte V_out;
  byte V_gain;
  byte V_ctrl;
  int opened;
  int write_sem;
  int read_sem;
  int open_sem;
  int conversion_sem;
};
```

Las clases, ax10410.cpp

```
/*************************************************************************
*
*  DEFINICIÓN DE LA CLASE AX10410 PARA EL VEHÍCULO CARRETILLA
*  AUTOR: Eduardo Gómez- Elegido Pedraza
*
*  En esta clase se definen los distintos métodos que se pueden usar para
*  acceder a la tarjeta AX10410 de Axiom, con los que actuaremos sobre la
*  carretilla.
*
*************************************************************************/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <iostream.h>
#include <fcntl.h>
#include <fstream.h>
#include <math.h>

#include "ax10410.hpp"


/*************** Constructor de la clase AX ******************/

AX::AX ()
{
  escritura.digital = 8;
  escritura.analogico[0] = 6*4096/10;
  escritura.analogico[1] = 6*4096/10;
}


/******************** Destructor de la clase AX ******************/

AX::~AX() { close_ax(); }


/********* open_ax, abre el archivo de dispositivo de la ax y la inicializa *********/

int AX::open_ax(char *ax_file = "/dev/ax10410")
{

  ax_handler = open(ax_file,O_RDWR);

  if (ax_handler <= 0)
    return ERROR;
  else
    return SUCCESS;

  write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));

}


/**** close_ax, cierra la tarjeta, en caso de que este abierta ******/

int AX::close_ax()
{
  escritura.digital &= 0x1f;
  write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
  if (close(ax_handler) <= 0)
    return ERROR;
  else
    return SUCCESS;

}
```

```
/**** incrementar_potenciometro, sube el cursor del potenciometro de traccion ******/

void AX::incrementar_potenciometro(int incr,int guardar,int vel,int digital)
{

  int i;
  int flag=1;

  escritura.digital |= 0x04;
  write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
  escritura.digital &= 0xf7;
  write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
  for (i = 0; i < incr ; i++)
  {
      if (flag)
      {
             escritura.digital |= 0x02;
             flag = 0;
      }
      else
      {
              escritura.digital &= 0xfd;
             flag = 1;
      }
      write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
      usleep(1000*vel);
  }
  if(!guardar)
  {
    escritura.digital |= 0x08;
    write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
    if (escritura.digital&0x02==0)
    {
      escritura.digital|=0x02;
      write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
    }
  }
  else
  {
    if (escritura.digital&0x02==0)
    {
      escritura.digital|=0x02;
      write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
    }
    escritura.digital |= 0x08;
    write(ax_handler, (char *) &escritura, sizeof(struct ax_10410_escritura));
  }
}


/**** decrementar_potenciometro, baja el cursor del potenciometro de traccion *****/

void AX::decrementar_potenciometro(int incr,int guardar,int vel,int digital)
{
  int i;
  int flag = 1;

  escritura.digital &= 0xfb;
  write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
  escritura.digital &= 0xf7;
  write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));


  for (i = 0; i < incr ; i++)
  {
      if (flag)
      {
             escritura.digital |= 0x02;
             flag = 0;
      }
      else
      {
             escritura.digital &= 0xfd;
             flag = 1;
      }
      write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
```

```
      usleep(1000*vel);
  }

  if(!guardar)
  {
    escritura.digital |= 0x08;
    write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
    if (escritura.digital&0x02==0)
    {
      escritura.digital|=0x02;
      write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
    }
  }
  else
  {
    if (escritura.digital&0x02==0)
    {
      escritura.digital|=0x02;
      write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
    }
    escritura.digital |= 0x08;
    write(ax_handler, (char *) &escritura, sizeof(struct ax_10410_escritura));
  }
}


/**** arrancar_marcha_adelante, arranca la carretilla y la pone a una velocidad ****/

void AX::arrancar_marcha_adelante(int velocidad)
{
  int incremento;
  int saveono;
  int veloc;
  int digital=0;

  digital = 0x10;
  escritura.digital |= 0x10;
  write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
  usleep(300000);


  incremento = velocidad;
  saveono = 0;
  veloc = 50;

  incrementar_potenciometro(incremento,saveono,veloc,digital);
}


/***** arrancar_marcha_atras, arranca la carretilla y la pone a una velocidad *****/

void AX::arrancar_marcha_atras(int velocidad)
{
  int incremento;
  int saveono;
  int veloc;
  int digital=0;

  incremento = 6;
  saveono = 0;
  veloc = 50;

  decrementar_potenciometro(incremento,saveono,veloc,digital);
  digital = 0x10;
  escritura.digital |= 0x10;
  write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
  usleep(300000);


  incremento = velocidad-6;
  saveono = 0;
  veloc = 50;

  decrementar_potenciometro(incremento,saveono,veloc,digital);
}
```

```
/******* parar_marcha_adelante, para la carretilla cuando va hacia delante *********/

void AX::parar_marcha_adelante(int velocidad)
{
  int incremento;
  int saveono;
  int veloc;
  int digital=0;


  incremento = velocidad;
  saveono = 0;
  veloc = 50;

  decrementar_potenciometro(incremento,saveono,veloc,digital);
  digital = 0xef;
  escritura.digital &= 0xef;
  usleep(300000);
  write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
}


/********* parar_marcha_atras, para la carretilla cuando va hacia detras **********/

void AX::parar_marcha_atras(int velocidad)
{
  int incremento;
  int saveono;
  int veloc;
  int digital=0;

  incremento = velocidad-6;
  saveono = 0;
  veloc = 50;

  incrementar_potenciometro(incremento,saveono,veloc,digital);
  digital = 0xef;
  escritura.digital &= 0xef;
  usleep(300000);
  write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));

  incremento = 6;
  saveono = 0;
  veloc = 50;

  incrementar_potenciometro(incremento,saveono,veloc,digital);
}


/*** centrar_potenciometro, 1º, el potenciometro en un extremo, luego, al centro **/

void AX::centrar_potenciometro() //hacerlo lo primero de todo, antes de encender
{                                // la carretilla, pero con el conmutador de
  int incremento;                // manual-automatico activado
  int saveono;
  int veloc;
  int digital=0;

  /*Lo ponemos en el cero*/
  incremento = 200; // con 200, este donde este, lo bajamos al tope
  saveono = 0;
  veloc=2;
  decrementar_potenciometro(incremento,saveono,veloc,digital);
  /*Lo ponemos en el medio*/
  incremento = 94;// este valor no es el centro, pero es el que funciona
  saveono = 1;
  veloc=2;
  incrementar_potenciometro(incremento,saveono,veloc,digital);
  escritura.digital &= 0xef;
  write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
}

/****************** girar_rueda, es evidente lo que hace, no? ********************/

int AX::girar_rueda(double giro) // giro ha de estar entre 0 y 3
{
  double voltios;
```

```
  int bits;
  extern int flag_giro;
  static int iteraciones_pid;
  iteraciones_pid ++;
  if((fabs(giro) <= 0.03))
    giro = 0;

  if((giro != 0) && (flag_giro == 0))
    flag_giro = 1;

  if((fabs(giro) <= 3) && flag_giro == 1 && giro != 0)
  {
    escritura.digital |= 0x60;
    voltios=giro+6;  //proximamente, se exigira una conversion de giro a voltios
    bits=(int)((voltios*4096)/10);
    escritura.analogico[1] = bits;
    write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
   }

   if(iteraciones_pid >= 20 && fabs(giro) <= 0.1)
   {
     iteraciones_pid = 0;
     voltios=6;
     bits=(int)((voltios*4096)/10);
     escritura.analogico[1] = bits;
     escritura.digital &= 0xdf;
     write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
     voltios=6;
     bits=(int)((voltios*4096)/10);
     escritura.analogico[1] = bits;
     escritura.digital &= 0x1f;
     write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
   }
    return SUCCESS;

 }


/************** levantar_palas, pues las levanta ******************/

int AX::levantar_palas(float velocidad) // velocidad ha de estar ente 0 y 3
{

  float voltios;
  int bits;

  if(velocidad<=3 && velocidad>=0)
  {
    escritura.digital |= 0x60;
    voltios=velocidad+6;  //aqui se exigira una conversion mas compleja
    bits=(int)((voltios*4096)/10);
    escritura.analogico[0] = bits;
    write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
    getchar();

    voltios=6;
    bits=(int)((voltios*4096)/10);
    escritura.analogico[0] = bits;
    escritura.digital &= 0xef;
    write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));

    voltios=6;
    bits=(int)((voltios*4096)/10);
    escritura.analogico[0] = bits;
    escritura.digital &= 0x1f;
    write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
    return SUCCESS;
  }
  else
  {
    return ERROR;
  }
}
```

```
/***************** bajar_palas, baja las palas **********************/

int AX::bajar_palas(float velocidad)
{

  float voltios;
  int bits;


  if(velocidad<3 && velocidad>0)
  {
    escritura.digital |= 0x60;
    voltios=6-velocidad;
    bits=(int)((voltios*4096)/10);
    escritura.analogico[0] = bits;
    write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
    getchar();

    voltios=6;
    bits=(int)((voltios*4096)/10);
    escritura.analogico[0] = bits;
    escritura.digital &= 0xef;
    write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));

    voltios=6;
    bits=(int)((voltios*4096)/10);
    escritura.analogico[0] = bits;
    escritura.digital &= 0x1f;
    write(ax_handler,(char *) &escritura, sizeof(struct ax_10410_escritura));
    return SUCCESS;
  }
  else
  {
    return ERROR;
  }
}
```

Las clases, ax10410.hpp

```
/*************************************************************************
*
*  AX10410.HPP
*
*  AUTOR: Eduardo Gómez- Elegido Pedraza
*
*  Declaración de los componentes de la clase ax, que se usarán en el fichero
*  ax10410.cpp.
*
*************************************************************************/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <linux/delay.h>

#include "./driver/ax10410.h"     //¡¡¡¡¡¡OJO!!!!!! MIRAR EL DIRECTORIO DE ESTO


class AX
{

  private:
     struct ax_10410_lectura lectura;
     struct ax_10410_escritura escritura;
     int ax_handler;

     //float giro=0.0;   //esto sera mas bien como entrada a la funciones
     //int digital=0;
     //int velocidad=0;
     //float subida;


  public:
     AX();
     ~AX();

     int open_ax(char *ax_file = "/dev/ax10410");

     int close_ax();

     void incrementar_potenciometro(int,int,int,int);

     void decrementar_potenciometro(int,int,int,int);

     void arrancar_marcha_adelante(int);

     void arrancar_marcha_atras(int);

     void parar_marcha_adelante(int);

     void parar_marcha_atras(int);

     int girar_rueda(double);

     void centrar_potenciometro();

     int levantar_palas(float);

     int bajar_palas(float);

};
```

Las clases, robosense.cpp

```
/*************************************************************************
*
* robosense.CPP
*
* Programa de comunicación con el láser Robosense de Siman
*
* Autor: Eduardo Gómez- Elegido Pedraza
* Basado en un programa de Rafael Martín de Agar Tirado
*
*
* ULTIMA MODIFICACION:
*************************************************************************/

// Para printf()
//#include <stdio.h>

// Para bzero()
#include <string.h>

// Para atoi(), atof(),...
#include <stdlib.h>


// Para el manejo de cadenas
#include <string.h>

// Para sleep
#include <unistd.h>

// Para gettimeofday()
#include <sys/time.h>


// Mis .h
#include "robosense.hpp"


/*************************************************************************
*
* init_robosense - inicializa el Robosense
*
* PARAMETROS:
*
* DEVUELVE:
*
*************************************************************************/
int Robosense::init_robosense()
{
  int i;
  i=init_puerto_serie();
  printf("puerto serie inicializado\n");
  return i;
}

/*************************************************************************
*
* mandar_comando_robosense - manda una orden al Robosense
*
* PARAMETROS:
*
* DEVUELVE:
*
*************************************************************************/
void Robosense::mandar_comando_robosense(int comando)
{
  unsigned char mensaje[15];
  unsigned char mensaje_corregido[15];
```

```
int checksum=0;
int flag=1;
int selector;
unsigned char target;
int frame;
int i,j;
int xmost,xleast,ymost,yleast=0;
int frecuencia;


if((comando==11||comando==12||comando==14||comando==17||comando==18||comando==21||
    comando==22||comando==23||comando==25||comando==27||comando==28||comando==29||
    comando==114||comando==54))
  selector=11;
else
  selector=comando;

switch(selector)
{
  case 11:
    mensaje[0]=0x0d;
    mensaje[1]=0x03;
    mensaje[2]=comando;
    checksum=256-((mensaje[1]+comando)%256);
    mensaje[3]=checksum;
  break;
  case 19:
    printf("\nSelecciona blanco:");
    do
      scanf("%u",&target);
    while(target==0);
    mensaje[0]=0x0d;
    mensaje[1]=0x04;
    mensaje[2]=comando;
    mensaje[3]=target;
    for(i=1;i<mensaje[1];i++)
      checksum+=(unsigned int) mensaje[i];
    checksum=256-(checksum%256);
    mensaje[4]=checksum;
  break;
  case 211:
    printf("\nSelecciona Frame:");
    scanf("%d",&frame);
    mensaje[0]=0x0d;
    mensaje[1]=0x04;
    mensaje[2]=comando;
    mensaje[3]=frame;
    for(i=1;i<mensaje[1];i++)
      checksum+=(unsigned int) mensaje[i];
    checksum=256-(checksum%256);
    mensaje[4]=checksum;
  break;
  case 212:
    mensaje[0]=0x0d;
    mensaje[1]=0x09;
    mensaje[2]=comando;
    mensaje[3]=xmost;
    mensaje[4]=xleast;
    mensaje[5]=ymost;
    mensaje[6]=yleast;
    mensaje[7]=0x00;
    mensaje[8]=0x00;
    for(i=1;i<mensaje[1];i++)
      checksum+=(unsigned int) mensaje[i];
    checksum=256-(checksum%256);
    mensaje[9]=checksum;
    break;
  case 24:
    printf("\nSelecciona frecuencia:");
    scanf("%d",&frecuencia);
    mensaje[0]=0x0d;
    mensaje[1]=0x04;
    mensaje[2]=comando;
    mensaje[3]=frecuencia;
    for(i=1;i<mensaje[1];i++)
      checksum+=(unsigned int) mensaje[i];
    checksum=256-(checksum%256);
```

```
    mensaje[4]=checksum;
    break;
  case 113:
    mensaje[0]=0x0d;
    mensaje[1]=0x09;
    mensaje[2]=comando;
    mensaje[3]=xmost;
    mensaje[4]=xleast;
    mensaje[5]=ymost;
    mensaje[6]=yleast;
    mensaje[7]=0x00;
    mensaje[8]=0x00;
    for(i=1;i<mensaje[1];i++)
      checksum+=(unsigned int) mensaje[i];
    checksum=256-(checksum%256);
    mensaje[9]=checksum;
    break;
  case 210:
    printf("\nSelecciona blanco:");
    scanf("%d",&target);
    if(target==0)
      printf("Blanco incorrecto");
    else
    {
      mensaje[0]=0x0d;
      mensaje[1]=0x04;
      mensaje[2]=comando;
      mensaje[3]=target;
      for(i=1;i<mensaje[1];i++)
        checksum+=(unsigned int) mensaje[i];
      checksum=256-(checksum%256);
      mensaje[4]=checksum;
    }
    break;
  case 20:
    mensaje[0]=0x0d;
    mensaje[1]=0x04;
    mensaje[2]=comando;
    mensaje[3]=0x01;
    for(i=1;i<mensaje[1];i++)
      checksum+=(unsigned int) mensaje[i];
    checksum=256-(checksum%256);
    mensaje[4]=checksum;
    break;
  default:
    flag=0;
    printf("\nComando incorrecto\n");
}
if(flag)
{
  j=1;
  mensaje_corregido[0]=mensaje[0];
  for(i=1;i<mensaje[1]+1;i++)
  {
    if(mensaje[i]==0x0d)
    {
      mensaje_corregido[j++]=0x1b;
      mensaje_corregido[j]=0xf2;
    }
    else if(mensaje[i]==0x1b)
    {
      mensaje_corregido[j++]=0x1b;
      mensaje_corregido[j]=0xd8;
    }
    else
      mensaje_corregido[j]=mensaje[i];
    j++;
  }

  for(i=0;i<j+2;i++)
    mensaje[i]=mensaje_corregido[i];
}
/*if(comando==14)
{
  enviar_cadena_robosense(mensaje,j);
  flag=0;
}*/
```

```
  while(flag && comando)
  {
    enviar_cadena_robosense(mensaje,j);
    usleep(250000);
    if(mensaje_recibido[2]!=1)
      flag=1;
    else
    {
      flag=0;
      mensaje_recibido[2]=0;
    }

  }

    //enviar_cadena_robosense(mensaje,j);

}

/*********************************************************************
*
* enviar_cadena_robosense - manda la orden en cadena estructurada al
* Robosense
*
* PARAMETROS:
*
* DEVUELVE:
*
*********************************************************************/
void Robosense::enviar_cadena_robosense(unsigned char *cadena,int j)
{
  unsigned char *cadena_a_enviar;
  int i;

  cadena_a_enviar=new unsigned char[j+2];
  for(i=0;i<=j;i++)
    cadena_a_enviar[i]=cadena[i];
  enviar_cadena_puerto_serie((char *)cadena_a_enviar);
  delete cadena_a_enviar;
}

/*********************************************************************
*
* hacer_lectura_robosense - lee un carácter del Robosense
*
* PARAMETROS:
*
* DEVUELVE:
*
*********************************************************************/
void Robosense::hacer_lectura_robosense()
{
  unsigned char ch;
  unsigned char mensaje[15];
  int j;
  int i=0;
  int k=0;
  int resul=0;
  int flag=0;

  do
  {
    ch = leer_caracter_puerto_serie();

  }while(ch!=0x0d);

  calc_time_mark_robosense(); //Llego el primer caracter, ponemos la marca de tiempo

  do
  {
    resul=procesar_caracter_robosense(ch, mensaje,&i);
    ch = leer_caracter_puerto_serie();


  //printf("\nchar leido: %d\n",ch);
  } while(resul == 0);
  //strcpy(mensaje,mensaje_recibido);
```

```
  for(j=0;j<(mensaje[1]+1);j++)
  {
    if(flag==1)
      flag=0;
    else
    {
      mensaje_recibido[k]=mensaje[j];
      k++;
    }
    if(j!=0 && (mensaje[j]==0x0d || mensaje[j]==0x1b))
      flag=1;
  //}
    //printf("\nSe ha recibido: %0x\n",(unsigned short int)*(mensaje+j));
  }
  i=0;
}
/*********************************************************************
*
* procesar_caracter_robosense - almacena el caracter y valida el
* mensaje
*
* PARAMETROS:
*
* DEVUELVE:
*
*********************************************************************/
int Robosense::procesar_caracter_robosense(unsigned char c, unsigned char
*mensaje_valido,int *i)
{
  //static int ii=*i; //Estatica porque es el indice que no ha de destruirse
  int longitud;
  int j=0;
  int checksum=0;


  if((*i)==0)
  {
    //printf("\nchar mio, pero que muy mio, de siempre de toda la vida:
%u\n",mensaje_valido[*i]);
    for(j=1;j<=14;j++)
    {
      mensaje_valido[j]=0;
    }
  }
  if((c!=0x0d)&&(*i)==0)
  {
    //perror("\nError de cabecera en la transmision\n");
    return 2;
  }
  else
  {
    mensaje_valido[*i]=c;     //guardamos el caracter en mensaje
  }

  if((*i)==1)
  {
    longitud=c;               //obtenemos la longitud del mensaje
  }
  if((*i)==14)
  {
    for(j=1;j<=mensaje_valido[1];j++)
    {
      checksum+=mensaje_valido[j];
    }
    //printf("\nchecksum:%d\n",checksum);

    checksum=checksum%256;

    if(checksum==0)
    {
      return 1;               //comprobamos el checksum
    }
    else
    {
      //printf("\nError de checksum en recepcion\n");
      return 1;
    }
```

```
//hemos llegado al final, comprobamos el checksum y paramos
  }
  else
  {
    if(mensaje_valido[*i]==0xf2)
    {
      if(mensaje_valido[(*i)-1]==0x1b)
        mensaje_valido[(*i)-1]=0x0d;
    }
    if(mensaje_valido[*i]==0xd8)
    {
       if(mensaje_valido[(*i)-1]==0x1b)
        mensaje_valido[(*i)-1]=0x1b;
    }
    (*i)++;
    return 0;  //no hemos acabado, seguimos
  }
}
/*********************************************************************
*
* obtener_resultados_robosense - obtiene los resultados de la lectura
*
* PARAMETROS:
*
* DEVUELVE:
*
*********************************************************************/
void Robosense::obtener_resultados_robosense()
{
  int i;
  int x1=0;
  int y1=0;
  int numblancos=0;
  int targetid=0;
  int confianza=0;
  int numerror=0;
  int status=0;
  int rango=0;
  int angulo=0;
  short int temp=0;
  int aux_x;
  int aux_y;
  short int aux_teta;
  switch(mensaje_recibido[2])
  {
    case 0x71:
      aux_x=mensaje_recibido[4];
      aux_x=aux_x<<8;
      aux_x|=mensaje_recibido[5];
      aux_y=mensaje_recibido[7];
      aux_y=aux_y<<8;
      aux_y|=mensaje_recibido[8];
      aux_teta=mensaje_recibido[9]<<8;
      temp=mensaje_recibido[10];
      aux_teta|=temp;
      confianza=mensaje_recibido[11];
      x=aux_x/1000.0;
      y=aux_y/1000.0;
      teta=aux_teta/1000.0;
      //printf("\nPosicion tras el
mapeo,x=%d,y=%d,teta=%u,confianza=%u",x,y,teta,confianza);
    break;
    case 0x6f:
      numblancos=mensaje_recibido[3];
      targetid=mensaje_recibido[4];
      x1=mensaje_recibido[5];
      x1=x1<<8;
      x1|=mensaje_recibido[6];
      y1=mensaje_recibido[7];
      y1=y1<<8;
      y1|=mensaje_recibido[8];
      //printf("\nNumero de blancos=%u,blanco
%u,x1=%d,y1=%d",numblancos,targetid,x1,y1);
    break;
    case 0x6e:
      numblancos=mensaje_recibido[3];
      targetid=mensaje_recibido[4];
```

```
      x1=mensaje_recibido[5];
      x1=x1<<8;
      x1|=mensaje_recibido[6];
      y1=mensaje_recibido[7];
      y1=y1<<8;
      y1|=mensaje_recibido[8];
      //printf("\nNumero de blancos=%u,blanco
%u,x1=%d,y1=%d",numblancos,targetid,x1,y1);
    break;
    case 0x0f:
      numblancos=mensaje_recibido[4];
      targetid=mensaje_recibido[5];
      x1=mensaje_recibido[6];
      x1=x1<<8;
      x1|=mensaje_recibido[7];
      y1=mensaje_recibido[8];
      y1=y1<<8;
      y1|=mensaje_recibido[9];
      //printf("\nNumero de blancos=%u,blanco
%u,x1=%d,y1=%d",numblancos,targetid,x1,y1);
    break;
    case 0x17:
      aux_x=mensaje_recibido[4];
      aux_x=aux_x<<8;
      aux_x|=mensaje_recibido[5];
      aux_y=mensaje_recibido[7];
      aux_y=aux_y<<8;
      aux_y|=mensaje_recibido[8];
      aux_teta=mensaje_recibido[9];
      aux_teta=aux_teta<<8;
      aux_teta|=mensaje_recibido[10];
      confianza=mensaje_recibido[11];
      x=aux_x/1000.0;
      y=aux_y/1000.0;
      teta=aux_teta/1000.0;
      //printf("\nPosicion tras la actualizacion,x=%d,y=%d,teta=%u,nivel de
confianza=%u",x,y,teta,confianza);
    break;
    case 0x19:
      confianza=mensaje_recibido[3];
      //printf("\nConfianza=%d",confianza);
    break;
    case 0x1d:
      aux_x=mensaje_recibido[3];
      aux_x=aux_x<<8;
      aux_x|=mensaje_recibido[4];
      aux_y=mensaje_recibido[5];
      aux_y=aux_y<<8;
      aux_y|=mensaje_recibido[6];
      aux_teta=mensaje_recibido[7];
      aux_teta=aux_teta<<8;
      aux_teta|=mensaje_recibido[8];
      x=aux_x/1000.0;
      y=aux_y/1000.0;
      teta=aux_teta/1000.0;
      //printf("\nPosicion recuperada,x=%d,y=%d,teta=%u",x,y,teta);
    break;
    case 0x2b:
      numerror=mensaje_recibido[3];
      //printf("\nError %u",numerror);
    break;
    case 0x2c:
      status=mensaje_recibido[3];
      //printf("\nStatus=%u",status);
    break;
    case 0xd2:
      targetid=mensaje_recibido[3];
      rango=mensaje_recibido[4];
      rango=rango<<8;
      rango|=mensaje_recibido[5];
      angulo=mensaje_recibido[6];
      angulo=angulo<<8;
      angulo|=mensaje_recibido[7];
      //printf("\nPosicion respecto a blanco %u:
Rango=%u,Angulo=%u",targetid,rango,angulo);
    break;
    case 0x2d:
```

```
      //printf("\nRollFrame: %u",mensaje_recibido[2]);
    break;
    case 1:
    break;
    default:
      printf("\nError");
  }

  //for(i=0;i<=mensaje_recibido[1];i++)
    //printf("\nSe ha recibido: %0x\n",mensaje_recibido[i]);
}


void Robosense::calc_time_mark_robosense()
{
  struct timeval time_mark;
  double seconds;

// Obtenemos una medida del tiempo
  gettimeofday(&time_mark, NULL);

// La pasamos a segundos
  seconds = time_mark.tv_sec + time_mark.tv_usec / 1000000.0;

// Actualizamos la variable privada
  time_mark_robosense = seconds;

}


double Robosense::get_time_mark_robosense()
{
  return time_mark_robosense;
}


double Robosense::get_x_robosense()
{
  return x;
}


double Robosense::get_y_robosense()
{
  return y;
}


double Robosense::get_teta_robosense()
{
  return teta;
}


int Robosense::get_confianza_robosense()
{
  return confianza;
}
```

Las clases, robosense.hpp

```
/*********************************************************************
*
* robosense_con_tiempo.HPP
*
* Declaración de constantes, clases y funciones para trabajar con el
* sensor láser Robosense de Siman.
*
* AUTOR: Eduardo Gomez- Elegido Pedraza.
* Basado en un programa de Rafael Martin de Agar Tirado.
*
*
*
* ULTIMA MODIFICACION:
*********************************************************************/

//Para entrada salida
#include <stdio.h>
#include <iostream.h>


//Para el manejo del puerto serie
#include "../serial/serial.hpp"

//Por si acaso...
#include <math.h>

// Declaracion de la clase Robosense
class Robosense: private Puerto_serie
{
  private:
// Variables
    unsigned char mensaje_recibido[15];
    double time_mark_robosense;
    int fin_lectura_robosense;
    double x;
    double y;
    double teta;
    int confianza;




// Funciones
    void enviar_cadena_robosense(unsigned char *cadena,int j);
    int procesar_caracter_robosense(unsigned char c, unsigned char *mensaje_validado,
int *i);
    void calc_time_mark_robosense();



  public:
// Funciones
    Robosense(char *port = "/dev/robosense") : Puerto_serie(port) { };
    ~Robosense() { };

    int init_robosense();

    void mandar_comando_robosense(int comando);

    void hacer_lectura_robosense();

    void obtener_resultados_robosense();

    double get_x_robosense();
```

```
    double get_y_robosense();
    double get_teta_robosense();
    int get_confianza_robosense();
    double get_time_mark_robosense();
};
```

Las clases, romeo.cpp

```
/*********************************************************************
*
* ROMEO.CPP
*
* Implementación de las funciones definidas en ROMEO.HPP
*
* AUTOR: Rafael Martín de Agar Tirado
* MODIFICADO PARA LA CARRETILLA POR: Eduardo Gómez- Elegido Pedraza
*
* ULTIMA MODIFICACION: 10-9-2002
*********************************************************************/

/*****************************
  FALLOS QUE HAY QUE CORREGIR:
*****************************

 *-> Para las marcas de tiempo, en el caso de la DCX y la AX5411 se obtienen
 directamente en romeo.cpp, mientras que en el caso de los dispositivos del
 puerto serie se obtienen a partir de funciones miembros de las clases
 asociadas. -> Crear funciones miembro para DCX y AX que nos devuelvan la marca
 de tiempo de la última medida recibida.

 *-> Orientación calculada a partir del gyro. La orientación inicial, cuando
 no es 0, se suma cada vez que llega una medida del Gyro -> MAL!!! -> Sólo hay
 que sumarlo una vez, al principio.

***************/




// Para cout
#include <iostream.h>

#include <math.h>
#include <stdio.h>

// Para gettimeofday()
#include <sys/time.h>
#include <unistd.h>

// Para bzero()
#include <string.h>

// Definición de la clase Romeo
#include "romeo.hpp"

/*********************************************************************
*
* romeo - constructor de la clase Romeo
*
* Inicializa variables privadas de la clase.
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - void
*********************************************************************/
Romeo::Romeo()
{
// DCX
  pulsos_centro = 0;

  dir_encoder = 0;
  trac_encoder = 0;
```

```
  ref_speed = 0;
  ref_curv = 0;

  bzero(dcx_data, DCX_SIZE_OF_QUEUE*sizeof(DCX_DATA));
  dcx_index = 0;
  new_dcx_data_flag = 0;

  bzero(&dcx_est, sizeof(ESTIMATED));
  bzero(&dcx_data_temp, sizeof(DCX_DATA));

// AX5411
  bzero(ax5411_data, AX5411_SIZE_OF_QUEUE*sizeof(AX5411_DATA));
  ax5411_index = 0;
  new_ax5411_data_flag = 0;

  bzero(&ax5411_est, sizeof(ESTIMATED));
  bzero(&ax5411_data_temp, sizeof(AX5411_DATA));

// ROBOSENSE
  robosense_index=0;
  new_robosense_data_flag=0;

  bzero(robosense_data, ROBOSENSE_SIZE_OF_QUEUE*sizeof(ROBOSENSE_DATA));

  bzero(&robosense_data_temp, sizeof(ROBOSENSE_DATA));
  bzero(&robosense_est, sizeof(ESTIMATED));
  flag_de_comienzo = 0;
// ax10410
  flag_marcha_atras = 0;
  flag_arrancado = 0;
  speed_ant = 0.0;

// Varios
  bzero(&estimated, ESTIMATED_SIZE_OF_QUEUE*sizeof(ESTIMATED));

  initial_time_ref = 0;
  initial_x = 0;
  initial_y = 0;
  initial_orientation = 0;

//declaracion de las distintas constantes del PID

  kr = 0.11;
  ti = 15;
  td = 0.3;


//declaracion de las variables necesarias para el PID

  ek = 0;      //error en el instante actual
  ek1 = 0;     //error en el instante anterior
  ek2 = 0;     //error dos instantes anterior al actual
  uk = 0;      //señal de control a aplicar
  uk1 = 0;     //señal de control apicada en el instante anterior

//inicializacion de los tiempos

  time_act_pid = 0;
  time_ant_pid = 0;

}

/*********************************************************************
*
* ~romeo - destructor de la clase Romeo
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - void
*********************************************************************/
Romeo::~Romeo()
{
}
```

```
/*********************************************************************
*
* init_romeo - inicialización del vehículo
*
* Esta función se encarga tanto de la inicialización de los
* dispositivos con los que va equipado el vehículo, como de llevar al
* mismo a un estado inicial adecuado.
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - void
*********************************************************************/
int Romeo::init_romeo()
{
/*
// DCX
  if(open_dcx() < 0)
    return ERROR;

  if(load_macro() < 0)
    return ERROR;

  if(set_speed(0.0) < 0)
    return ERROR;

  if(set_curv(0.0) < 0)
    return ERROR;

// AX5411
  if (open_ax5411() < 0)
    return ERROR;

  if (set_range_ax5411(BETA_REMOLQUE, LAST_SONAR) < 0)
// Para leer sólo el sensor del remolque
//  if (set_range_ax5411(BETA_REMOLQUE, BETA_REMOLQUE) < 0)
    return ERROR;

// Si va a funcionar sin sónares -> Ganancia 2 (para el sensor del remolque)
// Si lleva sónares, el rango de valores es mayor -> Ganancia 1
  if (set_gain_ax5411(1) < 0)
    return ERROR;

// De momento, disparamos todos los sónares a la vez y los vamos leyendo
  int i = 0;
  for (i = FIRST_SONAR; i <= LAST_SONAR; i++)
  {
    act_sonar(i);
  }



// Inicializamos la referencia de tiempo inicial
  set_initial_time_ref();

// Calculamos la posición inicial
//  set_initial_position();*/


// ROBOSENSE
  if (start_robosense() < 0)
    return ERROR;

// AX10410
  if(open_ax()<0)
    return ERROR;

  centrar_potenciometro();

// Inicializamos la referencia de tiempo inicial
  set_initial_time_ref();


  return SUCCESS;
}
```

```
/*********************************************************************
* end_romeo - lleva el vehículo a un estado final adecuado
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - void
*********************************************************************/
void Romeo::end_romeo()
{
/*// Referencia de velocidad y curvatura a 0
  set_speed(0.0);
  set_curv(0.0);

// Esperamos 2 segundos antes de mandar el reset
  sleep(2);

// Reinicialización de la DCX
  send_reset_dcx();

// Desactivamos los sónares
  int i = 0;
  for (i = FIRST_SONAR; i <= LAST_SONAR; i++)
  {
    deact_sonar(i);
  }

// Paramos el envío desde el GPS
  stop_gps();*/
  close_ax();

}

/*********************************************************************
*         F U N C I O N E S    P A R A    L A    D  C  X
*********************************************************************/

/***********
* PRIVADAS *
***********/

/*********************************************************************
*
* read_encoder - lee el valor de los encoders de dirección y tracción
*
* PARAMETROS:
* - type_of_encoder which: indica el encoder que queremos leer: puede
* tomar los valores "dir" o "trac".
*
* DEVUELVE:
* - long: valor del encoder leído.
*********************************************************************/
long Romeo::read_encoder(type_of_encoder which)
{
  DWORD ReturnValue = 0;
  DWORD temp = 0;
  char Reply[4096];

// Para leer los Encoders --> Comando ¿AT o TP?

  switch(which)
  {
    case dir:
//      writeDcx("AT", 2, 0, 6);  // Para leer el encoder de dirección
      write_dcx("TP", 2, 0, 6);
      read_dcx(Reply, 4096);
// Pasamos el valor leido a radianes
      dir_encoder = *((DWORD *)&Reply[2]);
      ReturnValue = dir_encoder;
      break;
```

```
    case trac:
//      write_dcx("AT", 1, 0, 6);  // Para leer el encoder de tracción
      write_dcx("TP", 1, 0, 6);  // Para leer el encoder de tracción
      read_dcx(Reply, 4096);
      trac_encoder = *((DWORD *)&Reply[2]);
      ReturnValue = trac_encoder;
      break;

    default:
      return ERROR;
  }
  return ReturnValue;

}

/*********************************************************************
*
* fin_carrera_direc - detecta el fin de carrera derecho de la dirección
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - int: 1 si se ha detectado el fin de carrera y 0 en caso contrario.
*********************************************************************/
int Romeo::fin_carrera_direc()
{
  int limit_left = 0;
  char respuesta[20];

// Le pedimos a la DCX el estado
  write_dcx("TS",2,(DWORD) 0,6);

// Lo leemos
  read_dcx(respuesta, 20);

// Viendo el formato del estado (pág. 33 User's Manual), el bit que indica
// si hemos alcanzado el fin de carrera DERECHO se consigue así
  limit_left = (respuesta[5] & 0x20) >> 5;

  return limit_left;
}

/*********************************************************************
*
* curv_a_pulsos - convierte de curvatura a pulsos de encoder
*
* PARAMETROS:
* - float curv: curvatura
*
* DEVUELVE:
* - long: pulsos de encoder
*********************************************************************/
long Romeo::curv_a_pulsos(float curv)
{
  float aux;
  long pulsos;

// Correciones de curvatura: CORRECCION_DRCHA, CORRECCION_IZQDA
// Esto es necesario porque había un error entre curvatura en el campo y
// curvatura estimada por la fórmula

//  Fórmula original
//  aux = -atan(E*curv)*PULSO_A_RAD_DIR;

  if (curv > 0)  // Curva hacia la izquierda
  {
    aux = -atan(E*curv*CORRECCION_IZQDA)*PULSO_A_RAD_DIR;
  }
  else  // Curva hacia la derecha
  {
    aux = -atan(E*curv*CORRECCION_DRCHA)*PULSO_A_RAD_DIR;
  }

  pulsos = (long) rint(aux);
```

```
return pulsos;

}

/*********************************************************************
*
* pulsos_a_curv - convierte pulsos de encoder a curvatura
*
* PARAMETROS:
* - long pulsos: pulsos de encoder
*
* DEVUELVE:
* - float: curvatura
*********************************************************************/
float Romeo::pulsos_a_curv(long pulsos)
{
  float curv;

// Fórmula original sin corregir
//  curv = -tan( ( (float) pulsos ) / PULSO_A_RAD_DIR) / ( (float) E );

  if (pulsos < 0) // Curva hacia la izquierda
  {
    curv = -tan( ( (float) pulsos ) / PULSO_A_RAD_DIR) / ( ((float)E) *
((float)CORRECCION_IZQDA));
  }
  else
  {
    curv = -tan( ( (float) pulsos ) / PULSO_A_RAD_DIR) / ( ((float) E) * ((float)
CORRECCION_DRCHA)) ;
  }

  return curv;
}


/***********
* PUBLICAS *
***********/
/*********************************************************************
*
* set_speed - ajusta la velocidad del vehículo ROMEO 4R
*
* PARAMETROS:
* - float sp: Velocidad deseada (en m/s)
*
* DEVUELVE:
* - int: Siempre SUCCESS
*********************************************************************/
int Romeo::set_speed(float sp)
{
  float aux = 0.0;

  ref_speed = sp;

  if (sp < 0)
  {
    sp = fabs(sp);

// Y cambiamos la dirección para que vaya marcha atrás (DI 1 1)
    write_dcx("DI", 1, (DWORD) 1, 6);
  }
  else
  {
    write_dcx("DI", 1, (DWORD) 0, 6);  // DI 1 0
  }
// Speed en m/s
  if (sp > VEL_MAX_TR)
    aux = VEL_MAX_TR * MS_PULSOS_TR;
  else
  {
    aux = sp * MS_PULSOS_TR;       // Pasamos los m/s a pulsos de encoder
  }
  write_dcx("SV",1,(DWORD) aux,6);             // Y lo mandamos a la DCX
  return SUCCESS;

}
```

```
/*************************************************************************
*
* set_ax_speed - ajusta la velocidad del vehículo Carretilla
*
* PARAMETROS:
* - float sp: Velocidad deseada (en algo que todavia no conocemos)
*
* DEVUELVE:
* - int: Siempre SUCCESS
*************************************************************************/

int Romeo::set_ax_speed(float sp)
{
  if(sp != 0)
  {
    speed_ant = sp;
  }

  if (sp < 0)
  {
    arrancar_marcha_atras((int) (-1)*sp);
    flag_marcha_atras = 1;
    flag_arrancado = 1;
  }
  if(sp > 0)
  {
    arrancar_marcha_adelante((int) sp);
    flag_marcha_atras = 0;
    flag_arrancado = 1;
  }
  if(flag_marcha_atras == 0 && sp == 0 && flag_arrancado == 1)
  {
    parar_marcha_adelante((int) speed_ant+2);
    flag_arrancado = 0;
  }
  if(flag_marcha_atras == 1 && sp == 0 && flag_arrancado == 1)
  {
    parar_marcha_atras((int) (-1)*(speed_ant-1));
    flag_arrancado = 0;
  }
// De momento asi, luego, ya veremos
}

/*************************************************************************
*
* set_curv - ajusta el ángulo de giro (curvatura) del vehículo ROMEO
* 4R
*
* PARAMETROS:
* - float curv: Curvatura deseada en m-1.
*
* DEVUELVE:
* - int: Siempre SUCCESS.
*************************************************************************/

int Romeo::set_curv(float curv)
{
  float aux = 0.0;

  if (fabs(curv) > CURV_MAX)
    curv = SIGNO(curv)*CURV_MAX;

/* Pasamos la curvatura a pulsos de encoder
  aux = -atan(E*curv)*PULSO_A_RAD_DIR-pulsos_centro;*/

  aux = curv_a_pulsos(curv) - pulsos_centro;

  write_dcx("MA",2,(DWORD) aux,6);

  //ref_curv = curv;

  return SUCCESS;
}
```

```
/*************************************************************************
*
* set_ax_curv - ajusta el ángulo de giro (curvatura) del vehículo
* Carretilla
*
* PARAMETROS:
* - float curv: Curvatura deseada en m-1.
*
* DEVUELVE:
* - int: Siempre SUCCESS.
*************************************************************************/

int Romeo::set_ax_curv(double curv)
{
  girar_rueda(curv);
  return SUCCESS;     //De momento, así, luego, ya veremos
}


/*************************************************************************
*
* set_ref_curv - pone la referencia de curvatura
*
*
*
*************************************************************************/

int Romeo::set_ref_curv(double nueva_curv)
{
  ref_curv = nueva_curv;
}


/*************************************************************************
*
* get_ref_speed - devuelve el valor de la última referencia de
* velocidad
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - void
*************************************************************************/
double Romeo::get_ref_speed()
{
  return ref_speed;
}

/*************************************************************************
*
* get_ref_curv - devuelve el valor de la última referencia de
* curvatura
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - void
*************************************************************************/
double Romeo::get_ref_curv()
{
  return ref_curv;
}
```

```
/*****************************************************************************
*
* centrar_volante_modo_posición - centra el volante en modo posición
*
* Esta función opera en modo posición, tratando de centrar el volante.
* Para ello, el procedicimiento que se sigue es ir girando el volante hacia
* la derecha hasta que se detecta el fin de carrera. En ese momento, y
* conocido el número de pulsos de encoder desde la derecha hasta el centro, se
* procede a centrar el volante. Además, se actualiza el valor de
* "pulsos_centro".
*
* NOTA: Tenemos el problema de que si en el fichero de configuración
* se activan los límites para el motor de dirección ("LN"), entonces el
* motor se queda parado cuando llega al final de carrera, a pesar de que le
* volvemos a hacer un "MN". Entonces, la solución a la que hemos llegado
* es no poner "LN 2 0" en inicial.mcr sino al final de esta función.
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - void
*****************************************************************************/
void Romeo::centrar_volante_modo_posicion()
{
  long pulsos = 0;
  long pulsos_centrado = 0;

  int incremento = 3;

// Vamos girando el volante hacia la derecha (se incrementa el valor del
// encoder) hasta que se detecta el fin de carrera
  while (fin_carrera_direc() == 0)
  {
    write_dcx("MA",2,(DWORD) pulsos,6);
    pulsos += incremento;
    usleep(5000);
  }

// Paramos el motor
  write_dcx("AB",2,(DWORD) 0, 6);

// ¿Por qué no se mueve el motor si están activados los límites con "LN"
// a pesar de que le volvemos a dar un "MN"
//  write_dcx("MN",2, (DWORD) 0, 6);

// Esperamos 1 segundo, para evitar cambios bruscos de movimiento en el motor
  write_dcx("WA",0,(DWORD) 1000,6);

// Calculamos los pulsos de encoder para tener el volante centrado
  pulsos_centrado = pulsos - PULSOS_CENTRO_DERECHA;

// Lo escribimos
  write_dcx("MA",2,(DWORD) pulsos_centrado,6);

// Actualizamos la variable privada "pulsos_centro"
  pulsos_centro = -pulsos_centrado;

// Volvemos a activar el control de los límites para el volante, de forma
// que se pare el motor si se alcanzan alguno de los dos límites (y evitar
// que se queme el motor)

// Pero antes nos esperamos 1 seg. para que dé tiempo a salir desactivarse
// el fin de carrera
  write_dcx("WA",0,(DWORD) 1000,6);
  write_dcx("LN",2,(DWORD) 0, 6);

// Nos esperamos varios segundos para que dé tiempo a centrarse el volante
  write_dcx("WA",0,(DWORD) 5000,6);


}
```

```
/*************************************************************************
*
* centrar_volante_modo_velocidad - centra el volante en modo velocidad
*
* NO IMPLEMENTADA AUN!
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - void
*************************************************************************/
void Romeo::centrar_volante_modo_velocidad()
{
}



/*************************************************************************
*
* centrar_volante_carretilla - centra el volante de la carretilla
*
* NO IMPLEMENTADA AUN!
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - void
*************************************************************************/
void Romeo::centrar_volante_carretilla()
{
}
/*************************************************************************
*
* update_dcx - función de lectura y actualizacion de estiamaciones de
* la DCX
*
* En primer lugar, toma dos medidas sucesivas del encoder de tracción,
* separadas por un tiempo que se pasa como parámetro, y
* a partir de las mismas realiza una estimación de la velocidad del
* vehículo.
* El resultado es almacenado en la estructura de datos de la DCX.
* En segundo lugar, lee el encoder de dirección y almacena el
* resultado en la estructura de datos.
* En tercer lugar, se guarda la marca de tiempo.
* Por último, se actualiza el índice del array y se activa la bandera
* de nuevos datos.
*
* PARAMETROS:
* - long t_espera: tiempo de espera entre dos medidas consecutivas del
* encoder
* de tracción para estimar la velocidad.
*
* DEVUELVE:
* - int: ERROR si se ha producido algún fallo o SUCCESS en caso
* contrario.
*
*************************************************************************/
int Romeo::update_dcx(long t_espera)
{
  double aux = 0;

  double t_inicial, t_final;
  double tiempo_total;
  long enc_inicial, enc_final, enc_total;


// ESTIMACION DE LA VELOCIDAD

// Primera medida del encoder
  enc_inicial = read_encoder(trac);
  t_inicial = get_relative_time();

// Tiempo que dejamos pasar entre mediciones: cto. mayor, más exacta la
// medición, pero más lenta la actualización. ¡Compromiso!
// Este usleep() es interrumpido en cada iteración del bucle ppal, porque
```

```
// usa el mismo temporizador que el timer. Por muy grande que le pongamos
// el valor, se verá interrumpida antes.
  usleep(t_espera);

  enc_final = read_encoder(trac);
  t_final = get_relative_time();

  tiempo_total = t_final - t_inicial;

// Nos basamos en la referencia de velocidad teórica para saber el sentido de
// giro.
// Sabiendo el sentido, podemos saber si se ha producido desbordamiento,
// en cuyo caso devolvemos ERROR.
  if ( (ref_speed > 0) && (enc_final < enc_inicial) || \
       (ref_speed < 0) && (enc_final > enc_inicial) )
  {
    perror("update_real_speed: desbordamiento\n");
    return ERROR;
  }

  enc_total = enc_final - enc_inicial;

  aux = (enc_total / tiempo_total) / (PULSOS_TR_METRO);
// Fin de estimación de velocidad

  dcx_data[dcx_index].dcx_speed = aux;

// Cálculo de la curvatura
  aux = pulsos_a_curv(read_encoder(dir) + pulsos_centro) ;

  dcx_data[dcx_index].dcx_curv = aux;

// Ponemos la marca de tiempo
  dcx_data[dcx_index].time = get_relative_time();

// Terminado el relleno, actualizamos el índice
  dcx_index++;
  dcx_index %= DCX_SIZE_OF_QUEUE;

// Activamos la bandera de nuevos datos
  new_dcx_data_flag = 1;

  return SUCCESS;
}

/*********************************************************************
*
* get_dcx_data - devuelve la estructura completa de datos leidos de la
* DCX
*
* Se le pasa un índice como parámetro, que indica el elemento del
* buffer al que queremos acceder. Un valor 0 indica el último, 1
* el anterior,...
*
* PARAMETROS:
* - int index: Indice del elemento al que queremos acceder.
*
* DEVUELVE:
* - DCX_DATA: estructura de datos de la DCX.
*********************************************************************/
DCX_DATA Romeo::get_dcx_data(int index)
{
  int aux;

  aux = dcx_index - index - 1;
  if ( aux < 0 )
    aux += DCX_SIZE_OF_QUEUE;

  return dcx_data[aux];
}
```

```
/*********************************************************************
*
* new_dcx_data - comprueba si existen nuevos datos leídos de la DCX
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - int: 1 si existen nuevos datos, y 0 en caso contrario.
*********************************************************************/
int Romeo::new_dcx_data()
{
  if (new_dcx_data_flag == 1)
  {
    new_dcx_data_flag = 0;
    return 1;
  }
  else
    return 0;
}

/*********************************************************************
*         F U N C I O N E S    D E    L A    A X 5 4 1 1
*********************************************************************/

/***********
* PRIVADAS *
***********/


/***********
* PUBLICAS *
***********/
/*********************************************************************
*
* update_ax5411 - función de lectura y actualiz. de estimaciones de la
* AX5411
*
* En primer lugar, realiza una nueva medida de las entradas analógicas
* programadas.
* Hecho esto, las almacenamos en nuestro array de estructuras,
* transformándolas convenientemente en las unidades deseadas.
* Y ya por último, le ponemos la marca de tiempo, actualizamos el
* índice del array y activamos la bandera de nuevos datos.
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - int: ERROR si se ha producido algún fallo o SUCCESS en caso
* contrario.
*
*********************************************************************/
int Romeo::update_ax5411(char *cadena_sonares)
{
// Programamos el rango de la AX5411 para que lea la entrada del remolque
  set_range_ax5411(BETA_REMOLQUE, BETA_REMOLQUE);
  update_all_input_ax5411();

// Leemos todas las entradas ya actualizadas y las almacenamos convenientemente
// Beta remolque

  ax5411_data[ax5411_index].beta_remolque =
     (read_analog_input_ax5411(BETA_REMOLQUE)-PULSOS_CENTRO)*PULSOS_A_VOLTIOS*
     VOLTIOS_A_GRADOS*GRADOS_A_RADIANES;



// Sónares
// Tenemos que tener en cuenta que los sónares están "enchufados" en las
// entradas 1-12 (en la 0 está el sensor del carrito)
  int i;
  for (i = FIRST_SONAR; i <= LAST_SONAR; i++)
  {
    ax5411_data[ax5411_index].sonar[i-FIRST_SONAR] = read_sonar(i-FIRST_SONAR+1);
//                            read_analog_input_ax5411(i-FIRST_SONAR+1);
```

```
  }

// Terminada la lectura ponemos la marca de tiempo
  ax5411_data[ax5411_index].time = get_relative_time();

// Y actualizamos el índice ax5411_index
  ax5411_index++;
  ax5411_index %= AX5411_SIZE_OF_QUEUE;

// Activamos la bandera de nuevos datos
  new_ax5411_data_flag = 1;

  return SUCCESS;
}

/*****************************************************************
*
* act_sonar - activa la lectura de un sónar
*
* PARAMETROS:
* - int i: sónar que queremos leer
*
* DEVUELVE:
* - void
*****************************************************************/
void Romeo::act_sonar(int i)
{
  write_digital_output_ax5411(i-1, 1);
}

/*****************************************************************
*
* deact_sonar - desactiva la lectura de un sónar
*
* PARAMETROS:
* - int i: sónar que queremos desactivar
*
* DEVUELVE:
* - void
*****************************************************************/
void Romeo::deact_sonar(int i)
{
  write_digital_output_ax5411(i-1, 0);

}

/*****************************************************************
*
* get_ax5411_data - devuelve la estructura completa de datos leidos de
* la AX5411
*
* Se le pasa un índice como parámetro, que indica el elemento del
* buffer al que queremos acceder. Un valor 0 indica el último, 1
* el anterior,...
*
* PARAMETROS:
* - int index: Indice del elemento al que queremos acceder.
*
* DEVUELVE:
* - AX5411_DATA: estructura de datos de la AX5411.
*****************************************************************/
AX5411_DATA Romeo::get_ax5411_data(int index)
{
  int aux;

  aux = ax5411_index - index - 1;
  if ( aux < 0 )
    aux += AX5411_SIZE_OF_QUEUE;

  return ax5411_data[aux];
}
```

```
/*********************************************************************
*
* new_ax5411_data - comprueba si existen nuevos datos leídos de la
* AX5411
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - int: 1 si existen nuevos datos, y 0 en caso contrario.
*********************************************************************/
int Romeo::new_ax5411_data()
{
  if (new_ax5411_data_flag == 1)
  {
    new_ax5411_data_flag = 0;
    return 1;
  }
  else
    return 0;
}


double Romeo::read_sonar(int sonar)
{
  double leido = 0.0;
  double metros = 0.0;

// Disparamos el sónar
//  act_sonar(sonar);

// Programamos la AX5411 para que sólo lea la entrada de dicho sónar
  set_range_ax5411(sonar + FIRST_SONAR - 1, sonar + FIRST_SONAR - 1);

// Realizamos una nueva medida de la entrada
//int i=0;
//while (i++ < 1000)
//  usleep(500000);

  update_all_input_ax5411();

// Leemos el valor medido
  leido = read_analog_input_ax5411(sonar + FIRST_SONAR - 1);

//cout << sonar << " " << get_relative_time() << "  " << leido << endl;



// Desactivamos el sónar para que no interfiera con los demás
//  deact_sonar(sonar);

//cout << get_relative_time() << "  " << leido << endl;
// Devolvemos la medida convertida a distancia según el sónar que sea
// Éstos son los de 6 m.
  if ((sonar == 1) || (sonar == 4) || (sonar == 6) ||
     (sonar == 7) || (sonar == 10) || (sonar == 12))
    return (leido*SONAR_6_A + SONAR_6_B);
// Éstos son los de 3 m.
  if ((sonar == 3) || (sonar == 5) || (sonar == 9) || (sonar == 11))
    return (leido*SONAR_3_A + SONAR_3_B);
}
/*********************************************************************
*               F U N C I O N E S    D E L    R O B O S E N S E
*********************************************************************/
```

```
/**********************************************************************
*
* start_robosense - función que prepara al robosense para navegar
*
* Antes de utilizar el robosense, hemos de aprender el mapa en el que
* se encuentra para luego poder navegar por él.
*
*
* PARÁMETROS:
* - void
*
* DEVUELVE:
* - int: ERROR si se ha producido algún fallo o SUCCESS en caso
* contrario.
*
**********************************************************************/
int Romeo::start_robosense()
{
int i=SUCCESS;

i=init_robosense();

/*mandar_comando_robosense(14);

sleep(10);

mandar_comando_robosense(12);*/

return i;
}

/**********************************************************************
*
* update_robosense - función de lectura y actual. de estimaciones del
* ROBOSENSE
*
* En primer lugar, hacemos una lectura del ROBO. Así, nos quedaremos
* bloqueados hasta que el ROBO mande nuevos datos.
* Una vez recibidos, almacenamos aquellos datos que nos interesen en
* una de las estructuras del array.
* Después, hacemos estimaciones de diversos parámetros a partir de los
* datos recibidos.
* Y ya por último, incrementamos el índice del array de las
* estructuras y activamos la bandera de nuevos datos del ROBO.
*
* PARAMETROS:
* - void
*
* DEVUELVE:
* - int: ERROR si se ha producido algún fallo o SUCCESS en caso
* contrario.
*
**********************************************************************/
int Romeo::update_robosense()
{
  double aux = 0;

  hacer_lectura_robosense();
  obtener_resultados_robosense();

// Ya tenemos la posicion del vehiculo. Ahora tenemos que rellenar
// la estructura de datos del Robosense


  robosense_data[robosense_index].x_robo = get_x_robosense();
  robosense_data[robosense_index].y_robo = get_y_robosense();
  robosense_data[robosense_index].time = get_time_mark_robosense() -
                                         initial_time_ref;
  robosense_data[robosense_index].robo_orientation = get_teta_robosense();
  if(robosense_data[robosense_index].robo_orientation < 0 ||
                robosense_data[robosense_index].robo_orientation >
                2 * 3.14159265)
    robosense_data[robosense_index].robo_orientation = 0;
  robosense_data[robosense_index].confianza = get_confianza_robosense();


// Calculo del incremento de teta
```

```
// Nos hacen falta datos anteriores del Robosense
  ROBOSENSE_DATA robosense_data_ant;
  robosense_data_ant = get_robosense_data(0);


  aux = estimate_robosense_delta_teta(robosense_data[robosense_index].robo_orientation,
robosense_data_ant.robo_orientation);

  robosense_data[robosense_index].robo_delta_teta = aux;

// Estimación de la velocidad a partir del Robosense
  aux = estimate_robosense_robo_speed(robosense_data[robosense_index].x_robo,
                                robosense_data[robosense_index].y_robo,
                                robosense_data[robosense_index].time,
                                robosense_data_ant.x_robo,
                                robosense_data_ant.y_robo,
                                robosense_data_ant.time);

  robosense_data[robosense_index].robo_speed = aux;

// Estimacion de la longitud de arco recorrido a partir del Robosense
  aux = estimate_robosense_delta_s(robosense_data[robosense_index].x_robo,
                                  robosense_data[robosense_index].y_robo,
                                  robosense_data_ant.x_robo,
                                  robosense_data_ant.y_robo);

  robosense_data[robosense_index].robo_delta_s = aux;

// Estimación de la curvatura a partir del robosense
  aux = estimate_robosense_curv(robosense_data[robosense_index].robo_delta_s,
                              robosense_data[robosense_index].robo_delta_teta);
  if(aux > 2)
    aux = 2;
      /* ||(robosense_data_ant.x_robo == 0 &&
                robosense_data_ant.y_robo == 0))*/
    robosense_data[robosense_index].robo_curv = aux;
  /*else
    robosense_data[robosense_index].robo_curv = robosense_data_ant.robo_curv;*/

// Terminado el relleno, actualizamos el índice
  robosense_index++;
  robosense_index %= ROBOSENSE_SIZE_OF_QUEUE;

// Activamos la bandera de nuevos datos
  new_robosense_data_flag = 1;
}

/*********************************************************************
*
* estimate_robosense_delta_teta - incremento de orientación a partir
* del ROBOSENSE
*
* Realiza una estimación del cambio de la orientación del vehículo a
* partir de datos obtenidos del Robosense.
* El proceso es restar las dos últimas orientaciones del Robosense.
* Este ángulo viene dado en RADIANES.
*
* PARAMETROS:
* - double teta_act: Orientacion actual.
* - double int teta_ant: Orientacion anterior.
*
* DEVUELVE:
* - double: El valor de la diferenciade orientacion (en RADIANES,
* insisto)
*********************************************************************/
double Romeo::estimate_robosense_delta_teta(double teta_act, double teta_ant)
{
  double delta_teta;
  if(teta_act > 5.80 && teta_ant < 0.5)
    teta_act -= 2 * 3.14159265;
  if(teta_act < 0.5 && teta_ant > 5.80)
    teta_act += 2 * 3.14159265;

  /*if(teta_ant > teta_act)
    delta_teta = -((double)(teta_act - teta_ant));
  else*/
    delta_teta = teta_act-teta_ant;
```

```
  return delta_teta;
}

/*********************************************************************
*
* estimate_robosense_robo_speed - estimación de la velocidad a partir
* del Robosense
*
* Esta función, a partir de dos puntos de la trayectoria y los tiempos
* en los que se pasó por cada uno, realiza una estimación de la
* velocidad.
*
* PARAMETROS:
* - double x_act: coordenada X del punto actual.
* - double y_act: coordenada Y del punto actual.
* - double t_act: instante de tiempo del punto actual.
* - double x_ant: coordenada X del punto anterior.
* - double y_ant: coordenada Y del punto anterior.
* - double t_ant: instante de tiempo del punto anterior.
*
* DEVUELVE:
* - double: estimación de la velocidad calculada.
*********************************************************************/
double Romeo::estimate_robosense_robo_speed(double x_act, double y_act,
               double t_act, double x_ant, double y_ant, double t_ant)
{
  double aux;
  aux = (double)sqrt((y_act-y_ant)*(y_act-y_ant) + (x_act-x_ant)*(x_act-x_ant));

  aux /= (t_act - t_ant);

  return aux;

}

/*********************************************************************
*
* estimate_robosense_delta_s - incremento del parametro arco a partir
* del ROBOSENSE
*
* Realiza una estimación del cambio del parametro arco a partir de
* datos obtenidos del Robosense.
* El proceso es hallar el espacio recorrido entre las dos ultimas
* estimaciones suponiendo que se va en linea recta.
*
* PARAMETROS:
*
*
* DEVUELVE:
* - double: El valor de la orientación calculado (en RADIANES,
* insisto)
*********************************************************************/

double Romeo::estimate_robosense_delta_s(double x_act, double y_act, double x_ant,
double y_ant)
{
  double delta_s;
  delta_s = (double)fabs(sqrt((y_act-y_ant)*(y_act-y_ant) + (x_act-x_ant)*(x_act-
x_ant)));
  return delta_s;
}
```

```
/*********************************************************************
*
* estimate_robosense_curv - estima la curvatura a partir del ROBOSENSE
*
* Realiza una estimación de la curvatura a partir de
* datos obtenidos del Robosense.
* Se trata de dividir el incremento de orientación entre el incremento
* de arco, la distancia recorrida.
*
* PARAMETROS:
*
*
* DEVUELVE:
* - double: El valor de la curvatura calculada
*********************************************************************/
double Romeo::estimate_robosense_curv(double deltas, double deltateta)
{
  double curv;
  if( deltas != 0)
  {
    curv = deltateta/deltas;
  }
  else
  {
    curv = 0;
  }
  return curv;
}
/*********************************************************************
*
* get_robosense_data - devuelve la estructura de datos del Robosense
* indicada por el indice del array.
*
* Si el índice es 0, se devuelve la última estructura de datos
* rellenada.
* Si es 1, la anterior. Y así sucesivamente.
*
* PARAMETROS:
* - int index: índice del array.
*
* DEVUELVE:
* - ROBOSENSE_DATA: estructura de datos del ROBOSENSE correspondiente.
*********************************************************************/
ROBOSENSE_DATA Romeo::get_robosense_data(int index)
{
  int aux;
  aux = robosense_index - index - 1;
  if ( aux < 0 )
    aux += ROBOSENSE_SIZE_OF_QUEUE;
    return robosense_data[aux];
}

/*********************************************************************
*
* new_robosense_data - función que determina si existe dato nuevo del
* Robosense.
*
* PARAMETROS:
*
* - void
*
* DEVUELVE:
* - int: 1 si existe nuevo dato del Robosense y 0 en caso contrario.
*********************************************************************/
int Romeo::new_robosense_data()
{
  if (new_robosense_data_flag == 1)
  {
    new_robosense_data_flag = 0;
    return 1;
  }
  else
    return 0;
}
```

```
// FUNCIONES DE TIEMPO
/*************************************************************************
*
* set_initial_time_ref - establece la referencia de tiempo inicial
*
* PARAMETROS:
* - void.
*
* DEVUELVE:
* - void.
*************************************************************************/
void Romeo::set_initial_time_ref()
{
  struct timeval temp;
  gettimeofday(&temp, NULL);

  initial_time_ref = temp.tv_sec + temp.tv_usec / 1000000.0;

}

/*************************************************************************
*
* get_relative_time - devuelve el instante actual referido al instante
* inicial.
*
* PARAMETROS:
* - void.
*
* DEVUELVE:
* - double: instante actual relativo.
*************************************************************************/
double Romeo::get_relative_time()
{
  struct timeval temp;

  gettimeofday(&temp, NULL);

  return ((temp.tv_sec + temp.tv_usec/1000000.0)- initial_time_ref);
}

// ODOMETRIA
/*************************************************************************
*
* odometria - función de odometría usando el modelo de la bicicleta.
*
* Se realiza una estimación de la posición y orientación actuales a
* partir de
* la posición y orientación anteriores, las diferencias de tiempo, y
* la velocidad y curvatura.
*
* PARAMETROS:
* - double *x: primero incluye el valor anterior de X y después se
* modifica con el valor actual.
* - double *y: primero incluye el valor anterior de Y y después se
* modifica con el valor actual.
* - double *orient: primero incluye el valor anterior de orientación y
* después se modifica con el valor actual.
* - double vel: velocidad en el instante anterior.
* - double curv: curvatura en el instante anterior.
* - double *time_ant: primero incluye el instante anterior de tiempo y
* después el instante actual.
* - double time_act: instante actual.
*
* DEVUELVE:
* - void.
*************************************************************************/
void Romeo::odometria(double *x, double *y, double *orient,
                      double vel, double curv,
                      double *time_ant, double time_act)
{
  double dx, dy, dorient;
```

```
  dx = vel * cos(*orient);
  dy = vel * sin(*orient);
  dorient = vel * curv;
  *x += dx * (time_act - *time_ant);
  *y += dy * (time_act - *time_ant);
  *orient += dorient * (time_act - *time_ant);
  *time_ant = time_act;
}

/*********************************************************************
* update_estimacion
*
*
*********************************************************************/
int Romeo::update_estimacion()
{
// Cuando comenzamos la iteración, calculamos el instante actual
    time_act = get_relative_time();
extern long iteraciones;
int iii = 0;
double curvatura_media = 0;
double divisor = 0;



/************************
* Nuevos datos de la DCX *
************************/
    if (new_dcx_data())
    {
// Leemos los nuevos valores de la DCX
      dcx_data_temp = get_dcx_data(0);

// Actualizamos los valores de la estimación anterior al instante en que
// se leyeron nuevos datos de la DCX
      odometria_dcx(&dcx_est.x, &dcx_est.y, &dcx_est.orient,
                  dcx_est.speed, dcx_est.curv,
                  &dcx_est.time, dcx_data_temp.time);
       // Actualizamos los valores desde el instante de los datos de la DCX
        // hasta el instante actual

// Para la velocidad tomamos el valor medio entre la medida anterior y la
       // actual
      dcx_est.speed = 0.5 * (dcx_data_temp.dcx_speed + dcx_est.speed);
      dcx_est.curv = dcx_data_temp.dcx_curv;
      odometria_dcx(&dcx_est.x, &dcx_est.y, &dcx_est.orient,
                  dcx_est.speed, dcx_est.curv,
                  &dcx_est.time, time_act);
    }

/***************************
* Nuevos datos de la AX5411 *
***************************/
     if (new_ax5411_data())
     {
// Actualizamos la estructura de datos de la AX5411
       ax5411_data_temp = get_ax5411_data(0);

// Posible procesamiento de los nuevos datos

     }


/*******************************
* Nuevos datos del Robosense   *
*******************************/
     if (new_robosense_data())
     {
// Leemos los nuevos datos del Robosense
       robosense_data_temp = get_robosense_data(0);
// Actualizamos los datos desde la estimación anterior hasta el instante
// en que llegaron los nuevos datos
       odometria(&robosense_est.x, &robosense_est.y, &robosense_est.orient,
            robosense_data_temp.robo_speed, robosense_data_temp.robo_curv,
                  &robosense_est.time, robosense_data_temp.time);
// Actualizamos desde que llegaron los nuevos datos hasta el instante actual
```

```
// Para ello, sustituimos el valor predicho por la odometría para x e y por el
// valor recibido del Robosense, SIEMPRE Y CUANDO la distancia entre ambos puntos
// sea menor que un valor límite. En caso contrario, mantenemos el punto
// predicho por la odometría.

       if ((sqrt( pow(fabs(robosense_est.x - robosense_data_temp.x_robo),2) +
                pow(fabs(robosense_est.y - robosense_data_temp.y_robo),2) +
                       pow(fabs(robosense_est.orient -
                       robosense_data_temp.robo_orientation),2))
                       < 2.2) || (flag_de_comienzo == 0));
       {
        robosense_est.x = robosense_data_temp.x_robo;
        robosense_est.y = robosense_data_temp.y_robo;
        robosense_est.orient = robosense_data_temp.robo_orientation;
        flag_de_comienzo = 1;
       }
       odometria(&robosense_est.x, &robosense_est.y, &robosense_est.orient,
                robosense_data_temp.robo_speed, robosense_data_temp.robo_curv,
                &robosense_est.time, time_act);

     }
// Y ahora realizamos la odometria normal, para cuando no haya datos del
// Robosense
     odometria(&robosense_est.x, &robosense_est.y, &robosense_est.orient,
                           robosense_data_temp.robo_speed,
                           robosense_data_temp.robo_curv,
                           &robosense_est.time, time_act);

// Aquí realizamos las estimaciones de los parámetros de interés según la
// fiabilidad de las medidas.
// Actualmente se está usando la odometría basada en el Gyro, sin GPS.

// Esto hay q descomentarlo cuando no estemos simulando

    estimated[est_index].speed = robosense_data_temp.robo_speed;
    //if (iteraciones<400){
    //estimated[est_index].curv -= 0.000007;
    //if(estimated[est_index].curv <= -2) estimated[est_index].curv = -2; }
    //else{
    //estimated[est_index].curv -= 0.000007;
    //if(estimated[est_index].curv >= 1) estimated[est_index].curv = 1; }
    for(iii = 1;iii == ROBOSENSE_SIZE_OF_QUEUE; iii++)
    {
      curvatura_media += get_robosense_data(iii).robo_curv;
    }
    if(fabs(robosense_data_temp.robo_curv - curvatura_media) < 1.5)
    {
      curvatura_media += robosense_data_temp.robo_curv;
      curvatura_media /= 5.;
    }
    else
      curvatura_media /=4.;
    //cout << curvatura_media << endl;
    //estimated[est_index].curv = robosense_data_temp.robo_curv;
    estimated[est_index].curv = curvatura_media;
    estimated[est_index].x = robosense_est.x;
    estimated[est_index].y = robosense_est.y;
    estimated[est_index].orient = robosense_est.orient;
    estimated[est_index].time = time_act;
    estimated[est_index].beta_remolque = 0;//ax5411_data_temp.beta_remolque;
// Terminado el relleno, actualizamos el índice
    est_index++;
    est_index %= ESTIMATED_SIZE_OF_QUEUE;
// Activamos la bandera de nuevos datos
  new_estimated_status_flag = 1;
  return SUCCESS;
}

ESTIMATED Romeo::get_estimated_status(int index)
{
  int aux;

  aux = est_index - index - 1;
  if ( aux < 0 )
    aux += ESTIMATED_SIZE_OF_QUEUE;

    return estimated[aux];
```

```
}

/*************************************************************************
*
* update_PID_direccion
*
*
*
*
*************************************************************************/

void Romeo::update_PID_direccion()
{
    double delta_time = 0;
    double a0;
    double a1;
    double a2;
    double uk_aux;

    time_act_pid = get_relative_time();

    delta_time = time_act_pid - time_ant_pid;

    while(delta_time < 0.005)
    {
      time_act_pid = get_relative_time();
      delta_time = time_act_pid - time_ant_pid;
    }
    a0 = kr*((1 + td/delta_time) + delta_time/(2*ti));
    a1 = kr*((-1 -2*td/delta_time) + delta_time/(2*ti));
    a2 = kr*(td/delta_time);
    uk = uk1 + a0*ek + a1*ek1 + a2*ek2;     //calculamos la señal de control
    //uk -= 0.01;
    if(uk < -0.1) uk_aux = uk -0.8;
    else if(uk > 0.1) uk_aux = uk + 0.3;
    else uk_aux = uk_aux;

    set_ax_curv(uk);  //actuamos
//actualizamos las variables para el siguiente instante
    uk1 = uk;
    ek1 = ek;
    ek2 = ek1;
    ek = ref_curv - get_estimated_status(0).curv;
    //cout << uk << endl;
    //ek = -1 - get_estimated_status(0).curv;

    time_ant_pid = time_act_pid;
}
```

Las clases, romeo.hpp

```
/*********************************************************************
*
* ROMEO.HPP
*
* Declaración de clases, funciones, tipos de datos, ... para usar en
* programas de control para el vehículo ROMEO 4R y para el vehículo
* Carretilla
*
* AUTOR: Rafael Martín de Agar Tirado
* MODIFICADO PARA LA CARRETILLA POR: Eduardo Gómez- Elegido Pedraza
*
* ULTIMA MODIFICACION: 10-9-2002
*********************************************************************/
// Sus .h
#include <alloc.h>


// Mis .h
#include "./dcx/dcx.hpp"
#include "./ax5411/ax5411.hpp"
#include "./robosense/robosense.hpp"
#include "./matematicas/matematicas.h"
#include "./ax10410/ax10410.hpp"
#include "romeo4.h"   //¡¡¡¡¡OJITO con los directorios!!!!!

// DCX
enum type_of_encoder {dir, trac} ;

#define DCX_SIZE_OF_QUEUE 5

// Estructura para albergar datos de la DCX con sus marcas de tiempo
typedef struct
{
  double dcx_speed;
  double dcx_curv;
  double time;
} DCX_DATA;

// AX5411
#define AX5411_SIZE_OF_QUEUE 5

typedef struct
{
  double sonar[NUM_SONAR];
  double beta_remolque;
  double time;
} AX5411_DATA;

// GPS

#define GPS_SIZE_OF_QUEUE 5

// Estructura para albergar datos del GPS con sus marcas de tiempo
typedef struct
{
  double x_utm;
  double y_utm;
  double time;
  double gps_speed;
  double gps_orientation;
  int dif_mode;
} GPS_DATA;

// ROBOSENSE

#define ROBOSENSE_SIZE_OF_QUEUE 5
```

```
// Estructura para albergar datos del Robosense con sus marcas de tiempo
typedef struct
{
  double x_robo;
  double y_robo;
  double time;
  double robo_orientation;
  int confianza;
  double robo_delta_teta;
  double robo_delta_s;
  double robo_speed;
  double robo_curv;
} ROBOSENSE_DATA;


#define ESTIMATED_SIZE_OF_QUEUE 5
// Estructura para mantener la última estimación de las variables de interés
typedef struct
{
  double x;
  double y;
  double orient;
  double dorient;
  double speed;
  double curv;
  double time;

  double beta_remolque;
} ESTIMATED;


// Clase Romeo. Hereda de tantas clases como dispositivos tenga incorporado.
class Romeo: public DCX, public AX5411, public AX, public Robosense
{
/****************
* P R I V A D O *
****************/
  private:

/******
* DCX *
******/
// Variables privadas

    long pulsos_centro;

    DWORD dir_encoder;
    DWORD trac_encoder;

    double ref_speed;
    double ref_curv;

    DCX_DATA dcx_data[DCX_SIZE_OF_QUEUE];
    int dcx_index;
    int new_dcx_data_flag;

    DCX_DATA dcx_data_temp;

    ESTIMATED dcx_est;
//declaracion de las distintas constantes del PID

  double kr;
  double ti;
  double td;


//declaracion de las variables necesarias para el PID

  double ek;   //error en el instante actual
  double ek1;  //error en el instante anterior
  double ek2;  //error dos instantes anterior al actual
  double uk;   //señal de control a aplicar
  double uk1;  //señal de control apicada en el instante anterior


// Funciones privadas
    long read_encoder(type_of_encoder);
```

```
    int fin_carrera_direc();

    long curv_a_pulsos(float curv);
    float pulsos_a_curv(long pulsos);
/*****
* AX *
*****/
// Variables privadas

    double ref_ax_speed;
    double ref_ax_curv;
    float speed_ant;

// Funciones privadas

/*********
* AX5411 *
*********/
// Variables privadas
    AX5411_DATA ax5411_data[AX5411_SIZE_OF_QUEUE];
    int ax5411_index;
    int new_ax5411_data_flag;

    AX5411_DATA ax5411_data_temp;

    ESTIMATED ax5411_est;

// Funciones privadas
  public:   // Para pruebas... QUITAR!!!
    void act_sonar(int i);
    void deact_sonar(int i);
  private:

/************
* Robosense *
************/
// Variables privadas

    ROBOSENSE_DATA robosense_data[ROBOSENSE_SIZE_OF_QUEUE];
    int robosense_index;
    int new_robosense_data_flag;

    ROBOSENSE_DATA robosense_data_temp;

    ESTIMATED robosense_est;

    int flag_de_comienzo;

// Funciones privadas
    double estimate_robosense_delta_teta(double teta_act, double teta_ant);
    double estimate_robosense_robo_speed(double x_act, double y_act,
                  double t_act,double x_ant, double y_ant, double t_ant);
    double estimate_robosense_delta_s(double x_act, double y_act,
                  double x_ant, double y_ant);
    double estimate_robosense_curv(double deltas,double deltateta);


/*********
* Varios *
*********/
// Variables privadas
    double initial_time_ref;
    double initial_x;
    double initial_y;
    double initial_orientation;

    ESTIMATED estimated[ESTIMATED_SIZE_OF_QUEUE];
    int est_index;
    int new_estimated_status_flag;

    double time_act;
    double time_act_pid;
    double time_ant_pid;

// Funciones privadas
    void set_initial_time_ref();
    void set_initial_orientation();
```

```
/****************
* P U B L I C O *
****************/
  public:

/********
* ROMEO *
********/
    Romeo();
    ~Romeo();

    int init_romeo();
    void end_romeo();

/******
* DCX *
******/
// Funciones públicas
    int set_speed(float sp);
    int set_curv(float curv);

    double get_ref_speed();
    double get_ref_curv();

    void centrar_volante_modo_posicion();
    void centrar_volante_modo_velocidad();
    void centrar_volante_carretilla();

    int update_dcx(long t_espera);

    DCX_DATA get_dcx_data(int index);

    int new_dcx_data();


/******
* AX *
******/
// Funciones públicas
    int set_ax_speed(float sp);
    int set_ax_curv(double curv);
    int set_ref_curv(double nueva_curv);

    double get_ref_ax_speed();
    double get_ref_ax_curv();

    void centrar_direccion_carretilla();

    int flag_marcha_atras;

    int flag_arrancado;
    /*void centrar_volante_modo_velocidad();

    int update_dcx(long t_espera);

    DCX_DATA get_dcx_data(int index);

    int new_dcx_data();*/


/*********
* AX5411 *
*********/
// Funciones públicas
    int update_ax5411(char *cadena_sonares);

    AX5411_DATA get_ax5411_data(int index);

    int new_ax5411_data();

    double read_sonar(int sonar);
```

```
/******
* GPS *
******/
// Funciones públicas
    int update_gps();

    GPS_DATA get_gps_data(int index);

    int new_gps_data();

    void estimate_gps_initial_position(int iterations);
    void estimate_gps_initial_orientation();

    double get_gps_initial_x();
    double get_gps_initial_y();


/************
* Robosense *
************/
// Funciones públicas
    int start_robosense();

    int update_robosense();

    ROBOSENSE_DATA get_robosense_data(int index);

    int new_robosense_data();

    /*void estimate_gps_initial_position(int iterations);
    void estimate_gps_initial_orientation();

    double get_gps_initial_x();
    double get_gps_initial_y();*/



/**********
*  Varios *
**********/
 // Funciones públicas
    int update_estimacion();
    ESTIMATED get_estimated_status(int index);

    double get_relative_time();
    void odometria(double *x, double *y, double *orient,
                   double vel, double curv,
                   double *time_ant, double time_act);
    void update_PID_direccion();
};
```

Los hilos, hilo_robosense.cpp

```
// Para perror()
#include <stdio.h>

// Para cout, cin,...
#include <iostream.h>

// Para los hilos
#include <pthread.h>

#include "../romeo.hpp"
#include "hilo_robosense.hpp"

pthread_t hilo_lectura_robosense;
pthread_attr_t atributos_hilo_lectura_robosense;


void iniciar_hilo_robosense()
{
  int ret;

// Inicialización de los atributos
  ret = pthread_attr_init(&atributos_hilo_lectura_robosense);
  if (ret != 0)
  {
    perror("Falló creación de atributos\n");
    exit(ERROR);
  }

// Configuración de los atributos: modo DETACHSTATE
  ret = pthread_attr_setdetachstate(&atributos_hilo_lectura_robosense,
                                    PTHREAD_CREATE_DETACHED);
  if (ret != 0)
  {
    perror("Fallo poniendo atributo 'detach'\n");
    exit(ERROR);
  }

// Lanzamos el hilo
  ret = pthread_create(&hilo_lectura_robosense, &atributos_hilo_lectura_robosense,
                       funcion_hilo_lectura_robosense, (void *) NULL);
}

void * funcion_hilo_lectura_robosense(void *)
{
  extern Romeo romeo;
  int ret;


  ret = pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
  if (ret != 0)
  {
    perror("Error en setcancelstate\n");
    exit(ERROR);
  }

  ret = pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
  if (ret != 0)
  {
    perror("Error en setcanceltype\n");
    exit(ERROR);
  }

  while(1)
  {
//    romeo.hacer_lectura_robosense();
    romeo.update_robosense();
  }
```

```
}


void destruir_hilo_robosense()
{
  pthread_cancel(hilo_lectura_robosense);
}
```

Los hilos, hilo_robosense.hpp

```
#define ERROR -1
#define SUCCESS 0

void iniciar_hilo_robosense();
void *funcion_hilo_lectura_robosense(void *);
void destruir_hilo_robosense();
```

Los hilos, hilo_estimacion.cpp

```
// Para perror()
#include <stdio.h>

// Para los hilos
#include <pthread.h>

#include "../romeo.hpp"
#include "hilo_estimacion.hpp"

pthread_t hilo_estimacion;
pthread_attr_t atributos_hilo_estimacion;


void iniciar_hilo_estimacion()
{
  int ret;

// Inicialización de los atributos
  ret = pthread_attr_init(&atributos_hilo_estimacion);
  if (ret != 0)
  {
    perror("Falló creación de atributos\n");
    exit(ERROR);
  }

// Configuración de los atributos: modo DETACHSTATE
  ret = pthread_attr_setdetachstate(&atributos_hilo_estimacion,
                                    PTHREAD_CREATE_DETACHED);
  if (ret != 0)
  {
    perror("Fallo poniendo atributo 'detach'\n");
    exit(ERROR);
  }

// Lanzamos el hilo
  ret = pthread_create(&hilo_estimacion, &atributos_hilo_estimacion,
                       funcion_hilo_estimacion, (void *) NULL);
}

void * funcion_hilo_estimacion(void *)
{
  extern Romeo romeo;
  int ret;


  ret = pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
  if (ret != 0)
  {
    perror("Error en setcancelstate\n");
    exit(ERROR);
  }

  ret = pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
  if (ret != 0)
  {
    perror("Error en setcanceltype\n");
    exit(ERROR);
  }

  while(1)
  {
    romeo.update_estimacion();
  }
}
```

```
void destruir_hilo_estimacion()
{
  pthread_cancel(hilo_estimacion);
}
```

Los hilos, hilo_estimacion.hpp

```
#define ERROR -1
#define SUCCESS 0

void iniciar_hilo_estimacion();
void *funcion_hilo_estimacion(void *);
void destruir_hilo_estimacion();
```

Los hilos, hilo_pid_direccion.cpp

```
// Para perror()
#include <stdio.h>

// Para los hilos
#include <pthread.h>

#include "../romeo.hpp"
#include "hilo_pid_direccion.hpp"

pthread_t hilo_pid_direccion;
pthread_attr_t atributos_hilo_pid_direccion;


void iniciar_hilo_pid_direccion()
{
  int ret;

// Inicialización de los atributos
  ret = pthread_attr_init(&atributos_hilo_pid_direccion);
  if (ret != 0)
  {
    perror("Falló creación de atributos\n");
    exit(ERROR);
  }

// Configuración de los atributos: modo DETACHSTATE
  ret = pthread_attr_setdetachstate(&atributos_hilo_pid_direccion,
                                    PTHREAD_CREATE_DETACHED);
  if (ret != 0)
  {
    perror("Fallo poniendo atributo 'detach'\n");
    exit(ERROR);
  }

// Lanzamos el hilo
  ret = pthread_create(&hilo_pid_direccion, &atributos_hilo_pid_direccion,
                       funcion_hilo_pid_direccion, (void *) NULL);
}

void * funcion_hilo_pid_direccion(void *)
{
  extern Romeo romeo;
  int ret;

  ret = pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
  if (ret != 0)
  {
    perror("Error en setcancelstate\n");
    exit(ERROR);
  }

  ret = pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
  if (ret != 0)
  {
    perror("Error en setcanceltype\n");
    exit(ERROR);
  }
  while(1)
  {
    romeo.update_PID_direccion();
  }
}

void destruir_hilo_pid_direccion()
{
  pthread_cancel(hilo_pid_direccion);
}
```

Los hilos, hilo_pid_direccion.hpp

```
#define ERROR -1
#define SUCCESS 0

void iniciar_hilo_pid_direccion();
void *funcion_hilo_pid_direccion(void *);
void destruir_hilo_pid_direccion();
```

El programa principal, main.cpp

```
/*********************************************************************
* MAIN.CPP
*
* Fallos detectados (funcionando como servidor):
* - Debido a fork() y a q, por tanto, los procesos se vuelven
* independientes,
* cuando un cliente cambia algún valor (x ej, la speed), ese cambio se
* produce en el vehículo realmente, pero sólo se refleja en la variable
* de ese cliente, de forma q si otro cliente lee la velocidad, el valor
* leido es erróneo.
* - Si un cliente hace un reset, la DCX se resetea, perdiendo su
* configuración.
* Si otro cliente intenta acceder, pueden ocurrir eventos
* CATASTROFICOS!!!!
* - Debido a fork() y a la forma en que hemos programado la funcion
* update_real_speed, si un usuario, en una conexión, pone el vehículo
* a una velocidad positiva y otro usuario le cambia la velocidad a una
* negativa, se produce desbordamiento en la función implementada.
*
*********************************************************************/

/*********************************************************************
*          A R C H I V O S    D E   C A B E C E R A
*********************************************************************/

/************************
* D E L   S I S T E M A *
************************/
// Entrada y salida básica
#include <iostream.h>
#include <stdio.h>

// Para atof()
#include <stdlib.h>

// Para manipulación de archivos
#include <fstream.h>
#include <iomanip.h>

// Para fork()
#include <unistd.h>

// Para bzero()
#include <string.h>

// Para floor()
#include <math.h>

// Para manejo de señales
#include <signal.h>
void signal_int_handler(int s);

// Para funciones de tiempo (ponerle el nombre al archivo de LOGS)
#include <time.h>

/************
* M I S  .H *
************/

// Definición de la clase Romeo
#include "romeo.hpp"

// Para los hilos
#include "./hilos/hilo_dcx.hpp"
#include "./hilos/hilo_ax5411.hpp"
#include "./hilos/hilo_estimacion.hpp"
#include "./hilos/hilo_robosense.hpp"
```

```
#include "./hilos/hilo_pid_direccion.hpp"

// Para el temporizador del bucle principal
#include "./timers/timers.hpp"
#include "./timers/ipc.h"

// Para funciones de comunicación cliente-servidor
#include "./socket/socket.hpp"
#include "./socket/mensajes_cliente/procesar.h"


/*****************************************************************
*         F I N    A R C H I V O S    D E    C A B E C E R A
*****************************************************************/

/*****************************************************************
*           D E F I N I C I O N    D E    C O N S T A N T E S
*****************************************************************/

// Velocidad del vehículo durante la prueba de algoritmos
#define SPEED_ROMEO 0  // En m/s.

// Curvatura inicial
#define CURV_INICIAL 0

// Período del bucle de control
#define PERIODO_BUCLE_CONTROL 50000  // En us.

// Tamaño del mensaje que se envía funcionando como cliente-servidor
//#define TAM_MENSAJE_CLIENTE 80


/*****************************************************************
*        F I N    D E F I N I C I O N    D E    C O N S T A N T E S
*****************************************************************/

/*****************************************************************
*  D E C L A R A C I O N   Y   D E F I N I C I O N   D E   F U N C I O * N E S
*****************************************************************/

// Función manejadora de la señal SIGINT (Ctrl-C) para salir del bucle de
// control
void signal_int_handler(int s)
{
  extern int fin;
  fin = 1;
}

// Función para monitorizar datos (cuerpo de la función después de main())
/*void monitor(int dcx, DCX_DATA dcx_data, ESTIMATED dcx_est,
             int ax5411, AX5411_DATA ax5411_data,
            int gps, GPS_DATA gps_data, ESTIMATED gps_est,
            int gyro, GYRO_DATA gyro_data, ESTIMATED gyro_est,
            int estimated, ESTIMATED estimated,
            double ref_speed, double ref_curv,
            long iteracion, double time_act);
*/
void monitor(ESTIMATED estimated);

/*****************************************************************
*      F I N   D E C L A R.   Y   D E F.   D E    F U N C I O N E S
*****************************************************************/

/*****************************************************************
*               V A R I A B L E S     G L O B A L E S
*****************************************************************/

// El objeto "romeo" debe ser global para permitir el acceso al mismo desde
// los hilos
Romeo romeo;

// Semáforo del timer del bucle principal: debe ser global para que el hilo
// del timer pueda modificar su valor
int timer_sem;

// Variable que marca la salida del bucle de control: debe ser global para
// que pueda ser modificada por la función manejadora de la señal SIGINT
```

```
int fin = 0;

// Monitorización de datos recibidos de cada dispositivo
// Archivo donde vamos a escribir estos datos. Necesita ser global para
// ser usado por el hilo de captura de datos. Aunque en la práctica este
// hilo no se lanza.
// También es necesario que sea global para que se pueda acceder a esta
// variable desde la función "monitor".
ofstream archivo;

// Variable que determina si el hilo de captura de datos debe escribir
// o no en un archivo. Debe ser global para permitir al hilo el acceso a la
// misma.
int escribir_archivo = 0;

// Variable que indica si se desea realizar una simulación o una prueba real
int simulate = 0;
int flag_giro = 0;
// Necesitamos que sea global para acceder a ella desde el bucle de control
// de la direccion, que esta en un hilo
ESTIMATED estimated;

/*********************************************************************
*           F I N    V A R I A B L E S     G L O B A L E S
*********************************************************************/

/*********************************************************************
*             A L G O R I T M O     D E     C O N T R O L
*********************************************************************/
// PURE PURSUIT
#include "./algoritmos/pure_pursuit/pure_pursuit.hpp"
#define LOOK_AHEAD 1.5
Pure_pursuit pp(LOOK_AHEAD, "camino.cab");

PUNTO_CART punto_objetivo;
double ref_curv_pp;
//double  ref_curv = 0;

// BACKING
#include "./algoritmos/backing/backing.hpp"
#define DESIRED_DIR M_PI/2
Backing backing;

double desired_dir = 0.0;
double gamma_backing = 0.3;
double gamma_backing_ant = 0.0;
double gamma_temp = 0.0;
double x_remolque = 0.0;
double y_remolque = 0.0;
int path_follow = 1;    // 1: indica seguimiento de trayectoria
                        // 0: seguimiento de dirección
int iteraciones=0;  // BORRRARRRRRRRRR LUEGOOOOOOOOOOOOO/////////////////
/*
// XFUZZY
#include "./algoritmos/xfuzzy/xfuzzy.hpp"
#include "./algoritmos/xfuzzy/XfuzzyRomeo4.hpp"
XfuzzyRomeo4 xfuzzyromeo;

#define X_INI 3.7
#define Y_INI 5
#define ORIENT_INI (M_PI/2-0.01)

double x_fuzzy, y_fuzzy, phi_fuzzy, v_fuzzy, curv_fuzzy, dir_fuzzy;
*/

// MANIOBRAS
//#include "./algoritmos/maniobras/maniobra.hpp"
//Maniobra maniobra;
//Maniobra maniobra(0., 0.431, 1./6., 0.1, 0.17, 0.17, 0.51, 0.2);

/*********************************************************************
*          F I N    A L G O R I T M O     D E     C O N T R O L
*********************************************************************/
```

```
/*********************************************************************
*                          M A I N
*********************************************************************/

main(int argc, char *argv[])
{
/*********************************************************************
*                V A R I A B L E S    L O C A L E S
*********************************************************************/

// Es una buena práctica de programación declarar como externas aquellas
// variables no definidas dentro de una función
  extern Romeo romeo;
  extern int timer_sem;
  extern int fin;
  extern ESTIMATED estimated;
  extern int flag_giro;
// Manejo de señales
  struct sigaction sa;

// Estructuras de datos para el bucle de control
// Estimación de los parámetros de interés

  bzero(&estimated, sizeof(ESTIMATED));

// Algunas variables generales necesarias
  double time_act = 0;
  double ref_speed = 0;
  double ref_curv = 0;
// Variables relacionadas con la conexión
//  Socket conexion;
//  char cadena[TAM_MENSAJE_CLIENTE];  // Almacena la información recibida de la
//  conexión

// Variables relacionadas con el archivo de LOGS (monitorización)
  time_t logs_time;
  struct tm *logs_struct;
  char logs_name[30];

// Monitorización de datos recibidos de cada dispositivo
// Un 1 en cualquiera de estas variables indica que se quiere monitorizar
// dicho dispositivo
  int monitor_dcx = 0;
  int monitor_ax5411 = 0;
  int monitor_gps = 0;
  int monitor_gyro = 0;
  int monitor_estimated = 0;

// Variable que indica si vamos marcha atrás
  int marcha_atras = 0;

// Array que indica el orden de disparo de los sónares
  char cadena_sonares[12];

/*********************************************************************
*              F I N   V A R I A B L E S    L O C A L E S
*********************************************************************/


/*********************************************************************
*                    I N I C I A L I Z A C I O N
*********************************************************************/
// Comprobación de los parámetros pasados en la línea de comandos
  if (argc != 3)
  {
    fprintf(stderr, "Sintaxis: main <veloc. inicial> <curv. inicial> \n");
    exit(ERROR);
  }
  else
  {
    ref_speed = atof(argv[1]);
    ref_curv = atof(argv[2]);
  }

// Configuración de la señal SIGINT para que nos saque del bucle de control
  sa.sa_handler = signal_int_handler;
  sigemptyset(&sa.sa_mask);
```

```
  sa.sa_flags = 0;
  sigaction(SIGINT, &sa, NULL);

// Inicialización del Romeo
  if(romeo.init_romeo() < 0)
  {
    cerr << "main: error inicializando Carretilla" << endl;
    exit(ERROR);
  }
  cout << "main: Carretilla inicializada!" << endl;
 getchar();
// Centramos el volante.Esto debemos hacerlo antes de iniciar el hilo del timer
// para evitar que incremente el semaforo del timer indiscriminadamente.
//  romeo.centrar_volante_modo_posicion();

// Le damos al vehículo la curvatura inicial
//  ref_curv = -1.0;
//  romeo.set_ax_curv(ref_curv);
// Y esperamos 2 segundos para que dé tiempo a moverse el volante
//  sleep(2);

//  ref_curv = 0;
//  romeo.set_ax_curv(ref_curv);

// Aquí ponemos el vehículo a la velocidad deseada
/*  getchar();
  ref_speed = 0;
  romeo.set_ax_speed(ref_speed);*/

// Inicializamos el archivo de LOGS:
// Primero preparamos el nombre con la fecha actual
  logs_time = time(NULL);
  logs_struct = localtime(&logs_time);
  strftime(logs_name, 30, "./log/%d_%m_%y-%H_%M_%S.log",logs_struct);

// Después lo abrimos
  archivo.open(logs_name);
  archivo.setf(ios::right);
  archivo.precision(10);
  archivo.width(10);

// Iniciamos los hilos necesarios
//  iniciar_hilo_dcx();
//  iniciar_hilo_ax5411(cadena_sonares);
  iniciar_hilo_robosense();


/*romeo.mandar_comando_robosense(14);

sleep(5);


romeo.mandar_comando_robosense(12);
cout << "main: hemos mandado parar" << endl;

sleep(25);
cout << "main: ya ha esperado suficiente" << endl;*/
romeo.mandar_comando_robosense(21);
sleep(2);
romeo.mandar_comando_robosense(23);
//sleep(10);
getchar();

  ref_speed = 37;
  romeo.set_ax_speed(ref_speed);
  sleep(1);
/*  romeo.set_ax_curv(-2);
  getchar();
  romeo.set_ax_curv(0);*/
//sleep(10);
//getchar();

/*romeo.mandar_comando_robosense(11);
getchar();

romeo.mandar_comando_robosense(11);
getchar();*/
```

```
/*romeo.mandar_comando_robosense(12);
sleep(5);*/
  iniciar_hilo_pid_direccion();
  iniciar_hilo_estimacion();

// Inicialización del temporizador del bucle de control
  setup_timer(PERIODO_BUCLE_CONTROL);

// Monitorización de variables: aquí ponemos a 1 las variables de aquellos
// dispositivos que queremos monitorizar
  monitor_dcx = 0;
  monitor_ax5411 = 0;
  monitor_gps = 0;
  monitor_gyro = 0;
  monitor_estimated = 1;

/*******************************
* Inicialización del algoritmo *
*******************************/

// PURE PURSUIT
  if (pp.init_pp() < 0)
  {
    cerr << "main: error inicializando algoritmo Pure Pursuit" << endl;
    exit(-1);
  }
  cout << "main: pure pursuit inicializado!" << endl;

/*****************************
* Fin inicial. del algoritmo *
*****************************/

// Para llevar la cuenta de iteraciones
  long iteracion = 0;

/*************************************************************************
*                 F I N    I N I C I A L I Z A C I O N
*************************************************************************/

/*************************************************************************
*                  B U C L E    D E    C O N T R O L
*************************************************************************/

/************************************************
* Inicialización del bucle: condiciones iniciales *
************************************************/
// Le ponemos la posición inicial a la estimación del GPS
// ¡¡ ESTO HAY QUE CORREGIRLO !!
/*  do
  {
    gps_data = romeo.get_gps_data(0);
    gps_est.x = gps_data.x_utm;
    gps_est.y = gps_data.y_utm;
  } while ( sqrt(fabs(pow(gps_est.x,2) + pow(gps_est.y,2))) > 100 );
*/
// Detectamos si vamos marcha atrás para activar la bandera de marcha atrás
  if (ref_speed < 0)
    marcha_atras = 1;

  cout << "main: entrando en bucle de control" << endl;

  while(!fin)
  {
    iteracion++;
    iteraciones++;

// Nos esperamos hasta que expire el temporizador
    semaphore_p_restart(timer_sem);

// Cuando comenzamos la iteración, calculamos el instante actual
    time_act = romeo.get_relative_time();
```

```
/*********************************************************************
*    B L O Q U E    D E    E S T I M A C I O N    D E    D A T O S
*********************************************************************/

   estimated = romeo.get_estimated_status(0);



/*********************************************************************
* F I N    B L O Q U E    D E    E S T I M A C I O N    D E    D A * T O S
*********************************************************************/

/*********************************************************************
*              A L G O R I T M O    D E    C O N T R O L
*********************************************************************/
/***** Alberto *****/
if (path_follow)
{
    ref_curv_pp = pp.algoritmo_pp(estimated.x, estimated.y,
estimated.orient,&punto_objetivo, &fin);

// Gamma
  if (simulate)
  {
    gamma_temp = gamma_backing;
  }
  else
  {
    gamma_temp = estimated.beta_remolque;
  }

// Cálculo de coordenadas del remolque
/*  x_remolque = estimated.x - L_REMOLQUE * cos(gamma_temp + estimated.orient) -
              L_UNION * cos(estimated.orient);
  y_remolque = estimated.y - L_REMOLQUE * sin(gamma_temp + estimated.orient) -
             L_UNION * sin(estimated.orient);*/
//cout << "x_remolque: " << x_remolque << "  y_remolque: " << y_remolque << endl;
//cout << "x_obj: " << punto_objetivo.x << "  y_obj: " << punto_objetivo.y << endl;

//cout <<  "des_orient: " << desired_dir << "  phi: " <<
//                          estimated.orient + gamma_temp + M_PI << endl;
//cout << x_remolque << " " << y_remolque << " " << desired_dir << " " <<
//        punto_objetivo.x << " " << punto_objetivo.y << endl;

// Cálculo de la dirección deseada
  desired_dir = atan2(punto_objetivo.y - y_remolque,
                      punto_objetivo.x - x_remolque);

}  // fin if (seguimiento)
else
{
    desired_dir = DESIRED_DIR;
}

//cout << "ref_curv: " << ref_curv << " gamma: " << gamma_backing << " theta: " <<
estimated.orient << endl;

if (simulate)
{
  gamma_backing += (-1* ref_speed*sin(gamma_backing)/L_REMOLQUE - ref_speed*ref_curv) *
0.050;

 estimated.beta_remolque = gamma_backing;
}

  ref_curv = backing.control(estimated.orient, estimated.beta_remolque,
                             desired_dir);
  ref_curv = 0;
  romeo.set_ref_curv(ref_curv);
/***** Fin Alberto *****/

/***** Fran & Lumi *****/

// Cambio de coordenadas
/*  x_fuzzy = estimated.x * sin(ORIENT_INI) - estimated.y * cos (ORIENT_INI) +
          X_INI;
  y_fuzzy = estimated.x * cos(ORIENT_INI) + estimated.y * sin (ORIENT_INI) +
```

```
          Y_INI;
  phi_fuzzy = (ORIENT_INI - estimated.orient) * 180.0 / M_PI;
  if (phi_fuzzy > 180.0)
  {
    phi_fuzzy -= 360.0;
  }
  if (phi_fuzzy < -180.0)
  {
    phi_fuzzy += 360.0;
  }

  v_fuzzy = estimated.speed;

  xfuzzyromeo.inference(x_fuzzy, phi_fuzzy, y_fuzzy, v_fuzzy,
                        &curv_fuzzy, &dir_fuzzy);

  ref_curv = -1 * curv_fuzzy;
  ref_speed = dir_fuzzy;

cout << "Referencias: " << ref_curv << " " << ref_speed << endl;
*/

/***** Fin Fran & Lumi *****/

/***** Maniobras ******/

/*
fin = maniobra.ControlManiobra(estimated.x, estimated.y, estimated.orient, -
(ax5411_data.beta_remolque), estimated.speed, estimated.curv, &ref_speed, &ref_curv,
estimated.time, 0.02);
// if ((maniobra.GetEstado()==5) && ref_curv >= 0.169 && (flag == 0))
*/
/*********************************************************************
*          F I N    A L G O R I T M O    D E    C O N T R O L
*********************************************************************/

/*********************************************************************
*              B L O Q U E    D E    A C T U A C I O N
*********************************************************************/
// Protección ante curvaturas demasiado elevadas. Éstas pueden provocar que
// se golpee el tope de la dirección, afectando de alguna forma al motor de
// dirección.
  if(ref_curv > CURV_MAX)
    ref_curv = CURV_MAX;
  if(ref_curv < -CURV_MAX)
    ref_curv = -CURV_MAX;

// Esto es provisional: es el algoritmo el q tiene q generar la consigna de
// curvatura correcta.
/*if (marcha_atras)
{
  ref_curv *= -1;
}
*/

// Aplicamos las consignas
//  romeo.set_curv(ref_curv);
//  romeo.set_speed(ref_speed);


/*********************************************************************
*        F I N    D E     B L O Q U E    D E    A C T U A C I O N
*********************************************************************/

/*********************************************************************
*             B L O Q U E    D E    M O N I T O R I Z A C I O N
*********************************************************************/
// Mostramos algunos datos de interés en pantalla
//cout << "ref speed " << ref_speed << " ref curv: " << ref_curv << endl;
//cout << estimated.x << " " << estimated.y << " " << estimated.orient << " " <<
ax5411_data.beta_remolque <<  endl;

/*monitor(monitor_dcx, dcx_data, dcx_est,
       monitor_ax5411, ax5411_data,
       monitor_gps, gps_data, gps_est,
       monitor_gyro, gyro_data, gyro_est,
       monitor_estimated, estimated,
```

```
       ref_speed, ref_curv,
       iteracion, time_act);
*/
monitor(estimated);

/*********************************************************************
*    F I N    D E    B L O Q U E    D E    M O N I T O R I Z A C I O N
*********************************************************************/

  }  // Fin while(!fin)

cout << "main: saliendo del bucle de control" << endl;

/*********************************************************************
*          F I N    D E L    B U C L E    D E    C O N T R O L
*********************************************************************/

/*********************************************************************
*             B L O Q U E    D E    T E R M I N A C I O N
*********************************************************************/
// Paramos
  romeo.set_ax_speed(0);
  romeo.mandar_comando_robosense(12);
// Destruimos los hilos iniciados
//  destruir_hilo_dcx();
//  destruir_hilo_ax5411();
  destruir_hilo_estimacion();
  destruir_hilo_robosense();
  destruir_hilo_pid_direccion();

  cout << "main: hilos destruidos" << endl;

// Para llegar al estado final adecuado
  romeo.end_romeo();

  cout << "main: romeo terminado" << endl;

  cout << "main: fin del programa" << endl;

  cout << "iteraciones: " << iteracion << endl;
/*********************************************************************
*        F I N    D E     B L O Q U E    D E    T E R M I N A C I O N
*********************************************************************/

}  // Fin main(...)


/*void monitor(int monitor_dcx, DCX_DATA dcx_data, ESTIMATED dcx_est,
             int monitor_ax5411, AX5411_DATA ax5411_data,
            int monitor_gps, GPS_DATA gps_data, ESTIMATED gps_est,
            int monitor_gyro, GYRO_DATA gyro_data, ESTIMATED gyro_est,
            int monitor_estimated, ESTIMATED estimated,
            double ref_speed, double ref_curv,
            long iteracion, double time_act)
            */
void monitor(ESTIMATED estimated)
{
  extern ofstream archivo;

/*  if (monitor_dcx || monitor_ax5411 || monitor_gps || monitor_gyro)
  {
    archivo << iteracion << " ";            // 1
    archivo << time_act << " ";                     // 2

    if(monitor_dcx)
    {
      archivo << ref_speed << " ";          // 3
      archivo << ref_curv << " ";           // 4
      archivo << dcx_data.time << " ";
      archivo << dcx_data.dcx_speed << " ";
      archivo << dcx_data.dcx_curv << " ";
      archivo << dcx_est.time << " ";
      archivo << dcx_est.x << " ";
      archivo << dcx_est.y << " ";          // 10
      archivo << dcx_est.orient << " ";             // 11

    } // if(dcx)
```

```
    if(monitor_ax5411)
    {
      archivo << ax5411_data.beta_remolque << " "; // 12
      int i = 0;
      for (i = FIRST_SONAR; i <= LAST_SONAR; i++)
      {
        archivo << ax5411_data.sonar[i-FIRST_SONAR] << " ";
      }

    } // if(ax5411)

    if(monitor_gps)
    {
      archivo << gps_data.time << " ";              // 25
      archivo << setw(10) << gps_data.x_utm << " "; // 26
      archivo << setw(10) << gps_data.y_utm << " "; // 27
      archivo << gps_data.gps_speed << " ";
      archivo << gps_data.gps_orientation << " ";
      archivo << gps_data.dif_mode << " ";  // 30
      archivo << gps_est.time << " ";
      archivo << gps_est.x << " ";
      archivo << gps_est.y << " ";
      archivo << gps_est.orient << " ";             // 34
    } // if(gps)

    if(monitor_gyro)
    {
      archivo << gyro_data.time << " ";             // 35
      archivo << gyro_data.angle_rate << " ";
      archivo << gyro_data.temp << " ";
      archivo << gyro_data.gyro_orientation << " ";
      archivo << gyro_est.time << " ";
      archivo << gyro_est.x << " ";         // 40
      archivo << gyro_est.y << " ";
      archivo << gyro_est.orient << " ";
    } // if(gyro)

    if(monitor_estimated)
    {
*/

      archivo << estimated.time << " ";             // 43
      archivo << estimated.curv << " ";
      archivo << estimated.speed << " ";
      archivo << estimated.x << " ";
      archivo << estimated.y << " ";
      archivo << estimated.orient << " ";   // 48
      // archivo << estimated.beta_remolque << " ";

/*
    } // if(estimated)
*/
   archivo << endl;
/*
  } // if (monitor_dcx || monitor_ax5411 || monitor_gps || monitor_gyro)
*/
} // end monitor(...)
```

Anexo II: Algunos documentos de interés para este proyecto.

Terminal Voltage 0V to +15V, 100 Taps

X9312

Digitally Controlled Potentiometer (XDCP™)

FEATURES

- **Solid-state potentiometer**
- **3-wire serial interface**
- **Terminal voltage, 0 to +15V**
- **100 wiper tap points**
  - **—Wiper position stored in nonvolatile memory and recalled on power-up**
- **99 resistive elements**
  - **—Temperature compensated**
  - **—End to end resistance range ± 20%**
- **Low power CMOS**
  - **—V_{CC} = 5V**
  - **—Active current, 3mA max.**
  - **—Standby current, 1mA max.**
- **High reliability**
  - **—Endurance, 100,000 data changes per bit**
  - **—Register data retention, 100 years**
- **R_{TOTAL} values = 10KΩ, 50KΩ, and 100KΩ**
- **Packages**
  - **—8-lead SOIC and DIP**

DESCRIPTION

The Xicor X9312 is a digitally controlled potentiometer (XDCP). The device consists of a resistor array, wiper switches, a control section, and nonvolatile memory. The wiper position is controlled by a 3-wire interface.

The potentiometer is implemented by a resistor array composed of 99 resistive elements and a wiper switching network. Between each element and at either end are tap points accessible to the wiper terminal. The position of the wiper element is controlled by the $\overline{CS}$, U/$\overline{D}$, and $\overline{INC}$ inputs. The position of the wiper can be stored in nonvolatile memory and then be recalled upon a subsequent power-up operation.

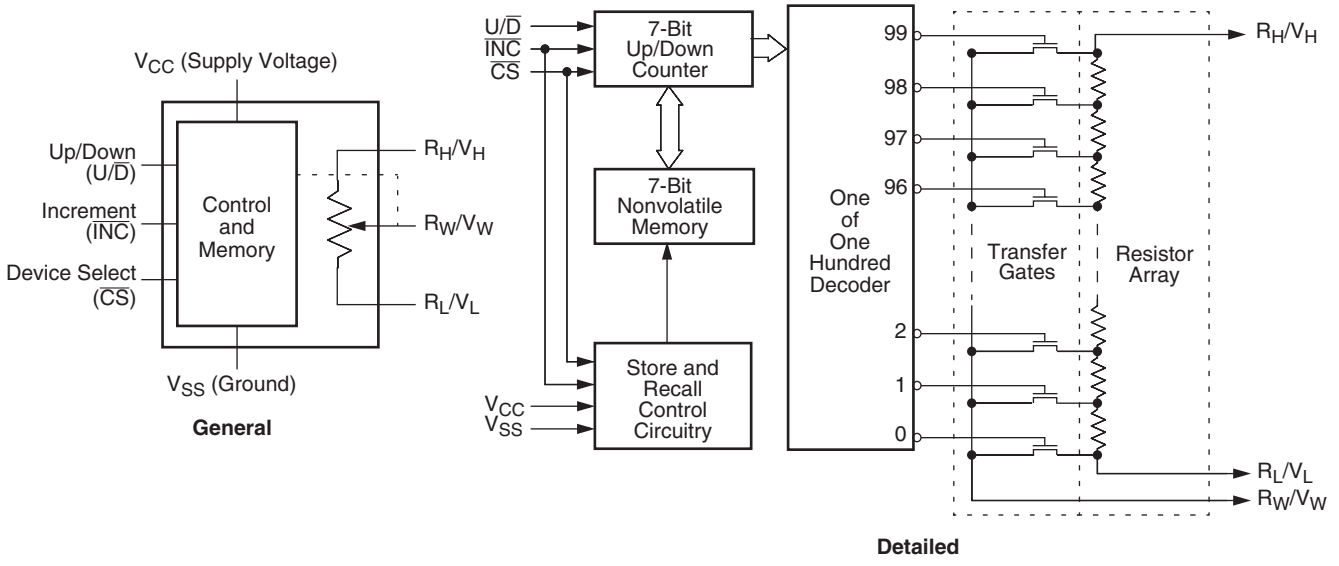
The device can be used as a three-terminal potentiometer or as a two-terminal variable resistor in a wide variety of applications including:

– control

– parameter adjustments

– signal processing

BLOCK DIAGRAM



XDCP is a trademark of Xicor, Inc.

PIN DESCRIPTIONS

R_H/V_H and R_L/V_L

The high (R_H/V_H) and low (R_L/V_L) terminals of the X9312 are equivalent to the fixed terminals of a mechanical potentiometer. The minimum voltage is 0V and the maximum is +15V. The terminology of R_L/V_L and R_H/V_H references the relative position of the terminal in relation to wiper movement direction selected by the U/$\overline{D}$ input and not the voltage potential on the terminal.

R_W/V_W

R_W/V_W is the wiper terminal and is equivalent to the movable terminal of a mechanical potentiometer. The position of the wiper within the array is determined by the control inputs. The wiper terminal series resistance is typically 40Ω.

Up/Down (U/$\overline{D}$)

The U/$\overline{D}$ input controls the direction of the wiper movement and whether the counter is incriminated or decremented.

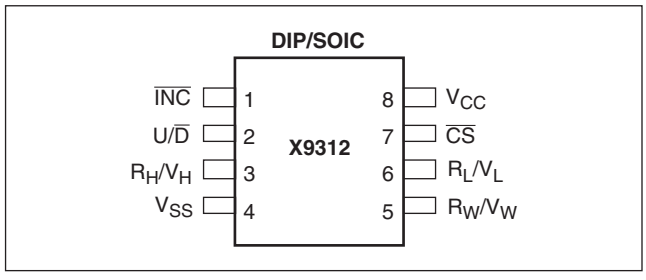
Increment ($\overline{INC}$)

The $\overline{INC}$ input is negative-edge triggered. Toggling $\overline{INC}$ will move the wiper and either increment or decrement the counter in the direction indicated by the logic level on the U/$\overline{D}$ input.

Chip Select ($\overline{CS}$)

The device is selected when the $\overline{CS}$ input is LOW. The current counter value is stored in nonvolatile memory when $\overline{CS}$ is returned HIGH while the $\overline{INC}$ input is also HIGH. After the store operation is complete the X9312 will be placed in the low power standby mode until the device is selected once again.

PIN CONFIGURATION



PIN NAMES

Symbol	Description
R_H/V_H	High terminal
R_W/V_W	Wiper terminal
R_L/V_L	Low terminal
V_{SS}	Ground
V_{CC}	Supply voltage
U/$\overline{D}$	Up/Down control input
$\overline{INC}$	Increment control input
$\overline{CS}$	Chip select control input

PRINCIPLES OF OPERATION

There are three sections of the X9312: the input control, counter and decode section; the nonvolatile memory; and the resistor array. The input control section operates just like an up/down counter. The output of this counter is decoded to turn on a single electronic switch connecting a point on the resistor array to the wiper output. Under the proper conditions the contents of the counter can be stored in nonvolatile memory and retained for future use. The resistor array is comprised of 99 individual resistors connected in series. At either end of the array and between each resistor is an electronic switch that transfers the potential at that point to the wiper.

The wiper, when at either fixed terminal, acts like its mechanical equivalent and does not move beyond the last position. That is, the counter does not wrap around when clocked to either extreme.

The electronic switches on the device operate in a “make before break” mode when the wiper changes tap positions. If the wiper is moved several positions, multiple taps are connected to the wiper for t_{IW} ($\overline{INC}$ to V_W change). The R_{TOTAL} value for the device can temporarily be reduced by a significant amount if the wiper is moved several positions.

When the device is powered-down, the last wiper position stored will be maintained in the nonvolatile memory. When power is restored, the contents of the memory are recalled and the wiper is set to the value last stored.

INSTRUCTIONS AND PROGRAMMING

The $\overline{INC}$, U/$\overline{D}$ and $\overline{CS}$ inputs control the movement of the wiper along the resistor array. With $\overline{CS}$ set LOW the device is selected and enabled to respond to the U/$\overline{D}$ and $\overline{INC}$ inputs. HIGH to LOW transitions on $\overline{INC}$ will increment or decrement (depending on the state of the U/$\overline{D}$ input) a seven bit counter. The output of this counter is decoded to select one of one hundred wiper positions along the resistive array.

The value of the counter is stored in nonvolatile memory whenever $\overline{CS}$ transitions HIGH while the $\overline{INC}$ input is also HIGH.

The system may select the X9312, move the wiper and deselect the device without having to store the latest wiper position in nonvolatile memory. After the wiper movement is performed as described above and once the new position is reached, the system must keep $\overline{INC}$ LOW while taking $\overline{CS}$ HIGH. The new wiper position will be maintained until changed by the system or until a powerup/down cycle recalled the previously stored data.

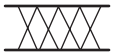
This procedure allows the system to always power-up to a preset value stored in nonvolatile memory; then during system operation minor adjustments could be made. The adjustments might be based on user preference, system parameter changes due to temperature drift, etc...

The state of U/$\overline{D}$ may be changed while $\overline{CS}$ remains LOW. This allows the host system to enable the device and then move the wiper up and down until the proper trim is attained.

MODE SELECTION

$\overline{CS}$	$\overline{INC}$	U/$\overline{D}$	Mode
L	↘	H	Wiper up
L	↘	L	Wiper down
↗	H	X	Store wiper position
H	X	X	Standby current
↗	L	X	No store, return to standby

SYMBOL TABLE

WAVEFORM	INPUTS	OUTPUTS
	Must be steady	Will be steady
	May change from Low to High	Will change from Low to High
	May change from High to Low	Will change from High to Low
	Don't Care: Changes Allowed	Changing: State Not Known
	N/A	Center Line is High Impedance

ABSOLUTE MAXIMUM RATINGS

Temperature under bias .....–65°C to +135°C
Storage temperature .....–65°C to +150°C
Voltage on $\overline{CS}$, $\overline{INC}$, U/$\overline{D}$ and V_{CC}
with respect to V_{SS} .....–1V to +7V
$\Delta V = |V_H - V_L|$..... 15V
Lead temperature (soldering 10 seconds).....300°C
I_W (10 seconds) ..... ±8.8mA

COMMENT

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only; functional operation of the device (at these or any other conditions above those listed in the operational sections of this specification) is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

RECOMMENDED OPERATING CONDITIONS

Temperature	Min.	Max.
Commercial	0°C	+70°C
Industrial	–40°C	+85°C

Supply Voltage (V_{CC})	Limits
X9312	5V ±10%

POTENTIOMETER CHARACTERISTICS (Over recommended operating conditions unless otherwise stated.)

Symbol	Parameter	Limits				Test Conditions/Notes
		Min.	Typ.	Max.	Unit	
	End to end resistance tolerance			±20	%	
V_{VH}	V_H terminal voltage	V_{SS}		15	V	V_{SS} = 0V
V_{VL}	V_L terminal voltage	V_{SS}		15	V	V_{SS} = 0V
	Power rating			25	mW	$R_{TOTAL} \geq 10$ KΩ
	Power rating			225	mW	R_{TOTAL} = 1 KΩ
R_W	Wiper resistance		40	100	Ω	I_W = 1mA, V_{CC} = 5V
I_W	Wiper current			±4.4	mA	
	Noise		-120		dBV	Ref: 1kHz
	Resolution		1		%	
	Absolute linearity(1)			±1	MI(3)	$R_{w(n)(actual)} - R_{w(n)(expected)}$
	Relative linearity(2)			±0.2	MI(3)	$R_{w(n+1)} - [R_{w(n)+MI}]$
	R_{TOTAL} temperature coefficient		±300		ppm/°C	
	Ratiometric temperature coefficient			±20	ppm/°C	
$C_H/C_L/C_W$	Potentiometer capacitances		10/10/25		pF	See circuit #3

Notes: (1) Absolute linearity is utilized to determine actual wiper voltage versus expected voltage = ($V_{w(n)}$(actual)–$V_{w(n)}$(expected)) = ±1 MI Maximum.
(2) Relative linearity is a measure of the error in step size between taps = $R_{W(n+1)} - [R_{w(n)} + MI] = \pm 0.2$ MI.
(3) 1 MI = Minimum Increment = $R_{TOT}/99$.

D.C. OPERATING CHARACTERISTICS (Over recommended operating conditions unless otherwise specified.)

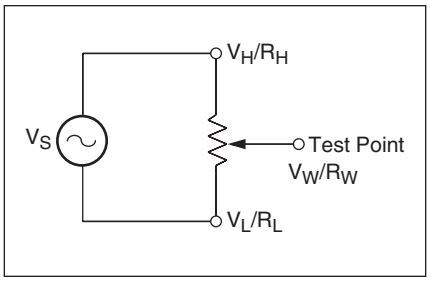
Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.(4)	Max.		
I_{CC}	V_{CC} active current (Increment)		1	3	mA	$\overline{CS}$ = V_{IL}, U/$\overline{D}$ = V_{IL} or V_{IH} and $\overline{INC}$ = 0.4V/2.4V @ max. t_{CYC}
I_{SB}	Standby supply current		500	1000	µA	$\overline{CS}$ = V_{CC} – 0.3V, U/$\overline{D}$ and $\overline{INC}$ = V_{SS} or V_{CC} – 0.3V
I_{LI}	$\overline{CS}$, $\overline{INC}$, U/$\overline{D}$ input leakage current			±10	µA	V_{IN} = V_{SS} to V_{CC}
V_{IH}	$\overline{CS}$, $\overline{INC}$, U/$\overline{D}$ input HIGH voltage	2		V_{CC} + 1	V	
V_{IL}	$\overline{CS}$, $\overline{INC}$, U/$\overline{D}$ input LOW voltage	–1		0.8	V	
C_{IN}(5)	$\overline{CS}$, $\overline{INC}$, U/$\overline{D}$ input capacitance			10	pF	V_{CC} = 5V, V_{IN} = V_{SS}, T_A = 25°C, f = 1MHz

ENDURANCE AND DATA RETENTION

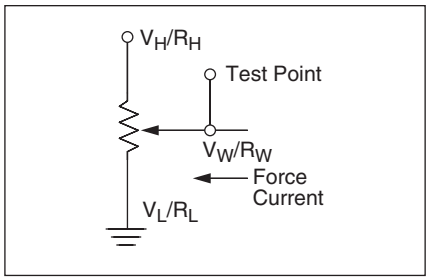
Parameter	Min.	Unit
Minimum endurance	100,000	Data changes per bit
Data retention	100	Years

Notes: (4) Typical values are for T_A = 25°C and nominal supply voltage.
(5) This parameter is periodically sampled and not 100% tested.

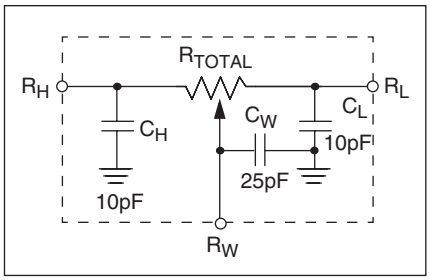
Test Circuit #1



Test Circuit #2



Circuit #3 SPICE Macro Model



A.C. CONDITIONS OF TEST

Input pulse levels	0V to 3V
Input rise and fall times	10ns
Input reference levels	1.5V

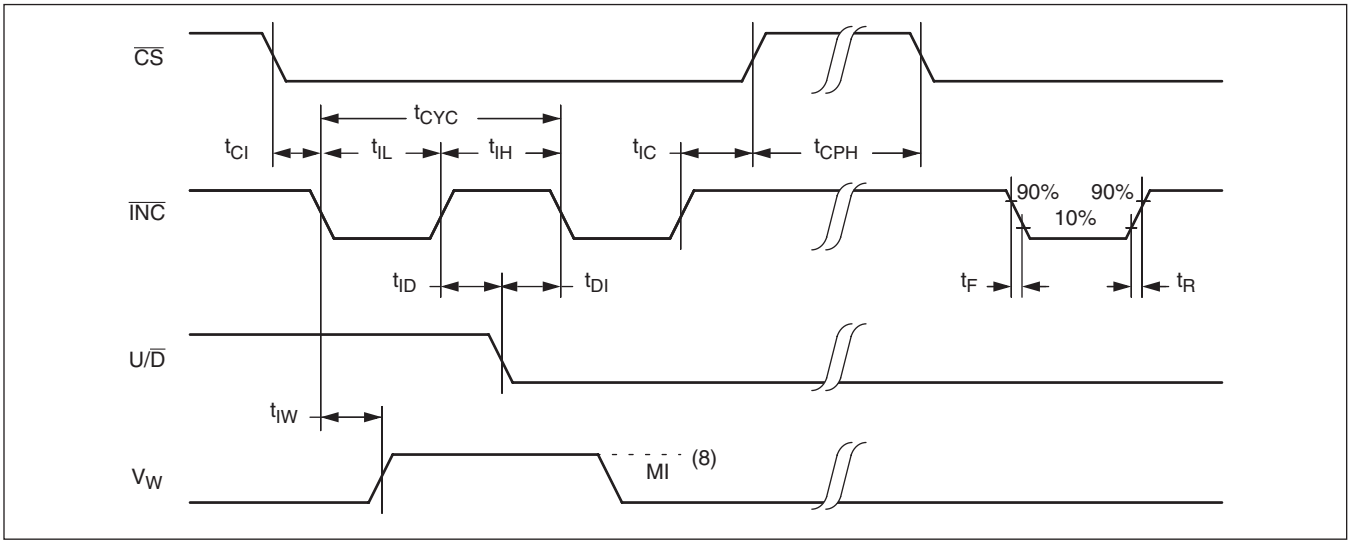
A.C. OPERATING CHARACTERISTICS (Over recommended operating conditions unless otherwise specified)

Symbol	Parameter	Limits			Unit
		Min.	Typ.[6]	Max.	
t_{CI}	$\overline{CS}$ to $\overline{INC}$ setup	100			ns
t_{ID}	$\overline{INC}$ HIGH to U/$\overline{D}$ change	100			ns
t_{DI}	U/$\overline{D}$ to $\overline{INC}$ setup	1			µs
t_{IL}	$\overline{INC}$ LOW period	1			µs
t_{IH}	$\overline{INC}$ HIGH period	1			µs
t_{IC}	$\overline{INC}$ inactive to $\overline{CS}$ inactive	1			µs
t_{CPH}	$\overline{CS}$ deselect time (STORE)	20			ms
t_{CPH}	$\overline{CS}$ deselect time (NO STORE)	100			ns
t_{IW}	$\overline{INC}$ to Vw change		100	500	µs
t_{CYC}	$\overline{INC}$ cycle time	4			µs
t_R, t_F[7]	$\overline{INC}$ input rise and fall time			500	µs
t_{PU}[7]	Power up to wiper stable			500	µs
t_R V_{CC}[7]	V_{CC} power-up rate	0.2		50	V/ms

POWER UP AND DOWN REQUIREMENTS

There are no restrictions on the sequencing of V_{CC} and the voltages applied to the potentiometer pins during power-up or power-down conditions. During power-up, the data sheet parameters for the DCP do not fully apply until 1 millisecond after V_{CC} reaches is final value. The V_{CC} ramp spec is always in effect.

A.C. TIMING

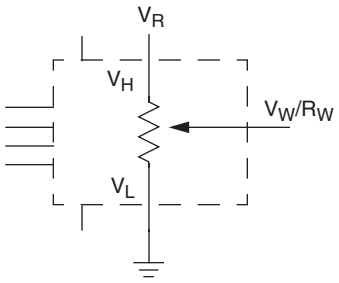


Notes: (6) Typical values are for T_A = 25°C and nominal supply voltage.
(7) This parameter is sample tested.
(8) MI in the A.C. timing diagram refers to the minimum incremental change in the V_W output due to a change in the wiper position.

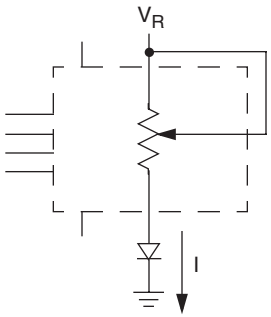
APPLICATIONS INFORMATION

Electronic digitally controlled (XDCP) potentiometers provide three powerful application advantages; (1) the variability and reliability of a solid-state potentiometer, (2) the flexibility of computer-based digital controls, and (3) the retentivity of nonvolatile memory used for the storage of multiple potentiometer settings or data.

Basic Configurations of Electronic Potentiometers



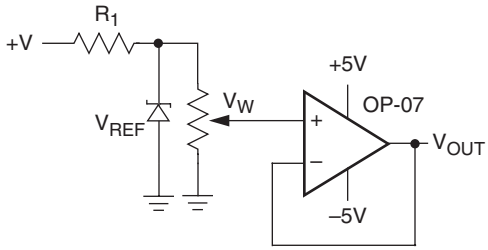
Three terminal potentiometer;
variable voltage divider



Two terminal variable resistor;
variable current

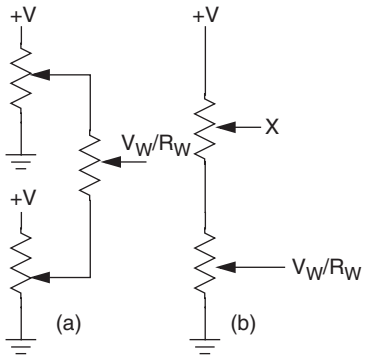
Basic Circuits

Buffered Reference Voltage

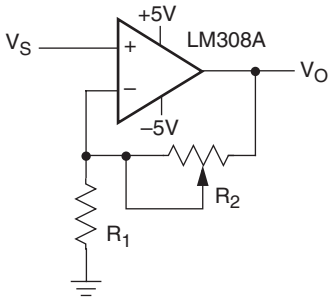


$V_{OUT} = V_W/R_W$

Cascading Techniques

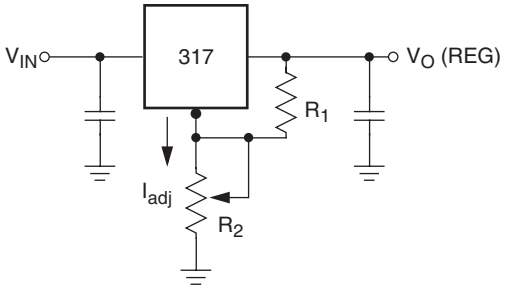


Noninverting Amplifier



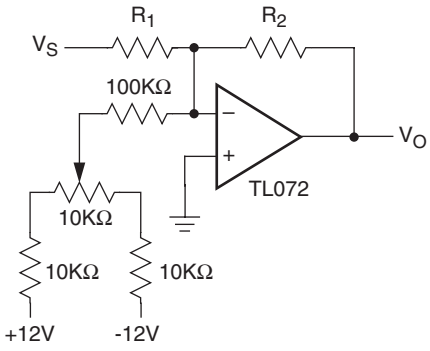
$V_O = (1+R_2/R_1)V_S$

Voltage Regulator

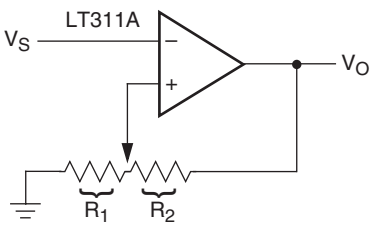


V_O (REG) = 1.25V $(1+R_2/R_1)+I_{adj}$ R_2

Offset Voltage Adjustment



Comparator with Hysteresis

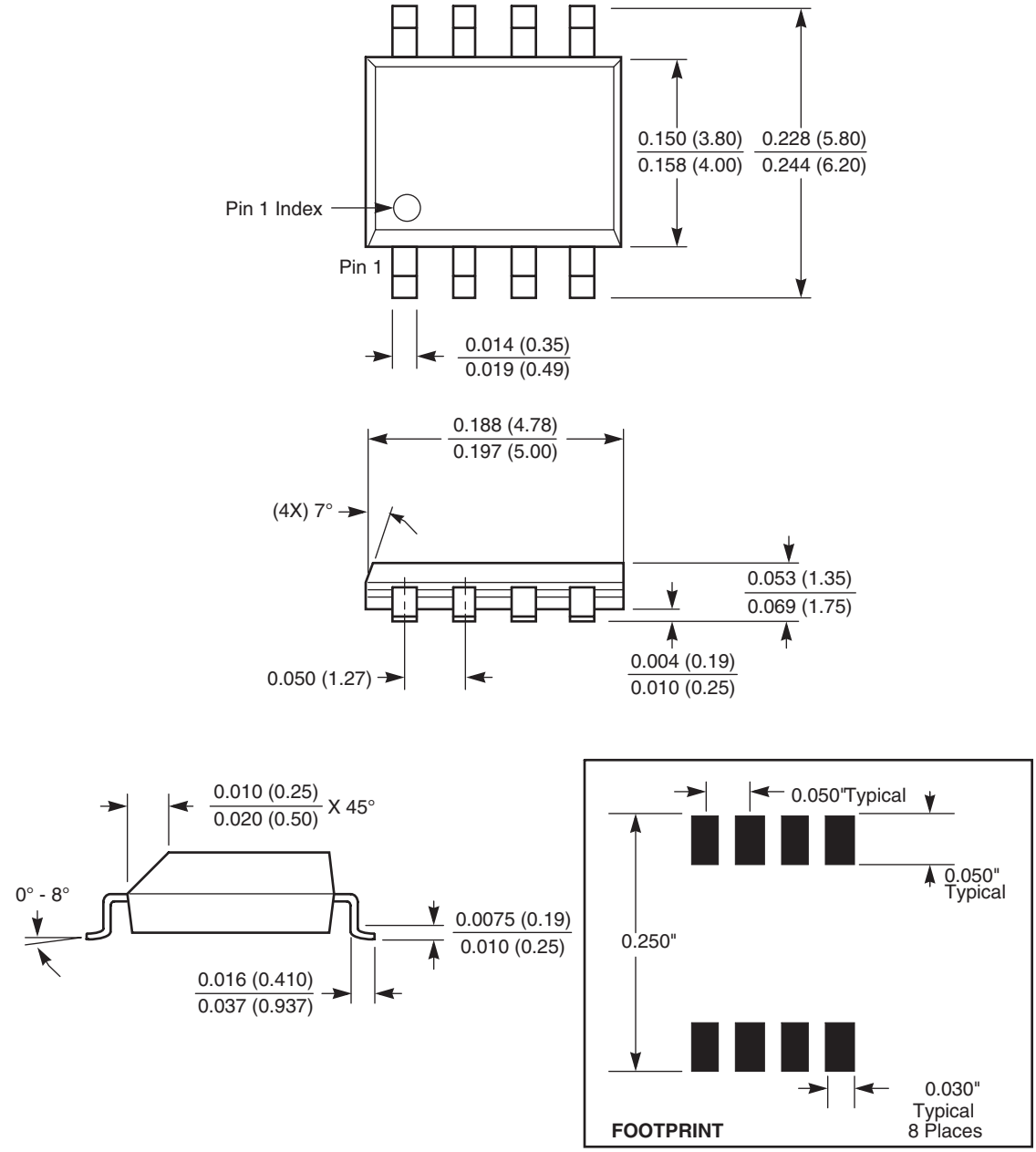


$V_{UL} = \{R_1/(R_1+R_2)\}\ V_O(max)$
$V_{LL} = \{R_1/(R_1+R_2)\}\ V_O(min)$

(for additional circuits see AN115)

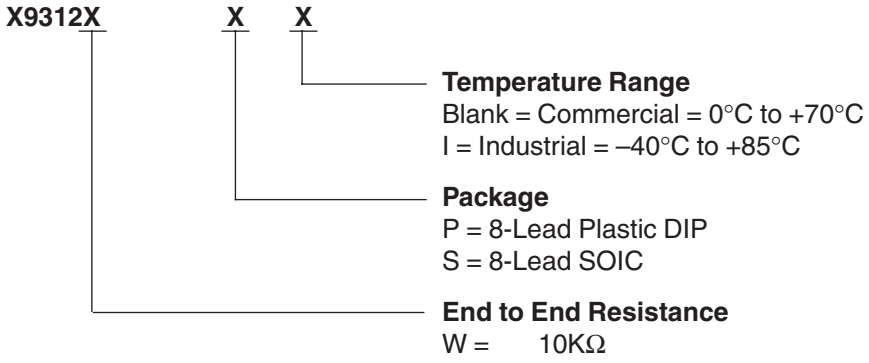
PACKAGING INFORMATION

8-Lead Plastic Small Outline Gull Wing Package Type S



NOTE: ALL DIMENSIONS IN INCHES (IN PARENTHESES IN MILLIMETERS)

ORDERING INFORMATION



Physical Characteristics

Marking Includes
Manufacturer's Trademark
Resistance Value or Code
Date Code

National Semiconductor

August 2000

LM124/LM224/LM324/LM2902
Low Power Quad Operational Amplifiers

General Description

The LM124 series consists of four independent, high gain, internally frequency compensated operational amplifiers which were designed specifically to operate from a single power supply over a wide range of voltages. Operation from split power supplies is also possible and the low power supply current drain is independent of the magnitude of the power supply voltage.

Application areas include transducer amplifiers, DC gain blocks and all the conventional op amp circuits which now can be more easily implemented in single power supply systems. For example, the LM124 series can be directly operated off of the standard +5V power supply voltage which is used in digital systems and will easily provide the required interface electronics without requiring the additional ±15V power supplies.

Unique Characteristics

- In the linear mode the input common-mode voltage range includes ground and the output voltage can also swing to ground, even though operated from only a single power supply voltage
- The unity gain cross frequency is temperature compensated
- The input bias current is also temperature compensated

Advantages

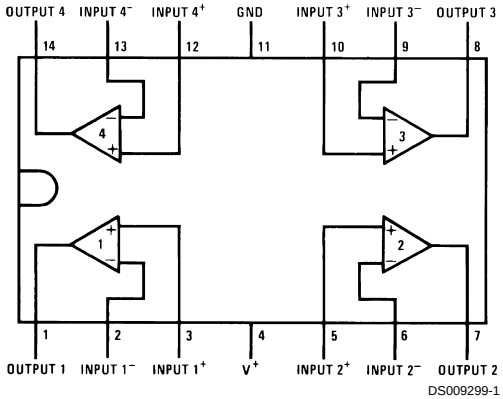
- Eliminates need for dual supplies
- Four internally compensated op amps in a single package
- Allows directly sensing near GND and V_{OUT} also goes to GND
- Compatible with all forms of logic
- Power drain suitable for battery operation

Features

- Internally frequency compensated for unity gain
- Large DC voltage gain 100 dB
- Wide bandwidth (unity gain) 1 MHz (temperature compensated)
- Wide power supply range:
  Single supply 3V to 32V
  or dual supplies ±1.5V to ±16V
- Very low supply current drain (700 µA)—essentially independent of supply voltage
- Low input biasing current 45 nA (temperature compensated)
- Low input offset voltage 2 mV and offset current: 5 nA
- Input common-mode voltage range includes ground
- Differential input voltage range equal to the power supply voltage
- Large output voltage swing 0V to V^+ − 1.5V

Connection Diagram

Dual-In-Line Package



DS009299-1

Top View

Order Number LM124J, LM124AJ, LM124J/883 (Note 2)**, LM124AJ/883** (Note 1)**, LM224J, LM224AJ, LM324J, LM324M, LM324MX, LM324AM, LM324AMX, LM2902M, LM2902MX, LM324N, LM324AN, LM324MT, LM324MTX or LM2902N LM124AJRQML and LM124AJRQMLV**(Note 3)

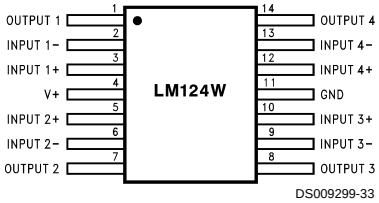
See NS Package Number J14A, M14A or N14A

Note 1: LM124A available per JM38510/11006

Note 2: LM124 available per JM38510/11005

Connection Diagram (Continued)

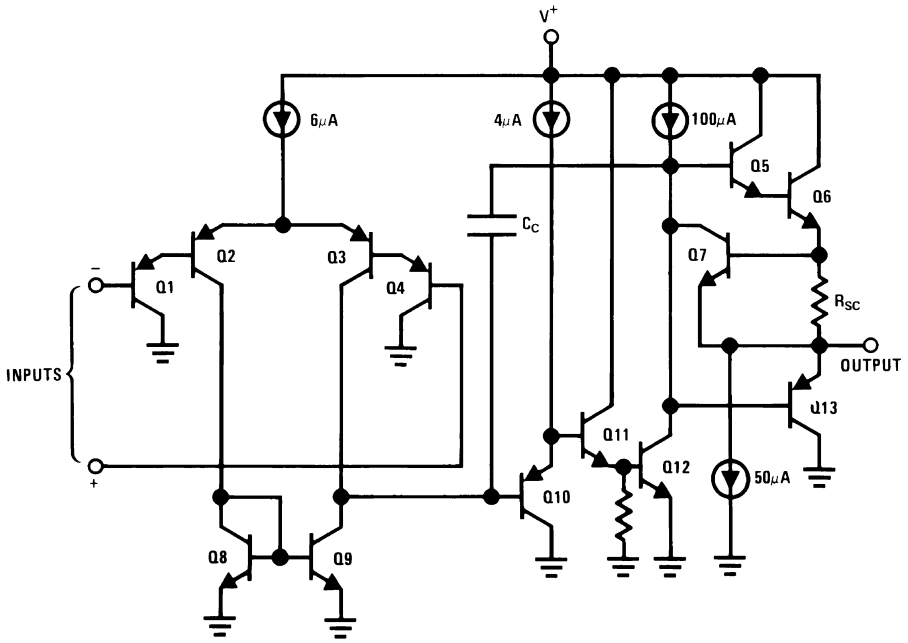
Note 3: See STD Mil DWG 5962R99504 for Radiation Tolerant Device



DS009299-33

Order Number LM124AW/883, LM124AWG/883, LM124W/883 or LM124WG/883
LM124AWRQML and LM124AWRQMLV(Note 3)
See NS Package Number W14B
LM124AWGRQML and LM124AWGRQMLV(Note 3)
See NS Package Number WG14A

Schematic Diagram (Each Amplifier)



DS009299-2

Absolute Maximum Ratings (Note 12)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

	LM124/LM224/LM324 LM124A/LM224A/LM324A	LM2902
Supply Voltage, V^+	32V	26V
Differential Input Voltage	32V	26V
Input Voltage	−0.3V to +32V	−0.3V to +26V
Input Current		
(V_{IN} < −0.3V) (Note 6)	50 mA	50 mA
Power Dissipation (Note 4)		
Molded DIP	1130 mW	1130 mW
Cavity DIP	1260 mW	1260 mW
Small Outline Package	800 mW	800 mW
Output Short-Circuit to GND		
(One Amplifier) (Note 5)		
$V^+ \leq 15V$ and $T_A = 25^\circ C$	Continuous	Continuous
Operating Temperature Range		−40˚C to +85˚C
LM324/LM324A	0˚C to +70˚C	
LM224/LM224A	−25˚C to +85˚C	
LM124/LM124A	−55˚C to +125˚C	
Storage Temperature Range	−65˚C to +150˚C	−65˚C to +150˚C
Lead Temperature (Soldering, 10 seconds)	260˚C	260˚C
Soldering Information		
Dual-In-Line Package		
Soldering (10 seconds)	260˚C	260˚C
Small Outline Package		
Vapor Phase (60 seconds)	215˚C	215˚C
Infrared (15 seconds)	220˚C	220˚C
See AN-450 “Surface Mounting Methods and Their Effect on Product Reliability” for other methods of soldering surface mount devices.		
ESD Tolerance (Note 13)	250V	250V

Electrical Characteristics

$V^+ = +5.0V$, (Note 7), unless otherwise stated

Parameter	Conditions	LM124A			LM224A			LM324A			Units
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Input Offset Voltage	(Note 8) $T_A = 25^\circ C$		1	2		1	3		2	3	mV
Input Bias Current (Note 9)	$I_{IN(+)}$ or $I_{IN(-)}$, $V_{CM} = 0V$, $T_A = 25^\circ C$		20	50		40	80		45	100	nA
Input Offset Current	$I_{IN(+)}$ or $I_{IN(-)}$, $V_{CM} = 0V$, $T_A = 25^\circ C$		2	10		2	15		5	30	nA
Input Common-Mode Voltage Range (Note 10)	$V^+ = 30V$, (LM2902, $V^+ = 26V$), $T_A = 25^\circ C$	0		V^+−1.5	0		V^+−1.5	0		V^+−1.5	V
Supply Current	Over Full Temperature Range $R_L = \infty$ On All Op Amps $V^+ = 30V$ (LM2902 $V^+ = 26V$) $V^+ = 5V$		 1.5 0.7	 3 1.2		 1.5 0.7	 3 1.2		 1.5 0.7	 3 1.2	mA
Large Signal Voltage Gain	$V^+ = 15V$, $R_L \geq 2k\Omega$, ($V_O = 1V$ to 11V), $T_A = 25^\circ C$	50	100		50	100		25	100		V/mV
Common-Mode Rejection Ratio	DC, $V_{CM} = 0V$ to $V^+ - 1.5V$, $T_A = 25^\circ C$	70	85		70	85		65	85		dB

Electrical Characteristics (Continued)

V^+ = +5.0V, (Note 7), unless otherwise stated

Parameter		Conditions		LM124A			LM224A			LM324A			Units
				Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Power Supply Rejection Ratio		V^+ = 5V to 30V (LM2902, V^+ = 5V to 26V), T_A = 25˚C		65	100		65	100		65	100		dB
Amplifier-to-Amplifier Coupling (Note 11)		f = 1 kHz to 20 kHz, T_A = 25˚C (Input Referred)			−120			−120			−120		dB
Output Current	Source	V_{IN}^+ = 1V, V_{IN}^- = 0V, V^+ = 15V, V_O = 2V, T_A = 25˚C		20	40		20	40		20	40		mA
	Sink	V_{IN}^- = 1V, V_{IN}^+ = 0V, V^+ = 15V, V_O = 2V, T_A = 25˚C		10	20		10	20		10	20		mA
		V_{IN}^- = 1V, V_{IN}^+ = 0V, V^+ = 15V, V_O = 200 mV, T_A = 25˚C		12	50		12	50		12	50		µA
Short Circuit to Ground		(Note 5) V^+ = 15V, T_A = 25˚C			40	60		40	60		40	60	mA
Input Offset Voltage		(Note 8)				4			4			5	mV
V_{OS} Drift		R_S = 0Ω			7	20		7	20		7	30	µV/˚C
Input Offset Current		$I_{IN(+)} - I_{IN(-)}$, V_{CM} = 0V				30			30			75	nA
I_{OS} Drift		R_S = 0Ω			10	200		10	200		10	300	pA/˚C
Input Bias Current		$I_{IN(+)}$ or $I_{IN(-)}$			40	100		40	100		40	200	nA
Input Common-Mode Voltage Range (Note 10)		V^+ = +30V (LM2902, V^+ = 26V)		0		V^+−2	0		V^+−2	0		V^+−2	V
Large Signal Voltage Gain		V^+ = +15V (V_OSwing = 1V to 11V) $R_L \geq 2$ kΩ		25			25			15			V/mV
Output Voltage Swing	V_{OH}	V^+ = 30V (LM2902, V^+ = 26V)	R_L = 2 kΩ	26			26			26			V
			R_L = 10 kΩ	27	28		27	28		27	28		V
	V_{OL}	V^+ = 5V, R_L = 10 kΩ			5	20		5	20		5	20	mV
Output Current	Source	V_O = 2V	V_{IN}^+ = +1V, V_{IN}^- = 0V, V^+ = 15V	10	20		10	20		10	20		mA
	Sink		V_{IN}^- = +1V, V_{IN}^+ = 0V, V^+ = 15V	10	15		5	8		5	8		mA

Electrical Characteristics

V^+ = +5.0V, (Note 7), unless otherwise stated

Parameter	Conditions	LM124/LM224			LM324			LM2902			Units
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Input Offset Voltage	(Note 8) T_A = 25˚C		2	5		2	7		2	7	mV
Input Bias Current (Note 9)	$I_{IN(+)}$ or $I_{IN(-)}$, V_{CM} = 0V, T_A = 25˚C		45	150		45	250		45	250	nA
Input Offset Current	$I_{IN(+)}$ or $I_{IN(-)}$, V_{CM} = 0V, T_A = 25˚C		3	30		5	50		5	50	nA
Input Common-Mode Voltage Range (Note 10)	V^+ = 30V, (LM2902, V^+ = 26V), T_A = 25˚C	0		V^+−1.5	0		V^+−1.5	0		V^+−1.5	V
Supply Current	Over Full Temperature Range R_L = ∞ On All Op Amps										mA
	V^+ = 30V (LM2902 V^+ = 26V)		1.5	3		1.5	3		1.5	3	mA
	V^+ = 5V		0.7	1.2		0.7	1.2		0.7	1.2	mA
Large Signal Voltage Gain	V^+ = 15V, $R_L \geq$ 2kΩ, (V_O = 1V to 11V), T_A = 25˚C	50	100		25	100		25	100		V/mV
Common-Mode Rejection Ratio	DC, V_{CM} = 0V to V^+ − 1.5V, T_A = 25˚C	70	85		65	85		50	70		dB
Power Supply Rejection Ratio	V^+ = 5V to 30V (LM2902, V^+ = 5V to 26V),	65	100		65	100		50	100		dB

Electrical Characteristics (Continued)

V^+ = +5.0V, (Note 7), unless otherwise stated

Parameter		Conditions		LM124/LM224			LM324			LM2902			Units
				Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
		T_A = 25˚C											
Amplifier-to-Amplifier Coupling (Note 11)		f = 1 kHz to 20 kHz, T_A = 25˚C (Input Referred)			−120			−120			−120		dB
Output Current	Source	V_{IN}^+ = 1V, V_{IN}^- = 0V, V^+ = 15V, V_O = 2V, T_A = 25˚C		20	40		20	40		20	40		mA
	Sink	V_{IN}^- = 1V, V_{IN}^+ = 0V, V^+ = 15V, V_O = 2V, T_A = 25˚C		10	20		10	20		10	20		mA
		V_{IN}^- = 1V, V_{IN}^+ = 0V, V^+ = 15V, V_O = 200 mV, T_A = 25˚C		12	50		12	50		12	50		µA
Short Circuit to Ground		(Note 5) V^+ = 15V, T_A = 25˚C			40	60		40	60		40	60	mA
Input Offset Voltage		(Note 8)				7			9			10	mV
V_{OS} Drift		R_S = 0Ω			7			7			7		µV/˚C
Input Offset Current		$I_{IN(+)} - I_{IN(-)}$, V_{CM} = 0V				100			150		45	200	nA
I_{OS} Drift		R_S = 0Ω			10			10			10		pA/˚C
Input Bias Current		$I_{IN(+)}$ or $I_{IN(-)}$			40	300		40	500		40	500	nA
Input Common-Mode Voltage Range (Note 10)		V^+ = +30V (LM2902, V^+ = 26V)		0		V^+−2	0		V^+−2	0		V^+−2	V
Large Signal Voltage Gain		V^+ = +15V (V_OSwing = 1V to 11V) $R_L \geq$ 2 kΩ		25			15			15			V/mV
Output Voltage Swing	V_{OH}	V^+ = 30V (LM2902, V^+ = 26V)	R_L = 2 kΩ	26			26			22			V
			R_L = 10 kΩ	27	28		27	28		23	24		V
	V_{OL}	V^+ = 5V, R_L = 10 kΩ			5	20		5	20		5	100	mV
Output Current	Source	V_O = 2V	V_{IN}^+ = +1V, V_{IN}^- = 0V, V^+ = 15V	10	20		10	20		10	20		mA
	Sink		V_{IN}^- = +1V, V_{IN}^+ = 0V, V^+ = 15V	5	8		5	8		5	8		mA

Note 4: For operating at high temperatures, the LM324/LM324A/LM2902 must be derated based on a +125˚C maximum junction temperature and a thermal resistance of 88˚C/W which applies for the device soldered in a printed circuit board, operating in a still air ambient. The LM224/LM224A and LM124/LM124A can be derated based on a +150˚C maximum junction temperature. The dissipation is the total of all four amplifiers — use external resistors, where possible, to allow the amplifier to saturate of to reduce the power which is dissipated in the integrated circuit.

Note 5: Short circuits from the output to V^+ can cause excessive heating and eventual destruction. When considering short circuits to ground, the maximum output current is approximately 40 mA independent of the magnitude of V^+. At values of supply voltage in excess of +15V, continuous short-circuits can exceed the power dissipation ratings and cause eventual destruction. Destructive dissipation can result from simultaneous shorts on all amplifiers.

Note 6: This input current will only exist when the voltage at any of the input leads is driven negative. It is due to the collector-base junction of the input PNP transistors becoming forward biased and thereby acting as input diode clamps. In addition to this diode action, there is also lateral NPN parasitic transistor action on the IC chip. This transistor action can cause the output voltages of the op amps to go to the V^+voltage level (or to ground for a large overdrive) for the time duration that an input is driven negative. This is not destructive and normal output states will re-establish when the input voltage, which was negative, again returns to a value greater than −0.3V (at 25˚C).

Note 7: These specifications are limited to −55˚C ≤ T_A ≤ +125˚C for the LM124/LM124A. With the LM224/LM224A, all temperature specifications are limited to −25˚C ≤ T_A ≤ +85˚C, the LM324/LM324A temperature specifications are limited to 0˚C ≤ T_A ≤ +70˚C, and the LM2902 specifications are limited to −40˚C ≤ T_A ≤ +85˚C.

Note 8: $V_O \simeq$ 1.4V, R_S = 0Ω with V^+ from 5V to 30V; and over the full input common-mode range (0V to V^+ − 1.5V) for LM2902, V^+ from 5V to 26V.

Note 9: The direction of the input current is out of the IC due to the PNP input stage. This current is essentially constant, independent of the state of the output so no loading change exists on the input lines.

Note 10: The input common-mode voltage of either input signal voltage should not be allowed to go negative by more than 0.3V (at 25˚C). The upper end of the common-mode voltage range is V^+ − 1.5V (at 25˚C), but either or both inputs can go to +32V without damage (+26V for LM2902), independent of the magnitude of V^+.

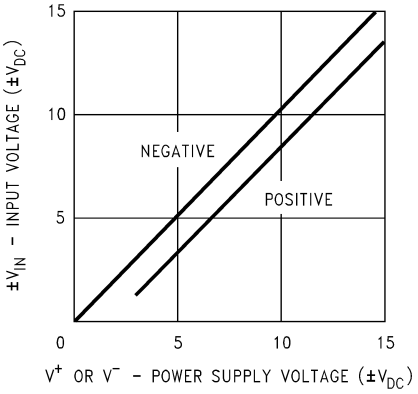
Note 11: Due to proximity of external components, insure that coupling is not originating via stray capacitance between these external parts. This typically can be detected as this type of capacitance increases at higher frequencies.

Note 12: Refer to RETS124AX for LM124A military specifications and refer to RETS124X for LM124 military specifications.

Note 13: Human body model, 1.5 kΩ in series with 100 pF.

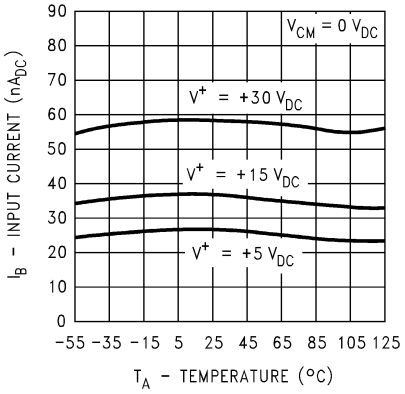
Typical Performance Characteristics

Input Voltage Range



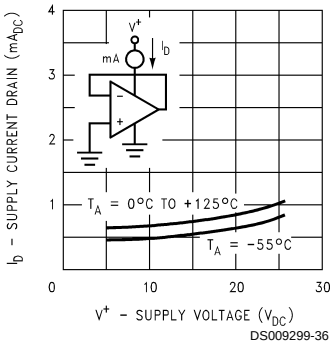
DS009299-34

Input Current



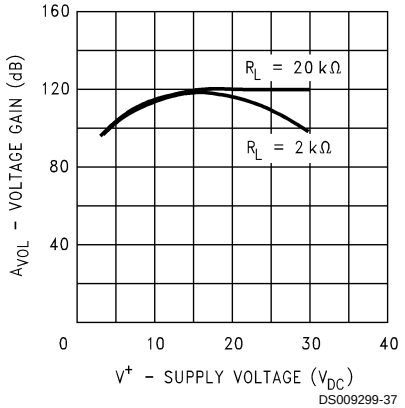
DS009299-35

Supply Current



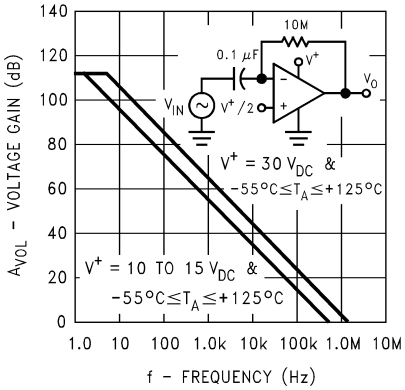
DS009299-36

Voltage Gain



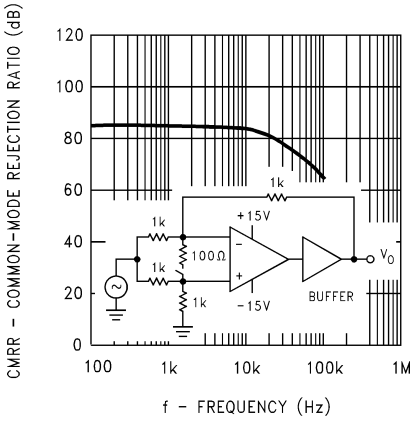
DS009299-37

Open Loop Frequency Response



DS009299-38

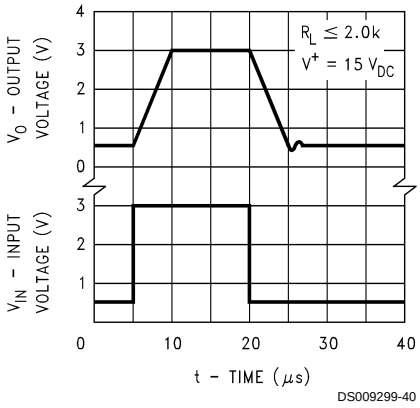
Common Mode Rejection Ratio



DS009299-39

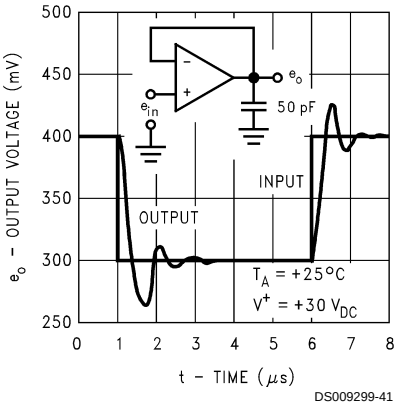
Typical Performance Characteristics (Continued)

Voltage Follower Pulse Response



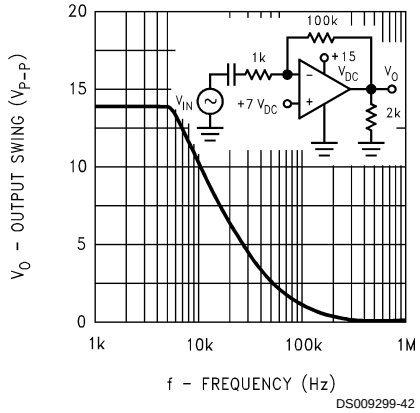
DS009299-40

Voltage Follower Pulse Response (Small Signal)



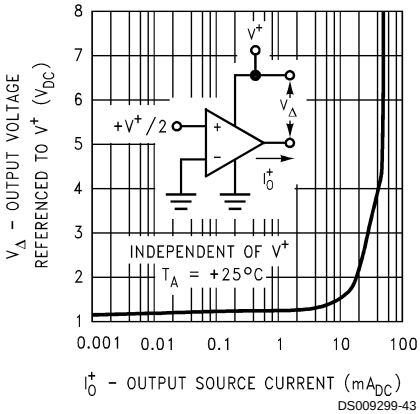
DS009299-41

Large Signal Frequency Response



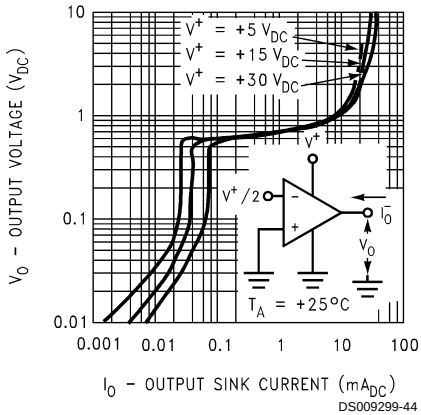
DS009299-42

Output Characteristics Current Sourcing



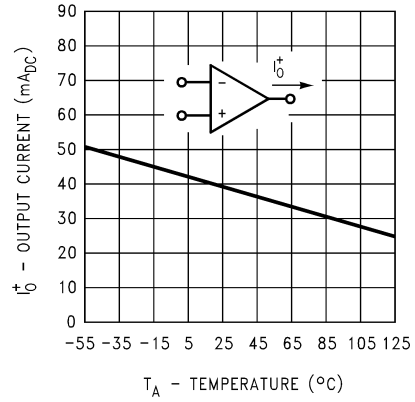
DS009299-43

Output Characteristics Current Sinking



DS009299-44

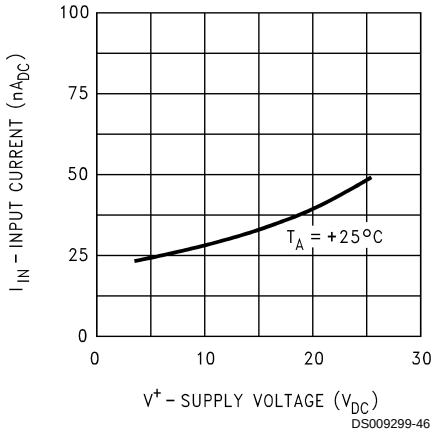
Current Limiting



DS009299-45

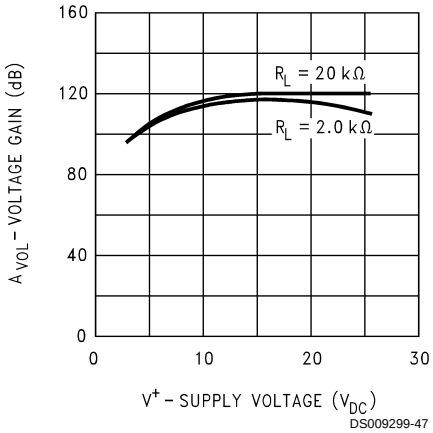
Typical Performance Characteristics (Continued)

Input Current (LM2902 only)



DS009299-46

Voltage Gain (LM2902 only)



DS009299-47

Application Hints

The LM124 series are op amps which operate with only a single power supply voltage, have true-differential inputs, and remain in the linear mode with an input common-mode voltage of 0 V_{DC}. These amplifiers operate over a wide range of power supply voltage with little change in performance characteristics. At 25˚C amplifier operation is possible down to a minimum supply voltage of 2.3 V_{DC}.

The pinouts of the package have been designed to simplify PC board layouts. Inverting inputs are adjacent to outputs for all of the amplifiers and the outputs have also been placed at the corners of the package (pins 1, 7, 8, and 14).

Precautions should be taken to insure that the power supply for the integrated circuit never becomes reversed in polarity or that the unit is not inadvertently installed backwards in a test socket as an unlimited current surge through the resulting forward diode within the IC could cause fusing of the internal conductors and result in a destroyed unit.

Large differential input voltages can be easily accommodated and, as input differential voltage protection diodes are not needed, no large input currents result from large differential input voltages. The differential input voltage may be larger than V^+ without damaging the device. Protection should be provided to prevent the input voltages from going negative more than −0.3 V_{DC} (at 25˚C). An input clamp diode with a resistor to the IC input terminal can be used.

To reduce the power supply drain, the amplifiers have a class A output stage for small signal levels which converts to class B in a large signal mode. This allows the amplifiers to both source and sink large output currents. Therefore both NPN and PNP external current boost transistors can be used to extend the power capability of the basic amplifiers. The output voltage needs to raise approximately 1 diode drop above ground to bias the on-chip vertical PNP transistor for output current sinking applications.

For ac applications, where the load is capacitively coupled to the output of the amplifier, a resistor should be used, from the output of the amplifier to ground to increase the class A bias current and prevent crossover distortion.

Where the load is directly coupled, as in dc applications, there is no crossover distortion.

Capacitive loads which are applied directly to the output of the amplifier reduce the loop stability margin. Values of 50 pF can be accommodated using the worst-case non-inverting unity gain connection. Large closed loop gains or resistive isolation should be used if larger load capacitance must be driven by the amplifier.

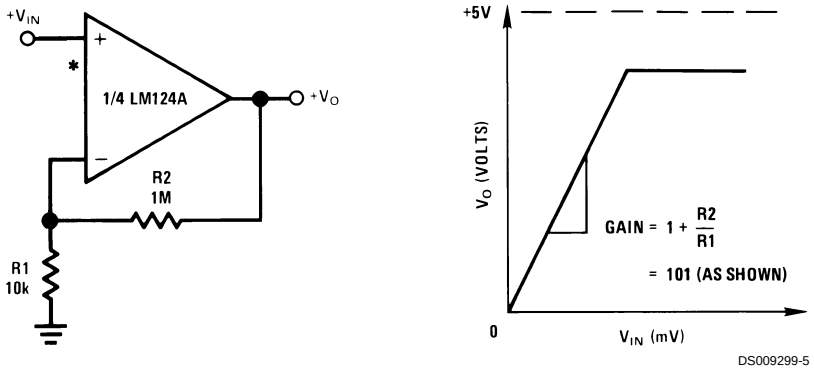
The bias network of the LM124 establishes a drain current which is independent of the magnitude of the power supply voltage over the range of from 3 V_{DC} to 30 V_{DC}.

Output short circuits either to ground or to the positive power supply should be of short time duration. Units can be destroyed, not as a result of the short circuit current causing metal fusing, but rather due to the large increase in IC chip dissipation which will cause eventual failure due to excessive junction temperatures. Putting direct short-circuits on more than one amplifier at a time will increase the total IC power dissipation to destructive levels, if not properly protected with external dissipation limiting resistors in series with the output leads of the amplifiers. The larger value of output source current which is available at 25˚C provides a larger output current capability at elevated temperatures (see typical performance characteristics) than a standard IC op amp.

The circuits presented in the section on typical applications emphasize operation on only a single power supply voltage. If complementary power supplies are available, all of the standard op amp circuits can be used. In general, introducing a pseudo-ground (a bias voltage reference of $V^+/2$) will allow operation above and below this value in single power supply systems. Many application circuits are shown which take advantage of the wide input common-mode voltage range which includes ground. In most cases, input biasing is not required and input voltages which range to ground can easily be accommodated.

Typical Single-Supply Applications (V^+ = 5.0 V_{DC})

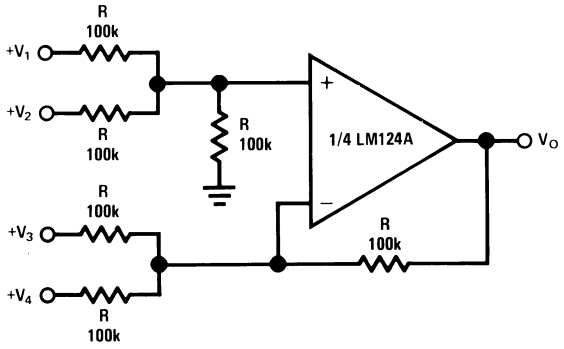
Non-Inverting DC Gain (0V Input = 0V Output)



DS009299-5

*R not needed due to temperature independent I_{IN}

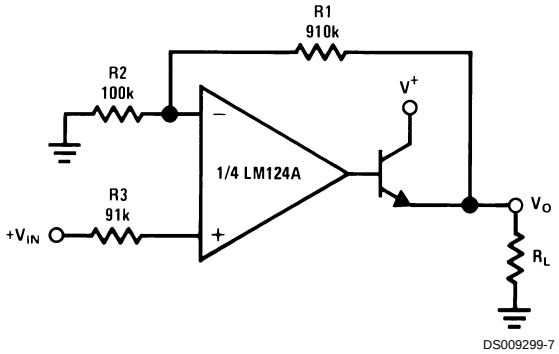
DC Summing Amplifier
($V_{IN'S} \geq 0\ V_{DC}$ and $V_O \geq V_{DC}$)



DS009299-6

Where: $V_0 = V_1 + V_2 - V_3 - V_4$
$(V_1 + V_2) \geq (V_3 + V_4)$ to keep $V_O > 0\ V_{DC}$

Power Amplifier

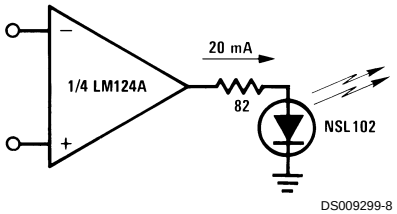


DS009299-7

$V_0 = 0\ V_{DC}$ for $V_{IN} = 0\ V_{DC}$
$A_V = 10$

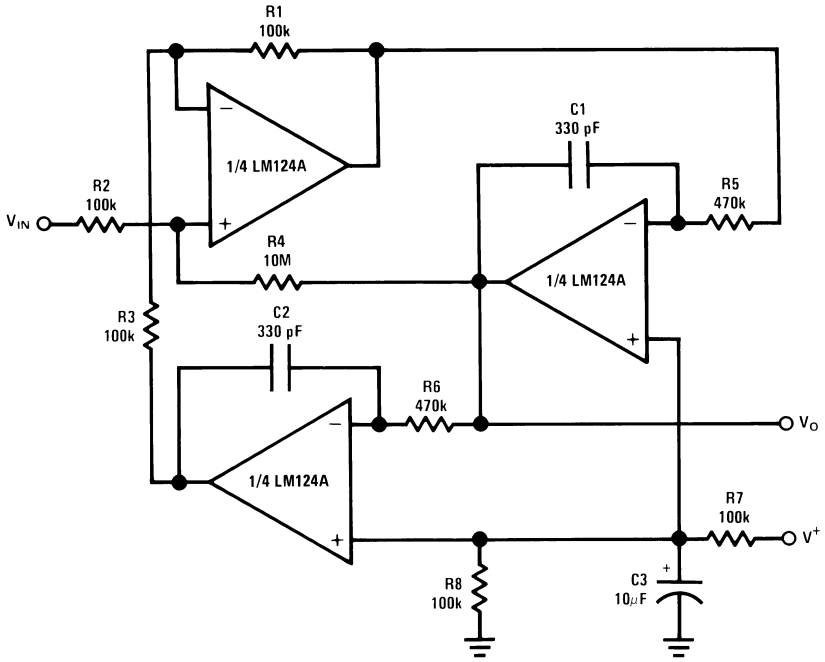
Typical Single-Supply Applications (V^+ = 5.0 V_{DC}) (Continued)

LED Driver



DS009299-8

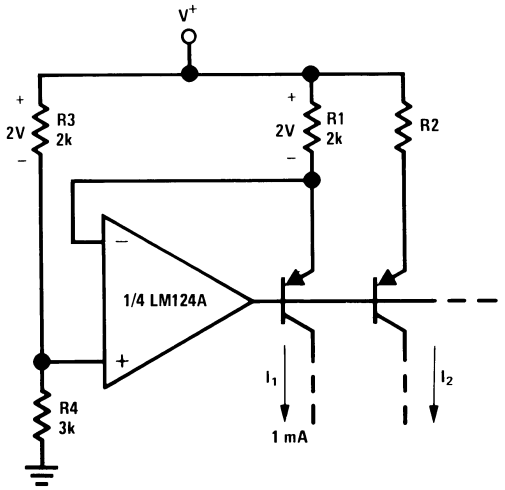
"BI-QUAD" RC Active Bandpass Filter



DS009299-9

f_o = 1 kHz
Q = 50
A_V = 100 (40 dB)

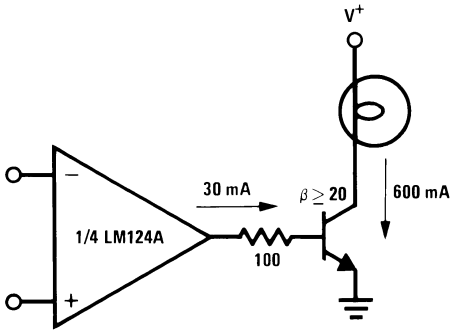
Fixed Current Sources



DS009299-10

$$I_2 = \left(\frac{R1}{R2}\right) I_1$$

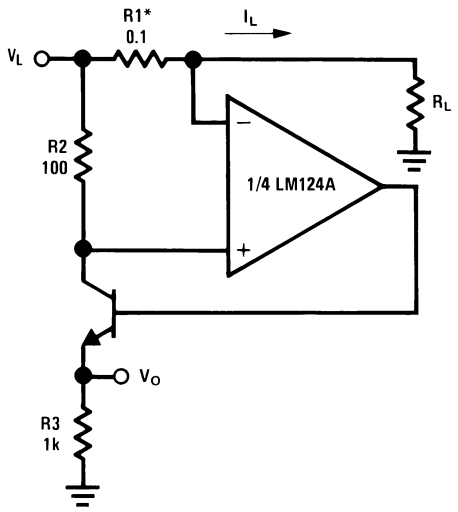
Lamp Driver



DS009299-11

Typical Single-Supply Applications ($V^+ = 5.0\ V_{DC}$) (Continued)

Current Monitor



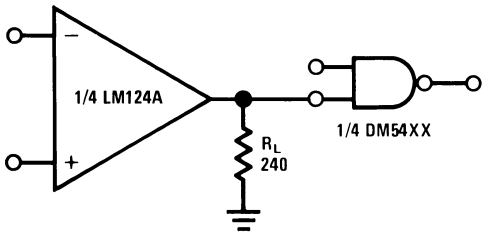
DS009299-12

$$V_O = \frac{1V(I_L)}{1A}$$

$V_L \leq V^+ - 2V$

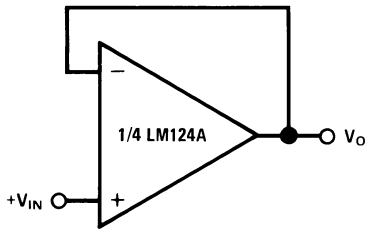
*(Increase R1 for I_L small)

Driving TTL



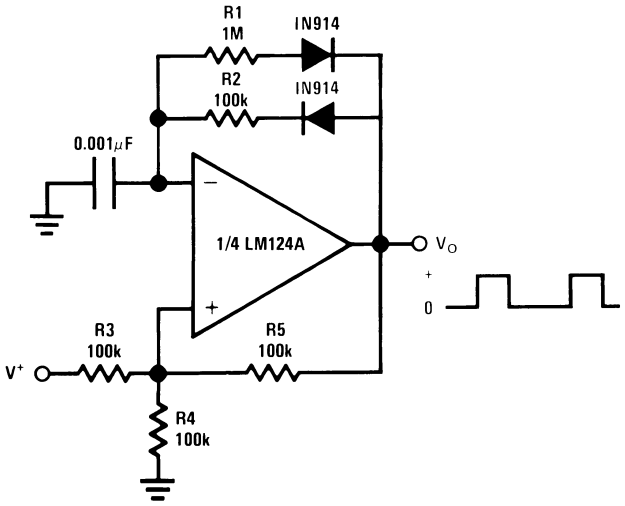
DS009299-13

Voltage Follower



DS009299-14

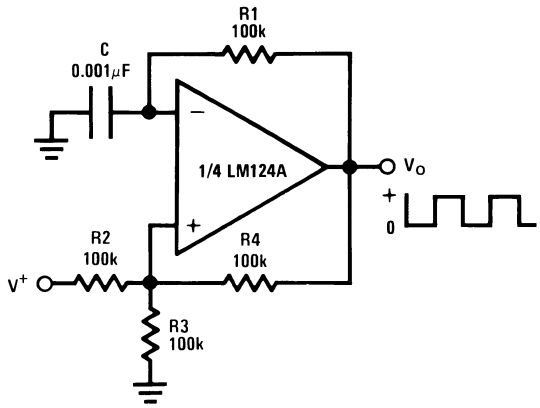
Pulse Generator



DS009299-15

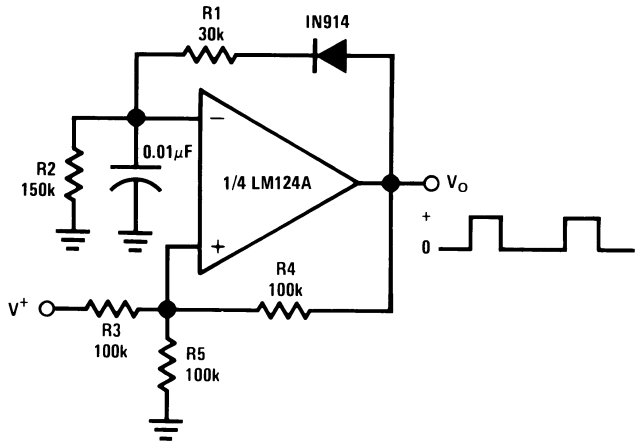
Typical Single-Supply Applications (V^+ = 5.0 V_{DC}) (Continued)

Squarewave Oscillator



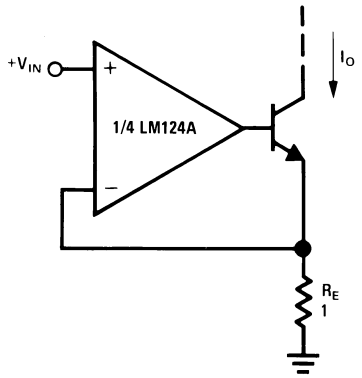
DS009299-16

Pulse Generator



DS009299-17

High Compliance Current Sink

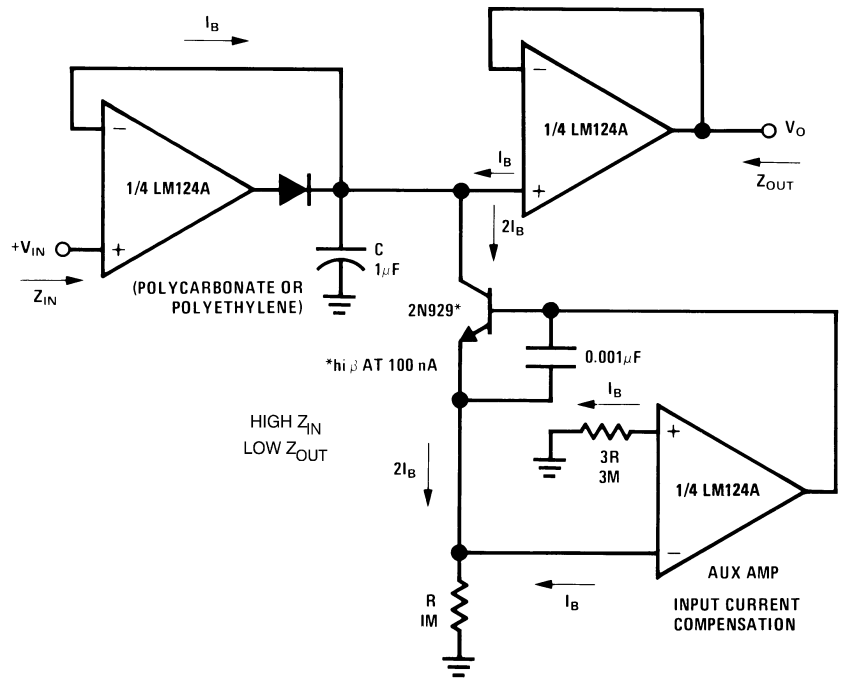


DS009299-18

I_O = 1 amp/volt V_{IN}
(Increase R_E for I_o small)

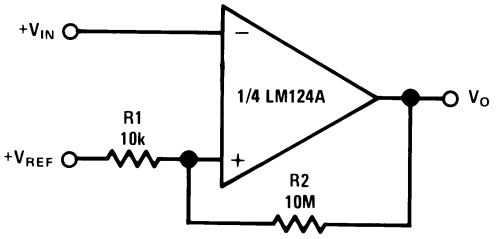
Typical Single-Supply Applications (V^+ = 5.0 V_{DC}) (Continued)

Low Drift Peak Detector



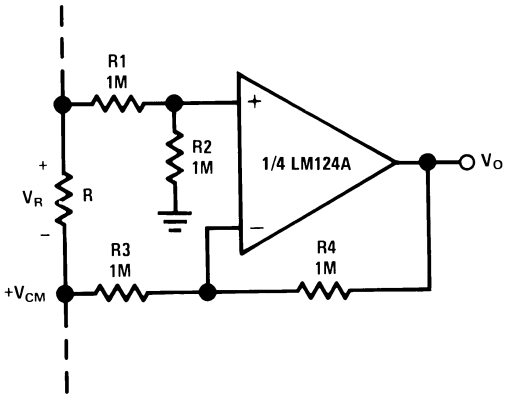
DS009299-19

Comparator with Hysteresis



DS009299-20

Ground Referencing a Differential Input Signal

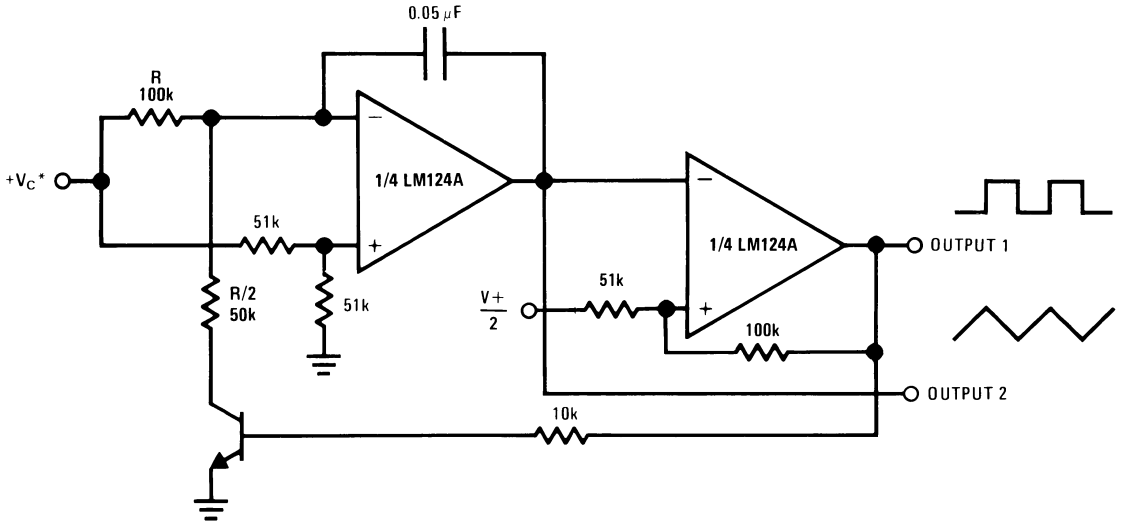


DS009299-21

$V_O = V_R$

Typical Single-Supply Applications (V^+ = 5.0 V_{DC}) (Continued)

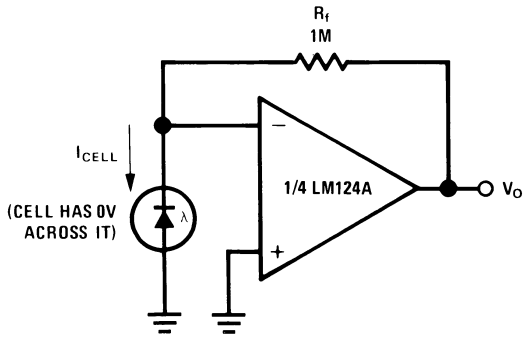
Voltage Controlled Oscillator Circuit



DS009299-22

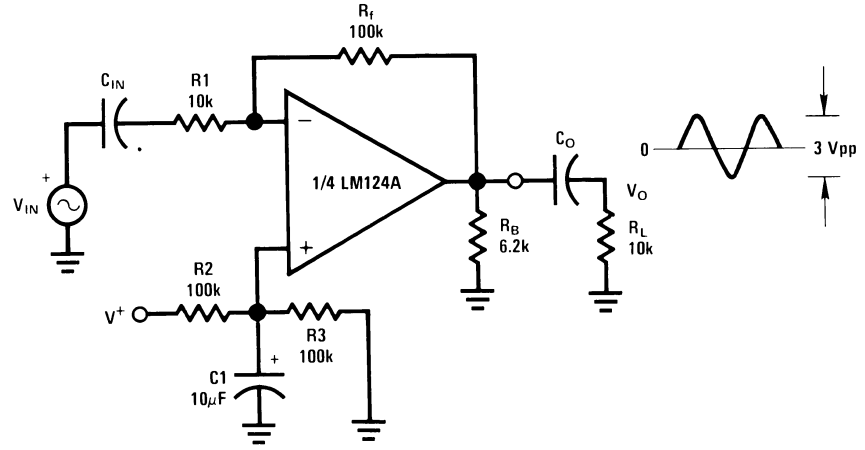
*Wide control voltage range: 0 $V_{DC} \leq V_C \leq 2$ ($V^+ - 1.5$ V_{DC})

Photo Voltaic-Cell Amplifier



DS009299-23

AC Coupled Inverting Amplifier

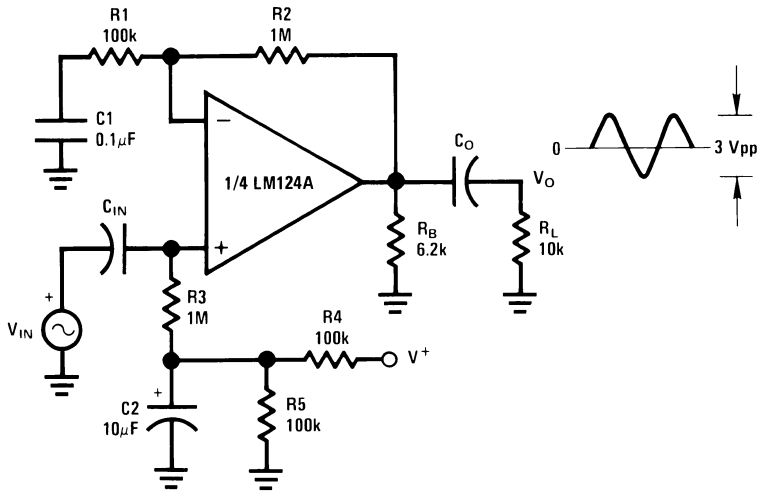


DS009299-24

$A_V = \frac{R_f}{R1}$ (As shown, $A_V = 10$)

Typical Single-Supply Applications (V^+ = 5.0 V_{DC}) (Continued)

AC Coupled Non-Inverting Amplifier

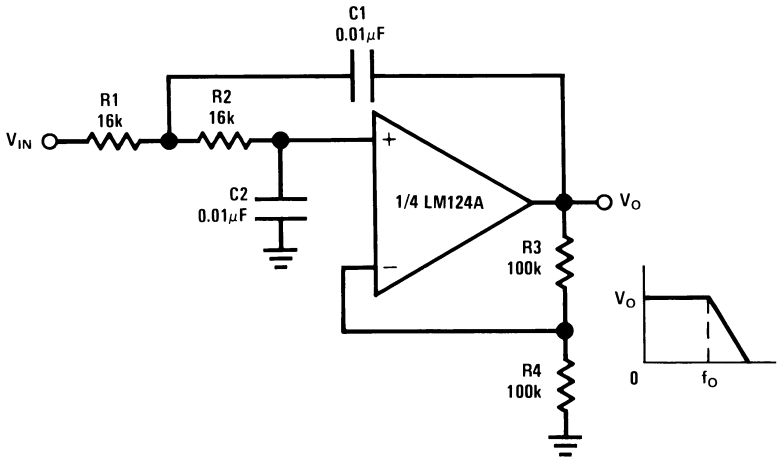


DS009299-25

$$A_V = 1 + \frac{R2}{R1}$$

A_V = 11 (As shown)

DC Coupled Low-Pass RC Active Filter

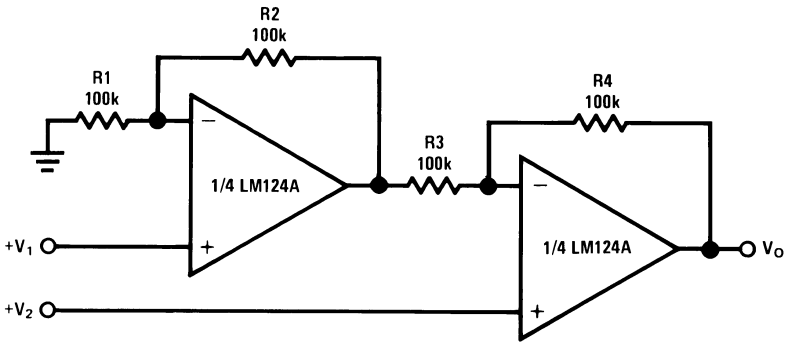


DS009299-26

f_O = 1 kHz
Q = 1
A_V = 2

Typical Single-Supply Applications (V^+ = 5.0 V_{DC}) (Continued)

High Input Z, DC Differential Amplifier



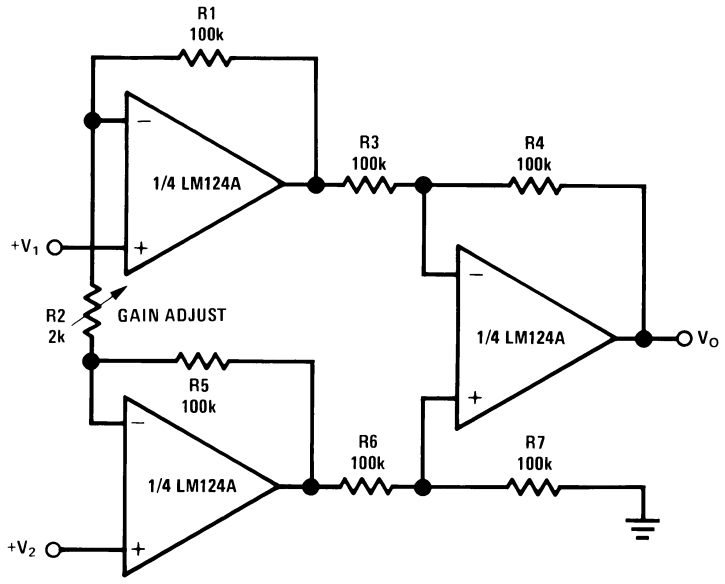
DS009299-27

For $\frac{R1}{R2} = \frac{R4}{R3}$ (CMRR depends on this resistor ratio match)

$V_O = 1 + \frac{R4}{R3}(V_2 - V_1)$

As shown: $V_O = 2(V_2 - V_1)$

High Input Z Adjustable-Gain
DC Instrumentation Amplifier



DS009299-28

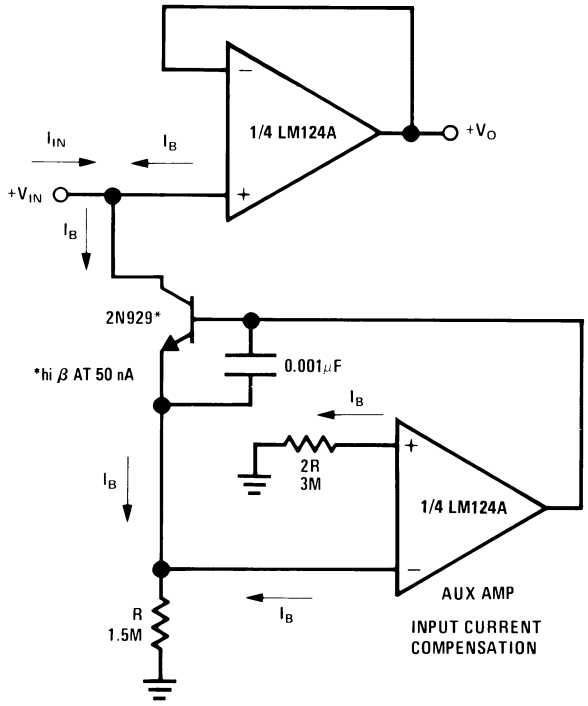
If R1 = R5 & R3 = R4 = R6 = R7 (CMRR depends on match)

$$V_O = 1 + \frac{2R1}{R2}(V_2 - V_1)$$

As shown $V_O = 101\ (V_2 - V_1)$

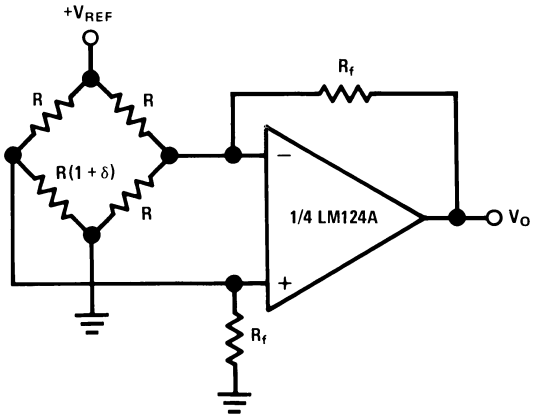
Typical Single-Supply Applications (V^+ = 5.0 V_{DC}) (Continued)

Using Symmetrical Amplifiers to Reduce Input Current (General Concept)



DS009299-29

Bridge Current Amplifier

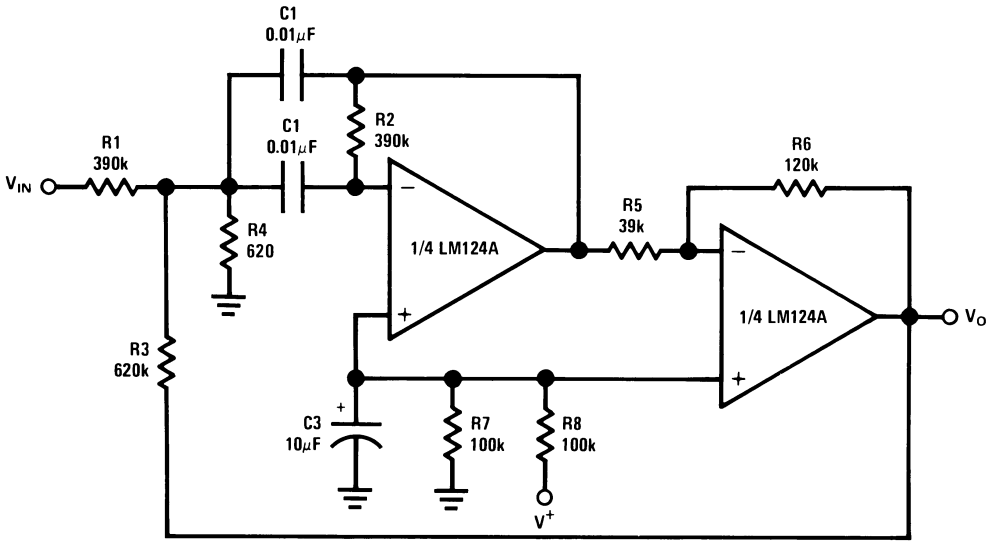


DS009299-30

For $\delta << 1$ and $R_f >> R$

$$V_O \cong V_{REF}\left(\frac{\delta}{2}\right)\frac{R_f}{R}$$

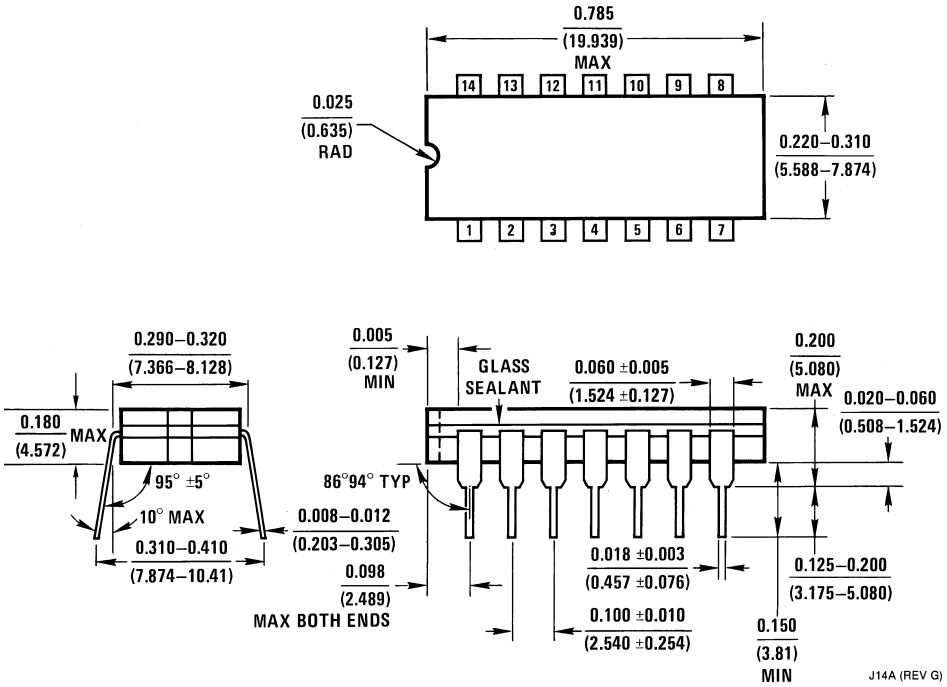
Bandpass Active Filter



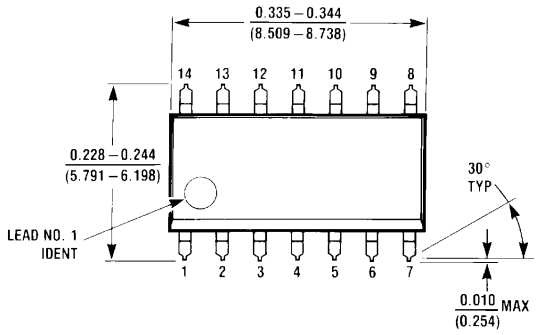
DS009299-31

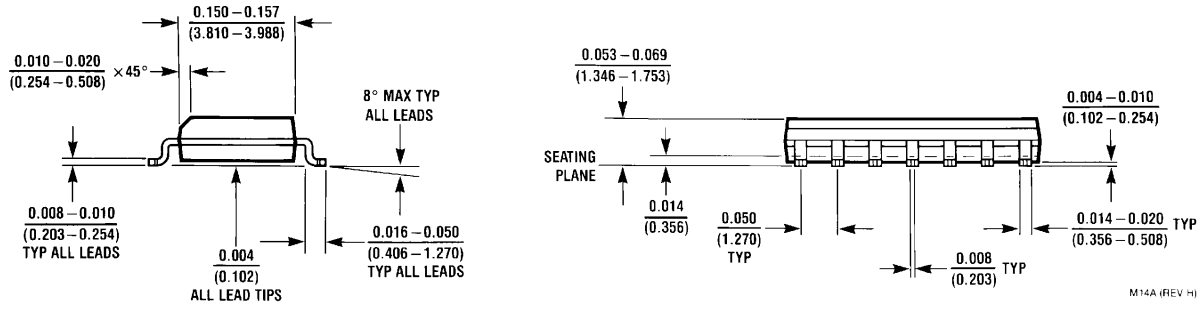
f_O = 1 kHz
Q = 25

Physical Dimensions inches (millimeters) unless otherwise noted



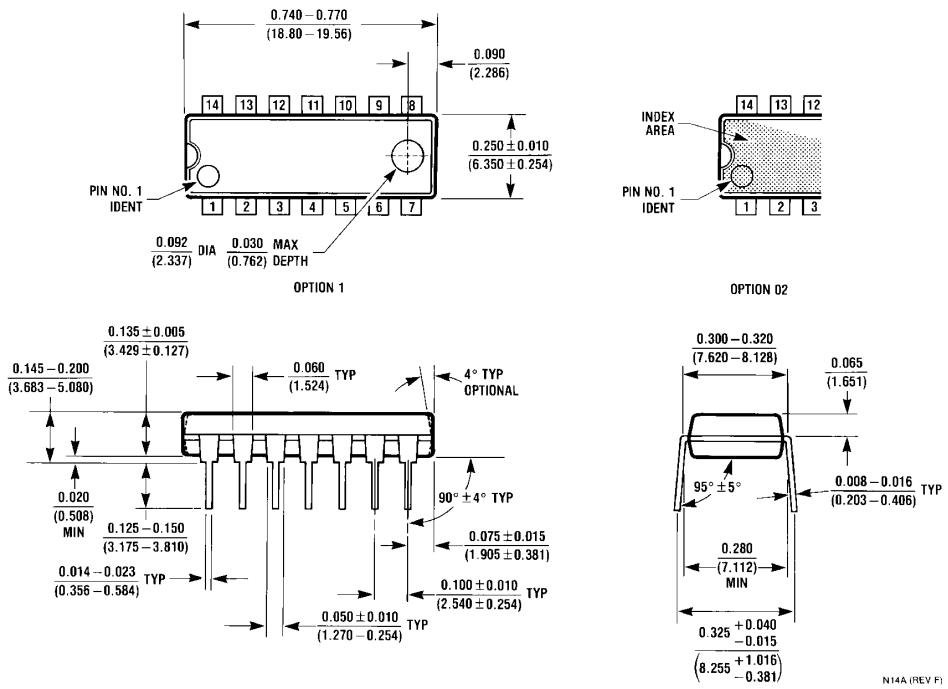
Ceramic Dual-In-Line Package (J)
Order Number JL124ABCA, JL124BCA, JL124ASCA, JL124SCA, LM124J, LM124AJ, LM124AJ/883, LM124J/883, LM224J, LM224AJ or LM324J
NS Package Number J14A



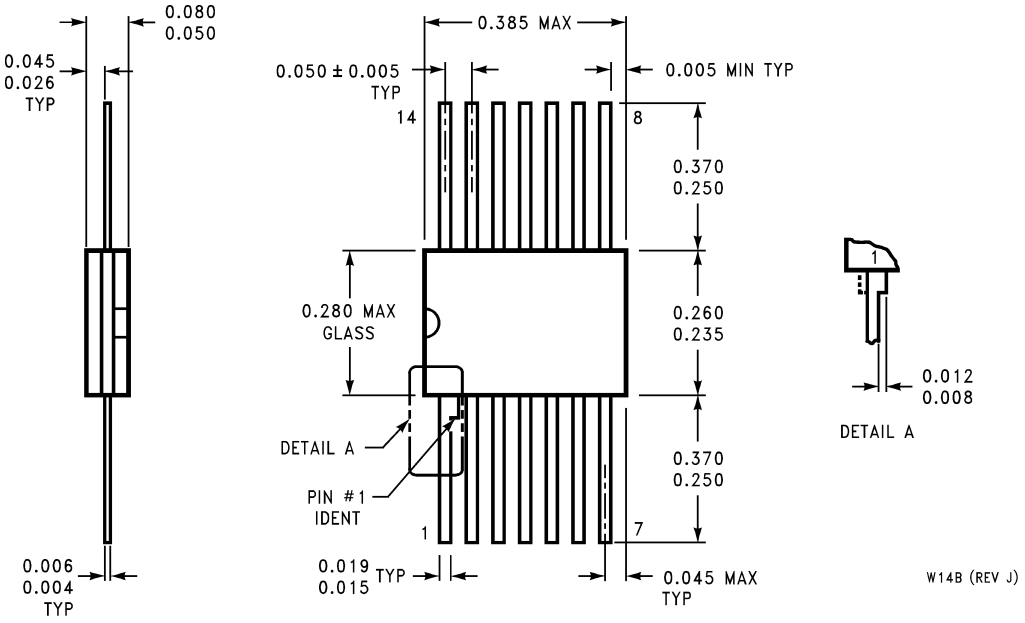


MX S.O. Package (M)
Order Number LM324M, LM324MX, LM324AM, LM324AMX, LM2902M or LM2902MX
NS Package Number M14A

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)

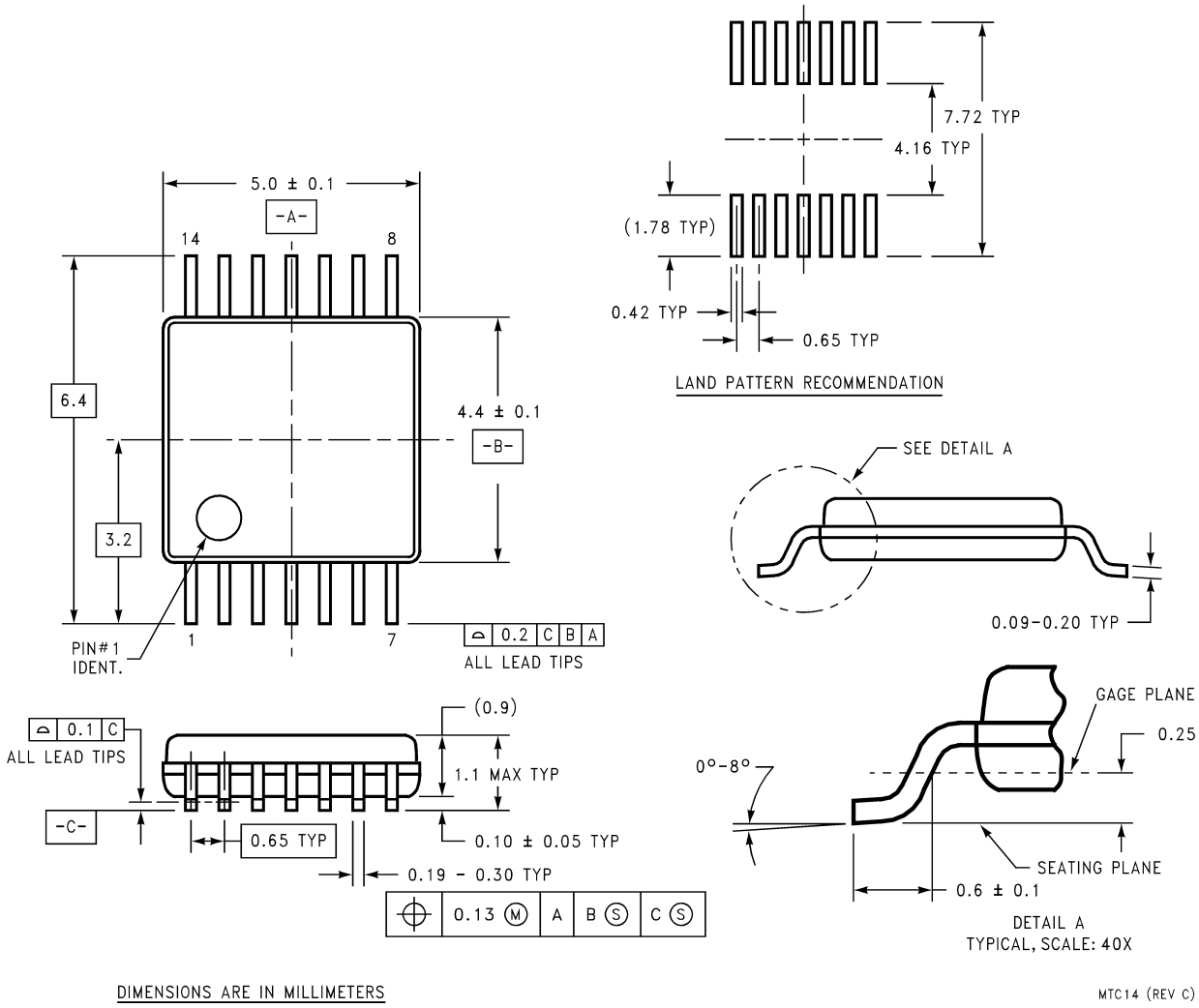


Molded Dual-In-Line Package (N)
Order Number LM324N, LM324AN or LM2902N
NS Package Number N14A



Ceramic Flatpak Package
Order Number JL124ABDA, JL124ABZA, JL124ASDA, JL124BDA, JL124BZA, JL124SDA, LM124AW/883, LM124AWG/883, LM124W/883 or LM124WG/883
NS Package Number W14B

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



14-Pin TSSOP
Order NumberLM324MT or LM324MTX
NS Package Number MTC14

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
Americas
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com
www.national.com

National Semiconductor Europe
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: ap.support@nsc.com

National Semiconductor Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507

PC/104 Specification

Version 2.4

August 2001

Please Note

This specification is subject to change without notice. While every effort has been made to ensure the accuracy of the material contained within this document, the PC/104 Embedded Consortium shall under no circumstances be liable for incidental or consequential damages or related expenses resulting from the use of this specification. If errors are found, please notify the PC/104 Consortium.

REVISION HISTORY

Version 1.0, March 1992 - Initial release.

Version 2.1, July 1994 - Revised specification incorporating changes to conform with IEEE P996.1 draft version D1.00:

a. Changed bus options. Eliminated the "option 2" configurations having right-angle P1 and P2 connectors. Created new "option 2" configurations similar to "option 1," but without the stackthrough pins. Added a statement indicating that a P2 connector may be included on 8-bit modules, if desired.
b. Added two additional mounting holes to 8-bit bus versions, making the mounting hole patterns of both 8- and 16-bit modules identical.
c. Added an I/O connector region along the bus edge of the module.
d. Increased widths of I/O mating-connector regions from 0.4" to 0.5".
e. Changed lengths of I/O mating-connector regions so that their edges align with the outer edges of the annular rings of adjacent mounting holes.
f. Reduced the bus drive requirement on the signals that had been specified at 6 mA to 4 mA.
g. Added specification of module power requirements.
h. In Appendix C, Section 3, changed minimum value of pullup resistance on shared interrupt line from 10K to 15K ohms.
i. Added a section defining levels of PC/104 conformance.

Version 2.2, September 1994

a. Added correction sheet showing revised schematic for Appendix C.

Version 2.3, June 1996

a. Incorporated correction to Appendix C schematic.
b. Changed P2 connector Pin 1 designation in 16-bit module dimension drawings.
c. Added metric dimensions, including metric versions of module dimension drawings.
d. Minor formatting changes.

Version 2.4, August 2001

a. Added Appendix D Connector Specifications.
b. Removed all specific company references.
c. Corrected Consortium address and phone numbers
d. Added new reference for ISA specification
e. Cleaned up mechanical drawings

TABLE OF CONTENTS

(This page left blank intentionally)

PC/104 SPECIFICATION

Version 2.4 – August 2001

1. INTRODUCTION

While the PC and PC/AT architectures have become extremely popular in both general purpose (desktop) and dedicated (non-desktop) applications, its use in *embedded* microcomputer applications has been limited due to the large size of standard PC and PC/AT motherboards and expansion cards.

This document supplies the mechanical and electrical specifications for a compact version of the ISA (PC and PC/AT) bus, optimized for the unique requirements of embedded systems applications. The specification is herein referred to as "PC/104", based on the 104 signal contacts on the two bus connectors (64 pins on P1, plus 40 pins on P2).

Briefly, the needs of embedded applications have been satisfied by PC/104, through the following key differences from standard ISA bus:

- Reducing the form-factor, to 3.550 by 3.775 inches (90 by 96 mm).
- Eliminating the need for backplanes or card cages, through its self-stacking bus.
- Minimizing component count and power consumption (to typically 1-2 Watts per module), by reducing required bus drive on most signals to 4 mA.

PC/104 specifies two module versions — 8-bit and 16-bit — which correspond to the PC and PC/AT bus implementations, respectively.

The remainder of this specification covers the *differences* from the ISA bus as detailed in Edward Solari's book ISA and EISA Theory and Operation published by Annabooks. Designers of modules and systems based on PC/104 should be familiar with the ISA specification. It is available from:

Annabooks
12860 Danielson Court
Poway, CA USA 92064
Tel 800.462.1042 or 858.435.2000
Fax 858.391.5616
On the web at http://www.annabooks.com

If errors are found in this document, please send a written copy of the suggested corrections to:

PC/104 Embedded Consortium
1060 North Fourth Street
San Jose, CA USA 95112
Tel 650.903.8304
Fax 408.999.0344
E-mail info@pc104.org
On the web at http://www.pc104.org

2. MECHANICAL SPECIFICATIONS

2.1 Module Dimensions

PC/104 modules can be of two bus types, 8-bit and 16-bit. These correspond to the PC and PC/AT buses, respectively. The detailed mechanical dimensions of these two PC/104 bus types are provided in Appendix A.

2.2 Bus Options

As shown in the figures in Appendix A, each of the two bus types (8-bit and 16-bit) offers two *bus options*, according to whether or not the P1 and P2 bus connectors extend through the module as "stackthrough" connectors. These options are provided to help meet the tight space requirements of embedded applications.

Figure 1 illustrates a typical module stack including both 8- and 16-bit modules, and shows the use of both the "stackthrough" and "non-stackthrough" bus options. As shown in Figure 1, when 8- and 16-bit modules are combined in a stack, the 16-bit modules must be stacked below (i.e., on the "secondary side" of) the 8-bit modules. A "passive" P2 bus connector may optionally be included in the design of 8-bit modules, to allow the use of these modules anywhere in a stack.

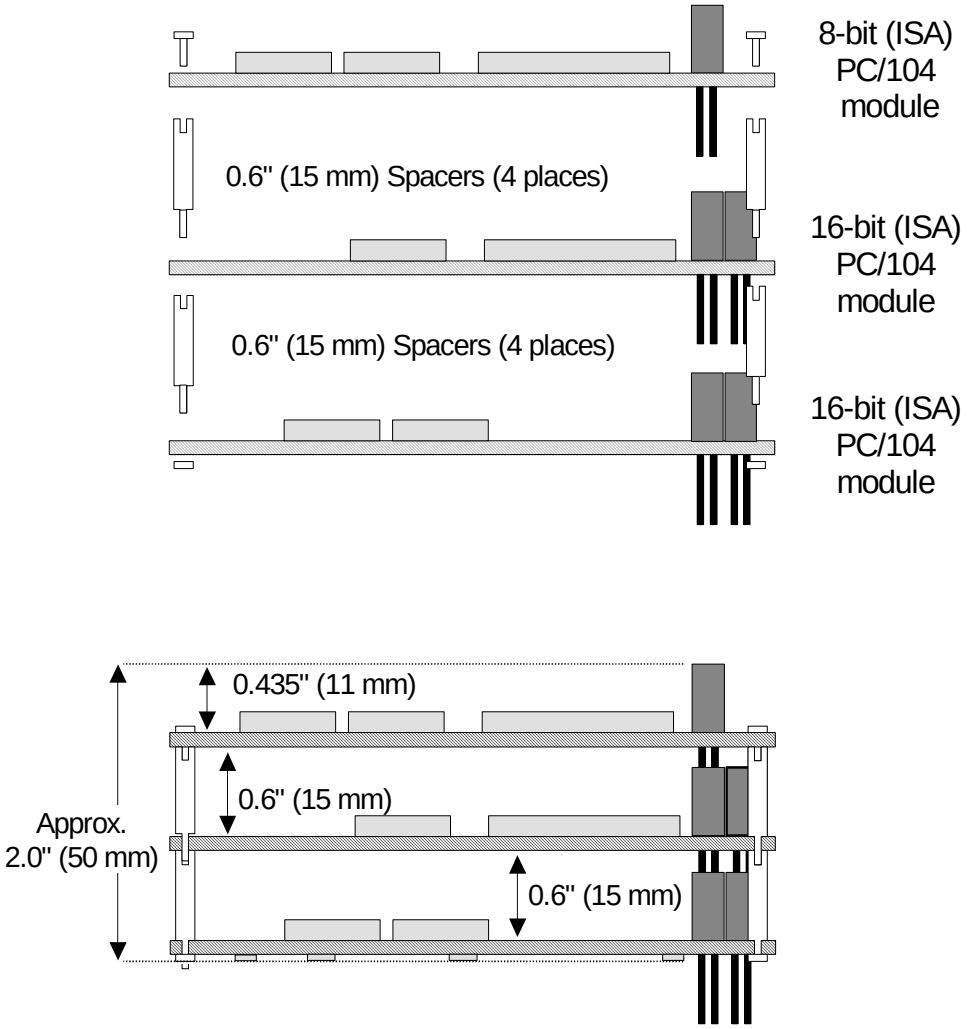


Figure 1. Typical Module Stack

3. ELECTRICAL SPECIFICATIONS

3.1 Signal Functions and Assignments

3.1.1 Signal Definitions

All PC/104 bus signals are identical in definition and function to their ISA counterparts.

3.1.2 Signal Assignments

Signals are assigned in the same order as on the edgecard connectors of ISA, but transformed to the corresponding header connector pins. Signal assignments for the J1/P1 and J2/P2 connectors are given in Appendix B.

3.1.3 Added Grounds

Several ground pins have been added, to maximize bus integrity. See Appendix B.

3.1.4 Key Locations

Key locations - consisting of omitted pins on P1 and P2, and plugged holes on J1 and J2 - have been designated on each bus connector, to help assure proper connector mating. See Appendix B.

3.2 AC Signal Timing

All PC/104 bus signals are identical in signal timing to their ISA counterparts.

3.3 DC Signal Levels

All PC/104 bus signal DC logic high and logic low voltage levels are identical to their ISA counterparts.

3.4 Bus Drive Current

To reduce component count and minimize power consumption and heat dissipation most bus signals have a reduced bus drive requirement of 4 mA. The exception is open collector driven signals which must drive 330 ohm pullup resistors defined by the ISA specification. This allows direct driving of the bus by many ASIC devices, and by HCT family logic.

Specifically, the following signals must be driven with devices capable of providing 20 mA sink current (as indicated in ISA):

MEMCS16*
IOCS16*
MASTER*
ENDXFR*

All other signals may be driven with devices capable of providing 4 mA sink current.

3.5 Interrupt-Sharing Option

The ISA specification briefly mentions an optional means to share a single bus interrupt line among multiple interrupting devices. Appendix C provides a design guideline which can help ensure compatibility of interrupt-sharing among PC/104 modules.

3.6 Bus Termination Option

As in ISA, termination of the PC/104 bus signals may be desired in some systems to increase data integrity and system reliability. When termination is included, *AC termination* networks must be used to provide termination close to the characteristic impedance of the signal lines without exceeding the DC output current capabilities of the drivers.

As in the ISA specification, the recommended network consists of a resistor-capacitor network of 40-60 ohms in series with 30-70 pF, connected between each bus signal and ground.

Whether termination is needed, and where it should be located, is dependent on the specific system configuration and must be determined by the system designer.

3.7 Module Power Requirements

The operating voltage range and maximum power requirements of each module are given in Table 1. Each module shall not draw more than the operating current indicated. The total power requirement of a PC/104 module stack is the sum of that required by each of the modules in the stack. Operating voltages, which refer to the voltage measured at the appropriate bus connector pins of any given module, are specified to ±5 percent. Only those voltages required by modules in a system need be supplied to the bus.

Table 1. Module Power Requirements

Nominal Voltage	Maximum Voltage	Minimum Voltage	Maximum Current
+12 Volts	+12.6 Volts	+11.4 Volts	1.0 Amp
+5 Volts	+5.25 Volts	+4.75 Volts	2.0 Amp
-5 Volts	-4.75 Volts	-5.25 Volts	0.2 Amp
-12 Volts	-11.4 Volts	-12.6 Volts	0.3 Amp

4. LEVELS OF CONFORMANCE

This section provides terminology intended to assist manufacturers and users of PC/104 bus-compatible products in defining and specifying conformance with the PC/104 Specification.

4.1 PC/104 "Compliant"

This refers to "PC/104 form-factor" devices that conform to all non-optional aspects of the PC/104 Specification, including both *mechanical* and *electrical* specifications.

4.2 PC/104 "Bus-compatible"

This refers to devices which are not "PC/104 form-factor" (i.e., do not comply with the module dimensions of the PC/104 Specification), but provide a male or female PC/104 *bus connector* that meets both the *mechanical* and *electrical* specifications provided for the PC/104 bus connector.

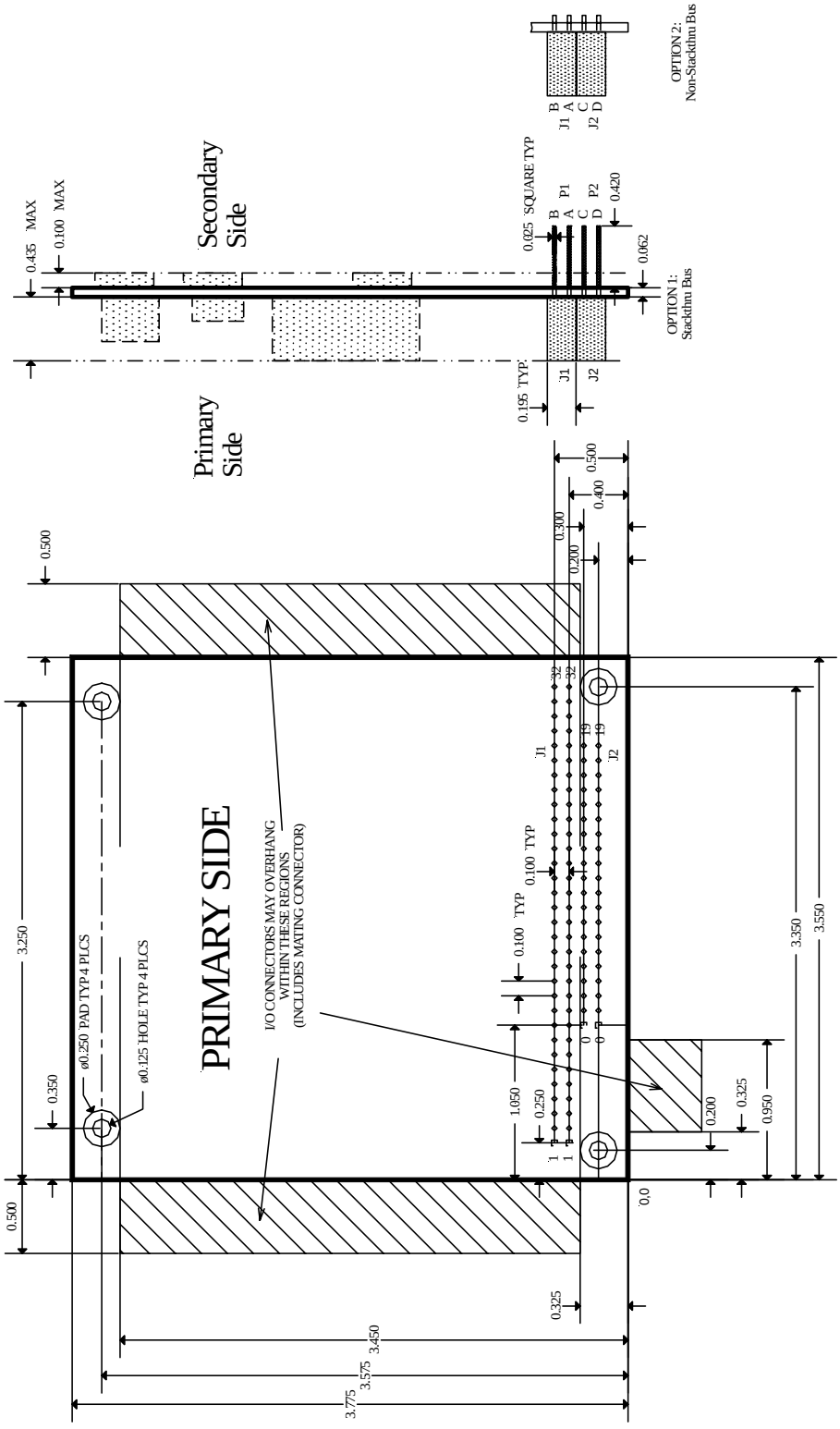
(This page left blank intentionally)

APPENDIX A

MODULE DIMENSIONS

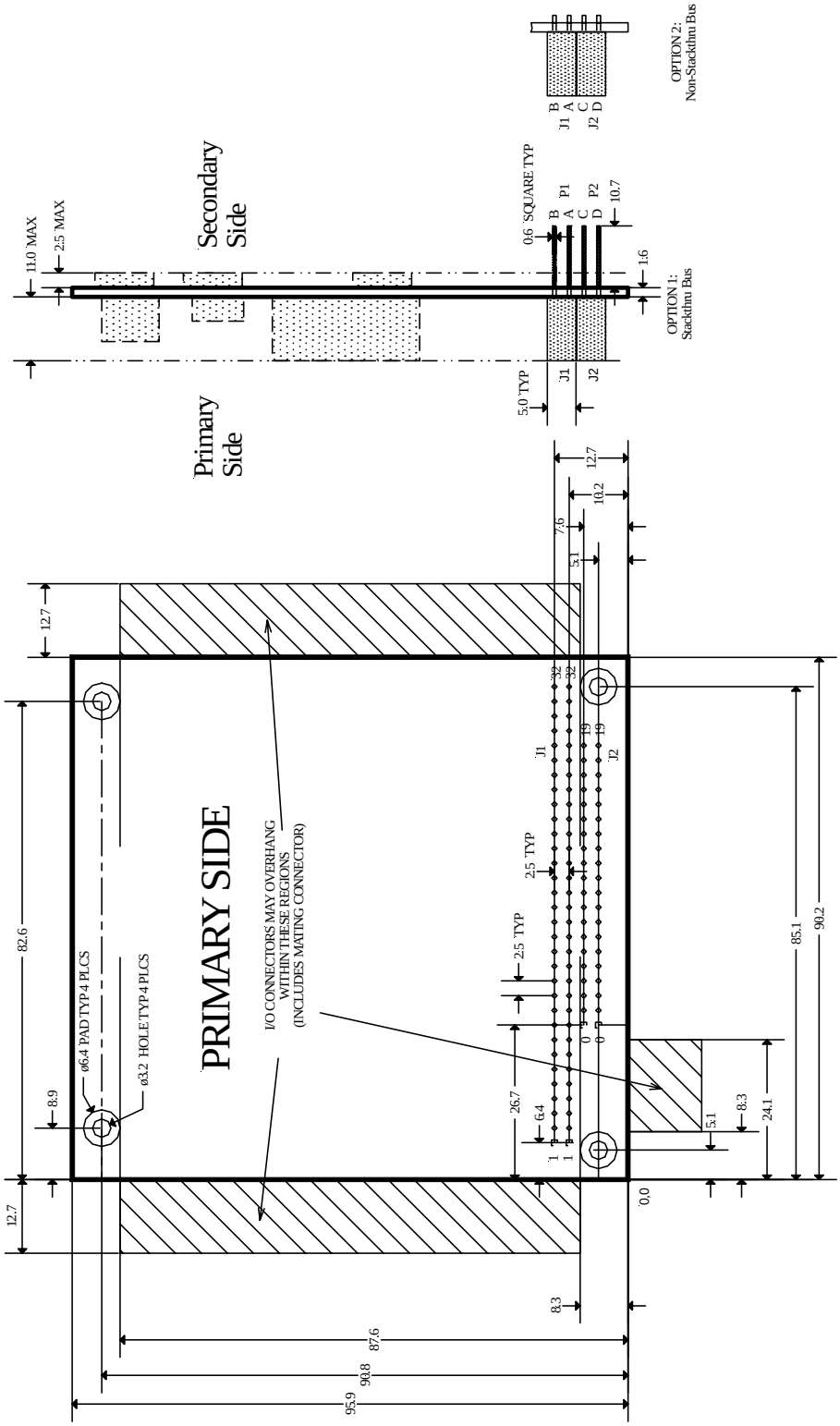
PC/104 16-BIT MODULE DIMENSIONS (ENGLISH)

Dimensions are in inches.



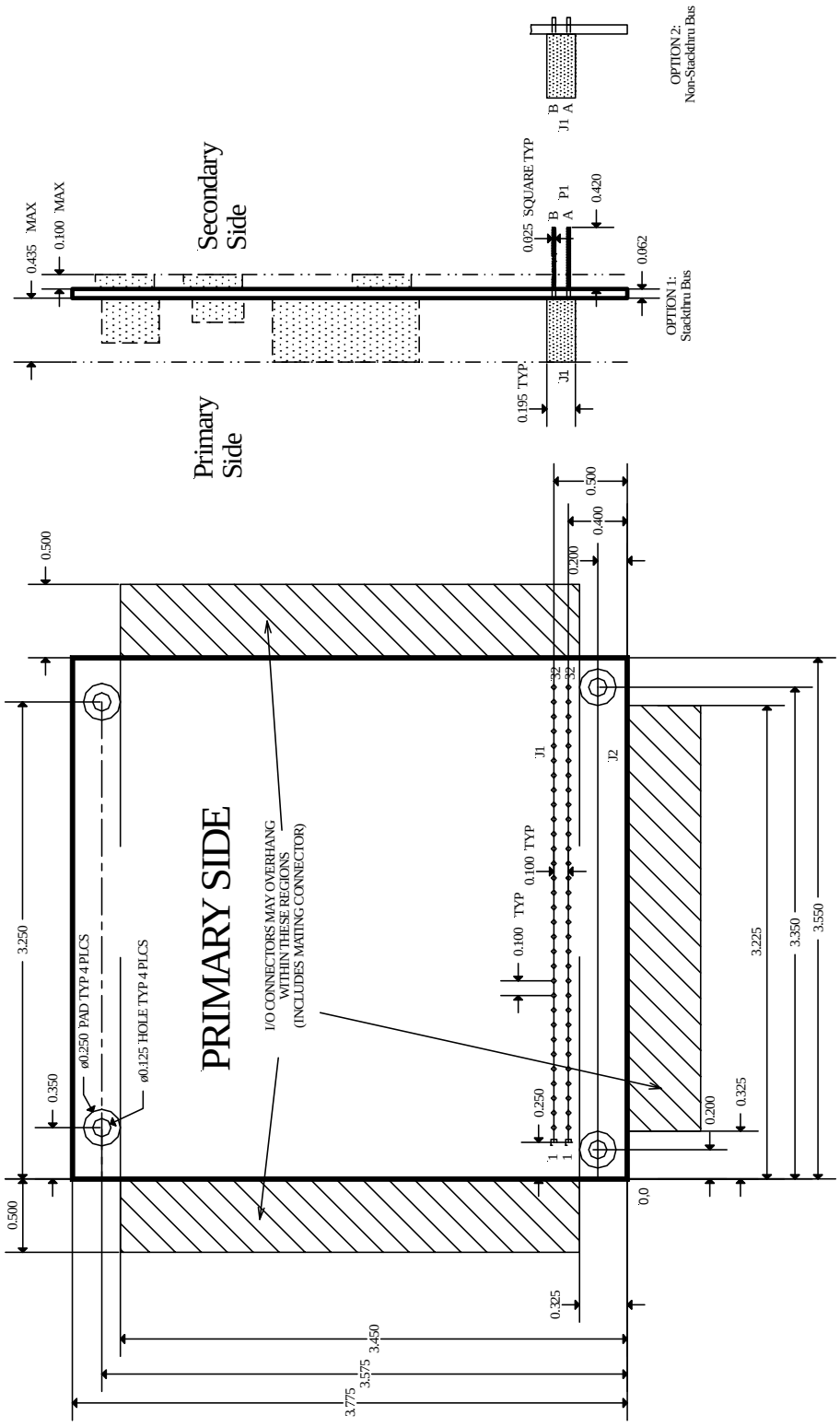
PC/104 16-BIT MODULE DIMENSIONS (METRIC)

Dimensions are in millimeters.



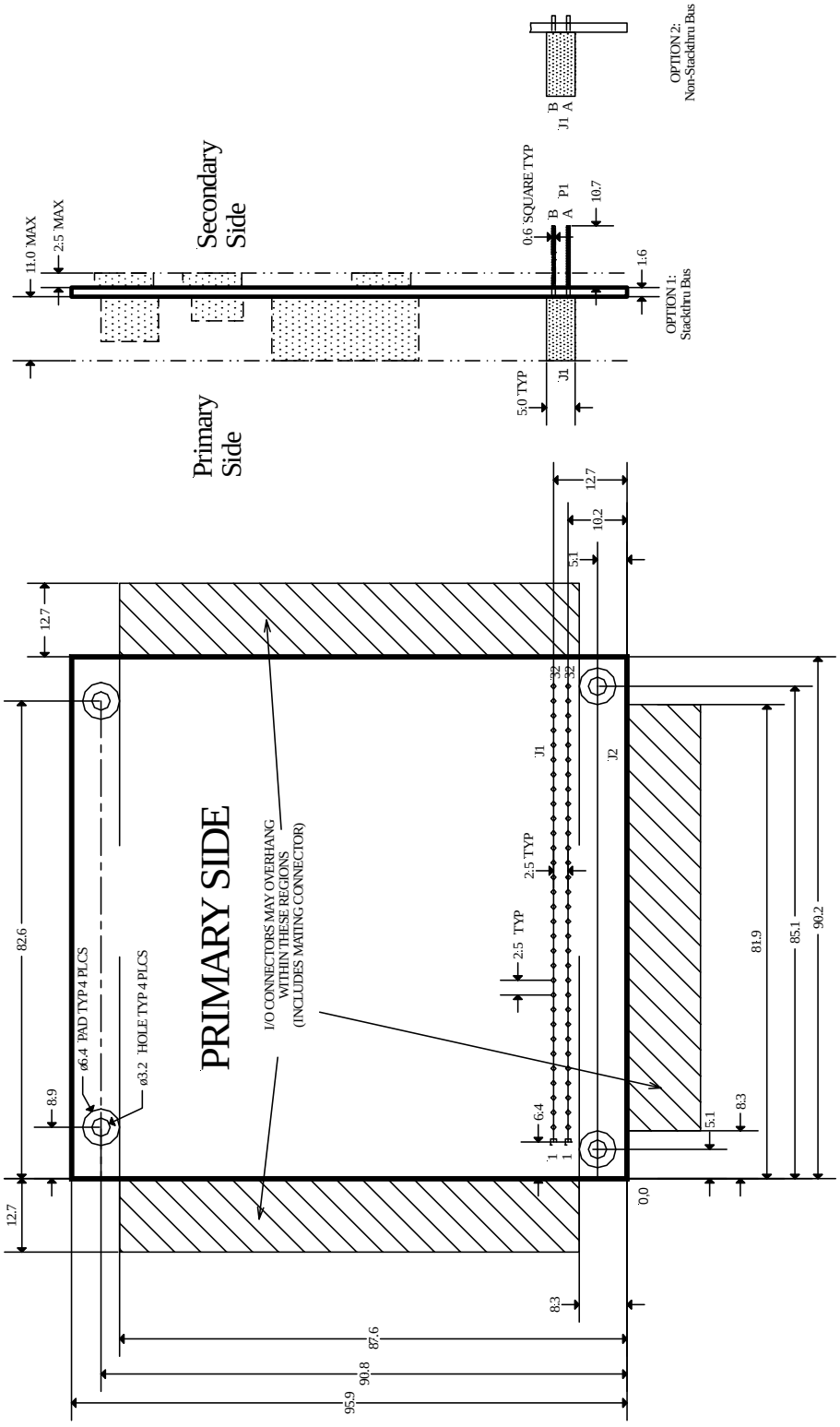
PC/104 8-BIT MODULE DIMENSIONS (ENGLISH)

Dimensions are in inches.



PC/104 8-BIT MODULE DIMENSIONS (METRIC)

Dimensions are in millimeters.



(This page left blank intentionally)

APPENDIX B

BUS SIGNAL ASSIGNMENTS

Appendix B. PC/104 Bus Signal Assignments

Pin Number	J1/P1 Row A	J1/P1 Row B	J2/P2 Row C[1]	J2/P2 Row D[1]
0	--	--	GND	GND
1	IOCHCHK*	GND	SBHE*	MEMCS16*
2	SD7	RESETDRV	LA23	IOCS16*
3	SD6	+5V	LA22	IRQ10
4	SD5	IRQ9	LA21	IRQ11
5	SD4	-5V	LA20	IRQ12
6	SD3	DRQ2	LA19	IRQ15
7	SD2	-12V	LA18	IRQ14
8	SD1	ENDXFR*	LA17	DACK0*
9	SD0	+12V	MEMR*	DRQ0
10	IOCHRDY	(KEY)[2]	MEMW*	DACK5*
11	AEN	SMEMW*	SD8	DRQ5
12	SA19	SMEMR*	SD9	DACK6*
13	SA18	IOW*	SD10	DRQ6
14	SA17	IOR*	SD11	DACK7*
15	SA16	DACK3*	SD12	DRQ7
16	SA15	DRQ3	SD13	+5V
17	SA14	DACK1*	SD14	MASTER*
18	SA13	DRQ1	SD15	GND
19	SA12	REFRESH*	(KEY)[2]	GND
20	SA11	SYSCLK	--	--
21	SA10	IRQ7	--	--
22	SA9	IRQ6	--	--
23	SA8	IRQ5	--	--
24	SA7	IRQ4	--	--
25	SA6	IRQ3	--	--
26	SA5	DACK2*	--	--
27	SA4	TC	--	--
28	SA3	BALE	--	--
29	SA2	+5V	--	--
30	SA1	OSC	--	--
31	SA0	GND	--	--
32	GND	GND	--	--

NOTES:
1. Rows C and D are not required on 8-bit modules. See Section 2.2.
2. B10 and C19 are key locations. See Section 3.1.4.
3. Signal timing and function are as specified in ISA specification.
4. Signal source/sink current differ from ISA values. See Section 3.4.

APPENDIX C

INTERRUPT-SHARING OPTION

APPENDIX C

INTERRUPT-SHARING OPTION

C.1 Introduction

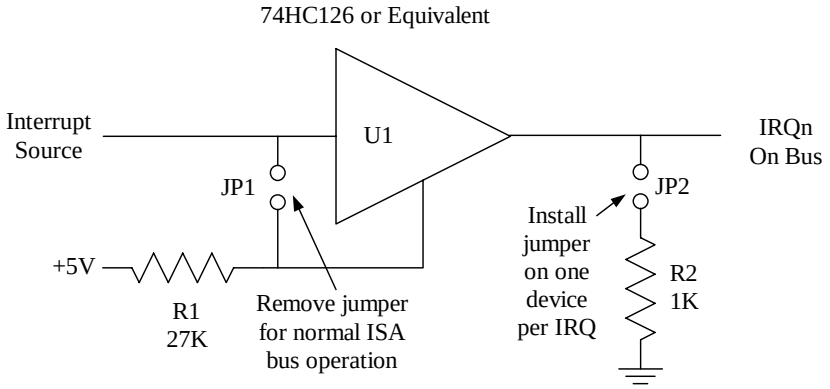
The Interrupt Request lines (IRQn's) on the ISA bus are *active high*. Consequently, the usual technique of wire-ORing open-collector driven *active low* bus signals cannot be used for interrupt-sharing in the PC bus architecture.

The ISA specification briefly mentions an optional means to share a single bus interrupt line among multiple interrupting devices. This appendix provides design guidelines which can help assure compatibility of interrupt-sharing among PC/104 modules.

C.2 Recommended Circuit

A circuit similar to that shown in the figure below can provide interrupt-sharing of the *active high* IRQ signals on the ISA bus, given a few system-level restrictions (see below).

> *NOTE: This recommendation does not comply with the ISA specification, since it is not possible to implement interrupt-sharing in an ISA compatible manner.*



Typical Interrupt-Sharing Circuit

C.3 Restrictions

All bus devices sharing a common interrupt must be equipped with a suitable interrupt-sharing circuit (see the Figure, above) and must meet the following two restrictions:

- The interrupt line being shared must not have a pullup resistance (to +5 volts) less than 15K ohms anywhere in the system. (Typically, the pullup resistance is located on the CPU module, so this is generally a restriction on the design of the CPU module.) Resistive bus termination will generally violate this restriction; use AC termination instead (Section 3.6).

- The interrupt line being shared must have one (and only one) pulldown resistor (1K ohms) connected between the IRQ line and ground. Resistive bus termination will generally violate this restriction; use AC termination instead.

C.4 "ISA Compatibility" Option Jumper

The ISA specification calls for using a 2.2K pullup resistor on each of the IRQ lines, which violates the 15K minimum pullup resistance allowed with the recommended interrupt-sharing circuit. In systems having this value of pullup, devices with the circuit shown in the above figure can be made compatible by disabling their interrupt-sharing circuit. This is accomplished by unshorting both JP1 and JP2, resulting in a normal ISA (non-shared) interrupt configuration (but with the reduced bus drive common to other PC/104 bus signals).

(This page left blank intentionally)

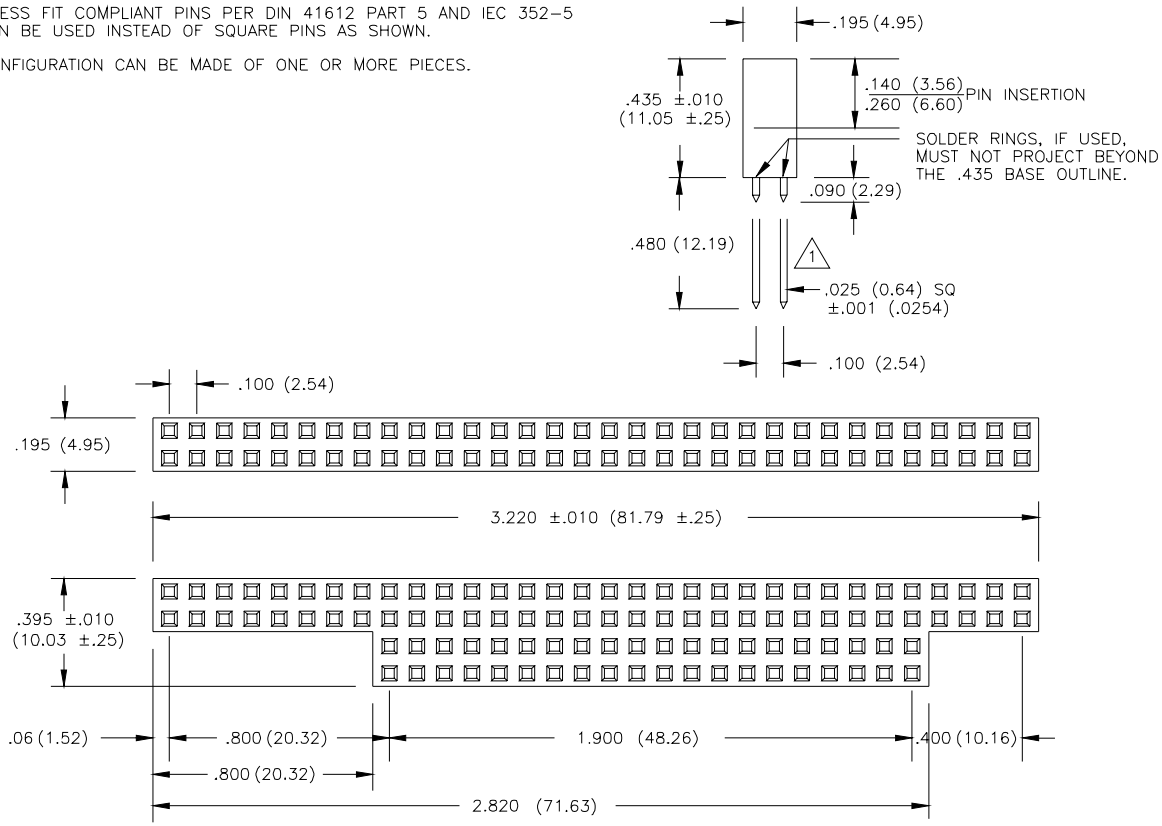
APPENDIX D

CONNECTOR SPECIFICATIONS

PC/104 8-BIT AND 16-BIT CONNECTOR DIMENSIONS

NOTES:

1. PRESS FIT COMPLIANT PINS PER DIN 41612 PART 5 AND IEC 352-5 CAN BE USED INSTEAD OF SQUARE PINS AS SHOWN.
2. CONFIGURATION CAN BE MADE OF ONE OR MORE PIECES.



PC/104 8-BIT AND 16-BIT CONNECTOR SPECIFICATIONS

MATERIALS

HOUSING:	HIGH TEMP THERMOPLASTIC, UL RATED 94-V0
CONTACT:	PHOSPHOR BRONZE
SOLDER:	TIN-LEAD (63-37), IF APPLICABLE
SOLDER CLIP:	ALUMINUM ALLOY, IF APPLICABLE

CONTACT FINISH

FEMALE INTERFACE:	20 MICROINCHES MINIMUM HARD GOLD
MALE INTERFACE:	GOLD FLASH MINIMUM
SOLDER TAIL:	100 MICROINCHES MINIMUM SOLDER
UNDERPLATE:	50 MICROINCHES MINIMUM NICKEL

MECHANICAL PERFORMANCE

INSERTION FORCE:	3.5 OUNCE PER PIN MAXIMUM
WITHDRAW FORCE:	1 OUNCE MINIMUM AVERAGE
NORMAL FORCE:	50 GRAMS MINIMUM (PER BEAM)
DURABILITY:	50 CYCLES MINIMUM
OPERATING TEMP:	-55° C TO +85° C

ELECTRICAL PERFORMANCE

CONTACT RESISTANCE:	<30 MILLIOHMS MAXIMUM
CURRENT CAPACITY:	1 AMP CONTINUOUS PER PIN
DIELECTRIC STRENGTH:	1000 VAC
INSULATION RESISTANCE:	5,000 MEGOHMS MINIMUM

Bibliografía.

- Aníbal Ollero, 2001. "Robótica, Manipuladores y Robots Móviles". Marcombo – Boixareu editores.
- Manuel Mazo y otros, 2000. "Automatización de una Carretilla Industrial". Departamento de Electrónica de la Universidad de Alcalá de Henares.
- R. K. Miller y colaboradores, 1994. "A Camera Space Control System for an Automated Forklift". IEEE Transactions on Robotics and Automation, Vol. 10, nº 5, octubre.
- A. Lara y otros, 1995. "Mobile Robot for an Industrial Environment". Departamento de Ingeniería Mecánica de la ETSII de la Universidad de Valladolid.
- J. Delgado y otros, 1995. "Automatic and Flexible Transport System Based on Mobile Robots". Departamentos de Expresión Gráfica y de Ingeniería de Control de Sistemas de la Universidad de Valladolid.
- Rafael Martín de Agar, 2002. "Interfaz para Aplicaciones de Alto Nivel que Implementen Algoritmos de Control para el Vehículo Romeo 4R". Proyecto Final de Carrera, Escuela Superior de Ingenieros, Sevilla.
- Matt Welsh, 1992 – 1996. "Linux, Instalación y Primeros Pasos". Traducido por el proyecto LuCAS en agosto de 1998.
- Alexandro Rubini y Jonathan Corbet, 1998. "Linux Device Drivers". O´Reilly editores.
- Siman, 1997. Manual de usuario del sensor láser Robosense de Siman.
- Axiom, 1994. Manual de usuario del módulo AX10410 de Axiom.
- Linde, 1999. Instrucciones de servicio de la carretilla elevadora Linde E16.
- Axiom, 1997. Manual de usuario de la placa base SBC 8251 de Axiom.
- Octagon Systems. Manual de usuario de la placa 5066 de Octagon Systems.

- Tech 80, 1996. Manual de usuario del módulo 5928 de Tech80.
- Axiom, 1994. Manuales de usuario de los módulos AX10425, AX10420, AX10465 y AX10445 de Axiom.
- Danfoss. Manual de la válvula proporcional tipo PVG 32.
- Atos, 2002. "Solenoid Directional Valves type DHI, DHU, DHO". Manual de características.