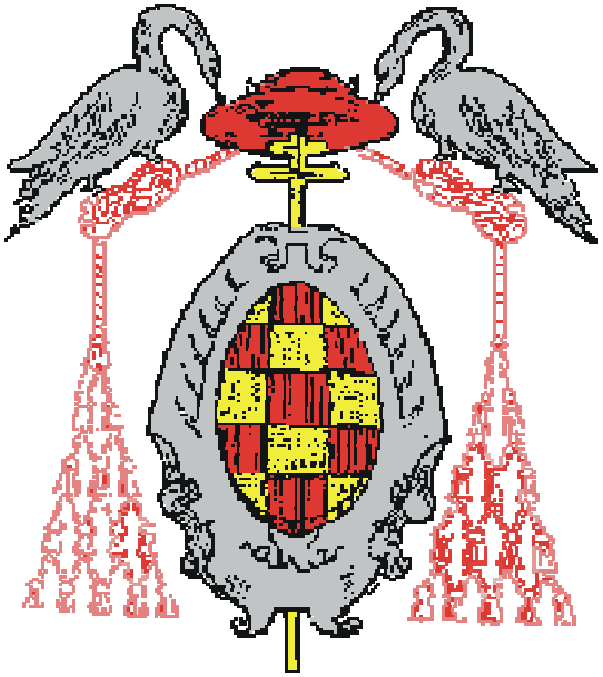
UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

INGENIERÍA TÉCNICA INDUSTRIAL
ESPECIALIDAD EN ELECTRÓNICA INDUSTRIAL

Trabajo Fin de Carrera



Guiado de una silla de ruedas mediante joystick y soplido por bus CAN

Juan Bellón Álvarez

Septiembre de 2009

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

INGENIERÍA TÉCNICA INDUSTRIAL
ESPECIALIDAD EN ELECTRÓNICA INDUSTRIAL

Trabajo Fin de Carrera

Guiado de una silla de ruedas mediante joystick y soplido por bus CAN

Autor: Juan Bellón Álvarez
Tutor: Juan Carlos García García

TRIBUNAL:

Presidente: D. Manuel Mazo Quintas

Vocal 1º: D. Marta Marrón Romera

Vocal 2º: D. Juan Carlos García García

CALIFICACIÓN.....

FECHA.....

A mis padres y mis hermanos y como no a toda la gente que he conocido estos años, algunos han sido un poco pesadotes pero se les echara de menos. Para que no quedéis en el olvido ay vais, Taberné contigo empezó todo pero abandonaste barco, luego llegaron gentuza como el "puto revi", los "pichones" y los "juevistas" que intentaremos seguir dando guerra. Sois muchos para nombraros a todos bueno esta odisea se acabó.
VINI VIDI VINCI

I. RESUMEN

El principal objetivo del proyecto SIAMO II es dotar a una silla de ruedas de las herramientas hardware y software adecuadas para conseguir un guiado semiautomático de la misma mediante técnicas robóticas y a su vez conseguir tener un sistema modulable, escalable y con interfaces de comunicación normalizados.

Este proyecto tiene como objetivo proseguir los trabajos de investigación procedentes del proyecto SIAMO, con la colaboración del profesor Paulo Amaral de la Universidad Federal de Espirito Santo (Brasil) y José Basterrechea alumno de la escuela politécnica de Alcalá

CAPÍTULO 1

INTRODUCCIÓN

1.1. Motivaciones

Nos basamos en el proyecto "Guiado semiautomático de una silla de ruedas" de Eduardo Sebastián Martínez, 1999, como principal antecedente. Este proyecto está basado en tecnología **LonWorks** y **NeuronChip**.

En nuestro proyecto nos basamos en una tecnología más moderna y actual la cuál puede ser fácilmente actualizable, esta tecnología se basa en el uso del microcontrolador LPC2129 que dispone del bus de comunicaciones normalizado CAN *(Controller Area Network),* esta tecnología nos permite configurar nuestro sistema mediante cualquier PC convencional.

El trabajo desarrollado a lo largo de este proyecto se halla englobado en el campo de la robótica móvil con el fin de apoyo a la discapacidad.

Es por eso, que en este proyecto se incorporan ciertos elementos tanto hardware como software a una silla de ruedas para conseguir un guiado semiautomático.

1.2. Objetivos perseguidos en el trabajo

Este proyecto se basa principalmente en algunas funciones de bajo nivel de la maquina como son el guiado mediante joystick, el guiado mediante soplido, el guiado mediante joystick de cabeza a través de un PC y el control de los distintos sensores que dispone.

El principal objetivo del trabajo realizado es conseguir un sistema modulable, de nodos independientes que se comunican entre si mediante el bus CAN. De esta forma nuestro sistema puede ser incrementado con nuevos nodos para la realización de nuevas tareas.

1.3. Estructura de la memoria

Esta memoria está distribuida en cinco capítulos. El contenido de éstos se describe a continuación:

Capítulo 1. Introducción: Capítulo actual, en él se introduce el trabajo desarrollado.

Capítulo 2. Estructura general del sistema: En este capítulo se presentan la estructura de la silla de ruedas tanto a nivel hardware como a nivel software.

Capítulo 3. Descripción del funcionamiento de los nodos del sistema: En este capítulo se presenta la estructura detallada del sistema.

Capítulo 4. Resultados obtenidos: En este capítulo se detallan los comportamientos obtenidos en el manejo de la silla. Se anexan distintas imágenes de los diferentes modos de funcionamiento y a su vez se adjuntan en un CD en formato digital diversos videos de los funcionamientos.

Capítulo 5. Conclusiones y trabajos futuros: En este capítulo incluimos los datos y las conclusiones obtenidas sobre el trabajo realizado y posibles mejoras sobre nuestro sistema.

A continuación de la memoria se incluyen el manual de usuario, el pliego de condiciones, el presupuesto, los planos y la bibliografía de este proyecto.

CAPÍTULO 2

ARQUITECTURA GENERAL DEL SISTEMA

Como se exponía en el capítulo de introducción en este capítulo se pasa a describir la arquitectura general del sistema y de cómo, partiendo de esa arquitectura, se pueden confeccionar distintos tipos de aplicaciones dependiendo del tipo de interfaz que se instale. En nuestro caso, primero incorporamos el sistema al robot **Alca-Roby** para posteriormente incorporarlo a **SIAMO II.**

2.1. Arquitectura del sistema

El sistema se basa en una red distribuida la cual se divide en dos niveles, por un lado se encuentra el sistema de alto nivel, el cual está formado por el monitor táctil y el PC con el cual el usuario puede realizar distintas tareas como obtener información referente al sistema, tener conexión a internet, etc. Por otra parte se encuentra el sistema de bajo nivel constituido por los distintos nodos del sistema, como son la tarjeta de los sensores o la del joystick.

La interconexión entre los distintos nodos y niveles se realiza principalmente por medio del bus de datos CAN.

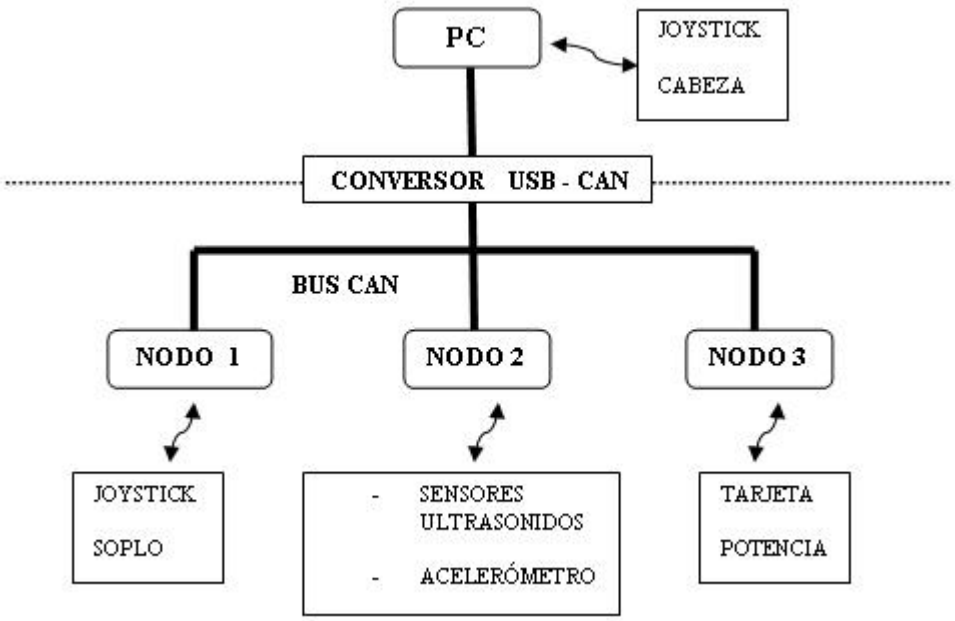


Figura 2. 1 Arquitectura general

La Figura 2. 1 muestra el esquema general, en el se encuentran representados los nodos de que dispone el sistema y los distintos tipos de interfaz que existen como son el joystick, el sistema de comando de soplo, etc.

2.1.1. Elementos y dispositivos hardware

La *arquitectura hardware* del sistema está formada por los distintos elementos que constituyen el sistema. Este se divide en tres partes fundamentales:

- Sistema de alimentación
- Sistema de Alto Nivel
- Sistema de Bajo Nivel

La **¡Error! No se encuentra el origen de la referencia.** muestra la distribución de los distintos sistemas y de los elementos que los constituyen.

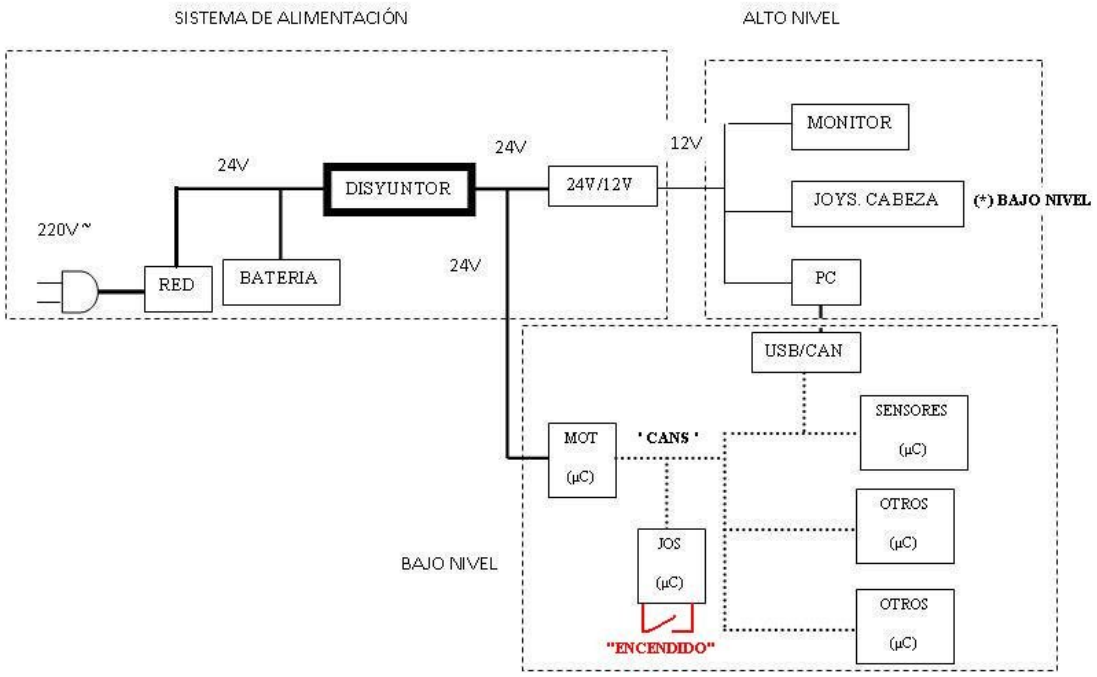


Figura 2. 2 Sistema de alimentación, alto y bajo nivel (incluye el joystick de cabeza)

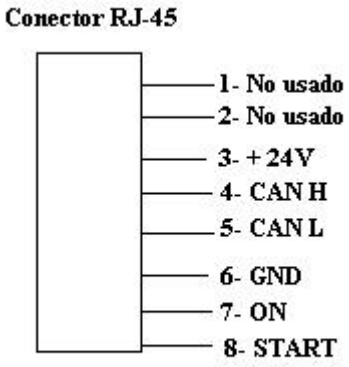


Figura 2. 3 Distribución de cables en el bus de bajo nivel

2.1.1.1. Sistema de alimentación

Tal y como muestra la Figura 2. 2 el sistema de alimentación está constituido por dos baterías conectadas en serie que proporcionan una tensión de 24V, un transformador de 220V-24V y un conversor de 24V-12V.

El sistema se alimenta de la energía proporcionada por las baterías, estas van conectadas al transformador 24V-12V del cual se obtiene la alimentación necesaria para el funcionamiento del PC, monitor. La energía utilizada por los distintos nodos proviene directamente de las baterías, para ello todos los nodos disponen de fuentes conmutadas que

transforman los 24V provenientes de las baterías en 5V para el funcionamiento de los distintos nodos. Se utilizan fuentes conmutadas porque tienen un mayor rendimiento.

Entre las baterías y el conversor se ha instalado un interruptor magneto-térmico que será el encargado de cortar el suministro de energía en caso de que ocurra algún fallo en la alimentación.

El transformador 220V-24V es utilizado para realizar la carga de las baterías a través de la red eléctrica normal, éste se encuentra conectado directamente a ellas.

El encendido del bajo nivel realiza mediante un interruptor situado en el nodo del joystick. La activación de éste provoca el encendido de todos los nodos de bajo nivel. En los elementos del alto nivel se realiza en los interruptores de encendido de cada elemento.

2.1.1.2. Sistema de Alto Nivel

El Alto nivel esta formador por el monitor y el PC. Éste sistema engloba todas las funciones que puede realizar el usuario a nivel físico interactuando él mismo directamente con el sistema.

En este caso las funciones que el usuario puede desempeñar son las concernientes a manejar SIAMO II con el joystick de cabeza, para ello se dispone de una aplicación software. Además el usuario dispone en el monitor de información concerniente al sistema.

(*) En la **¡Error! No se encuentra el origen de la referencia.** se incluye también el joystick de cabeza como un elemento del sistema de Alto Nivel. En realidad éste es un elemento que pertenece al sistema de Bajo Nivel. El motivo de su ubicación en esa zona es que se trataba de una interfaz ya construida y tiene una alimentación de 12V. En caso de que éste se construyera de nuevo la ubicación sería en la zona de bajo nivel con el resto de los nodos.

Para la elección del PC se eligió la placa VIA EPIA MII, con una forma Mini-ITX y unas dimensiones de 17cm x 17cm y un consumo bajo. También cabe destacar que dispone de puerto Fire Wire para la conexión futura de cámaras de video. Está optimizada para las aplicaciones de digitales más demandadas actualmente, es totalmente compatible con los sistemas operativos Windows y Linux. La Tabla 2.1 muestra las características de placa elegida.

Procesador	VIA C3/EDEN EBGA, C3 1 GHz
Chipset	VIA CLE266 North Bridge VT8235M South Bridge
Memoria de sistema	1 ranura DDR266 DIMM Hasta 1 GB de tamaño de memoria
VGA	Gráficos VIA Unichrome AGP integrados con acelerador MPEG-2
Ranuras de expansión	1 PCI
IDE en placa	2 conectores UltraDMA 133/100/66
Floppy en placa	1 conector FDD
LAN en placa	VIA VT6103 10/100 Base-T Ethernet PHY
Audio en placa	Códec AC'97 VIA VT1616 de 6 canales
Salida de TV en placa	Salida de TV VIA VT1622
1394 en placa	VIA VT6307S IEEE 1394 Firewire
Panel E/S posterior	• 1 conector USB para 2 puertos USB 2.0 adicionales • 2 conectores 1394 para 2 puertos 1394 • 1 conector de audio en el panel frontal (Mic y Line Out) • 1 conector CD Audio-in • 1 Buzzer • 1 conector FIR • 1 conector CIR (conmutable para KB/MS) • 1 conector Wake-on-LAN • CPU/Sys FAN/Fan 3 • 1 conector SMBUS • 1 conector para conector de puerto serial LVDS módulo1 para un segundo puerto COM • Conector de energía ATX
BIOS	• Award BIOS • Memoria flash 2/4 Mbit

Tabla 2. 1. Datos técnicos del PC

El monitor elegido ha sido un monitor táctil de la marca ELO TOUCH, se trata de un monitor de 12'' para que su integración en el sistema no sea demasiado voluminosa. El monitor táctil permite al usuario manejar el ordenador, ya que el uso de un ratón convencional para el manejo de éste es una opción poco viable.

2.1.1.3. Sistema de Bajo Nivel

La **¡Error! No se encuentra el origen de la referencia.** muestra los elementos que constituyen el sistema de Bajo Nivel. En este nivel se localizan todos los nodos desarrollados, la tarjeta de potencia y el conversor USB – CAN.

La línea de 24V une las baterías con el nodo de la tarjeta de motores. Desde ésta se distribuyen esos 24V a la tarjeta de potencia y a los restantes nodos. La conexión entre los distintos nodos se realiza a través de un cable estándar de red con conector RJ-45. En este cable a parte de la alimentación de los nodos discurre otras señales como el bus CAN. El empleo de un cable de red estándar es porque se trata de un elemento de conexión fácil de encontrar además de tener un precio reducido. A este cable le denominaremos en adelante como 'CANS'. En **¡Error! No se encuentra el origen de la referencia.** se muestran la distribución de las señales que circulan por el bus CANS.

La conexión de todos los buses CANS se realiza a través de una regleta con varios conectores RJ-45 hembras conectados en paralelo de tal forma que se puedan añadir fácilmente nuevos nodos al sistema. La Imagen 2. 1 muestra la forma que tiene la regleta.

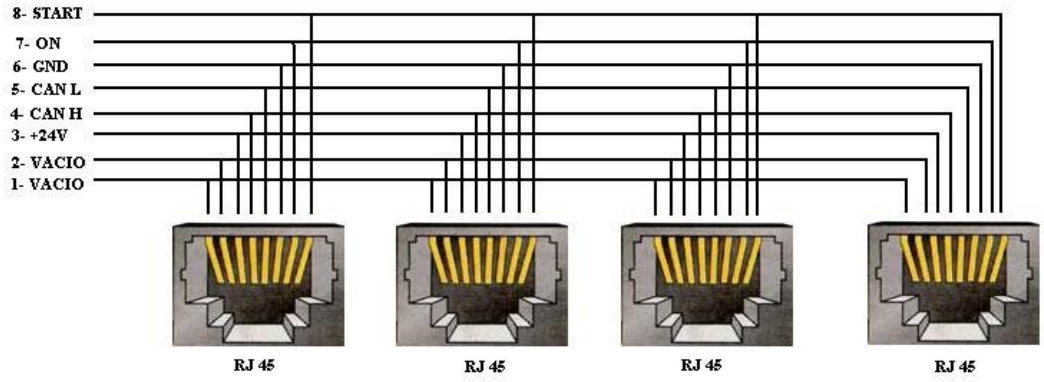


Imagen 2. 1. Regleta de conexión del bus CANS

La conexión entre el Alto Nivel y el Bajo Nivel se realiza por medio de un conversor CAN – USB. Mediante el uso de este conversor se realiza la comunicación del PC con el resto de los nodos mediante mensajes CAN. El conversor elegido es de la casa LAWICEL. Éste se muestra en la Imagen 2. 2.

Imagen 2. 2. Conversor USB - CAN

La tarjeta de potencia usada es una AX3500 de ROBOTEQ. Es la encargada de controlar los motores. Ésta tarjeta esta diseñada para trabajar con niveles de alimentación entre 12V y 24V además de soportar corrientes de hasta 60A.

La tarjeta AX3500 dispone de la posibilidad de controlar dos motores simultáneamente, este puede ser individual o mixto. También posee dos tipos de control sobre los motores, en lazo abierto y en lazo cerrado ya que dispone de la posibilidad de conectarle dos sensores de posición. La Tabla 2.2 muestra las características técnicas de la tarjeta.

Voltaje de funcionamiento	12V-24V DC
Número de canales	2
Corriente máxima	60ª
Aumento de corriente	>250ª
MOSFETs por canal	8
Resistencia de ON	5m Ohn
Rectificación síncrona	Si
Límite de corriente	Por la reducción automática de la potencia de salida en función de la carga y el tiempo transcurrido
Temperatura de protección	80ºC
Protección de voltaje	Apagar la salida por debajo de 13V y por encima de 43V

Tabla 2. 2. Datos técnicos de la tarjeta de potencia

A continuación, en la Imagen 2. 3 se muestra la tarjeta de potencia utilizada.

Imagen 2. 3. Tarjeta de potencia

Los distintos nodos del sistema están formados fundamentalmente por las tarjetas LPC2129 de Embedded Artists. La elección de estas tarjetas esta basada en el reducido tamaño lo cual permite una mejor inserción del sistema en las distintas plataformas donde se quiera llevar a cabo su instalación.

Otro factor a parte del tamaño ha sido que disponga del bus de datos CAN, el cual es la principal vía de comunicación entre los distintos nodos y el PC.

La tabla siguiente muestra las características técnicas del microcontrolador escogido.

Procesador	Microcontrolador NXP's ARM7TDMI LPC2129
Memoria Flash	256Mb
Memoria de datos	16Kb
CAN	Dos canales Can con transmisor TJA1040 ó TJA1041
Reloj	12MHZ para máxima velocidad de ejecución y velocidades estándar de CAN
Dimensiones	55 x 58 mm
Alimentación	5V DC
Conectores	2x16 puertos de entrada/salida, RS232, DSUB-9
Otros	256Kb de memoria E2PROM para I2C, descarga de programas simple y automática por canal serie, 4 capas de PCB para una mejor inmunidad al ruido, fácil de conectar a señales JTAG

Tabla 2. 3. Datos técnicos del microcontrolador

La Imagen 2. 4 nos muestra el diseño de la tarjeta LPC2129.

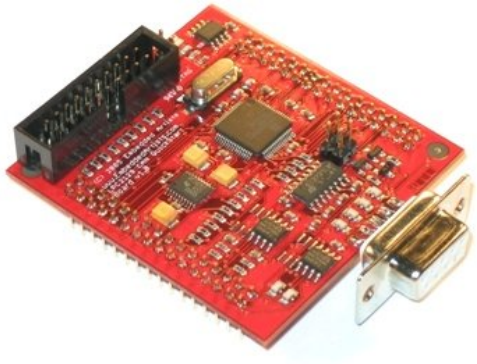


Imagen 2. 4. Tarjeta del microcontrolador

2.1.2. Nodos del sistema

El sistema completo se compone de varios nodos como se mostraba en la **¡Error! No se encuentra el origen de la referencia.**, en ella aparecen tres nodos nombrados como Nodo 1, Nodo 2 y Nodo 3. Cada uno de ellos se encarga de una función dentro del sistema.

En primer lugar se encuentra el Nodo 1, aquí se llevan a cabo las tareas de recoger y de gestionar datos provenientes de las distintas interfaces Hombre – Máquina para realizar los distintos movimientos del sistema. Las interfaces que se pueden conectar al sistema dependen de la aplicación donde esté instalado y de la persona que va a manejar ésta, una de las interfaces mas genéricas podría ser un joystick manual para realizar distintos movimientos.

Este nodo también dispone de un display en el que se puede mostrar información referente al sistema o, si es necesario, modificar algún parámetro del sistema, además también se dispone de un teclado. La conexión entre el display y el Microcontrolador se realiza mediante el bus Serie.

En segundo lugar está el Nodo 2. En este nodo se encuentran situados los distintos sensores de que dispone el sistema, todos ellos se comunican con el Microcontrolador por medio del bus I2C. Todos los datos recogidos por los sensores son enviados al microcontrolador. Una vez que éste los ha recibido y procesado se envían por bus CAN a los otros nodos.

En tercer lugar se encuentra el Nodo 3. Es el nodo encargado de controlar la tarjeta de potencia de los motores. Éste nodo recibe toda la información proveniente de los demás nodos, con la cual manda las órdenes para realizar los distintos movimientos recogidos por las distintas interfaces Hombre – Máquina. La comunicación entre el Nodo 3 y la tarjeta de potencia se realiza mediante el bus Serie.

2.1.3. Buses de comunicación

Para la comunicación entre los distintos elementos del sistema se utilizan varios sistemas de comunicación. Los buses utilizados son los siguientes:

- Bus CAN
- Bus Serie
- Bus I2C

El bus CAN comunica los distintos nodos del sistema además de comunicar éstos con el PC a través del conversor USB – CAN por medio del bus CANS.

El bus I2C comunica los distintos sensores del sistema con el Nodo 2.

El bus Serie comunica el nodo del joystick con el display además de comunicar el nodo de la tarjeta de motores con la tarjeta de potencia para transmitir los distintos comandos a los motores.

2.1.3.1. Bus CAN

Can-Bus es un protocolo de comunicación en serie desarrollado por Bosch para el intercambio de información entre unidades de control electrónicas del automóvil. En la actualidad el uso de este tipo de bus esta cada vez mas presente en sectores como lo son la automatización industrial o el área de control. También es usado en arquitecturas de bus industrial en aplicaciones de Tiempo Real distribuidas. CAN es un protocolo abierto para uso industrial además de tener una alta seguridad.

El significado de las siglas CAN es Controller Area Network (Red de área de control) y el significado de Bus, en el campo de la informática, se entiende como un elemento que permite transportar una gran cantidad de información.

Este sistema permite compartir una gran cantidad de información entre las unidades de control abonadas al sistema, lo que provoca una reducción de la cantidad de cables que componen la instalación eléctrica.

En la Imagen 2. 5 se representa el esquema general de conexiones de un sistema distribuido empleando el bus CAN, se puede observar la existencia de tres nodos conectados a la misma línea de bus.

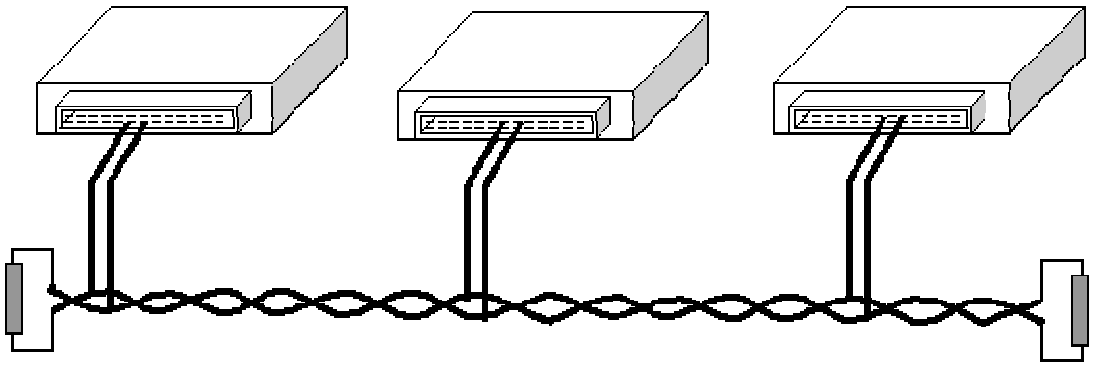


Imagen 2. 5. Bus CAN

La información que circula entre las unidades de mando a través de los dos cables (bus) son paquetes de 0 y 1 (bit) con una longitud limitada y con una estructura definida de campos que conforman el mensaje. Uno de esos campos actúa de identificador del tipo de dato que se transporta, de la unidad de mando que lo trasmite y de la prioridad para trasmitirlo respecto a otros. El mensaje no va direccionado a ninguna unidad de mando en concreto, cada una de ellas reconocerá mediante este identificador si el mensaje le interesa o no. Todas las unidades de mando pueden ser trasmisoras y receptoras, y la cantidad de las mismas abonadas al sistema puede ser variable.

La información transcurre por un par trenzado que une todas las unidades que forman el sistema. La Imagen 2. 6 muestra la forma que tienen los cables del bus. En un hilo del par los valores de tensión varían entre 0V y 2.25V, por lo que se denomina CAN L (Low) y en el otro, el CAN H (High) lo hacen entre 2.75V. y 5V. En caso de que se interrumpa la línea H o que se derive a masa, el sistema trabajará con la señal de Low con respecto a masa, en el caso de que se interrumpa la línea L, ocurrirá lo contrario. Esta situación permite que el sistema siga trabajando con uno de los cables cortados o comunicados a masa, incluso con ambos comunicados también sería posible el funcionamiento, quedando fuera de servicio solamente cuando ambos cables se cortan.

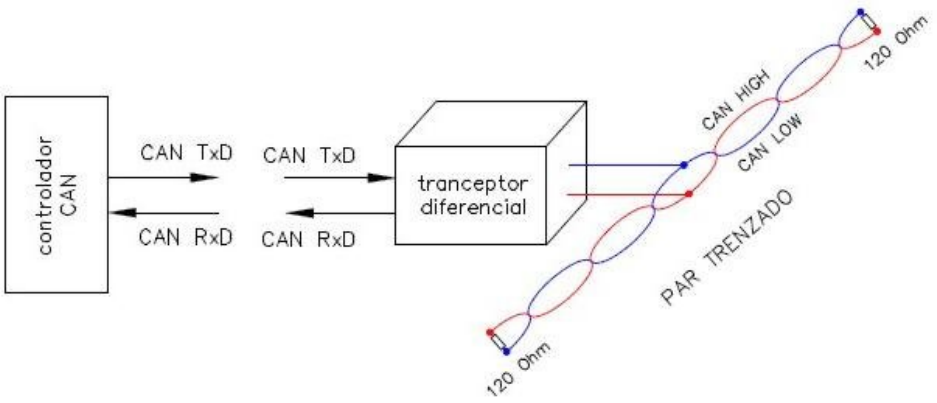


Imagen 2. 6. Esquema del bus CAN

La Imagen 2. 7 representa el esquema el formato que tienen los mensajes enviados a través del bus.

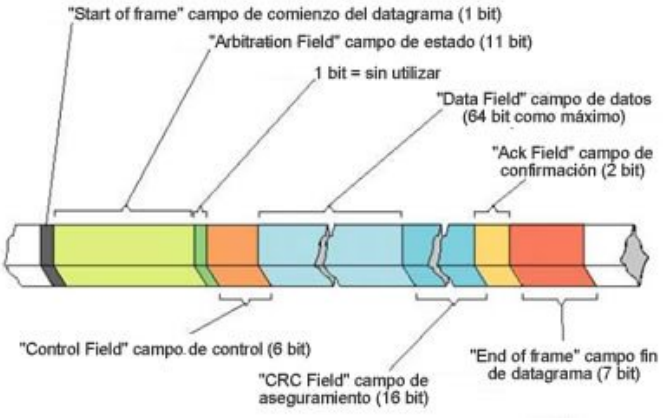


Imagen 2. 7. Estructura de un mensaje CAN

2.1.3.2. Bus Serie

Es la forma común de realizar transmisiones de datos entre fuentes serie asíncrono (por ejemplo entre PC – modem ó PC - PC). Está diseñado para distancias cortas, de hasta 15 metros según la norma, y para velocidades de comunicación bajas. La velocidad se mide en baudios (bits/segundo).

Los voltajes enviados por los pines pueden ser en 2 estados, encendido o apagado. Encendido (valor binario de 1) significa que el pin esta transmitiendo una señal entre -3 y -25 voltios, mientras que apagado (valor binario de 0) quiere decir que esta transmitiendo una señal entre +3 y +25 voltios.

El puerto serie es del tipo asíncrono, utiliza cableado simple desde 3 hilos hasta 25 y que conecta computadoras o microcontroladores a todo tipo de periféricos, como lo pueden ser las impresoras o los módems. El tipo de conector original era el DB – 25, pero el conector más común de encontrar es el DB – 9. La Tabla 2. 4 muestra la distribución de los pines para ambos conectores.

Conector 25 pines	Conector 9 pines	Nombre	Descripcion
1	1	-	Masa chasis
2	3	TxD	Transmit Data
3	2	RxD	Receive Data
4	7	RTS	Request to send
5	8	CTS	Clear to send
6	6	DSR	Data Set Ready
7	5	SG	Signal Ground
8	1	DCD	Data Carrier Detect
15	-	TxC	Transmit Clock
17	-	RxC	Receive Clock
20	4	DTR	Data Terminal Ready
22	9	RI	Ring Indicator
24	-	RTxC	Transmin/Receive Clock

Tabla 2. 4 Distribución de pines en el bus Serie

Los tipos de canal pueden ser simplex, half duplex o full duplex. En un canal simplex los datos solo viajarán en una dirección, en un canal half duplex, los datos pueden viajar en una u otra dirección, pero sólo durante un determinado periodo de tiempo, luego la línea debe ser conmutada antes que los datos puedan viajar en la otra dirección. En un canal full duplex, los datos pueden viajar en ambos sentidos simultáneamente. Las líneas de protocolo ó *handshaking* de la RS-232 se usan para resolver los problemas asociados con este modo de operación, tal como en qué dirección los datos deben viajar en un instante determinado.

Las UART o U(S)ART (*Transmisor y Receptor Síncrono Asíncrono Universal*) se diseñaron para convertir las señales que maneja la CPU y transmitirlas al exterior. Las UART deben resolver problemas tales como la conversión de voltajes internos del DCE (*Equipo de Comunicación de datos*) con respecto al DTE (*Equipo terminal de datos*), gobernar las señales de control, y realizar la transformación desde el bus de datos de señales en paralelo a serie y viceversa. Debe ser robusta y deberá tolerar circuitos abiertos, cortocircuitos y escritura simultánea sobre un mismo pin, entre otras consideraciones. Es en la UART en donde se implementa la interfaz.

2.1.3.3. Bus I2C

El bus I2C es un tipo de bus serie que utiliza sólo dos hilos trenzados y una masa común para la interconexión de los distintos periféricos. La velocidad máxima a la que se puede transmitir con este bus es de 100 kHz (actualmente se está implantando la frecuencia de 400 kHz). Con el mismo circuito (dos hilos) se puede llegar a controlar hasta 128 dispositivos. La principal utilidad de este tipo de bus es la de comunicación entre periféricos cuando la distancia entre ellos no es excesivamente elevada.

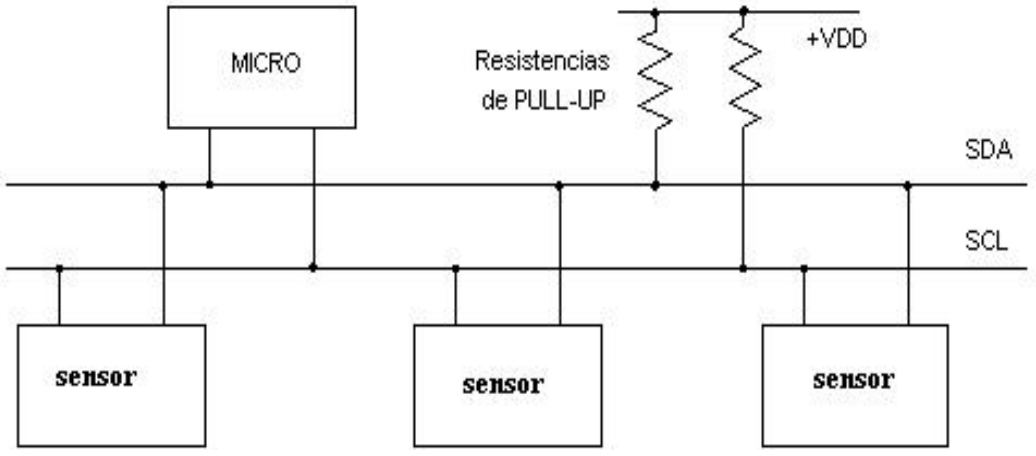


Imagen 2. 8. Esquema de conexión del bus I2C

La Imagen 2. 8 nos muestra el esquema de conexión de un bus I2C en el que se observa los distintos sensores conectados a un microcontrolador, además se observa que al principio del bus hay dos resistencias de pull-up, necesarias para realizar este tipo de comunicación.

El microcontrolador (maestro) es el encargado de iniciar y terminar la transferencia de información y es el que genera la señal de reloj (SCL), los sensores (esclavos) son los dispositivos direccionados por el maestro mediante tramas de 7 bits. La línea de datos (SDA) es utilizada tanto por el maestro como por el esclavo para la transmisión de información.

Cuando el maestro inicia una trama de comunicación, envía a través de la línea de datos la dirección del esclavo con el que se pretende establecer una comunicación. Todos los esclavos reciben dicha dirección, pero es uno sólo el que responderá y el resto permanece en espera de que se inicie una nueva trama.

La Imagen 2. 9 y la Imagen 2. 10 muestran el envío y recepción de mensajes a través de un bus I2C.

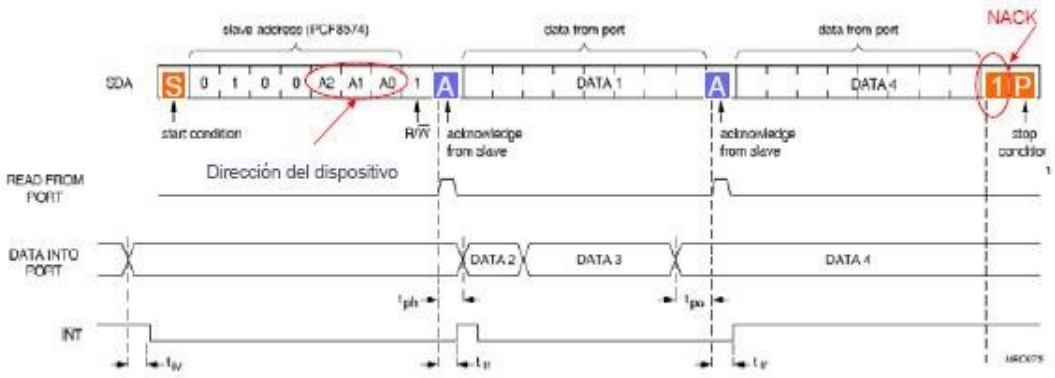


Imagen 2. 9. Lectura de un dato del bus I2C

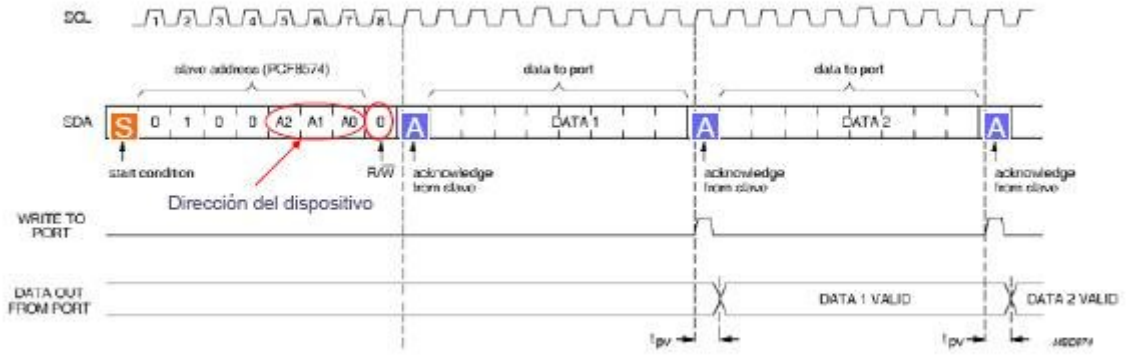


Imagen 2. 10. Escritura de un dato en el bus I2C

2.2. Portabilidad

La principal característica que posee el sistema diseñado, como se mencionó anteriormente, es la posibilidad de incorporarla a distintas aplicaciones.

En nuestro caso, la primera aplicación donde fue instalado el sistema fue el robot **Alca-Roby**, este robot fue diseñado y construido para ser un puesto de información en la celebración de ferias y congresos. Posteriormente el sistema fue acoplado a **SIAMO II**. La diferencia entre ambos radica en su estructura mecánica y funcional aunque la arquitectura ya que en las dos aplicaciones se trata de un robot móvil diferencial. Por lo tanto el paso de una aplicación a otra es relativamente sencillo.

Tanto el robot **Alca-Roby** como la silla **SIAMO II** ya habían sido objeto de proyectos de desarrollo anteriores en el Departamento de Electrónica. Sin embargo su arquitectura presentaba importantes problemas para proseguir con los trabajos sobre ellas. Como principal inconveniente se puede mencionar que ambos robots utilizaban la tecnología **LonWorks** con comunicación mediante bus **Lon**. El problema se encontraba cuando se procedía a programar los distintos módulos del sistema para lo cual era necesario el uso de su propio hardware. Con este nuevo sistema disponemos de grandes ventajas, por un lado la programación de cada uno de los módulos se puede realizar fácilmente desde cualquier ordenador convencional; por otra parte el bus Can esta más extendido entre diversos fabricantes que el bus LON.

En la Imagen 2. 11 se muestran los dos sistemas finales incorporados a **Alca-Roby** y a **SIAMO II**.

Imagen 2. 11. Alca-Roby y SIAMO II

2.3. Interfaz Hombre – Máquina

Una interfaz Hombre - Máquina es una serie de dispositivos y procedimientos con el que se permite al hombre poder interactuar con la máquina para realizar distintas operaciones dependiendo de los fines para los que este destinada ésta.

En este caso, **SIAMO II** dispone de tres tipos de interfaces, estas tienen como objetivo facilitar y proporcionar una vida más confortable a las personas con movilidad reducida. Las interfaces instaladas en nuestro sistema son un joystick manual, un sistema de comando de soplo y otro joystick para ser comandado por movimientos de la cabeza.

Junto con el joystick manual se han incorporado distintos elementos para realizar distintas funciones dentro del sistema, éstos se encuentran situados en el mismo compartimento del joystick manual. Los elementos incorporados más relevantes son un display, un teclado y un potenciómetro. La principal función de los dos primeros es la de poder cambiar de modo de funcionamiento, también nos ofrece el nivel de carga de las baterías. Por otra parte el usuario puede regular la velocidad con uso del potenciómetro.

En el mismo emplazamiento se encuentran un pulsador de color negro cuya función es la de activar el claxon y un interruptor de color rojo acompañado de una lámpara que sirven para encender o apagar el sistema y para indicar el estado del sistema respectivamente.

El sistema de soplido es una interfaz capaz de mover el sistema a través de soplos o aspiraciones de aire por parte del usuario. Este se encuentra conectado directamente a la misma tarjeta que controla los movimientos ejecutados a través del joystick manual. Se trata de una interfaz de fácil manejo y que proporciona buenos resultados. La Imagen 2. 12 muestra los distintos elementos del joystick manual y del sistema de comando por soplo.

Por ultimo la Imagen 2. 13 muestra el joystick de cabeza. Esta interfaz está conectada directamente al PC, los movimientos del sistema se realizan a través de giros de la cabeza que el usuario efectúa, el manejo del mismo se asemeja al de los ratones de los ordenadores convencionales. En la imagen se observan los elementos que componen esta interfaz.

Imagen 2. 12 Joystick manual y sistema de soplo

Imagen 2. 13 Joystick de cabeza

CAPÍTULO 3

DESCRIPCIÓN DE LOS NODOS DEL SISTEMA

Como se detallaba en el capítulo de introducción en este capítulo se pasa a proceder a explicar los distintos algoritmos desarrollados para la realización de cada nodo del sistema y de las distintas herramientas utilizadas para la elaboración de los mismos.

Los elementos que se han incorporado han sido:

- Nodo con Joystick manual y sistema de soplo, dotado además de un teclado y un display
- Nodo del Joystick de cabeza
- Nodo de la tarjeta de motores
- Nodo de la tarjeta de sensores

En este TFC se van a detallar los nodos concernientes al joystick manual y al control de los motores. Los nodos restantes serán descritos en el TFC elaborado por José Basterrechea.

3.1 Nodo del joystick

Este nodo es el nodo principal del sistema ya que es el encargado de seleccionar el modo de funcionamiento del sistema y dependiendo del modo de funcionamiento los nodos tienen distintas funciones.

3.1.1 Arquitectura física

En la figura 3.1 se muestra un esquema de la estructura hardware del nodo del joystick.

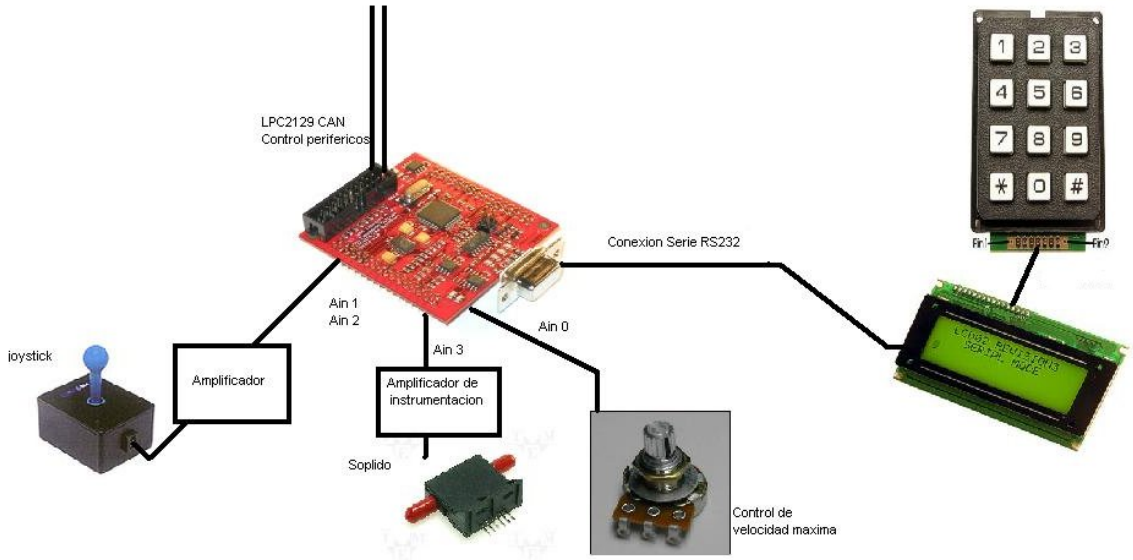


Figura 3. 1 Esquemático hardware

Como podemos observar en la figura anterior este nodo utiliza varios recursos del microcontrolador LPC2129 los cuales detallamos:

El joystick está conectado a la tarjeta del micro mediante las entradas analógicas AIN1 y AIN2 del mismo a través de un amplificador de acondicionamiento.

El sensor de soplido está conectado a la tarjeta mediante la entrada analógica AIN3 a través de un amplificador de instrumentación.

El potenciómetro de control de velocidad está a través de la entrada analógica AIN0 de la tarjeta del micro.

El potenciómetro se encarga de regular la entrada entre 3,3V y 0V.

El teclado matricial está conectado directamente al display LCD, el cual está conectado a la tarjeta del microcontrolador mediante el puerto serie, configurado a 9600bps, sin paridad y un bit de parada.

La interconexión entre los distintos nodos se realiza por medio del bus de datos CANS.

3.1.1.1 Arquitectura física del joystick

El joystick utilizado, es un joystick convencional de cinco pines, dos de ellos se utilizan para alimentarlo, otro de los pines llamado referencia nos indica el valor que tiene en reposo y los otros dos se utilizan para obtener la medida procedente del mismo.

En la figura 3.2 observamos el circuito de acondicionamiento mediante el cual las señales procedentes del joystick son amplificadas para su posterior conversión.

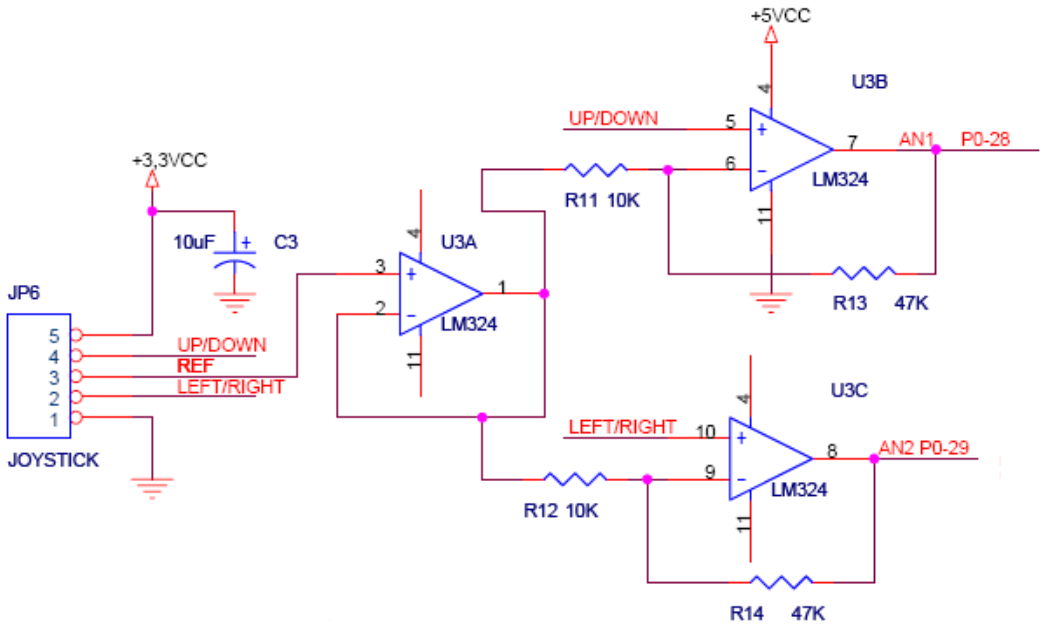


Figura 3. 2 Amplificador joystick

3.1.1.2 Arquitectura física del soplido

El sensor de soplido utilizado es de la casa HONEYWELL y alguna de sus características se adjuntan en la tabla 1.

Características del sensor	Valor
Tiempo de respuesta	3mseg
Consumo de potencia	30mW
Resistencia de los sensores	5KΩ
Rango de temperaturas de funcionamiento	-25ºC a 85ºC
Presión máxima soportada	25psi
Repetividad e histéresis máxima	0,35% de la media
Deriva en función de la temperatura	+22% de la lectura a -25ºC -22% de la lectura a 85ºC
Voltaje de offset a 25ºC	± 1mV(5 cuentas con 8bits)
Sensibilidad	10mV/(pulgada de agua)
Tolerancia de presión	Positiva 3,5mV 5,5%F.S. Negativa 7mV 11%F.S

Tabla 1.-Características del sensor

En la figura 3.3 se muestra una imagen del aspecto externo del sensor

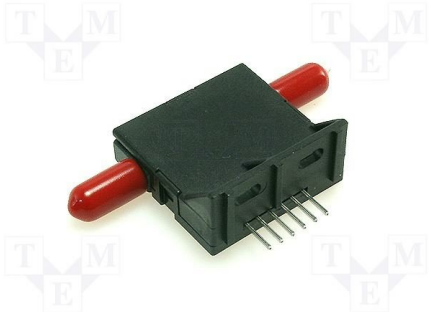


Figura 3. 3 Sensor AWM2000V

El circuito acondicionador del sensor se puede ver en la figura 3.4, ha sido extraído de las hojas características del fabricante. Se trata de una configuración como amplificador de instrumentación clásica, adaptado a una fuente simple de alimentación.

Se utiliza el amplificador operacional 124 por sus mejores prestaciones con alimentación simple.

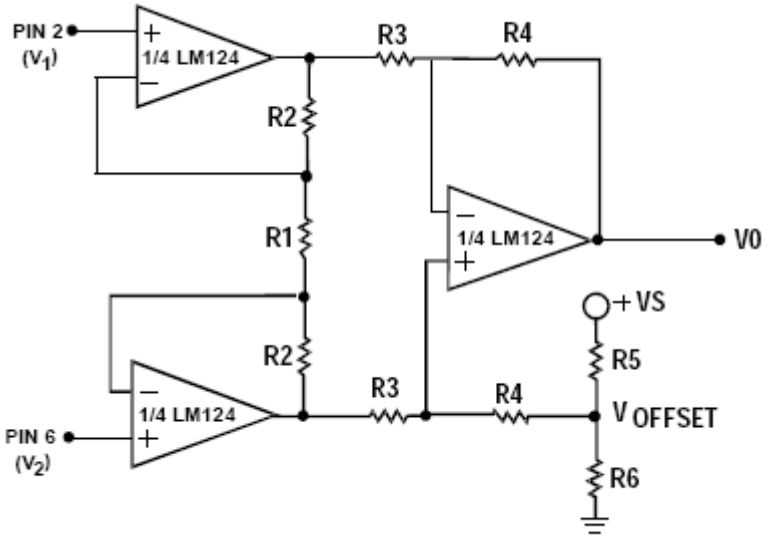


Figura 3. 4 Amplificador de instrumentación del soplido

3.1.1.3 Arquitectura física del potenciómetro

El potenciómetro que utilizamos para regular la velocidad máxima es un potenciómetro lineal que regula la tensión de la entrada AIN0 entre 3,3 V y 0 V.

En la figura 3.5 observamos su conexión.

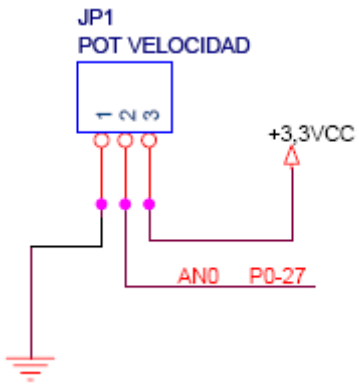


Figura 3. 4 Conexión potenciómetro

3.1.1.4 Arquitectura física del display

El display utilizado en el joystick es el "DISPLAY LCD SERIE + I2C 4 X 20 LCD03 S310118" de la casa *superrobotica*, este display puede ser conectado tanto en modo conexión serie como en modo conexión I2C. Este display a su vez lo tenemos conectado directamente al teclado "TECLADO MATRICIAL 4 X 3 TECLAS S310119" de la casa *superrobotica,* pudiendo leer su estado mediante el display.

La conexión que hemos seleccionado para nuestro display es mediante el puerto serie y se detalla en la figura 3.6.

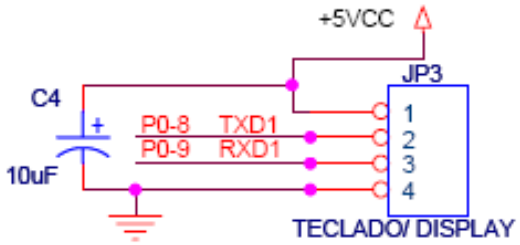


Figura 3. 5 display serie

3.1.2 Arquitectura funcional

En el siguiente diagrama, figura 3.7 se describe el funcionamiento general del modo de funcionamiento del sistema.

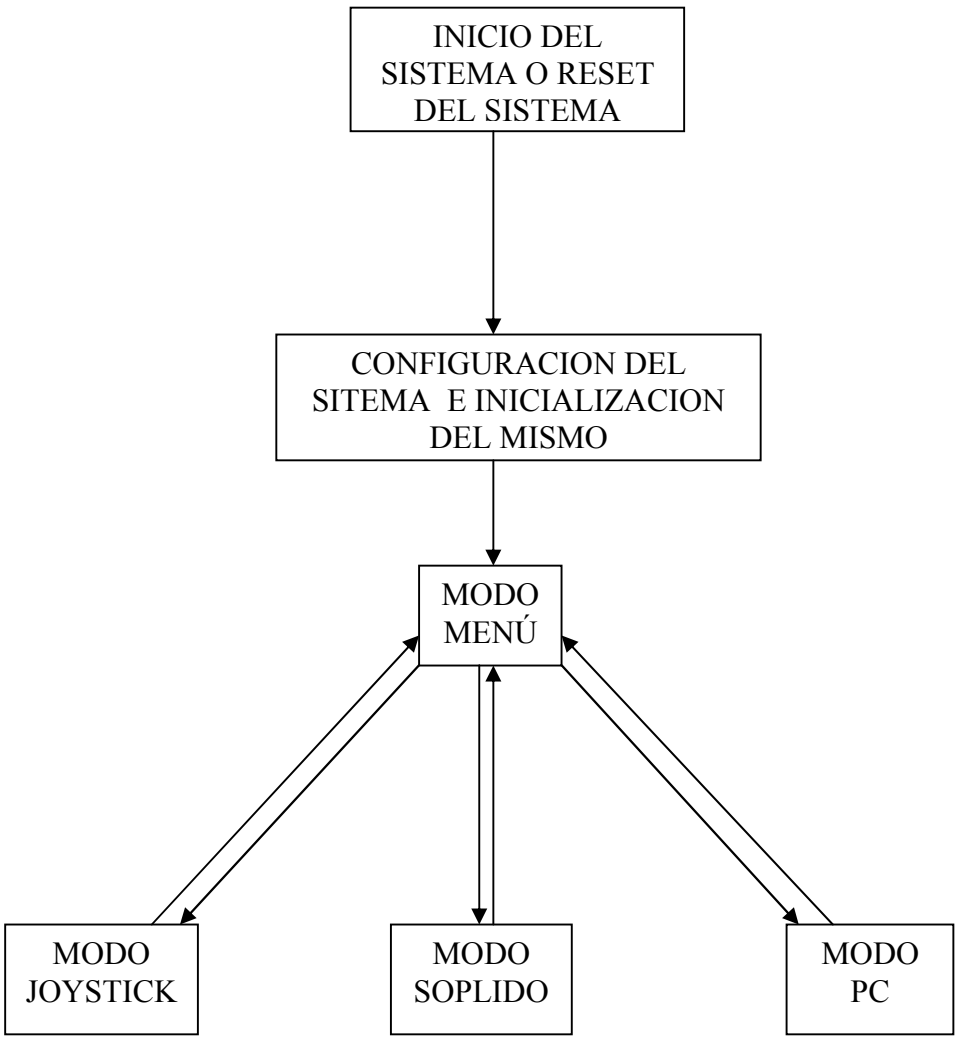


Figura 3.7 diagrama de modo de funcionamiento

Como podemos observar en la figura 3.7 al encender o resetear nuestra silla, entraremos en el modo menú en el que podremos seleccionar el modo de funcionamiento deseado. Para cambiar el modo de funcionamiento, es obligatorio pasar por el modo menú de nuevo. El modo de funcionamiento seleccionado será enviado a través del bus CANS con la etiqueta 0x111 cada 500mseg para que el resto de nodos sepan el estado de funcionamiento actual. El formato del mensaje se muestra en la tabla2.

ETIQUETA	DATO B	DATO A	
0X110			MODO

Tabla 2.-Mensaje CAN modo

El dato MODO es un dato de 8 bits.

Asociados a todos los modos de funcionamiento tenemos la función **display** que es una tarea cíclica que se ejecuta cada 150 mseg y se encarga de llenar un buffer con los datos a representar en el display, ya sea el modo de funcionamiento o el estado de la batería. Otras funciones asociadas a cualquier modo de funcionamiento serian las funciones **can** y **txserie** ejecutadas cada 10mseg.

La función **can,** se encarga de comprobar si ha llegado algún mensaje que nos interese para el funcionamiento de nuestro nodo.

La función **txserie** comprueba si hay algún si el buffer de datos para representar en el display está vacío y en caso contrario manda por el puerto serie un carácter.

3.1.2.1 Arquitectura funcional del joystick

Las tareas asociadas al modo joystick son tareas cíclicas que se ejecutan cada 100 mseg y en la figura 3.8 se detallan.

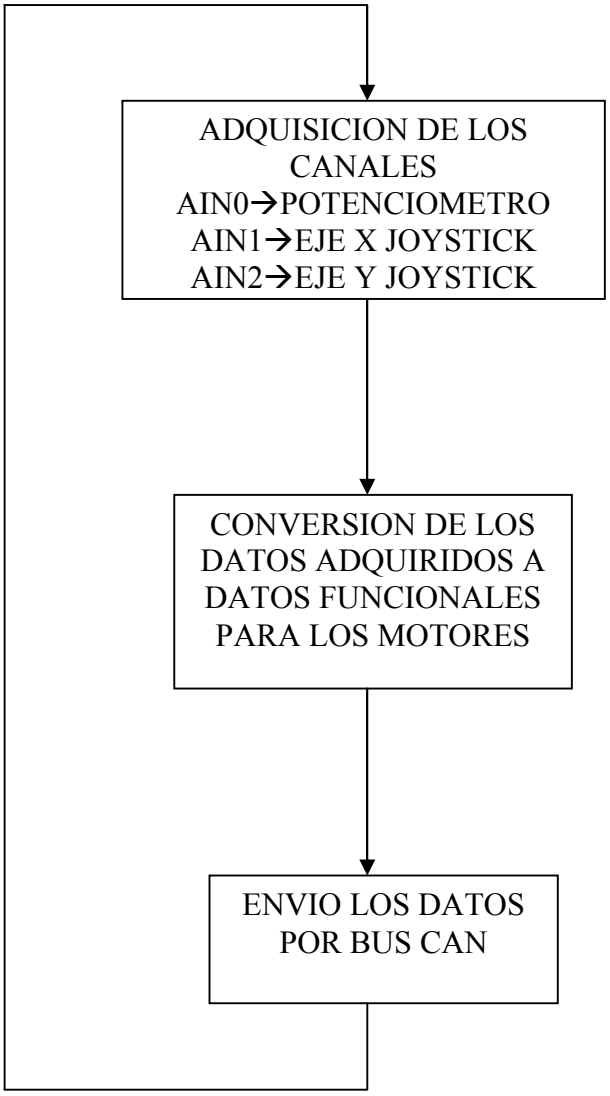


Figura 3.8 funcionamiento del modo joystick

En la tarea de adquisición de los canales, adquirimos los valores de los ejes del joystick manual mediante los cuales conseguiremos unos datos validos de velocidad respecto a la velocidad máxima regulada por el potenciómetro.

En la tarea de conversión, mediante el modelo matemático de la figura 3.9 implementaremos el control de movimiento de la silla.

Este modelo va a ser necesario no solo para analizar el control de posición que se diseña sino también para poder obtener la posición real del móvil.

Figura 3.9 Modelo matemático de la silla

Las entradas del modelo son la velocidad de avance (V) y la velocidad de giro de la rueda(Ω) de la silla.

Las salidas (x,y) informa de las coordenadas cartesianas del centro de movimiento de la silla, en un sistema definido en el espacio en el que se desenvuelve la silla.

La salida (θ) informa del ángulo formado por el eje de movimiento de la silla y el semieje positivo de abscisas del sistema de coordenadas elegido, en sentido anti horario.

A partir del modelo matemático para conseguir las velocidades angulares de cada una de las ruedas motrices (ωd rueda derecha y ωi rueda izquierda) utilizamos las siguientes ecuaciones:

$$\omega d = (1/RADIO)(V+(R*D)/2).$$

$$\omega i = (1/RADIO)(V-(R*D)/2).$$

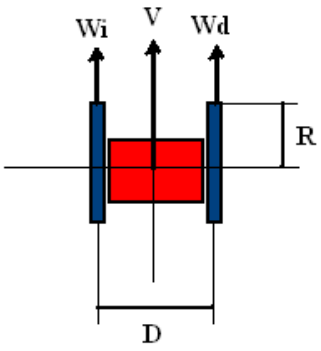


Figura 3.10 Diagrama de la silla de ruedas

ωd→ Dato a enviar al motor derecho.

ωi→ Dato a enviar al motor izquierdo.

RADIO→ Radio de la rueda de nuestra silla de ruedas.

D→ Distancia del eje de nuestra silla de ruedas.

V→ Dato adquirido del ejeX en función de su velocidad máxima.

R→ Dato adquirido del ejeY en función de su velocidad máxima.

En la tarea de enviar datos por el bus can, mediante la etiqueta 0x110 enviamos los datos adquiridos y los datos convertidos de los motores. En la tabla 3 se muestra la estructura del mensaje a enviar.

ETIQUETA	DATO B		DATO A		
0X110	VALOR POT	VALOR EJEY	VALOR EJEX	WI	WD

Tabla 3.-Mensaje CAN motores

El valor del potenciómetro *(valor pot)*, del ejey *(valor ejey)* y del ejex *(valor ejex)* son datos de 16 bits mientras que, los valores de ωd y ωi *(ωd,ωi)* son datos de 8 bits.

3.1.2.2 Arquitectura funcional del soplido

Las tareas asociadas al modo soplido son tareas cíclicas que se ejecutan cada 50 mseg y en la figura 3.11 se detallan.

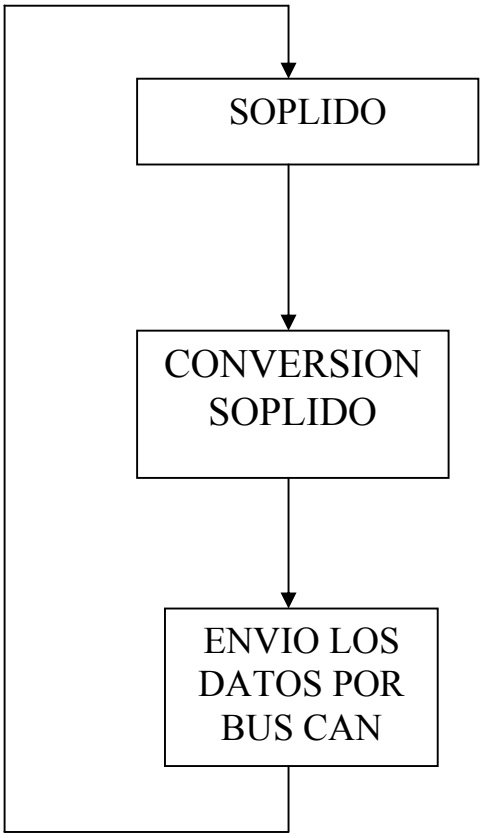


Figura 3.11 funcionamiento del modo soplido

Las funciones *soplido()* y *conversión soplido()* están basadas en dos máquinas de estados. La función *soplido* está basada en la máquina de estados reflejada en la figura 3.12 y detallada a continuación.

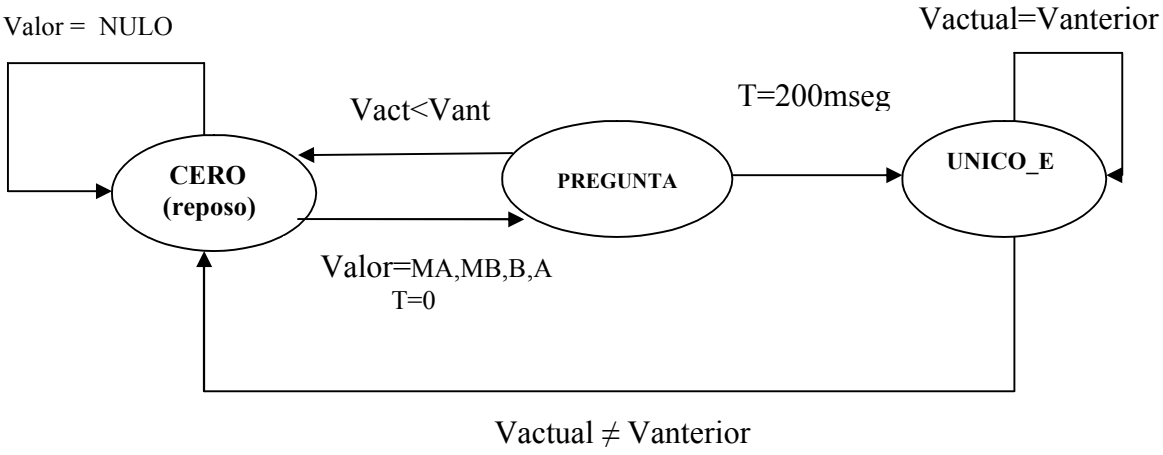


Figura 3.12 máquina de estados de la función soplido, encarga de validar el nivel de soplo detectado

Antes de comentar la máquina de estados es conveniente declarar los diferentes niveles de soplo. Estos son los siguientes:

MA→ Soplido fuerte.

MB→ Aspiración fuerte.

A→ Soplido flojo

B→ Aspiración floja

NULO→ No hay ni aspiración ni soplido.

En esta máquina de estados podemos apreciar tres estados denominados CERO, PREGUNTA y UNICO_E. En el estado CERO nos mantendremos hasta que nos llegue un dato de soplo correcto es decir, MA, MB, B o A. Una vez nos llegue un dato concreto pasaremos al estado PREGUNTA en el que estaremos 200 mseg comprobando si el dato sigue siendo el mismo. En el caso de que el dato sea inferior al anterior volveremos al estado CERO y estaremos a la espera de otro nuevo soplido. Una vez trascurridos los 200 mseg si es dato no ha variado pasaremos al estado UNICO_E en el que tendremos un dato valido hasta que el valor de soplido varié.

El valor valido de soplido únicamente se obtendrá estando en el estado UNICO_E, en el resto de estados obtendremos un valor NULO.

En el estado PREGUNTA podemos pasar de tener un valor A a un valor MA pero no al revés, y lo mismo ocurriría con B y MB.

El comando resultante tras validar la amplitud del soplo, es el comando utilizado en la función *conversión soplido* figura 3.13 para conseguir el comando final de velocidad.

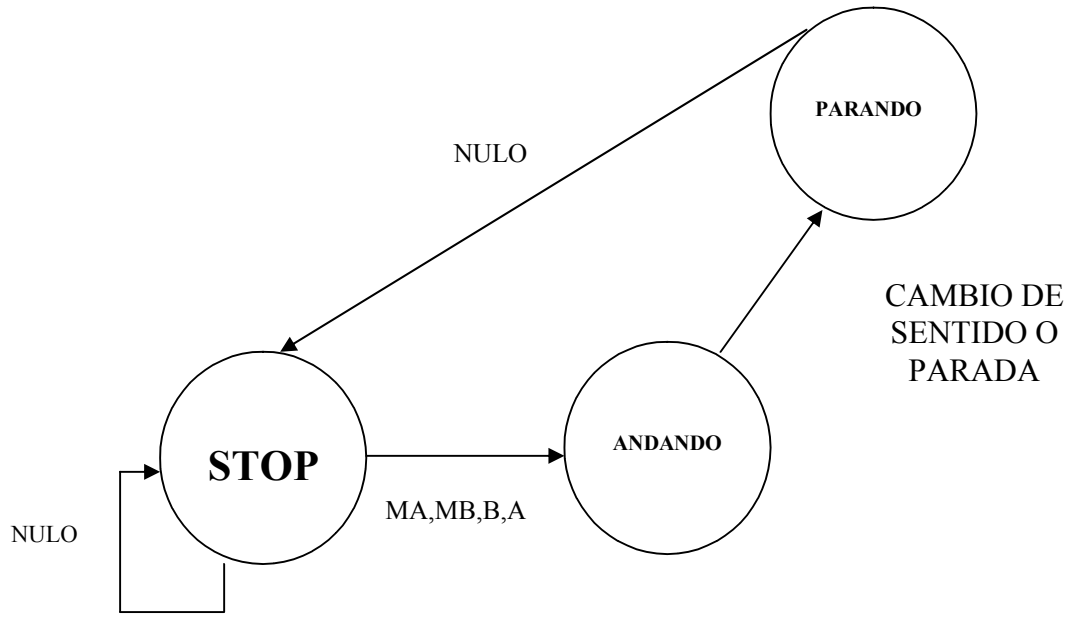


Figura 3.13 máquina de estados para la conversión del soplido, encargada de elaborar el comando final de velocidad.

En esta máquina de estados tenemos tres estados STOP, ANDANDO y PARANDO, en el estado STOP nuestra silla se encontrara parada y no ejecutara ninguna maniobra hasta que no obtengamos ningún dato valido en la función soplido. Una vez nos obtengamos algún dato valido, pasaremos al estado ANDANDO y la silla ejecutara la maniobra correspondiente al dato obtenido que pueden ser las siguientes:

1.- En el caso de que el dato obtenido sea MA, la silla aumentara su velocidad lineal. En el caso en el que su velocidad fuera negativa y tuviéramos que cambiar el sentido pasaríamos al estado PARANDO, es decir para cambiar el sentido de nuestra silla es obligatorio parar la silla.

2.- En el caso de que el dato obtenido sea MB, la silla disminuiría su velocidad lineal. En el caso en el que su velocidad fuera positiva y tuviésemos que cambiar el sentido pasaríamos al estado PARANDO.

3.-En el caso de que el dato obtenido sea A, la silla giraría hacia la derecha, está acción es momentánea y en el momento que dejáramos de soplar, la silla dejaría de girar.

4.-En el caso de que el dato obtenido sea B, la silla giraría hacia la izquierda, está acción es momentánea y en el momento que dejáramos de inspirar, la silla dejaría de girar.

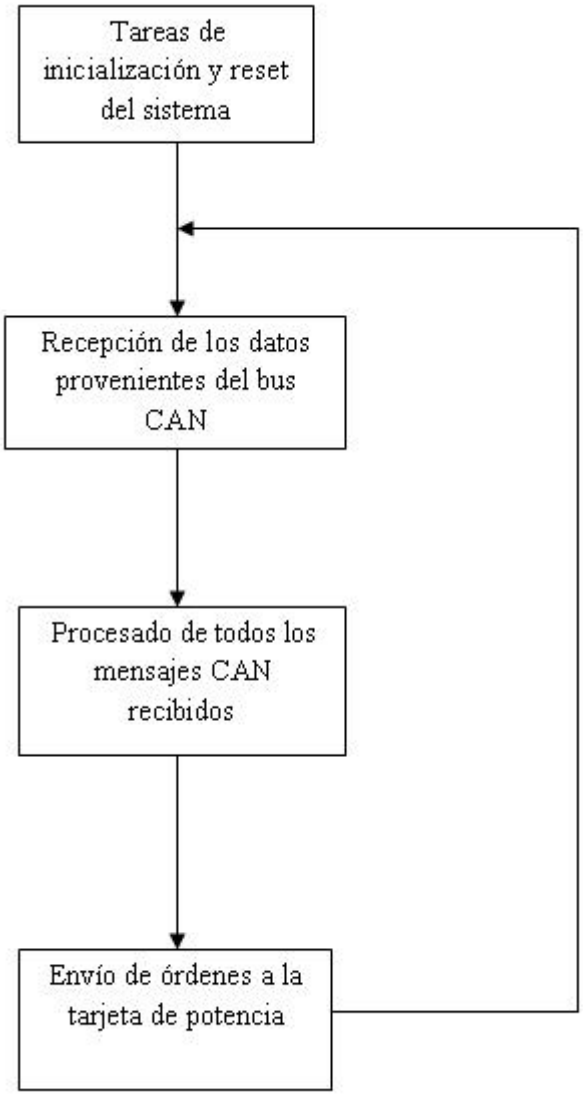
5.-En el caso de que el dato obtenido sea NULO, la silla no sufre ninguna modificación en su velocidad.

Estando en el estado ANDANDO si al modificar la velocidad de la silla llegáramos a las condiciones de parar la silla o cambiar el sentido de la misma, pasaríamos al estado PARANDO del que solo saldríamos con un dato NULO, es decir, dejando de ejercer ninguna acción sobre el sensor de soplo. Al obtener el dato NULO pasaríamos al estado STOP y estaríamos en el estado inicial de nuevo.

3.2 Nodo de la tarjeta de motores

El nodo de la tarjeta de motores es el encargado de mandar las órdenes de movimiento a los motores. Este recibe los comandos de movimiento procedentes del resto de los nodos a través del bus CANS y envía los comandos de movimiento a la tarjeta de potencia por medio del puerto serie.

3.2.1. Diagrama de flujo de funcionamiento



3.2.2. Arquitectura física

El nodo de la tarjeta de motores esta formado por la tarjeta del microcontrolador además del sistema de alimentación que se alimenta directamente de las baterías y que sirve como fuente de alimentación del resto de los nodos a través del bus CANS.

El aspecto físico que presenta el este nodo se representa en la imagen siguiente.

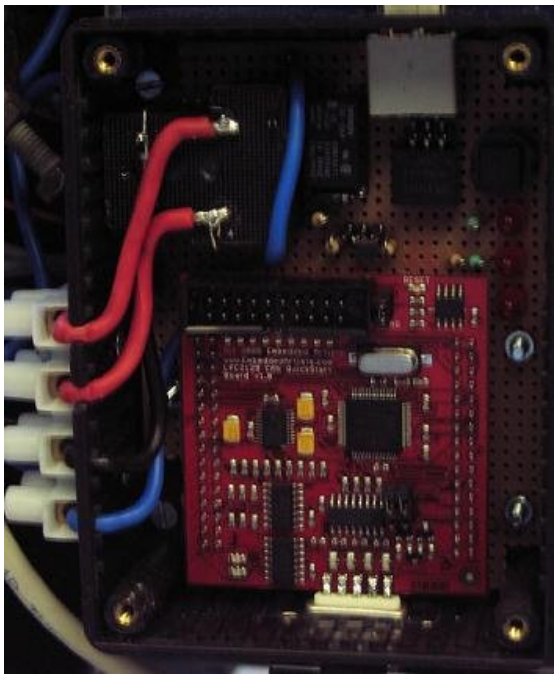


Figura 3. 14. Nodo de la tarjeta de motores

En la parte inferior de la imagen se puede observar la tarjeta del microcontrolador y en la parte superior se encuentra el sistema de alimentación del nodo.

Al sistema de alimentación le llegan los 24V procedentes de las baterías, estos son distribuidos por el bus CANS al resto de los nodos. La alimentación de la tarjeta del microcontrolador es de 5V, para ello se dispone de una fuente conmutada que trasforma los 24V en 5V. Estas fuentes se encuentran situadas en todos los nodos. Una vez que todos los nodos poseen su correspondiente alimentación el sistema queda a la espera de que sea activado, para ello es necesaria una señal de START que se produce cuando en el nodo del joystick se pulsa la tecla de encendido, tras activar el sistema se envía por el bus CANS la correspondiente señal de START lo que provoca en este nodo es que se active una señal de PWRCTRL que provoca la activación de la tarjeta de potencia, además se activa la señal de ON que provoca el encendido del microcontrolador. A continuación se muestra la figura 3.15 con el esquema eléctrico de la alimentación.

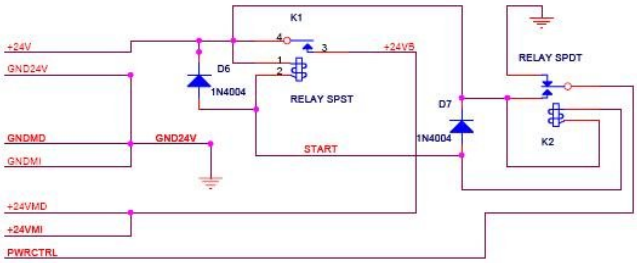


Figura 3. 15. Alimentación del nodo

3.2.3. Arquitectura funcional

La función de este nodo es la de ser un paso intermedio en el que el resto de los nodos le mandan ordenes mediante mensajes por el bus CANS y este tiene que procesar esos mensajes. Una vez procesados esos mensajes éste le manda las ordenes necesarias a la tarjeta de potencia que es la encarga de controlar los motores.

3.2.3.1. Mensajes CAN

Los mensajes recibidos por el bus CAN tienen una estructura determinada, se tratan de mensajes que tienen una dimensión de 8 bytes más un identificador del mensaje. Este identificador sirve para distinguir unos mensajes de otros, cuando el identificador del mensaje coindice con el mensaje que se espera éste es procesado. Los identificadores son un dato importante dentro de los mensajes ya que por el bus circulan varios mensajes simultáneamente.

En el caso de la tarjeta de los motores los mensajes que relevantes son los que vienen procedentes de de todos los nodos además de los procedentes del PC. En la Tabla 3.2 se detallan estos.

IDENTIFICADOR	DATO A		DATO B		
0x110 (Joys)	Potenciómetro	Eje X joystick	Eje Y joystick	Wd	Wi
0x111 (Joys)					Modo
0x120 (PC)				Wd	Wi
0x201 (Sen)	SLDD	SLIT	SDI	STD	
0x202 (Sen)	SLDT	STI	SLID	SDD	
0x203 (Sen)	Acel. X	Acel. Y	Acel. Z	SJOYS	

Tabla 3. 2. Mensajes CAN atendidos

La comprobación de si hay algún mensaje circulando en el bus es una tarea cíclica que tiene un periodo de 10 milisegundos.

Este nodo además de recibir mensajes por el bus también envía mensajes. Los datos que se envían son los referentes al nivel de carga de las baterías y al de las lecturas realizadas por los encoders. Estos datos son enviados al nodo del joystick para que visualice en el display el estado en el que se encuentran las baterías y el las lecturas de los encoders son enviadas al PC. La estructura de estos mensajes se muestra en la Tabla 3.3.

IDENTIFICADOR	DATO A	DATO B
0x210	Tensión del controlador	Tensión de la batería
0x101	Tiempo Motor D	Encoder ABS Derecho
0x102	Tiempo Motor I	Encoder ABS Izquierdo

Tabla 3. 3. Mensajes CAN enviados por el nodo de motores

El mensaje de la lectura del estado de las baterías se envía con un periodo de 2 segundos ya que se trata de un dato de poca importancia, por otra parte el los mensajes que contienen los datos de las lecturas de los encoders se envían con un periodo de 100 milisegundos.

3.2.3.2. Desarrollo software

El método de programación seguido es el de un ciclo continuo *while(1)* con reparto de turnos. Para ello se dispone de un timer de 100 microsegundos que cada fin de cuenta incrementa los contadores de las distintas tareas que se deben realizar. El periodo de tiempo de cada tarea es el que se describió en el apartado 3.2.3.1 de este capítulo. Cuando el contador de cada tarea llega a su fin de cuenta se pone a nivel alto un flag, se ejecuta la tarea y se desactiva el flag y se vuelve al bucle principal. La figura 3.16 muestra el esquema de la aplicación.

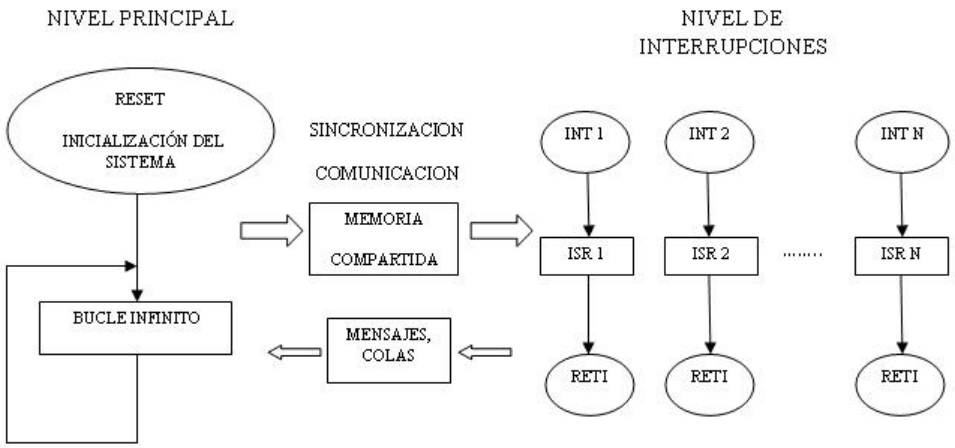


Imagen 3. 16. Planificación

En la Tabla 3.4 se muestran las funciones principales para la elaboración de este nodo.

NOMBRE	**FUNCIÓN**
init_serial()	Inicializar el puerto serie
config()	Configuración del sistema
sensores()	Procesar los datos recibidos de los sensores
aceleracion()	Realizar una aceleración software del sistema
acelerometro()	Procesar los datos del acelerometro
miputchar()	Enviar los comandos a la tarjeta de potencia
lee_encoderA()	Leer los datos del encoder A
envio_canA()	Enviar datos del encoder A
lee_encoderB()	Leer los datos del encoder B
envio_canB()	Enviar los datos del encoder A
lee_battery()	Leer el nivel de carga de la batería
envio_battery()	Enviar el nivel de batería

Tabla 3. 4. Funciones del nodo de motores

Para el desarrollo de este nodo la primera tarea que se lleva a cabo es la del reset e inicialización del sistema, aquí es donde se llevan a cabo todas las configuraciones necesarias para un correcto funcionamiento de este. Para ello en primer lugar se procede a la inicialización del puerto serie a través de la función *init_serial()* que se encuentra en el archivo "serial.h", posteriormente se realiza la configuración del sistema con la función *config(),* aquí se realizan todas las inicializaciones de los timers empleados así como la inicialización del bus CAN y de los filtros para los mensajes que va a recibir. Esta función se localiza en el archivo "setup.h". Las funciones empleadas, tanto como para la configuración del bus can como para el envío y recepción de mensajes se hallan en el archivo "LPC_FullCAN_SW.h".

Para el manejo de los distintos registros del microcontrolador se utiliza el archivo "driver.h", la función que tiene este fichero es la de hacer el sistema portable a otros

micros ya que es aquí donde se manejan los registros de este a través de funciones creadas para el manejo de estos.

Una vez realizada esta tarea el sistema queda a la espera de la recepción de los datos para procesarlos y enviárselos a la tarjeta de los motores. Los datos procesados serán los procedentes de los nodos de los sensores, del joystick manual y del sistema de soplido ya que los datos provenientes del PC han sido ya procesados previamente y son enviados directamente. Para el procesamiento de los datos se han realizado distintos algoritmos.

El primer algoritmo es el encargado de procesar los datos provenientes de los sensores de ultrasonidos, la función encargada de ello es *sensores()*. El algoritmo desarrollado plantea una comparación de las distancias recibidas con unas distancias fijadas, estas distancias son de 300 cm, 200 cm, 100 cm, 40 cm y de 5cm Si la distancia recibida es mayor de los 300 centímetros la silla funcionará con normalidad y los comandos mandados a la tarjeta de los motores son los recibidos de las diferentes interfaces. Si la distancia recibida es igual o menor a los 300 centímetros se procede asignar unos comandos de velocidad prefijados dependiendo de los márgenes en los que se encuentre hasta llegar a parar los motores si la distancia medida oscila entre los 40 y los 5 centímetros.

El siguiente algoritmo es el concerniente al sistema de aceleración por software, para ello se dispone la función *aceleracion()*, la función que tiene esta es la de hacer que la velocidad de avance o de giro se incremente o decremente paulatinamente dependiendo de lo que se desee hacer, esto hace que los movimientos realizados no sean bruscos y que el manejo sea más confortable.

El desarrollo de este algoritmo es sencillo, para su elaboración se parte de los datos que provienen del joystick o del sistema de soplido. Partiendo de esos datos, estos se van incrementando o decrementando hasta llegar al valor que las interfaces han enviado.

Posteriormente se realiza la llamada a la función *acelerometro()* encargada de leer los datos provenientes del acelerómetro. La misión de esta es la de gestionar los datos provenientes del acelerómetro. Este es utilizado para detectar los choques del sistema. En el algoritmo implementado se realiza una comparación de éstos datos con unos valores fijos. Si los valores recibidos referentes al eje X o al eje Y son mayores de 700 o menores de 400 provocara un paro instantáneo del sistema. Estas funciones se localizan en el archivo “funciones.h”

Una vez que se han procesado los datos por los distintos algoritmos estos son enviados por el puerto serie. Los comandos enviados son los datos de velocidad reales llamados RealD y RealI. Si los dos son cero se activará el freno instantáneamente. La función empleada para ello tiene el nombre de *miputchar()* ubicada en el archivo “serial.h”, con la que se envían caracteres hexadecimales por el puerto serie a la tarjeta de potencia.

Si el modo de funcionamiento es a través del joystick de cabeza la única función que se realiza aquí es la de activar o desactivar el freno, para ello se utiliza el mismo método del sistema de soplido o del joystick manual, si los valores recibidos son '0' el freno se activará instantáneamente, si no este desactivará hasta que los valores recibidos sean nulos.

Si estando funcionando el sistema con normalidad el usuario presiona la tecla de volver al menú principal el sistema se detendrá y el freno será activado.

Además del envío de datos a los motores también se leen los datos procedentes de los encoders se envían al PC para un procesamiento posterior.

Las funciones encargadas de realizar esto son *lee_encoderA()* y *envio_canA()* para leer y enviar los datos del encoder A y *lee_encoderB()* y *envio_canB()* para realizar las mismas operaciones con el encoder B.

Para la lectura y envío del nivel de carga de la batería se usan las funciones *lee_battery()* y *envio_battery()* respectivamente.

CAPÍTULO 4

RESULTADOS

En este capítulo se van a describir las pruebas realizadas al sistema desarrollado, así como los resultados obtenidos de ellas. Las pruebas se han desarrollado en el interior de la Escuela Politécnica. En el CD adjunto se muestran unos videos de las distintas pruebas que se han realizado con **SIAMO II**.

4.1. Pruebas del movimiento con el joystick de mano

Para la realización de este movimiento se ha procedido a mover a **SIAMO II** por distintas partes del edificio de la Escuela Politécnica. El comportamiento ofrecido por el sistema, tanto en espacios reducidos como en espacios más amplios, se puede calificar como un comportamiento bastante bueno. Solo cabe destacar que al realizar movimientos en espacios reducidos y debido al uso de sensores de ultrasonidos para evitar los choques, estos producen unos ecos que provocan que el sistema tenga alguna lectura errónea. Esto se produce porque cuando el sensor realiza una medida el haz que manda no es totalmente perpendicular al objeto o pared a medir, por lo tanto el rebote de la onda no es sale con un cierto ángulo y provoca que en ocasiones ese rebote lo reciba otro sensor.

Para finalizar con las pruebas realizadas con el joystick de mano, también cabe destacar que el trato que recibe el usuario es un trato confortable. En la Imagen 4. 1 se observa el funcionamiento del joystick manual.

Imagen 4. 1. Movimiento con joystick manual

4.2. Pruebas del movimiento con el sistema de soplido

La realización de estas pruebas se ha llevado a cabo también en los pasillos y las diferentes plantas del edificio de la Escuela Politécnica. El resultado que se ha obtenido de ellas es muy bueno.

Primeramente, y por tratarse de una interfaz poco común en el mundo de las sillas de ruedas, las pruebas se realizaron en trayectorias rectas, tras pasar estas de forma satisfactoria se procedió ha realizar giros sobre alrededor de columnas y ascensores dando los mismos resultados. Por último se procedió ha realizar pruebas para entrar y salir de los ascensores, esta una de las pruebas más complicadas ya que el usuario ha de manejar correctamente el sistema debido a que va a moverse en un entorno de dimensiones muy reducidas. Los resultados obtenidos tras esta última prueba fueron buenos. La imagen siguiente os muestra el funcionamiento del sistema.

Imagen 4. 2. Movimiento con el sistema de soplido

4.3. Pruebas del movimiento con el joystick de cabeza

Las pruebas realizadas para comprobar el comportamiento de esta interfaz se llevaron a cabo en las diferentes plantas del edificio. Se trata de un sistema difícil de manejar que requiere un cierto entrenamiento previo.

Al igual que con el sistema de movimiento por soplido, los movimientos iníciales han sido trayectorias rectilíneas, los resultados obtenidos fueron aceptables. A continuación se realizaron las pruebas alrededor de las columnas y de los ascensores dando resultados también aceptables.

El inconveniente que presenta este tipo de interfaz es que hay que tener especial cuidado con los movimientos realizados con la cabeza mientras se esta usando este ya que un movimiento de esta puede variar la trayectoria sustancialmente. A continuación se muestra la imagen de puesta en funcionamiento de esta interfaz.

Imagen 4. 3. Movimiento con el joystick de cabeza

CAPÍTULO 5

CONCLUSIONES Y TRABAJOS FUTUROS

5.1. Conclusiones

La finalidad de este proyecto ha sido la de confeccionar un nuevo sistema con el que poder realizar desplazamientos en distintos tipo de plataformas, en nuestro caso para incorporarlo a **SIAMO II.**

El uso y desarrollo de este nuevo sistema hace que el abanico de personas a la que va destinado sea más amplio puesto que hoy en día cada vez hay más personas que poseen movilidad reducida, además de que los tipos de minusvalía cada vez es más amplio.

La mayor dificultad que nos encontramos a la hora de realizar este proyecto vino dada por la estructura física de la silla. El problema fue el de incorporar a ésta todo el sistema de manera que ninguno de los elementos instalados no aumentase las dimensiones, lo que hubiera acarreado inconvenientes a la hora de acceder a determinados sitios. Por

otra parte se trataba de insertar el sistema donde ya existía otro anterior, por lo tanto todo lo el sistema de alimentación se ha mantenido.

5.2. Trabajos Futuros

a) Mejorar el sistema de sensores

Uno de los objetivos de este proyecto era el de incorporar sensores al sistema para la detección de obstáculos. Las pruebas realizadas en los distintos modos de funcionamiento dan como resultado que hay que mejorar este sistema.

En nuestro caso se han utilizado sensores de ultrasonidos, el problema que tienen éstos es que cuando la emisión del haz no es perpendicular al objeto el rebote de éste no es recibido por el mismo sensor, si no que o es un haz perdido o es recibido por otro sensor diferente. Una solución podría ser instalar más sensores o utilizar otro tipo de sensores complementarios a éstos.

b) Hacer que el movimiento del sistema sea autónomo

Actualmente se está trabajando en este aspecto aunque todavía no hay resultados que resaltar.

Se trata de una herramienta de gran utilidad, ya que a partir de unas rutas determinadas el sistema se puede mover de forma autónoma. Una de las grandes ventajas de esta herramienta vendría a la hora de moverse por edificios públicos donde el usuario podría desplazarse sin necesidad de ayuda, solo con introducir la ruta deseada en el sistema, éste le llevará al punto deseado.

c) Crear un programa de entrenamiento para manejar el sistema

Uno de los grandes problemas a la hora de manejar el sistema es que dispone de nuevas interfaces que no son muy comunes en el mercado. Es por ello que antes de usar el sistema en espacios abiertos es conveniente que el usuario realice un entrenamiento previo para familiarizarse y manejar correctamente el sistema.

III. MANUAL DE USUARIO

En esta parte del proyecto se pasará a explicar las acciones a realizar para obtener un funcionamiento óptimo del proyecto **SIAMO II.**

III.1. Interfaz hombre-máquina.

Lo primero que necesitamos para obtener un funcionamiento correcto es familiarizarnos con las diferentes interfaces hombre máquina y de esta forma saber el funcionamiento de cada dispositivo.

En las siguientes imágenes detallamos los distintos interfaces para familiarizarnos con ellos.

I.- Joystick Manual.

Imagen III.1 Joystick manual

En la figura III.1 podemos identificar:

1.-Interruptor de encendido→ Enciende y apaga la silla de ruedas.
2.-Led ON→ Lámpara que nos indica el estado de la silla.
3.-Interruptor bocina→ Pulsador para activar la bocina
4.-Potenciometro velocidad→ Regula la velocidad máxima de la silla.
5.-Teclado 3x4→ Teclado matricial usado para la selección del modo de funcionamiento.
6.-Joystick→ Joystick analógico de dos ejes X-Y
7.-Display LCD→ Dispositivo que nos ofrece distintas informaciones sobre el funcionamiento de la silla así como el modo de funcionamiento actual.

II.- Sensor de soplido.

Este interfaz consiste únicamente en un conducto por el que se tendrán que realizar las diferentes acciones de soplo para conseguir el guiado de la silla.

En la figura III.2 podemos identificar el aspecto del sensor de soplido.

Imagen III. 2 Sensor soplido

III.- Joystick de cabeza.

Este interfaz está constituido por un acelerómetro y un sensor óptico agregados artesanalmente a unas gafas.

El sensor óptico mediante una pajita realiza la función de clic de un ratón convencional.

En la figura III.3 se muestra una imagen del joystick de cabeza.

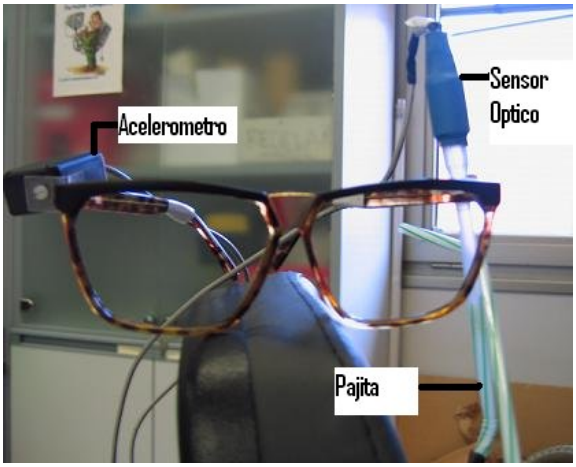


Imagen III. 3 Joystick de cabeza

IV.- Pantalla táctil.

Esta interfaz nos permite seleccionar la función del joystick de cabeza, ya que se puede utilizar como un ratón convencional o como un joystick para poder guiar la silla. También este dispositivo nos permite tener información sobre los parámetros de la silla.

La imagen III.4 nos muestra la pantalla táctil.

Imagen III. 4 Pantalla táctil

III.2. Puesta en marcha del sistema.

Para poner en marcha nuestro sistema lo primero que tenemos que realizar es activar el interruptor magnetotérmico situado en la parte posterior de la silla de ruedas, cerca de la batería.

Una vez accionado tenemos que tener en cuenta el modo de funcionamiento que vamos a utilizar, ya que, en algunos no necesitamos tener encendido el PC.

El PC solo es necesario para el guiado mediante el joystick de cabeza pero nos puede ser útil en los demás modos ya que nos proporciona diferentes informaciones de nuestro sistema.

Para encender el PC la única acción que tenemos que realizar es pulsar el botón de encendido del mismo.

Para encender el resto de mecanismos de la silla de ruedas tenemos que pulsar el botón rojo de interruptor de encendido situado en el joystick manual.

Una vez encendido, el display nos da la opción de elegir el modo de funcionamiento que deseamos, que son los siguientes:

1.-Si pulsamos el 1 en nuestro teclado se activará el modo de funcionamiento del joystick manual.

2.-Si pulsamos el 2 en nuestro teclado se activará el modo de funcionamiento mediante soplido.

3.-Si pulsamos el 3 en nuestro teclado se activará el modo de funcionamiento mediante PC.

Si estando en cualquier modo de funcionamiento pulsamos el 4 el sistema se detendrá y volverá al menú de selección de modo.

En nuestro joystick disponemos de un pulsador que activara una bocina para señalar las situaciones de peligro.

III.3. Instrucciones de guiado.

III.3.1 Guiado en modo joystick.

Este guiado es el más fácil de manejar para el usuario ya que, únicamente tenemos que manejar un joystick para seleccionar la dirección en la que queremos dirigirnos. Mediante el potenciómetro que disponemos en el joystick podemos regular la velocidad máxima para evitar una velocidad excesiva en espacios pequeños.

III.3.2 Guiado en modo soplido.

En este guiado también disponemos de potenciómetro para regular la velocidad. Para poder conseguir tener un buen manejo de la silla es recomendable entrenarse en espacios amplios para conseguir un buen manejo de la misma.

Las instrucciones de este guiado son las siguientes:

1.-Soplido fuerte→ Mediante un soplido fuerte la silla aumenta su velocidad y en el caso de que estuviera funcionando marcha atrás, disminuiría su velocidad hasta pararla.

2.-Aspiracion fuerte→Mediante una aspiración fuerte la silla disminuye su velocidad hasta pararla y en el caso de estar parada, se pondría a funcionar marcha atrás.

Estos dos tipos de soplidos son permanentes, es decir, tu soplas o aspiras un tiempo determinado para conseguir cierta velocidad y al dejar de soplar esa velocidad no será variada hasta una nueva aspiración o soplido.

3.-Soplido flojo→ Mediante un soplido flojo la silla girara hacia la derecha.

4.-Aspiracion floja→ Mediante una aspiración floja la silla girara hacia la izquierda.

Estos dos tipos de soplidos son momentáneos, es decir, en el momento que soplas la silla gira pero en el momento que dejas de soplar la silla deja de girar. Estos soplidos son muy usados para corregir la posición de la silla cuando está funcionando a cierta velocidad lineal constante.

III.3.3 Guiado en modo PC mediante joystick de cabeza.

En modo PC lo primero que debemos hacer es ejecutar la aplicación de nuestro programa. Una vez ejecutada, en la parte superior derecha tenemos una casilla con el nombre J.CABEZA que si la pulsamos activamos una pantalla en la que podemos activar el joystick de cabeza y seleccionar si queremos ir marcha atrás.

Una vez situados sobre la casilla para activar el joystick de cabeza la podemos activar soplando por la pajita o haciendo clic en la pantalla táctil. Si en cualquier momento volvemos a pulsar esa casilla el joystick se desactiva deteniendo la silla.

Del mismo modo que el guiado mediante soplo es recomendable entrenarse en espacios amplios para evitar accidentes.

Las instrucciones del guiado mediante joystick de cabeza son las siguientes:

1.-Giro de la cabeza hacia adelante→ La silla en función del grado de declinación de la cabeza adquiere velocidad.

2.-Giro de la cabeza hacia detrás→ La silla se detiene.

3.-Giro de la cabeza hacia la derecha→ La silla tiende a girar hacia la derecha.

4.-Giro de la cabeza hacia la izquierda→La silla tiende a girar hacia la izquierda.

Si pulsamos la casilla para ejecutar el guiado marcha atrás las instrucciones a seguir serian las mismas teniendo en cuenta que iría en sentido opuesto.

III.4. Desconexión del sistema

Para desconectar el sistema tenemos que desconectar tanto el PC como el resto del sistema. Para apagar nuestro sistema nos basta con pulsar el interruptor de encendido.

Para una correcta desconexión del sistema es recomendable esperar a que se apague adecuadamente el PC antes de apagar el magnetotérmico.

IV. PLIEGO DE CONDICIONES

A continuación se detallan los elementos físicos y las herramientas software utilizadas durante el desarrollo de este proyecto.

IV.1. Equipos Físicos

- Silla de ruedas motorizada

Dimensiones	108 cm x 58 cm x 110 cm (largo - ancho - altura)
Radio de la rueda	15,5 cm
Distancia entre ejes	52,5 cm
Velocidad máxima	2 m/s

- Robot Alcarroby.
- Encoders "HEDS-5500".
- Batería 24V
  2 x shimastu npc26-12(12V26AH)
- Display"Display LCD serie + i2c S310118".
- Teclado" Teclado matricial 4 x 3 teclas S310119".

- Joystick
- Sensor de soplido" awm2000v".
- Cargador de baterías"EMS Power 7969".
- Convertidor dc-dc" Mascot 8862000079".
- Tarjeta de potencia Roboteq AX-3500.
- Tarjeta micro controladora Philips LPC2129.
- Sensores de ultrasonidos "SRF02".
- Acelerómetro" ADXL330".
- Conversor de puerto USB a bus CAN "Lawicel CANUSB".

IV.2. Software

- Sistema operativo Windows XP.
- Procesador de textos (Microsoft Word).
- Roborun.
- Keil V3 + Ulink2.

V. PRESUPUESTO

En esta parte del proyecto se hará una estimación del coste total que supone la ejecución del mismo. En los apartados siguientes aparecen los gastos agrupados según su origen, y en el último apartado se detalla el presupuesto total.

A continuación se va a desglosar el presupuesto de ejecución material en los tres elementos siguientes:

- Coste de materiales.
- Coste del software utilizado.
- Coste de mano de obra por el tiempo empleado.

Coste de materiales

MATERIAL	PRECIO	UNIDADES	TOTAL
Silla de ruedas	9000 €	1	9000 €
Encoder	54,27 €	2	108,54 €
Mecanizado encoder	2000 €	2	4000 €
Bateria	198 €	1	198 €
Cargador Bateria	135 €	1	135 €
Convertidor dc-dc	86,36 €	1	86,36 €
Mini PC	500 €	1	500 €
Pantalla tactil	735 €	1	735 €
Joystick	14,85 €	1	14,85 €
teclado	5,75 €	1	5,75 €
Display	32,8 €	1	32,8 €
Sensor ultrasonidos	13,75 €	9	123,75 €
Sensor soplido	73 €	1	73 €
Tarjeta LPC2129	55 €	3	165 €
Tarjeta Ax-3500	522 €	1	522 €
Acelerometro	42 €	2	84 €
conversor USB-CAN	132 €	1	0 €
Material electrico	600 €	1	600 €
		TOTAL	**16516,05 €**

Coste del software utilizado

MATERIAL	PRECIO
Microsoft Windows XP Profesional	135 €
Microsoft Office XP	200 €
Roborun	0 €
Keil V3 + ulink2	175 €
Material de oficina	120 €
TOTAL	**630,0 €**

Coste de la mano de obra por el tiempo empleado

FUNCION	Nº HORAS	€/HORA	TOTAL
Ingenieria	950	50	45000 €
Mecanografiado	100	12	1200 €
		TOTAL	**46200 €**

Coste total del presupuesto de ejecución material

PARTIDA	PRECIO
Coste materiales	**16516,05 €**
Coste software	**630,0 €**
Coste mano de obra	**46200,0 €**
TOTAL	**63346,05 €**

A continuación se enmarcan los gastos generados por las instalaciones utilizadas durante la realización del proyecto más el beneficio industrial. A efectos de cálculo se estima este apunte como el 20 % del presupuesto de ejecución material del proyecto.

El valor de los gastos generales y beneficio industrial asciende a **12669,21 €.**

Se han calculado, tanto los honorarios de dirección como los de redacción, como el 7 % del presupuesto de ejecución material.

El valor de los honorarios por dirección asciende a **4434,22€.**

El valor de los honorarios por redacción asciende a **4434,22€.**

Presupuesto total

Este concepto incluye el presupuesto de ejecución material, los gastos generales y el beneficio industrial

Presupuesto de ejecución material	**63346,05 €**
Gastos generales y beneficio industrial	**12669,21 €**
Honorarios por dirección	**4434,22€**
Honorarios por redacción	**4434,22€**
Total(sin iva)	**84883,707 €**
IVA(16%)	**13581,39 €**
TOTAL	**98465,10 €**

El presupuesto total del proyecto asciende a la cantidad de noventa y ocho mil y cuatro cientos sesenta y cinco euros con diez céntimos de **euro.**

Alcalá de Henares a 9 de Septiembre de 2009.

Firmado: Juan Bellón Álvarez

Ingeniero Técnico Industrial

VI. PLANOS

En esta parte del proyecto se adjuntan la imagen de la silla en su estado final, los esquemáticos de los circuitos diseñados y los códigos que se han generado para el desarrollo de los distintos nodos del sistema. Primero se muestra la imagen de la silla seguida de los esquemáticos de los circuitos.

A continuación aparecen los códigos necesarios para desarrollar los distintos nodos. Primero se muestran los archivos que contienen las configuraciones de los microcontroladores y del bus CAN, a continuación se muestranel resto de códigos pertenecientes a cada nodo.

VI.1. Imagen general de la silla

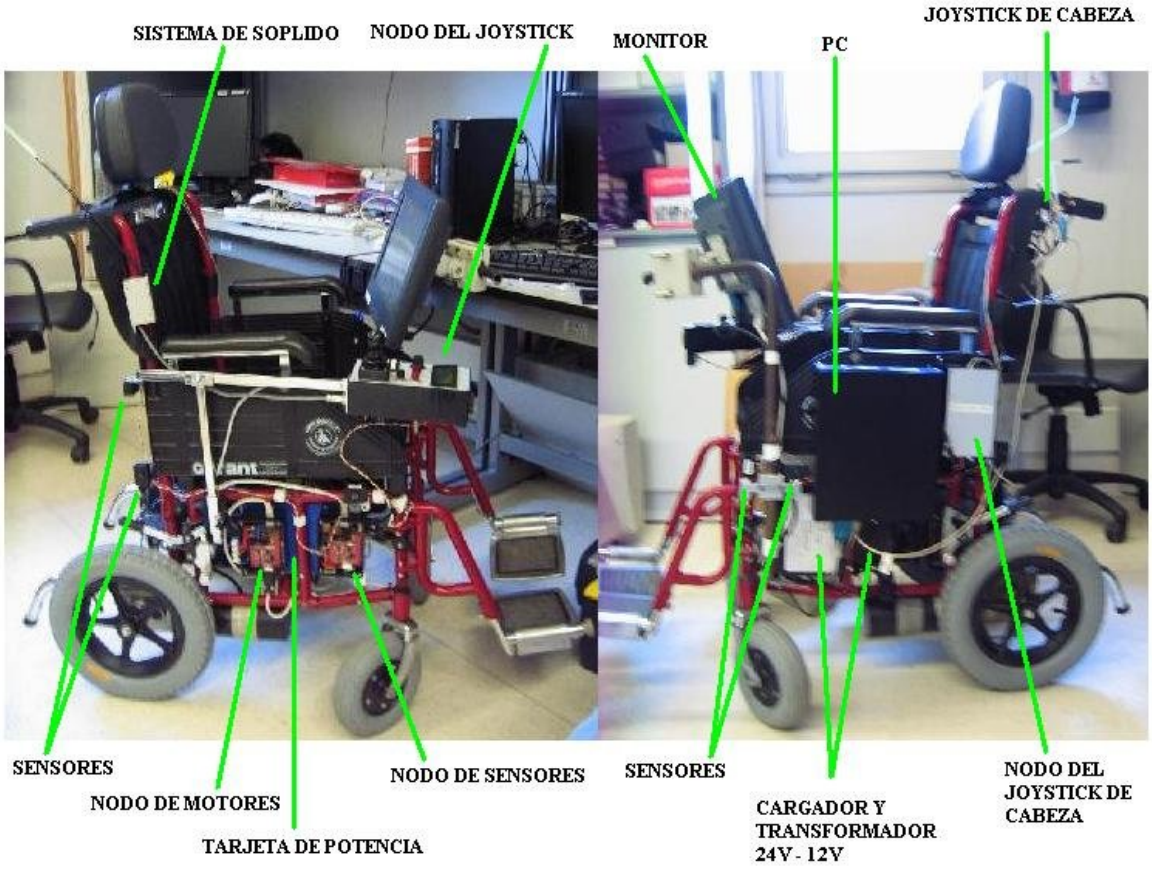


Imagen VI. 1. Situación de todos los elementos

VI.2. Esquemáticos

a) Esquemas de aliemntacion y buses

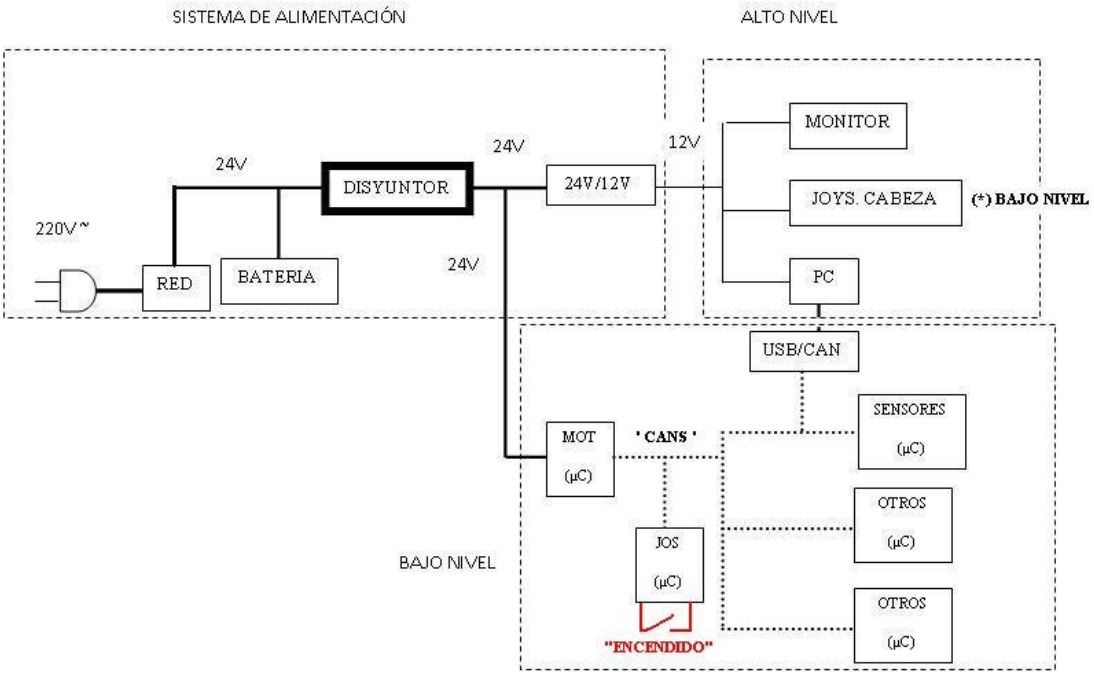


Imagen VI. 2. Esquema general del sistema

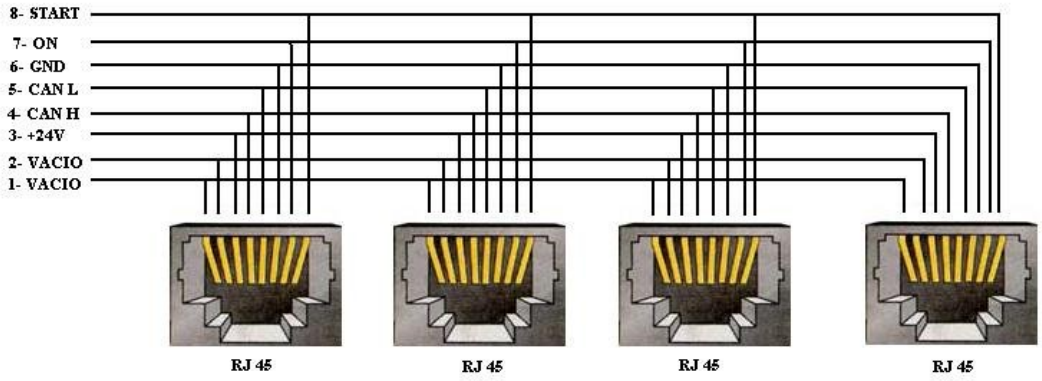


Imagen VI. 3. Regleta del bus CAN

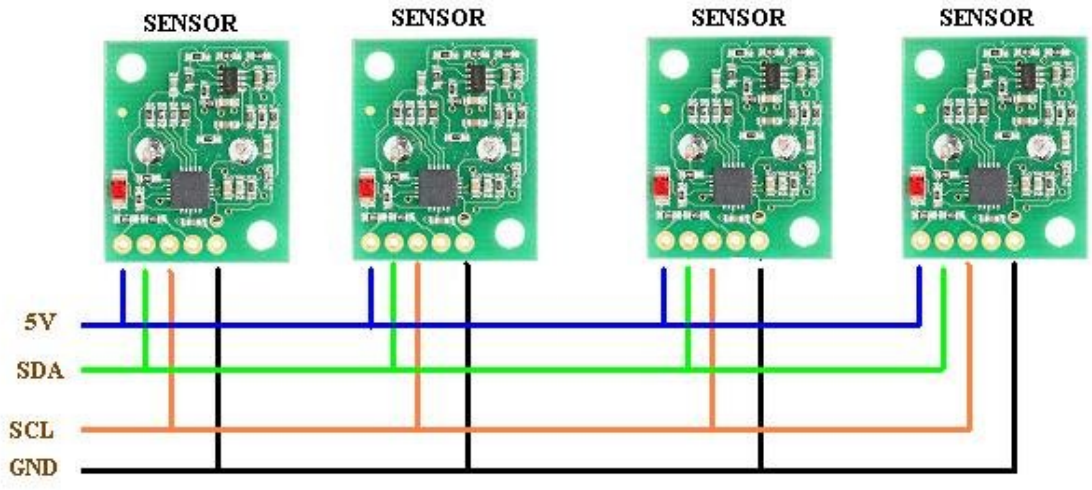


Imagen VI. 4. Bus I2C

b) Nodo joystick

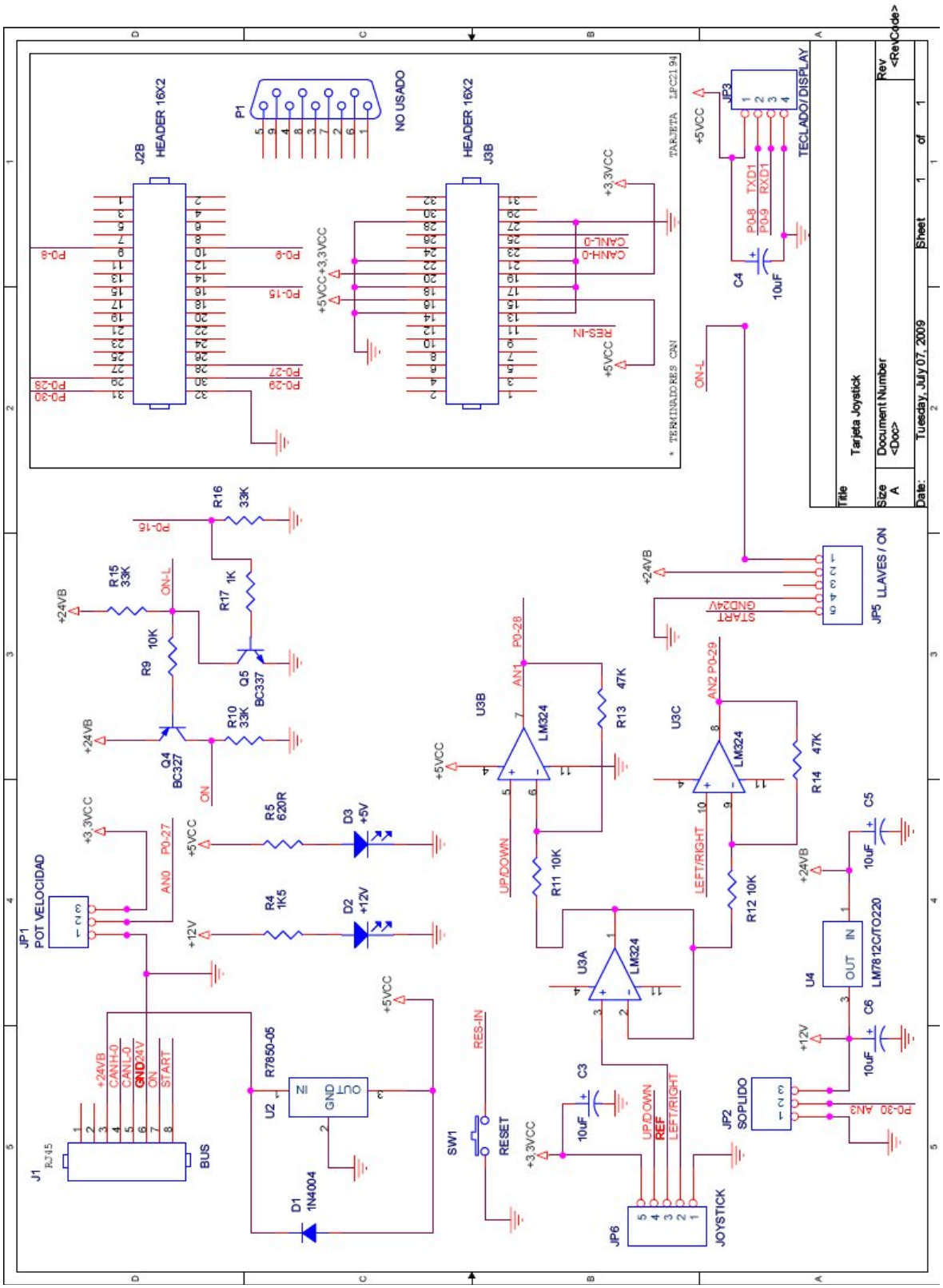


Imagen VI. 5. Tarjeta del nodo del joystick

c) Nodo tarjeta de motores

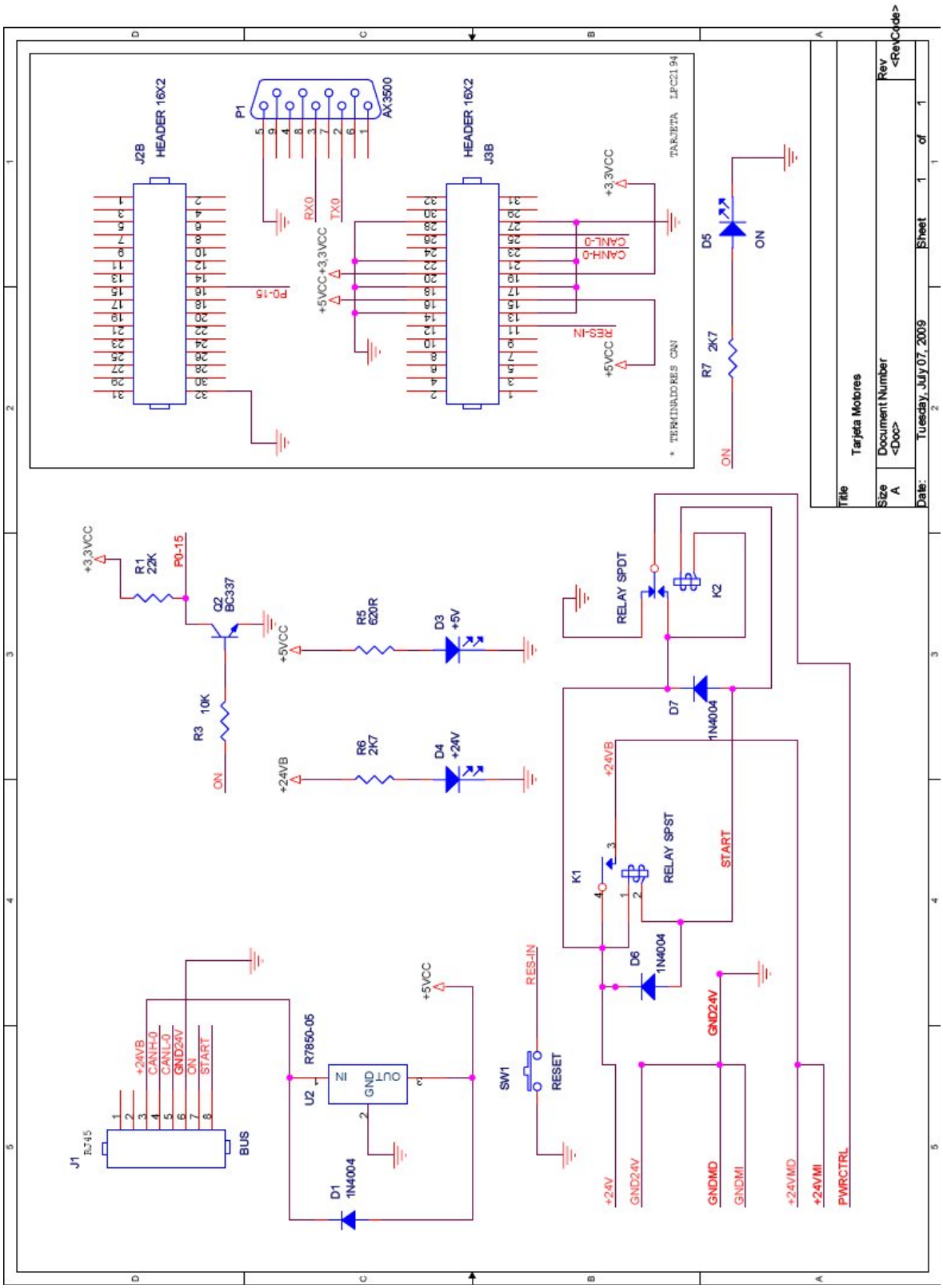


Imagen VI. 6. Tarjeta del nodo de motores

d) Nodo tarjeta de sensores

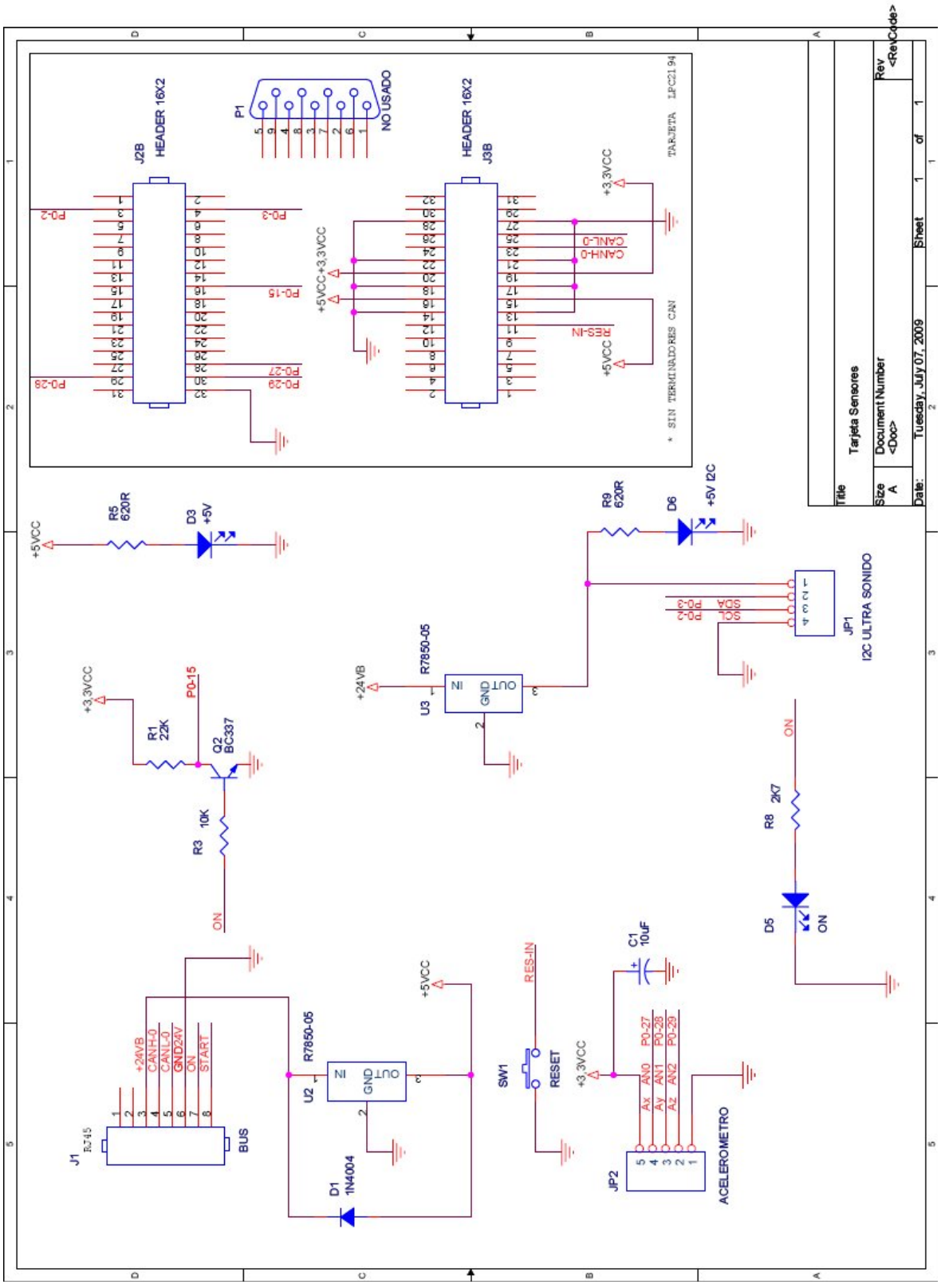


Imagen VI. 7. Tarjeta del nodo de sensores

VI.3. Programación de los nodos

En este apartado se muestran todos los códigos generados para el desarrollo de los nodos por los que esta compuesto el sistema.

VI.3.1. Codigos del Nodo joystick

En este subapartado se detallan los códigos de los archivos necesarios para trabajar la programación del nodo del joystick que son los siguientes:

1.- LPC_FullCAN_SW.c
2.- LPC_FullCAN_SW.h
3.- Driver.c
4.- Driver.h
5.- Setup.c
6.- Setup.h
7.- Serial.c
8.- Serial.h
9.- Funciones.c
10.- Funciones.h
11.- Main.c
12.- LPC21XX.h

Las arhivos LPC_FullCAN_SW son archivos originales, usados para la configuración de los dispositivos CAN y sus respectivas funciones de envio y recepción de datos.

El archivo main.c es el archivo principal en el que incluyendo los diferentes archivos .h conseguimos enlazarlo con las respectivas funciones de los archivos .c.

Los archivos driver.c y driver.h son archivos que nos facilitan la llamada a funciones frecuentemente usadas.

Los archivos setup.c y setup.h se encargan de la configuración general del sistema.

Las funciones serial.c y serial.h se encargan de la gestión del puerto serie.

Los archivos funciones.c y funciones.h se encargan de la gestión de las funciones principales de nuestro sistema.

A continuación se detallan los códigos de cada uno de los archivos creados, y las diversas funciones de cada uno de ellos.

```
#*******************************************************************************//
#*******************************************************************************//
# ARCHIVO: MAIN.C                                                                //
# AUTOR: Juan Bellón                                                            //
# Tarjeta joystick                                                               //
#                                                                                //
#*******************************************************************************//
#*******************************************************************************//



#include <LPC21xx.H>
#include "LPC_FullCAN_SW.h"
#include "setup.h"
#include "driver.h"
#include "funciones.h"
#include "serial.h"



#*******************************************************************************//
# NOMBRE: main                                                                   //
# AUTOR: Juan Bellón                                                            //
# FUNCIÓN: Ejecución global del nodo del joystick                                //
# PARAMETROS RECIBIDOS: No tiene                                                 //
# PARAMETROS DEVUELTOS: No tiene                                                 //
#*******************************************************************************//



int main(void)
{

        int i;
        funcion=4;     // funcion=1 soplido funcion=0 joystick funcion=2 pc funcion=4 menú
        estado=CERO;   // Variables inicializacion soplido
        modo=STOP;     // Variables inicializacion soplido
        tiempo=0;      // Variables inicializacion soplido
        velocidad=0;   // Variables inicializacion soplido

        config();      //Configuración general del sistema setup.h
        init_serial(); // Configuración serial serial.h
        for (i=0;i<=1000000;i++);
        borrar_display(); //borramos display

        dir_io(0x00008000); //encendemos led_on
        set_io(0x00008000);

        while(1)
        {
                /* Tcan controla el sondeo del bus can y el envio de datos serie 10mseg */
                if(Tcan==1)
                {

                        Timercan=100;
                        Active_can=1;
                        Tcan=0;
                        desactivoISR();
                        can();
                        activoISR();
                        desactivoISR();
                        tx_serie();
                        activoISR();

                }
                /* Tmodo controla el de sondeo envio continuo del modo de funcionamiento  */
                if(Tmodo==1)
                {
                        desactivoISR();
                        Timermodo=5000;
                        Active_modo=1;
                        Tmodo=0;
                        can_modo();
                        activoISR();
                }

                /* T3 controla l sondeo del jostyck 100mseg*/
```

```
                if((T3==1)&&(funcion!=1))
                {
                Timer3=1000;
                Timer0=1000;
                        Active_T3=1;
                        T3=0;
                        adquiere_jostyck();
                        conversion_jostyck();
                        desactivoISR();
                        Can_adquisicion();
                        activoISR();
                }

                /* T1 controla el sondeo del soplido 50mseg*/
                if((T1==1)&&(funcion==1))
                {
                        Timer1=500;
                        Active_T1=1;
                        T1=0;
                        soplido();
                        conversion_soplido();
                        desactivoISR();
                        Can_adquisicion();
                        activoISR();
                }

                /*T2 controla la actualización del display y el teclado 150mseg */
                if(T2==1)
                {
                        Timer2=1500;
                        Active_T2=1;
                        T2=0;
                        desactivoISR();
                        display();
                        activoISR();
                }
        }
}
```

```
#*********************************************************************************//
#*********************************************************************************//
# ARCHIVO: FUNCIONES.H                                                            //
# AUTOR: Juan Bellón                                                            //
# Tarjeta joystick                                                                //
#                                                                                 //
#*********************************************************************************//
#*********************************************************************************//



int velocidad;
int datomotorD,datomotorI;

int nivel_battery;

enum tipo_de_estado {CERO, PREGUNTA, UNICO_E};
static enum tipo_de_estado estado;
static int tiempo;

enum tipo_veloc {STOP, ANDANDO, PARANDO};
static enum tipo_veloc modo;


void can(void);
void can_modo(void);
void adquiere_jostyck(void);
void conversion_jostyck(void);
int adquiere_soplido(void);
void conversion_soplido(void);
void soplido(void);
void Can_adquisicion(void);

void initDisplay(void);
void setDisplay(unsigned char digit);
```

```
*******************************************************************************//
#******************************************************************************//
# ARCHIVO: FUNCIONES.C                                                          //
# AUTOR: Juan Bellón                                                          //
# Tarjeta joystick                                                              //
#                                                                               //
#******************************************************************************//
#******************************************************************************//



#******************************************************************************//
# NOMBRE: adquiere_jostyck                                                      //
# AUTOR: Juan Bellón                                                          //
# FUNCION: adquiere los datos procedentes del joystick y del potenciómetro    //
# PARAMETROS RECIBIDOS: No tiene                                                //
# PARAMETROS DEVUELTOS: valor2(0V - 3.3V); valor(0V - 3.3V); valor1 (0v - 3.3V)//
#******************************************************************************//



void adquiere_jostyck(void){

        estado=CERO;
        tiempo=0;
        Dato=NULO;
        Vlineal=0;
        Vgiro=0;


        ADCR = 0x012E0401;    // Adquisición del potenciometro

        do
        {
                valor2 = ADDR;
        } while ((valor2 & 0x80000000) == 0);       // Espera hasta fin de adquisición

        ADCR &= ~0x01000000;
        valor2 = (valor2 >> 6) & 0x03FF;     // Extracion del valor


        ADCR = 0x012E0402;    // Adquisición del eje X
        do
        {
                valor = ADDR;
        } while ((valor & 0x80000000) == 0); // Espera hasta fin de adquisición

        ADCR &= ~0x01000000;
        valor = (valor >> 6) & 0x03FF;       // Extracción del valor

        ADCR = 0x012E0404;    // Adquisición del eje Y
        do
        {
        valor1 = ADDR;
        } while ((valor1 & 0x80000000) == 0);       // Espera hasta fin de adquisición

        ADCR &= ~0x01000000;
        valor1 = (valor1 >> 6) & 0x03FF;     // Extración del valor


}
```

```
#*********************************************************************************//
# NOMBRE: conversion_jostyck                                                      //
# AUTOR: Juan Bellón                                                        //
# FUNCION: Convertir los datos procedentes del joystick y del potenciometro       //
# PARAMETROS RECIBIDOS: valor valor(0V - 3.3V); valor1(0V - 3.3V); valor2 (0v - 3.3V) //
# PARAMETROS DEVUELTOS: datomotorD (0-7F); datomotorI (0-7F)                      //
#*********************************************************************************//



void conversion_jostyck(void)
{

        int ejey,ejex;
        int V, R;
        float wd, wi;


        if ((valor>0x1e0)&&(valor<0x220))
        {
                ejex=0;
                V=0;
        }
        else
        {
                ejex=valor;
                V=(((ejex*75)/100)-385);
        }

        if ((valor1>0x1e0)&&(valor1<0x220))
        {
                ejey=0;
                R=0;
        }
        else
        {
                ejey=valor1;
                R=(((ejey*150)/100)-770);
        }

        wd=((1000/RADIO)*(V-((R*DISEJES)/2000)));
        wi=((1000/RADIO)*(V+((R*DISEJES)/2000)));

        if(wd >0)
        datomotorD=(((wd/18)/1050)*valor2);

        else if(wd==0)
        datomotorD=0;
        else
        datomotorD=(((wd/18)/1050)*valor2);


        if(wi >0)
        datomotorI=(((wi/18)/1050)*valor2);
        else if(wi==0)
        datomotorI=0;
        else
        datomotorI=(((wi/18)/1050)*valor2);

}
```

```
#*****************************************************************************//
# NOMBRE: adquiere_soplido                                                      //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Adquiere los datos procdedentes del sistema de soplido y del         //
#          potenciometro                                                      //
# PARAMETROS RECIBIDOS: No tiene                                                //
# PARAMETROS DEVUELTOS: valor2 (0V - 3.3V); valor (0V - 3.3V)                   //
#*****************************************************************************//



int adquiere_soplido(void)
{

        ADCR = 0x012E0401;    // Comienzo adquisición potenciometro
        do
        {
                valor2 = ADDR;
        } while ((valor2 & 0x80000000) == 0);       // Espera hasta fin de adquisición

        ADCR &= ~0x01000000;
        valor2 = (valor2 >> 6) & 0x03FF;     // Extracción del dato

        ADCR = 0x012E0408;    // Comienzo adquisición del soplido
        do
        {
                valor = ADDR;
        } while ((valor & 0x80000000) == 0);        // Espera hasta fin de adquisición

        ADCR &= ~0x01000000;
        valor = (valor >> 6) & 0x03FF;       // Extracción del dato

        return valor;


}
```

```
#*******************************************************************************//
# NOMBRE: soplido                                                               //
# AUTOR: Juan Bellón                                                           //
# FUNCION: Manejar los datos adquiridos en la dunción adquiere_soplido para     //
#          realizar los movimientos                                            //
#          Maquina_estado_soplido                                               //
# PARAMETROS RECIBIDOS: No tiene                                                //
# PARAMETROS DEVUELTOS: Dato (NULO, MA, MB,A,B)                                 //
#*******************************************************************************//


void soplido(void)
{
	int val;


	switch (estado)
	{
		case CERO:
			Dato=NULO;
			tiempo=0;
			val=adquiere_soplido();

			if ((val>0x2AA)&&(val<0x3E0)){
				entrada=A;
				estado=PREGUNTA;
			}
			else if(val>=0x3E0){
				entrada=MA;
				estado=PREGUNTA;
			}
			else if ((val<0x180)&&(val>0x50)){
				entrada=B;
				estado=PREGUNTA;
			}
			else if (val<=0x50){
				entrada=MB;
				estado=PREGUNTA;
			}

			else{
				entrada=NULO;
				estado=CERO;
			}

		break;

		case PREGUNTA:

			val=adquiere_soplido();
			tiempo++;
			if (tiempo>=4)
				estado=UNICO_E;
			switch (entrada)
			{
				case MA:
					if(val<0x3E0){
						estado=CERO;
						tiempo=0;
						entrada=NULO;
					}
				break;
				case MB:
					if(val>0x50){
						estado=CERO;
						tiempo=0;
						entrada=NULO;
					}
				break;
				case A:
					if(val>=0x3E0){
						entrada=MA;
					}
					else if(val<0x2AA){
						estado=CERO;
						tiempo=0;
```

```
                                        entrada=NULO;
                                }
                        break;
                        case B:
                                if(val<=0x50){
                                        entrada=MB;
                                }
                                else if(val>0x180){
                                        estado=CERO;
                                        tiempo=0;
                                        entrada=NULO;
                                }

                        break;
                        case NULO:
                        break;
                }
        break;

        case UNICO_E:
                val=adquiere_soplido();
                switch (entrada)
                {
                        case MA:
                                if(val<0x3E0){
                                        Dato=NULO;
                                        estado=CERO;
                                        tiempo=0;
                                        entrada=NULO;
                                }
                                else{
                                        Dato=MA;
                                        estado=UNICO_E;
                                }
                        break;
                        case MB:
                                if(val>0x50){
                                        estado=CERO;
                                        tiempo=0;
                                        entrada=NULO;
                                        Dato=NULO;}
                                else{
                                        Dato=MB;
                                        estado=UNICO_E;
                                }
                        break;
                        case A:
                                if(val<0x2AA){
                                        estado=CERO;
                                        tiempo=0;
                                        entrada=NULO;
                                        Dato=NULO;
                                }
                                else{
                                        Dato=A;
                                        estado=UNICO_E;
                                }
                        break;
                        case B:
                                if(val>0x180){
                                        estado=CERO;
                                        tiempo=0;
                                        entrada=NULO;
                                        Dato=NULO;
                                }
                                else{
                                        Dato=B;
                                        estado=UNICO_E;
                                }
                        break;
                        case NULO:
                        break;
                }

        break;
    }
}
```

```
#*********************************************************************************//
# NOMBRE: conversion_soplido                                                      //
# AUTOR: Juan Bellón                                                            //
# FUNCION: Manejar los datos de la función soplido para obtener los datos de    //
#          los motores                                                          //
# Maquina_estados_conversion_soplido                                            //
# PARAMETROS RECIBIDOS: Dato (NULO, MA, MB, A, B)                               //
# PARAMETROS DEVUELTOS: datomotorD (0-7F), datomotorI (0-7F)                    //
#*********************************************************************************//

void conversion_soplido(void)
{
        switch (modo)
        {
                case STOP:
                        if(Dato!=NULO){

                                modo=ANDANDO;}
                        else{

                                modo=STOP;}
                break;
                case ANDANDO:
                        switch (Dato)
                        {
                                case MA:

                                        Vgiro=0;
                                        if(Vlineal>=0){
                                                Vlineal++;
                                                if (Vlineal>80)
                                                        Vlineal=80;
                                                }
                                        else if(Vlineal<0){
                                                Vlineal+=6;
                                                if(Vlineal>=0){
                                                        Vlineal=0;
                                                        modo=PARANDO;
                                                }}
                                        break;
                                        case MB:

                                                Vgiro=0;
                                                if(Vlineal<=0){
                                                        Vlineal--;
                                                        if (Vlineal<-60)
                                                                Vlineal=-60;
                                                        }
                                                else if(Vlineal>0){
                                                        Vlineal-=6;
                                                                if(Vlineal<=0){
                                                                Vlineal=0;
                                                                modo=PARANDO;
                                                        }}
                                                break;
                                                case A:

                                                        Vgiro=20;
                                                break;
                                                case B:

                                                        Vgiro=-20;
                                                break;
                                                case NULO:
                                                        Vgiro=0;
                                                break;
                                }
                break;
                case PARANDO:
                        if(Dato==NULO)
                                modo=STOP;
                break;

        }
        datomotorI=(((Vlineal+Vgiro)*valor2)/1023);
        datomotorD=(((Vlineal-Vgiro)*valor2)/1023);
}
```

```
#*****************************************************************************//
# NOMBRE: Can_adquisicion                                                      //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Enviar los datos la tarjeta de motores                              //
# PARAMETROS RECIBIDOS: datomotorD (0 - 7F), datomotor I (0 - 7F)              //
# PARAMETROS DEVUELTOS: Mensajes CAN con wD y wI                               //
#*****************************************************************************//



void Can_adquisicion(void){

        short potvel,poty,potx;
        signed char velI,velD;
        velI=(char)datomotorI;
        velD=(char)datomotorD;
        potvel=(short)valor2;
        poty=(short)valor1;
        potx=(short)valor;
        FULLCAN_MSG MsgBuf;
        MsgBuf.Dat1 = 0x00080110L;
        MsgBuf.DatA = ((potx<<16)|(velI<<8));       // Envio del dato wD
        MsgBuf.DatA = (MsgBuf.DatA|(velD&0xFF));
        MsgBuf.DatB = ((potvel<<16)|poty);          // Envio del dato wI
        // Transmit initial message on CAN 1
        FullCAN_PushMessage(1,&MsgBuf);
}



#*****************************************************************************//
# NOMBRE: can_modo                                                            //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Envio del modo de funcionaminto                                    //
# PARAMETROS RECIBIDOS: Modo de funcionamiento: 0 = Joystick;  1 = Soplido     //
#                                               2 = Ordenador; 3 = Menú       //
# PARAMETROS DEVUELTOS: Emnsaje can con el modo de funcionamiento             //
#*****************************************************************************//



void can_modo(void){

        FULLCAN_MSG MsgBuf;
        MsgBuf.Dat1 = 0x00080111L;
        MsgBuf.DatB= 0;
        MsgBuf.DatA = funcion; // Envio modo: 0=joystick; 1=soplido; 2=ordenador; 4=modomenu

        FullCAN_PushMessage(1,&MsgBuf); // Envio del mensaje por le bus CAN
}
```

```
#*******************************************************************************//
#*******************************************************************************//
# ARCHIVO: SETUP.C                                                              //
# AUTOR: Juan Bellón                                                          //
# Tarjeta joystick                                                              //
#                                                                               //
#*******************************************************************************//
#*******************************************************************************//



#*******************************************************************************//
# NOMBRE: config                                                                //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Configura el sistema                                                 //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: Timercan = 100; Timermodo = 5000; Timer0 = 1000        //
#                       Timer1 = 1000;  Timer2 = 1500;    Timer3 = 1000       //
#                       Active_can = 1; Active_modo = 1;  Active_T0 = 1       //
#                       Active_T1 = 1;  Active_T2 = 1;    Active = 1          //
#                       Tcan = 0;       Tmodo = 0;        T0 = 0              //
#                       T2 = 0;         T3 = 0                                //
#*******************************************************************************//



void config(void){

        Timercan=100;
        Active_can=1;
        Tcan=0;

        Timermodo=5000;
        Active_modo=1;
        Tmodo=0;

        Timer0=1000;
        Active_T0=1;
        T0=0;

        Timer1=1000;
        Active_T1=1;
        T1=0;

        Timer2=1500;
        Active_T2=1;
        T2=0;

        Timer3=1000;
        Active_T3=1;
        T3=0;

        VPBDIV = 1;
        VICIntEnClr = 0xFFFFFFFFL;
        VICIntSelect = 0x00000000L;
        VICDefVectAddr = (unsigned long) DefaultISR;

        // Inicialización del bus CAN
        FullCAN_Init(1,0,CANBitrate001m_12MHz);
        FullCAN_SetErrIRQ(2);

        // Initialize Timer Interrupt
        T0MR0 = 5999; // 100 microsegundoss = 6.000-1 cuentas
        T0MCR = 3;
        T0TCR = 1;

        VICVectAddr3 = (unsigned long) Timer0ISR;
        VICVectCntl3 = 0x20 | 4;
        VICIntEnable = 0x00000010;
        // Filtros para la aceptación de  mensajes por el bus CAN
        FullCAN_SetFilter(1,0x210);

        // Esperamos 10 milisegundos para una correcta configuración
        SlowDown(100);
}
```

```
#*****************************************************************************//
# NOMBRE: Timer0ISR                                                           //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Control de los timers del sistema                                  //
# PARAMETROS RECIBIDOS: Timercan; Timermodo; Timer0; Timer1; Timer2; Timer3   //
#                       Active_can; Active_modo; Active_T0; Active_T1;        //
#                       Active_T2; Active_T3                                  //
# PARAMETROS DEVUELTOS: Active_can = 0; Active_modo = 0; Active_T0 = 0        //
#                       Active_T3 = 0;  Active_T1 = 0;   Active_T2 = 0        //
#                       Tcan = 1; Tmodo = 1; T0 = 1; T1 = 1; T2 = 1; T3 = 1   //
#*****************************************************************************//


void Timer0ISR (void)
{

  gTimerTick++;
  cleanISR();//
  if (Active_can)
               Timercan--;
  if (Timercan==0){
               Tcan=1;
               Active_can=0;
               }

  if (Active_modo)
               Timermodo--;
  if (Timermodo==0){
               Tmodo=1;
               Active_modo=0;
               }

  if (Active_T0)
               Timer0--;
  if (Timer0==0){
               T0=1;
               Active_T0=0;
               }

  if (Active_T3)
               Timer3--;
  if (Timer3==0){
               T3=1;
               Active_T3=0;
               }

  if (Active_T1)
               Timer1--;
  if (Timer1==0){
               T1=1;
               Active_T1=0;
               }

  if (Active_T2)
               Timer2--;
  if (Timer2==0){
               T2=1;
               Active_T2=0;
               }

  VICVectAddr = 0xFFFFFFFF;
}
```

```
#*******************************************************************************//
#*******************************************************************************//
# ARCHIVO: SERIAL.H                                                             //
# AUTOR: Juan Bellón                                                          //
# Tarjeta joystick                                                              //
#                                                                               //
#*******************************************************************************//
#*******************************************************************************//



int funcion;


void borrar_display(void);
void display(void);
void tx_serie(void);



void init_serial (void);

void miputchar (int ch);
char migetchar (void);
void puthex (int hex);



#*******************************************************************************//
#*******************************************************************************//
# ARCHIVO: SERIAL.C                                                             //
# AUTOR: Juan Bellón                                                          //
# Tarjeta joystick                                                              //
#                                                                               //
#*******************************************************************************//
#*******************************************************************************//



#*******************************************************************************//
# NOMBRE: init_serial                                                           //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Inicialización del bus serie                                         //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*******************************************************************************//



void init_serial (void)  {    //Inicialización del bus serie

  PINSEL0 = 0x00050000;       // Activación de los puertos TxD1 y RxD1
  U1LCR = 0x83;       // 8 bits, sin paridad, un bit de parada
  U1DLL = 0x84;
  U1DLM = 0x1; // 9600 Bauidos, reloj VPB de 15MHz
  U1LCR = 0x03;
}
```

```
#*****************************************************************************//
# NOMBRE: tx_serie                                                            //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Control buffer serie                                               //
# PARAMETROS RECIBIDOS: Txbuf, txrd[]                                         //
# PARAMETROS DEVUELTOS: teclado                                               //
#*****************************************************************************//



void tx_serie(void){
       if(v1!=v2)
       {
              int x,Borrar;
              txrd=Txbuf;
              aux=txrd[v2];
              if (aux==16)
              {
                     x=-1;
                     Borrar=1;

                     if ((U1LSR & 0x01))
                     {
                            aux1=U1RBR;
                     }
                     else
                            Borrar=0;
                     miputchar(16);
                                    do{
                            while (!(U1LSR & 0x01));

                            if (x==-1)
                            {
                                   aux1=U1RBR;
                                   x=0;
                            }
                            else{
                                   aux2=U1RBR;
                                   x=2;
                            }
                     }while(x!=2);

                     teclado=((aux2<<8)|aux1);

              }

              else
                     miputchar(aux);
              v2++;
              if(v2>=255)
                     v2=0;
       }
}



#*****************************************************************************//
# NOMBRE: miputchar                                                           //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Escribir caracteres en el puerto serie                             //
# PARAMETROS RECIBIDOS: Dato a transmitir                                     //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//



void miputchar (int ch)  {

  int n;
  for(n=0;n<=10000;n++);
  while (!(U1LSR & 0x20));
  U1THR=ch;
}
```

```
#*******************************************************************************//
# NOMBRE: migetchar                                                             //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Leer caracteres del puerto serie                                     //
# PARAMETROS RECIBIDOS: No tiene                                                //
# PARAMETROS DEVUELTOS: Dato recibido                                           //
#*******************************************************************************//


char migetchar (void)  {

  while (!(U1LSR & 0x01));

  return (U1RBR);

}


#*******************************************************************************//
# NOMBRE: display                                                               //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Visualizar datos en el display y el control del teclado              //
# PARAMETROS RECIBIDOS: Txbuf, nivel de carga de la batería, modo de funcionamiento   //
# PARAMETROS DEVUELTOS: Modo de funcionamiento                                  //
#*******************************************************************************//


void display(void){

        txwr=Txbuf;
        if (nivel_battery>290||nivel_battery<220)
                nivel_battery=battery;
        else
                battery=nivel_battery;

        unsigned char r1,r2,r3,c1,c2,c3;
        c1=(char)(nivel_battery/10);
        r1=(char)(nivel_battery%10);
        c2=c1/10;
        r2=c1%10;
        c3=c2/10;
        r3=c2%10;

        txwr[v1]=0x02;
        v1++;
        if(v1>=255)
                v1=0;
        txwr[v1]=0x0A;
        v1++;
        if(v1>=255)
                v1=0;
        txwr[v1]=(r3+0x30);
        v1++;
        if(v1>=255)
                v1=0;
        txwr[v1]=(r2+0x30);
        v1++;
        if(v1>=255)
                v1=0;
        txwr[v1]=('.');
        v1++;
        if(v1>=255)
                v1=0;
        txwr[v1]=(r1+0x30);
        v1++;
        if(v1>=255)
                v1=0;
        txwr[v1]=(0x56);
        v1++;
        if(v1>=255)
        v1=0;
        txwr[v1]=(0x20);
        v1++;
        if(v1>=255)
```

```
        v1=0;
txwr[v1]=(4);
v1++;
if(v1>=255)
        v1=0;
txwr[v1]=(0x04);
v1++;
if(v1>=255)
        v1=0;
if (funcion==4){
        if (primera==1){
                primera=0;
                short j;
                char fila1[20]="";
                char fila2[20]="Seleccione modo";
                char fila3[20]="1.- joystick";
                char fila4[20]="2.- soplido  3.-PC";
                int letra;
                j=0;


                txwr[v1]=(0x13);
                v1++;
                if(v1>=255)
                        v1=0;
                txwr[v1]=(0x02);
                v1++;
                if(v1>=255)
                        v1=0;
                txwr[v1]=(0x01);
                v1++;
                if(v1>=255)
                        v1=0;
                while (fila1[j]!='\0'){
                        letra=fila1[j];
                        txwr[v1]=(letra);
                        v1++;
                        if(v1>=255)
                                v1=0;
                        j++;
                }
                j=0;
                txwr[v1]=(0x02);
                v1++;
                if(v1>=255)
                        v1=0;
                txwr[v1]=(0x15);
                v1++;
                if(v1>=255)
                        v1=0;
                while (fila2[j]!='\0'){
                        letra=fila2[j];
                        txwr[v1]=(letra);
                        v1++;
                        if(v1>=255)
                                v1=0;
                        j++;
                }
                j=0;
                txwr[v1]=(0x02);
                v1++;
                if(v1>=255)
                        v1=0;
                txwr[v1]=(0x29);
                v1++;
                if(v1>=255)
                v1=0;
                while (fila3[j]!='\0'){
                        letra=fila3[j];
                        txwr[v1]=(letra);
                        v1++;
                        if(v1>=255)
                                v1=0;
                        j++;
                }
                j=0;
                txwr[v1]=(0x02);
```

```
                v1++;
                if(v1>=255)
                        v1=0;
                txwr[v1]=(0x3D);
                v1++;
                if(v1>=255)
                        v1=0;
                while (fila4[j]!='\0'){
                        letra=fila4[j];
                        txwr[v1]=(letra);
                        v1++;
                        if(v1>=255)
                        v1=0;
                        j++;
                }
                j=0;
        }
        txwr[v1]=(16);
        v1++;
        if(v1>=255)
                v1=0;
        if((teclado&0x1)==1){
                borrar_display();
                funcion=0;
                primera=1;
                teclado=0;}
        else if((teclado&0x2)!=0){
                borrar_display();
                funcion=1;
                primera=1;
                teclado=0;}
        else if((teclado&0x4)!=0){
                borrar_display();
                funcion=2;
                primera=1;
                teclado=0;
        }

}
else if (funcion==0){
        if (primera==1){
        primera=0;
        short j;
        char fila1[20]="";
        char fila2[20]="Modo joystick";
        char fila3[20]="Pulse 4 cambiar modo";
        char fila4[20]="";
        int letra;
        j=0;
        txwr[v1]=(0x13);
        v1++;
        if(v1>=255)
        v1=0;
        txwr[v1]=(0x02);
        v1++;
        if(v1>=255)
        v1=0;
        txwr[v1]=(0x01);
        v1++;
        if(v1>=255)
        v1=0;
        while (fila1[j]!='\0'){
                letra=fila2[j];
                txwr[v1]=(letra);
                v1++;
                if(v1>=255)
                v1=0;
                j++;
        }
        j=0;
        txwr[v1]=(0x02);
        v1++;
        if(v1>=255)
        v1=0;
        txwr[v1]=(0x15);
        v1++;
        if(v1>=255)
```

```
        v1=0;
        while (fila2[j]!='\0'){
                letra=fila2[j];
                txwr[v1]=(letra);
                v1++;
                if(v1>=255)
                v1=0;
                j++;
        }
        j=0;
        txwr[v1]=(0x02);
        v1++;
        if(v1>=255)
        v1=0;
        txwr[v1]=(0x29);
        v1++;
        if(v1>=255)
        v1=0;
        while (fila3[j]!='\0'){
                letra=fila3[j];
                txwr[v1]=(letra);
                v1++;
                if(v1>=255)
                v1=0;
                j++;
        }
        j=0;
        txwr[v1]=(0x02);
        v1++;
        if(v1>=255)
                v1=0;
        txwr[v1]=(0x3D);
        v1++;
        if(v1>=255)
                v1=0;
        while (fila4[j]!='\0'){
                letra=fila4[j];
                txwr[v1]=(letra);
                v1++;
                if(v1>=255)
                v1=0;
                j++;
        }
        j=0;
}

txwr[v1]=(16);
v1++;
if(v1>=255)
        v1=0;
if((teclado&0x8)!=0){
        borrar_display();
        funcion=4;
        primera=1;
        teclado=0;
}
}
else if (funcion==1){
        if (primera==1){
                primera=0;
                short j;
                char fila1[20]="";
                char fila2[20]="Modo soplido";
                char fila3[20]="Pulse 4 cambiar modo";
                char fila4[20]="";
                int letra;
                j=0;
                txwr[v1]=(0x13);
                v1++;
                if(v1>=255)
                        v1=0;
                txwr[v1]=(0x02);
                v1++;
                if(v1>=255)
                        v1=0;
                txwr[v1]=(0x01);
                v1++;
```

```
                if(v1>=255)
                        v1=0;

                while (fila1[j]!='\0'){
                        letra=fila2[j];
                        txwr[v1]=(letra);
                        v1++;
                        if(v1>=255)
                                v1=0;
                        j++;
                }
                j=0;
                txwr[v1]=(0x02);
                v1++;
                if(v1>=255)
                        v1=0;
                txwr[v1]=(0x15);
                v1++;
                if(v1>=255)
                        v1=0;
                while (fila2[j]!='\0'){
                        letra=fila2[j];
                        txwr[v1]=(letra);
                        v1++;
                        if(v1>=255)
                                v1=0;
                        j++;
                }
                j=0;
                txwr[v1]=(0x02);
                v1++;
                if(v1>=255)
                        v1=0;
                txwr[v1]=(0x29);
                v1++;
                if(v1>=255)
                        v1=0;
                while (fila3[j]!='\0'){
                        letra=fila3[j];
                        txwr[v1]=(letra);
                        v1++;
                        if(v1>=255)
                                v1=0;
                        j++;}
                j=0;
                txwr[v1]=(0x02);
                v1++;
                if(v1>=255)
                        v1=0;
                txwr[v1]=(0x3D);
                v1++;
                if(v1>=255)
                        v1=0;
                while (fila4[j]!='\0'){
                        letra=fila4[j];
                        txwr[v1]=(letra);
                        v1++;
                        if(v1>=255)
                                v1=0;
                        j++;
                }
                j=0;
        }
        txwr[v1]=(16);
        v1++;
        if(v1>=255)
                v1=0;

        if((teclado&0x8)!=0){
                borrar_display();
                funcion=4;
                primera=1;
                teclado=0;
        }
}
else if (funcion==2){
```

```
		if (primera==1){
			primera=0;
			short j;
			char fila2[20]="Modo PC";
			char fila3[20]="Pulse 4 cambiar modo";
			char fila4[20]="";
			int letra;
			j=0;
			txwr[v1]=(0x02);
			v1++;
			if(v1>=255)
				v1=0;
			txwr[v1]=(0x15);
			v1++;
			if(v1>=255)
				v1=0;
			while (fila2[j]!='\0'){
				letra=fila2[j];
				txwr[v1]=(letra);
				v1++;
				if(v1>=255)
					v1=0;
				j++;
			}
			j=0;
			txwr[v1]=(0x02);
			v1++;
			if(v1>=255)
				v1=0;
			txwr[v1]=(0x29);
			v1++;
			if(v1>=255)
				v1=0;
			while (fila3[j]!='\0'){
				letra=fila3[j];
				txwr[v1]=(letra);
				v1++;
				if(v1>=255)
					v1=0;
				j++;
			}
			j=0;
			txwr[v1]=(0x02);
			v1++;
			if(v1>=255)
				v1=0;
			txwr[v1]=(0x3D);
			v1++;
			if(v1>=255)
				v1=0;
			while (fila4[j]!='\0'){
				letra=fila4[j];
				txwr[v1]=(letra);
				v1++;
				if(v1>=255)
					v1=0;
				j++;
			}
			j=0;
		}
		txwr[v1]=(16);
		v1++;
		if(v1>=255)
			v1=0;

		if((teclado&0x8)!=0){
			borrar_display();
			funcion=4;
			primera=1;
			teclado=0;
		}
	}
}
)
```

```
#*******************************************************************************//
# NOMBRE: borrar_display                                                        //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Borrar el display                                                    //
# PARAMETROS RECIBIDOS: No tiene                                                //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//



void borrar_display(void){
        int n;
        txwr=Txbuf;
        txwr[v1]=0x0C;
        v1++;
        if(v1>=255)
        v1=0;
        for(n=0;n<=10000;n++);

}
```

```
#*****************************************************************************//
#*****************************************************************************//
# ARCHIVO: DRIVER.H                                                           //
# AUTOR: Juan Bellón                                                        //
# Tarjeta joystick                                                            //
#                                                                             //
#*****************************************************************************//
#*****************************************************************************//



void dir_io(int direccion);
void set_io(int pines);
void clear_io(int pines);

void cleanISR(void);

void init_timer(void);

void activoISR(void);
void desactivoISR(void);



#*****************************************************************************//
#*****************************************************************************//
# ARCHIVO: DRIVER.C                                                           //
# AUTOR: Juan Bellón                                                        //
# Tarjeta joystick                                                            //
#                                                                             //
#*****************************************************************************//
#*****************************************************************************//



#*****************************************************************************//
# NOMBRE: dir_io                                                              //
# AUTOR: Juan Bellón                                                        //
# FUNCION: Declarar los puertos como entradas                                 //
# PARAMETROS RECIBIDOS: Direccion de los puertos                            //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//



void dir_io(int direccion){
        IODIR0 = direccion;
}



#*****************************************************************************//
# NOMBRE: set_io                                                            //
# AUTOR: Juan Bellón                                                        //
# FUNCION: Poner a '1' los pines                                            //
# PARAMETROS RECIBIDOS: Pines a modificar                                   //
# PARAMETROS DEVUELTOS: No tiene                                            //
#*****************************************************************************//



void set_io(int pines){
        IOSET0 = pines;
}
```

```
#*****************************************************************************//
# NOMBRE: clear_io                                                            //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Poner a '0' los pines                                              //
# PARAMETROS RECIBIDOS: Pines a modificar                                     //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//


void clear_io(int pines){
       IOCLR0 = pines;
}


#*****************************************************************************//
# NOMBRE: init_timer                                                          //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Activar el timer                                                   //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//


void init_timer(void){
       VICIntEnable = 0x00000010;
}


#*****************************************************************************//
# NOMBRE: cleanISR                                                            //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Limpiar el flag de atención a la interrupción                      //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//


void cleanISR(void){
       T0IR = 1;
}


#*****************************************************************************//
# NOMBRE: activoISR                                                           //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Activar de las interrupciones                                      //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//


void activoISR(void){
       VICIntEnable = 0x00000010;
}


#*****************************************************************************//
# NOMBRE: desactivoISR                                                        //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Desactivar las interrupciones                                      //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//


void desactivoISR(void){
       VICIntEnable = 0x00000000;
}
```

VI.3.2. Codigos del Nodo de la tarjeta de los motores

En este subapartado se detallan los códigos de los archivos necesarios para trabajar la programación del nodo del joystick que son los siguientes:

1.- LPC_FullCAN_SW.c
2.- LPC_FullCAN_SW.h
3.- Driver.c
4.- Driver.h
5.- Setup.c
6.- Setup.h
7.- Serial.c
8.- Serial.h
9.- Funciones.c
10.- Funciones.h
11.- Main.c
12.- LPC21XX.h

Las arhivos LPC_FullCAN_SW son archivos originales, usados para la configuración de los dispositivos CAN y sus respectivas funciones de envio y recepción de datos.

El archivo main.c es el archivo principal en el que incluyendo los diferentes archivos .h conseguimos enlazarlo con las respectivas funciones de los archivos .c.

Los archivos driver.c y driver.h son archivos que nos facilitan la llamada a funciones frecuentemente usadas.

Los archivos setup.c y setup.h se encargan de la configuración general del sistema.

Las funciones serial.c y serial.h se encargan de la gestión del puerto serie para la conexión con la tarjeta de potencia Roboteq AX-3500.

Los archivos funciones.c y funciones.h se encargan de la gestión de las funciones principales de nuestro sistema.

A continuación se detallan los códigos de cada uno de los archivos creados, y las diversas funciones de cada uno de ellos.

```
#*******************************************************************************//
#*******************************************************************************//
# ARCHIVO: MAIN.C                                                               //
# AUTOR: Juan Bellón                                                          //
# Tarjeta de los motores                                                        //
#                                                                               //
#*******************************************************************************//
#*******************************************************************************//



#include <LPC21xx.H>
#include "LPC_FullCAN_SW.h"
#include "setup.h"
#include "driver.h"
#include "funciones.h"
#include "serial.h"



#*******************************************************************************//
# NOMBRE: main.c                                                                //
# AUTOR: Juan Bellón                                                            //
# FUNCIÓN: Ejecución global del nodo de los motores                             //
# PARAMETROS RECIBIDOS: No tiene                                                //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//



int main(void){


        dir_io(0x00000000);
        init_serial();
        config();
        encoderA=0;
        encoderB=0;

        while(1){

                /* T1 controla el sondeo del can 10mseg*/
                if(Tcan==1){
                        Timercan=100;
                        Active_can=1;
                        Tcan=0;
                        can();
                }
                /* Tbat controla el sondeo y el envio del nivel bat 2 seg*/
                if(Tbat==1){
                        Timerbat=20000;
                        Active_bat=1;
                        Tbat=0;
                        lee_battery();
                        desactivoISR();
                        envio_battery();
                        activoISR();
                }

                /* T1 controla lectura encoders y envio dato matores 100mseg*/
                if(T1==1){

                        Timer1=1000;
                        Active_T1=1;
                        T1=0;

                        lee_encoderA();
                        desactivoISR();
                        envio_canA();
                        activoISR();

                        lee_encoderB();
                        desactivoISR();
                        envio_canB();
                        activoISR();

                        if((modo==0) ||(modo==1))
```

```
                        envio_datos();
                    else if(modo==2)
                        envio_datospc();
                    else if(modo==4){
                        auxD=0;
                        auxI=0;
                        envio_datos();}
                }


        }
}
```

```
#*****************************************************************************//
#*****************************************************************************//
# ARCHIVO: FUNCIONES.C                                                        //
# AUTOR: Juan Bellón                                                          //
# Tarjeta de motores                                                          //
#                                                                             //
#*****************************************************************************//
#*****************************************************************************//



#*****************************************************************************//
# NOMBRE: envio_datospc                                                       //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Envio de datos a los motores procedentes de los motores            //
# PARAMETROS RECIBIDOS: datoI (0-7F), datoD (0,7F)                            //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//


void envio_datospc(void){
        int aux;
        if ((datoI==0)&&(datoD==0))
        {
                Active_freno=1;
                miputchar('!');
                miputchar('A');
                miputchar('0');
                miputchar('0');
                miputchar('\n');
                do{
                        while (!(U0LSR & 0x01));
                }
                while(U0RBR!=0x0D);

                miputchar('!');
                miputchar('B');
                miputchar('0');
                miputchar('0');
                miputchar('\n');
                do{
                        while (!(U0LSR & 0x01));
                }
                while(U0RBR!=0x0D);

                if(Tfreno==1){
                        Timer_freno=7500;
                        Active_freno=0;
                        Tfreno=0;
                        miputchar('!');
                        miputchar('c');
                        miputchar('\n');
                 }
        }
        else
        {
                Timer_freno=7500;
                Timer0=1000;
                Active_freno=0;
                if (datoD>=0)
                {
                        miputchar('!');
                        miputchar('C');
                        miputchar('\n');
                        miputchar('!');
                        miputchar('A');
                        puthex((datoD>>4)&0x0F);
                        puthex((datoD)&0x0F);
                        miputchar('\n');
                        do{
                        while (!(U0LSR & 0x01));
                        }
                        while(U0RBR!=0x0D);
                }
                else if(datoD<0)
                {
```

```
                        aux=datoD*-1;
                        miputchar('!');
                        miputchar('C');
                        miputchar('\n');
                        miputchar('!');
                        miputchar('a');
                        puthex((aux>>4)&0x0F);
                        puthex((aux)&0x0F);
                        miputchar('\n');
                        do{
                        while (!(U0LSR & 0x01));
                        }
                        while(U0RBR!=0x0D);
                }
                if (datoI>=0)
                {
                        miputchar('!');
                        miputchar('C');
                        miputchar('\n');
                        miputchar('!');
                        miputchar('B');
                        puthex((datoI>>4)&0x0F);
                        puthex((datoI)&0x0F);
                        miputchar('\n');
                        do{
                        while (!(U0LSR & 0x01));
                        }
                        while(U0RBR!=0x0D);
                }
                else if(datoI<0)
                {
                        aux=datoI*-1;
                        miputchar('!');
                        miputchar('C');
                        miputchar('\n');
                        miputchar('!');
                        miputchar('b');
                        puthex((aux>>4)&0x0F);
                        puthex((aux)&0x0F);
                        miputchar('\n');
                        do{
                        while (!(U0LSR & 0x01));
                        }
                        while(U0RBR!=0x0D);
                }

        }
}
```

```
#*********************************************************************************//
# NOMBRE: envio_datos                                                             //
# AUTOR: Juan Bellón                                                              //
# FUNCION:Envia los datos procedentes del joystick, con aceleración y sensores    //
# PARAMETROS RECIBIDOS: datoD (0-7F), datoI (0-7F)                                //
# PARAMETROS DEVUELTOS: No tiene                                                  //
#*********************************************************************************//



void envio_datos(void){

        int  aux;
        datoI=auxI;
        datoD=auxD;
        sensores();
        aceleracion();
        acelerometro();
        if ((realI==0)&&(realD==0))
        {
                Active_freno=1;
                miputchar('!');
                miputchar('A');
                miputchar('0');
                miputchar('0');
                miputchar('\n');
                do{
                        while (!(U0LSR & 0x01));
                }
                while(U0RBR!=0x0D);

                miputchar('!');
                miputchar('B');
                miputchar('0');
                miputchar('0');
                miputchar('\n');
                do{
                        while (!(U0LSR & 0x01));}
                while(U0RBR!=0x0D);

                if(Tfreno==1){

                        Timer_freno=7500;
                        Active_freno=0;
                        Tfreno=0;
                        miputchar('!');
                        miputchar('c');
                        miputchar('\n');

                 }


        }
        else
        {
                Timer_freno=7500;
                Active_freno=0;
                if (realD>0)
                {
                        miputchar('!');
                        miputchar('C');
                        miputchar('\n');
                        miputchar('!');
                        miputchar('A');
                        puthex((realD>>4)&0x0F);
                        puthex((realD)&0x0F);
                        miputchar('\n');
                        do{
                        while (!(U0LSR & 0x01));
                        }
                        while(U0RBR!=0x0D);
                }
                else if(realD<0)
                {
                        aux=realD*-1;
                        miputchar('!');
                        miputchar('C');
```

```
                        miputchar('\n');
                        miputchar('!');
                        miputchar('a');
                        puthex((aux>>4)&0x0F);
                        puthex((aux)&0x0F);
                        miputchar('\n');
                        do{
                        while (!(U0LSR & 0x01));
                        }
                        while(U0RBR!=0x0D);
                }
                else
                {
                        miputchar('!');
                        miputchar('a');
                        puthex((realD>>4)&0x0F);
                        puthex((realD)&0x0F);
                        miputchar('\n');
                        do{
                        while (!(U0LSR & 0x01));
                        }
                        while(U0RBR!=0x0D);
                }
                if (realI>0)
                {
                        miputchar('!');
                        miputchar('C');
                        miputchar('\n');
                        miputchar('!');
                        miputchar('B');
                        puthex((realI>>4)&0x0F);
                        puthex((realI)&0x0F);
                        miputchar('\n');
                        do{
                        while (!(U0LSR & 0x01));
                        }
                        while(U0RBR!=0x0D);
                }
                else if(realI<0)
                {
                        aux=realI*-1;
                        miputchar('!');
                        miputchar('C');
                        miputchar('\n');
                        miputchar('!');
                        miputchar('b');
                        puthex((aux>>4)&0x0F);
                        puthex((aux)&0x0F);
                        miputchar('\n');
                        do{
                        while (!(U0LSR & 0x01));
                        }
                        while(U0RBR!=0x0D);
                }
                else
                {
                        miputchar('!');
                        miputchar('b');
                        puthex((realI>>4)&0x0F);
                        puthex((realI)&0x0F);
                        miputchar('\n');
                        do{
                        while (!(U0LSR & 0x01));
                        }
                        while(U0RBR!=0x0D);
                }

        }
}
```

```
#*********************************************************************************//
# NOMBRE: aceleracion                                                             //
# AUTOR: Juan Bellón                                                              //
# FUNCION: Produce una aceleración por software a la silla                        //
# PARAMETROS RECIBIDOS: datoD (0-7F), datoI (0-7F)                                //
# PARAMETROS DEVUELTOS: realD (0-7F), realI (0-7F)                                //
#*********************************************************************************//



void aceleracion (void){

        int  incrementoI, incrementoD;
        incrementoI= 7;   // Incremento aceleracion
        incrementoD= 7;   // Incremento aceleracion

        // Aceleración del motor derecho
        if (datoD>0)
        {
                if (datoD>realD)
                        realD+=incrementoD;
                else if (datoD<realD)
                        realD-=incrementoD;
                if(realD>=0x7F)
                        realD=0x7F;

        }
        else if (datoD<0)
        {
                if (datoD<realD)
                        realD-=incrementoD;
                else if (datoD>realD)
                        realD+=incrementoD;
                if(realD<=(0x7F*-1))
                        realD=(0x7F*-1);

        }
        else if (datoD==0)
        {
                if (datoD<realD)
                        realD-=incrementoD;
                else if (datoD>realD)
                        realD+=incrementoD;
                if(((realD<=7)&&(realD>=0))||((realD>=-7)&&(realD<=0)))
                        realD=0;
        }
        // Aceleración del motor izquierdo
        if (datoI>0)
        {
                if (datoI>realI)
                        realI+=incrementoI;
                else if (datoI<realI)
                        realI-=incrementoI;
                if(realI>=0x7F)
                        realI=0x7F;
        }
        else if(datoI<0)
        {
                if (datoI<realI)
                        realI-=incrementoI;
                else if (datoI>realI)
                        realI+=incrementoI;
                if(realI<=(0x7F*-1))
                        realI=(0x7F*-1);
        }
        else if (datoI==0)
        {
                if (datoI<realI)
                        realI-=incrementoI;
                else if (datoI>realI)
                        realI+=incrementoI;
                if(((realI<=7)&&(realI>=0))||((realI>=-7)&&(realI<=0)))
                        realI=0;
        }


}
```

```
#*******************************************************************************//
# NOMBRE: sensores                                                              //
# AUTOR: Juan Bellón                                                            //
# FUNCION: Modificar la velocidad de la silla en función de los datos de los sensores //
# PARAMETROS RECIBIDOS: SDI: Sensor delantero izquierdo                         //
#                       SDD: Sensor delantero derecho                           //
#                       SJO: Sensor delantero joystick                          //
#                       STI: Sensor trasero izquierdo                           //
#                       STD: Sensor trasero derecho                             //
# PARAMETROS DEVUELTOS: datoD (0-7F), datoI (0-7F)                              //
#*******************************************************************************//


void sensores (void){
        // Sensor delantero izquierdo SDI
        if((SDI<=300)&&(SDI>200)&&(datoD>0)&&(datoI>0)){
                if (datoD>90)
                        datoD=(90);
                if (datoI>90)
                        datoI=(90);
        }
        else if ((SDI<=200)&&(SDI>100)&&(datoD>0)&&(datoI>0)){
                if (datoD>70)
                        datoD=(70);
                if (datoI>70)
                        datoI=(70);
        }
        else if((SDI<=100)&&(SDI>40)&&(datoD>0)&&(datoI>0)){
                if (datoD>40)
                        datoD=(40);
                if (datoI>40)
                        datoI=(40);
        }
        else if((SDI<=40)&&(SDI>5)&&(datoD>0)&&(datoI>0)){
                datoD=(0);
                datoI=(0);
        }
        // Sensor delantero derecho SDD
        if((SDD<=300)&&(SDD>200)&&(datoD>0)&&(datoI>0)){
                if (datoD>90)
                        datoD=(90);
                if (datoI>90)
                        datoI=(90);
        }
        else if ((SDD<=200)&&(SDD>100)&&(datoD>0)&&(datoI>0)){
                if (datoD>70)
                        datoD=(70);
                if (datoI>70)
                        datoI=(70);
        }
        else if((SDD<=100)&&(SDD>40)&&(datoD>0)&&(datoI>0)){
                if (datoD>40)
                        datoD=(40);
                if (datoI>40)
                        datoI=(40);
        }
        else if((SDD<=40)&&(SDD>5)&&(datoD>0)&&(datoI>0)){
                datoD=(0);
                        datoI=(0);
        }
        // Sensor delantero joystick SJO
        if((SJO<=300)&&(SJO>200)&&(datoD>0)&&(datoI>0)){
                if (datoD>90)
                        datoD=(90);
                if (datoI>90)
                datoI=(90);
        }
        else if ((SJO<=200)&&(SJO>100)&&(datoD>0)&&(datoI>0)){
                if (datoD>70)
                        datoD=(70);
                if (datoI>70)
                datoI=(70);
        }
```

```
        else if((SJO<=100)&&(SJO>40)&&(datoD>0)&&(datoI>0)){
                if (datoD>40)
                        datoD=(40);
                if (datoI>40)
                        datoI=(40);
        }
        else if((SJO<=40)&&(SJO>5)&&(datoD>0)&&(datoI>0)){
                datoD=(0);
                datoI=(0);
        }

        // Sensor trasero izquierdo STI
        if((STI<=300)&&(STI>200)&&(datoD<0)&&(datoI<0)){
                if (datoD<(-90))
                        datoD=(-90);
                if (datoI<(-90))
                        datoI=(-90);
        }
        else if ((STI<=200)&&(STI>100)&&(datoD<0)&&(datoI<0)){
                if (datoD<(-60))
                        datoD=(-60);
                if (datoI<(-60))
                        datoI=(-60);
        }
        else if((STI<=100)&&(STI>40)&&(datoD<0)&&(datoI<0)){
                if (datoD<(-35))
                        datoD=(-35);
                if (datoI<(-35))
                        datoI=(-35);
        }
        else if((STI<=40)&&(STI>5)&&(datoD<0)&&(datoI<0)){
                datoD=(0);
                datoI=(0);
        }

        // Sensor trasero derecho STD
        if((STD<=300)&&(STD>200)&&(datoD<0)&&(datoI<0)){
                if (datoD<(-90))
                        datoD=(-90);
                if (datoI<(-90))
                        datoI=(-90);
        }
        else if ((STD<=200)&&(STD>100)&&(datoD<0)&&(datoI<0)){
                if (datoD<(-60))
                        datoD=(-60);
                if (datoI<(-60))
                        datoI=(-60);
        }
        else if((STD<=100)&&(STD>40)&&(datoD<0)&&(datoI<0)){
                if (datoD<(-35))
                        datoD=(-35);
                if (datoI<(-35))
                        datoI=(-35);
        }
        else if((STD<=40)&&(STD>5)&&(datoD<0)&&(datoI<0)){
                datoD=(0);
                datoI=(0);
        }
}
```

```
#*****************************************************************************//
# NOMBRE: acelerometro                                                        //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Parar la silla si existe algún impacto                             //
# PARAMETROS RECIBIDOS: EJEX (0 - 800), EJEY (0 - 800)                        //
# PARAMETROS DEVUELTOS: alarma = 0 => PARO; alarma =1 => MARCHA               //
#*****************************************************************************//


void acelerometro(void){


        if ((EJEX<=400)||(EJEX>=700)){
                alarma=1;
        }
        if ((EJEY<=400)||(EJEY>=700)){
                alarma=1;
        }
        if (alarma==1){

                        realD=0;
                        realI=0;
                        datoD=0;
                        datoI=0;
                        if ((auxD==0)&&(auxI==0))
                                alarma=0;
        }

}



*****************************************************************************//
# NOMBRE: envio_battery                                                       //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Enviar el nivel de carga de batería                                //
# PARAMETROS RECIBIDOS: nivel_battery (0V - 26V)                              //
# PARAMETROS DEVUELTOS: Mensaje con el nivel de batería                       //
#*****************************************************************************//


void envio_battery(void){

        FULLCAN_MSG MsgBuf;
        MsgBuf.Dat1 = 0x00080210L;
        MsgBuf.DatA = nivel_battery;
        MsgBuf.DatB = 0;

        // Transmisión del mensaje
        FullCAN_PushMessage(1,&MsgBuf);

}



#*****************************************************************************//
# NOMBRE: envio_CanA                                                          //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Enviar por el bus CAN los datos del encoder A                      //
# PARAMETROS RECIBIDOS: Datos absolutos del encoder A                         //
# PARAMETROS DEVUELTOS: Mensaje con los datos del encoder A                   //
#*****************************************************************************//



void envio_canA(void){


        // Envío  datos del encoder A
        FULLCAN_MSG MsgBuf;
        MsgBuf.Dat1 = 0x00080101L;
        MsgBuf.DatA = encoderAABS;
        MsgBuf.DatB = cuenta;

        FullCAN_PushMessage(1,&MsgBuf);
}
```

```
#*******************************************************************************//
# NOMBRE: envio_CanB                                                            //
# AUTOR: Juan Bellón                                                            //
# FUNCION: Enviar por el bus Can los datos del encoder B                        //
# PARAMETROS RECIBIDOS: Datos absolutos del encoder B                           //
# PARAMETROS DEVUELTOS: Mensaje con los datos del encoder B                     //
#*******************************************************************************//


void envio_canB(void){


        // Envío datos del encoder B
        FULLCAN_MSG MsgBuf;
        MsgBuf.Dat1 = 0x00080102L;
        MsgBuf.DatA = encoderBABS;
        MsgBuf.DatB = cuenta;

        FullCAN_PushMessage(1,&MsgBuf);
}
```

```
#*******************************************************************************//
# NOMBRE: lee_encoderA                                                          //
# AUTOR: Juan Bellón                                                            //
# FUNCION: Leer datos del encoder A                                             //
# PARAMETROS RECIBIDOS: No tiene                                                //
# PARAMETROS DEVUELTOS: Datos del encoder A                                     //
#*******************************************************************************//


void lee_encoderA(void)
{
        encoderA=0;
        while (U0LSR & 0x01)
        eco=U0RBR;
        miputchar('?');
        miputchar('q');
        miputchar('4');
        miputchar('\n');
        x=-1;

        do
        {
                while (!(U0LSR & 0x01));
                if (U0RBR!=0x0D)
                {
                if (x==-1)
                {
                        x=(U0RBR-0x30);
                        if(x>9) x=x-7;
                        if (x<8) encoderA=x;
                        else (encoderA= 0xFFFFFFF0 | x);
                }
                else{
                        x=(U0RBR-0x30);
                        if(x>9) x=x-7;
                        encoderA=(encoderA<<4)|x;
                }
                }
        }
        while(U0RBR!=0x0D);
        encoderAABS+=encoderA;
}
```

```
#*****************************************************************************//
# NOMBRE: lee_encoderB                                                        //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Leer datos del encoder B                                           //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: Datos del encoder B                                   //
#*****************************************************************************//



void lee_encoderB(void)
{
        encoderB=0;
        while (U0LSR & 0x01)
        eco=U0RBR;
        miputchar('?');
        miputchar('q');
        miputchar('5');
        miputchar('\n');
        y=-1;
        do
        {
                while (!(U0LSR & 0x01));
                if (U0RBR!=0x0D)
                {
                if (y==-1)
                {
                        y=(U0RBR-0x30);
                        if(y>9) y=y-7;
                        if (y<8) encoderB=y;
                        else (encoderB= 0xFFFFFFF0 | y);
                }
                else{
                        y=(U0RBR-0x30);
                        if(y>9) y=y-7;
                        encoderB=(encoderB<<4)|y;
                }
                }
        }
        while(U0RBR!=0x0D);
    encoderBABS+=encoderB;
}



#*****************************************************************************//
# NOMBRE: lee_battery                                                         //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Realizar la lectura del estado de carga de la bateria              //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: Carga de la batería (0V - 26V)                        //
#*****************************************************************************//



void lee_battery(void)
{
        short z;
        nivel_battery=0;
        nivel_micro=0;

        while (U0LSR & 0x01)
        eco=U0RBR;
        miputchar('?');
        miputchar('E');
        miputchar('\n');

        z=-1;
        do{

                while (!(U0LSR & 0x01));

                        if (z==-1)
                        {
                                nivel_battery=((U0RBR-0x30))*0x10;
                                z=0;
                        }
```

```
                        else if (z==0){
                                nivel_battery+=(U0RBR-0x30);
                                z=1;
                        }
                        else if (z==1){
                                eco=U0RBR;
                                z=2;
                        }

                        else if (z==2){

                                nivel_micro=((U0RBR-0x30))*0x10;
                                z=3;
                        }
                        else if (z==3){
                                nivel_micro+=(U0RBR-0x30);
                                z=4;
                        }
                        else if (z==4){
                                eco=U0RBR;
                                z=5;
                        }
                }while(z!=5);

        nivel_battery=((nivel_battery)*550/256);
        nivel_micro=((nivel_micro)*285/256);
}
```

```
#*******************************************************************************//
#*******************************************************************************//
# ARCHIVO: SETUP.H                                                              //
# AUTOR: Juan Bellón                                                            //
# Tarjeta de los motores                                                        //
#                                                                               //
#*******************************************************************************//
#*******************************************************************************//



unsigned int volatile gTimerTick;
unsigned int volatile cuenta;
unsigned int Active_can, Timercan, Tcan;
unsigned int Active_bat, Timerbat, Tbat;
unsigned int Active_T0, Timer0, T0;
unsigned int Active_T1, Timer1, T1;
unsigned int Active_freno, Timer_freno, Tfreno;

void config(void);



#*******************************************************************************//
#*******************************************************************************//
# ARCHIVO: SETUP.C                                                              //
# AUTOR: Juan Bellón                                                            //
# Tarjeta de los motores                                                        //
#                                                                               //
#*******************************************************************************//
#*******************************************************************************//



#*******************************************************************************//
# NOMBRE: config                                                                //
# AUTOR: Juan Bellón                                                            //
# FUNCION: Configuración del sistema                                            //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: Timercan = 100; Timerbat = 20000; Timer1 = 1000       //
#                       Timer_freno = 7500;                                     //
#                       Active_can = 1; Active_bat = 1; Active_T1 = 1           //
#                       Active_freno = 0                                      //
#                       Tcan = 0; Tbat = 0; T1 = 0; Tfreno = 0;                 //
#*******************************************************************************//



void config(void){

        cuenta=0;

        Timercan=100;
        Active_can=1;
        Tcan=0;

        Timerbat=20000;
        Active_bat=1;
        Tbat=0;

        Timer1=1000;
        Active_T1=1;
        T1=0;

        Timer_freno=7500;
        Active_freno=0;
        Tfreno=0;

        VPBDIV = 1;
        IODIR0 = 0x00FF0000; // Puertos P1.16..23 defindos como salidas
        VICIntEnClr = 0xFFFFFFFFL;
        VICIntSelect = 0x00000000L;
        VICDefVectAddr = (unsigned long) DefaultISR;
        // Inicialización del bus CAN
        FullCAN_Init(1,0,CANBitrate001m_12MHz);
        FullCAN_SetErrIRQ(2);
        T0MR0 = 5999; // 100 microsegundos = 6.000-1 cuentas
```

```
        T0MCR = 3;
        T0TCR = 1;

        VICVectAddr3 = (unsigned long) Timer0ISR;
        VICVectCntl3 = 0x20 | 4;
        VICIntEnable = 0x00000010;

        // Filtros para la recepción de mensajes por el bus CAN
        FullCAN_SetFilter(1,0x110);
        FullCAN_SetFilter(1,0x111);
        FullCAN_SetFilter(1,0x120);
        FullCAN_SetFilter(1,0x201);
        FullCAN_SetFilter(1,0x202);
        FullCAN_SetFilter(1,0x203);

        // Espera de 10 milisegundos para una correcta inicialización
        SlowDown(100);
}



#*****************************************************************************//
# NOMBRE: Timer0ISR                                                           //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Control de los timers                                              //
# PARAMETROS RECIBIDOS: Timercan, Timerbat, Timer1, Timer_freno              //
#                       Active_can, Active_bat, Active_T1, Active_freno      //
# PARAMETROS DEVUELTOS: Tcan = 1, Active_can = 0;                             //
#                       Tbat = 1, Active_bat = 0;                             //
#                       T1 = 1, Active_T1 = 0;                                //
#                       Tfreno = 1, Active_freno = 0;                         //
#*****************************************************************************//



void Timer0ISR (void)
{

        cuenta++;
        gTimerTick++;
        cleanISR();//

        if (Active_can)
                Timercan--;
        if (Timercan==0){
                Tcan=1;
                Active_can=0;
        }
        if (Active_bat)
                Timerbat--;
        if (Timerbat==0){
                Tbat=1;
                Active_bat=0;
        }
        if (Active_T1)
                Timer1--;
        if (Timer1==0){
                T1=1;
                Active_T1=0;
        }
        if (Active_freno)
                Timer_freno--;
        if (Timer_freno==0){
                Tfreno=1;
                Active_freno=0;
        }

        VICVectAddr = 0xFFFFFFFF;
}
```

```
#*****************************************************************************//
#*****************************************************************************//
# ARCHIVO: SERIAL.H                                                           //
# AUTOR: Juan Bellón                                                          //
# Tarjeta de los motores                                                      //
#                                                                             //
#*****************************************************************************//
#*****************************************************************************//


void init_serial (void);
int miputchar (int ch);
char migetchar (void);
void puthex (int hex);


#*****************************************************************************//
#*****************************************************************************//
# ARCHIVO: SERIAL.C                                                           //
# AUTOR: Juan Bellón                                                          //
# Tarjeta de los motores                                                      //
#                                                                             //
#*****************************************************************************//
#*****************************************************************************//


#*****************************************************************************//
# NOMBRE: init_serial                                                         //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Inicializar el puerto serie                                        //
# PARAMETROS RECIBIDOS: No tiene                                             //
# PARAMETROS DEVUELTOS: No tiene                                             //
#*****************************************************************************//


void init_serial (void)  {   // Inicializacion del bus serie

        PINSEL0 = 0x00000005; // Activación de RxD1 y TxD1
        U0LCR = 0x8A;  // 8 bits, sin pariad, un bit de parada
        U0DLL = 0x84;
        U0DLM = 0x1;   // 9600 Baudios,reloj VPB de 15MHz
        U0LCR = 0x1A;
}


#*****************************************************************************//
# NOMBRE: miputchar                                                           //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Escribir datos en el puerto serie                                  //
# PARAMETROS RECIBIDOS: Datos a escribir                                     //
# PARAMETROS DEVUELTOS: No tiene                                             //
#*****************************************************************************//


int miputchar (int ch)  {     // Escribir caracteres en el puerto serie

        if (ch == '\n')  {
                while (!(U0LSR & 0x20));
                U0THR = CR;                         /* output CR */
                while (!(U0LSR & 0x01));
                return U0RBR;
        }

        while (!(U0LSR & 0x20));
        U0THR = ch;
        while (!(U0LSR & 0x01));
        return U0RBR;
}
```

```
#*******************************************************************************//
# NOMBRE: migtchar                                                              //
# AUTOR: Juan Bellón                                                            //
# FUNCION:Leer datos del puerto serie                                           //
# PARAMETROS RECIBIDOS: No tiene                                               //
# PARAMETROS DEVUELTOS: Datos leidos                                           //
#*******************************************************************************//


char migetchar (void)  {      // Leer caracteres del puerto serie
       while (!(U0LSR & 0x01));
       return (U0RBR);
}




#*******************************************************************************//
# NOMBRE: puthex                                                                //
# AUTOR: Juan Bellón                                                            //
# FUNCION:escribir datos Hexadecimales en el puerto serie                       //
# PARAMETROS RECIBIDOS: Datos a enviar                                         //
# PARAMETROS DEVUELTOS: No tiene                                               //
#*******************************************************************************//


void puthex (int hex){ // Escribie carácteres hexadecimals en el puerto serie

       if (hex > 9) miputchar('A' + (hex - 10));
       else            miputchar ('0' + hex);
}
```

```
#*****************************************************************************//
#*****************************************************************************//
# ARCHIVO: DRIVER.H                                                           //
# AUTOR: Juan Bellón                                                          //
# Tarjeta de motores                                                          //
#                                                                             //
#*****************************************************************************//
#*****************************************************************************//



void dir_io(int direccion);
void set_io(int pines);
void clear_io(int pines);

void cleanISR(void);

void init_timer(void);

void activoISR(void);
void desactivoISR(void);



#*****************************************************************************//
#*****************************************************************************//
# ARCHIVO: DRIVER.C                                                           //
# AUTOR: Juan Bellón                                                          //
# Tarjeta de motores                                                          //
#                                                                             //
#*****************************************************************************//
#*****************************************************************************//


#*****************************************************************************//
# NOMBRE: dir_io                                                              //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Declarar los puertos como entradas                                 //
# PARAMETROS RECIBIDOS: Direccion de los puertos                          //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//


void dir_io(int direccion){
        IODIR0 = direccion;
}



#*****************************************************************************//
# NOMBRE: set_io                                                              //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Poner a '1' los pines                                              //
# PARAMETROS RECIBIDOS:   Pines a modificar                               //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//



void set_io(int pines){
        IOSET0 = pines;
}



#*****************************************************************************//
# NOMBRE: clear_io                                                            //
# AUTOR: Juan Bellón                                                          //
# FUNCION: Poner a '0' los pines                                              //
# PARAMETROS RECIBIDOS:   Pines a modificar                               //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//



void clear_io(int pines){
        IOCLR0 = pines;}
```

```
#*******************************************************************************//
# NOMBRE: init_timer                                                            //
# AUTOR: Juan Bellón                                                            //
# FUNCION: Activar el timer                                                     //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//


void init_timer(void){
	VICIntEnable = 0x00000010;
}


#*******************************************************************************//
# NOMBRE: cleanISR                                                              //
# AUTOR: Juan Bellón                                                            //
# FUNCION: Limpiar el flag de atención a la interrupción                        //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//


void cleanISR(void){
	T0IR = 1;
}


#*******************************************************************************//
# NOMBRE: activoISR                                                             //
# AUTOR: Juan Bellón                                                            //
# FUNCION: Activar de las interrupciones                                        //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//


void activoISR(void){
	VICIntEnable = 0x00000010;
}


#*******************************************************************************//
# NOMBRE: desactivoISR                                                          //
# AUTOR: Juan Bellón                                                            //
# FUNCION: Desactivar las interrupciones                                        //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//


void desactivoISR(void){
	VICIntEnable = 0x00000000;
}
```

VI.3.3. Codigos del Nodo de la tarjeta de los sensores

En este subapartado se detallan los códigos de los archivos necesarios para trabajar la programación del nodo del joystick que son los siguientes:

1.- LPC_FullCAN_SW.c
2.- LPC_FullCAN_SW.h
3.- Driver.c
4.- Driver.h
5.- Setup.c
6.- Setup.h
7.- Can.c
8.- Can.h
9.- Funciones_i2c.c
10.- Funciones_i2c.h
11.- Main.c
12.- LPC21XX.h

Las arhivos LPC_FullCAN_SW son archivos originales, usados para la configuración de los dispositivos CAN y sus respectivas funciones de envio y recepción de datos.

El archivo main.c es el archivo principal en el que incluyendo los diferentes archivos .h conseguimos enlazarlo con las respectivas funciones de los archivos .c.

Los archivos driver.c y driver.h son archivos que nos facilitan la llamada a funciones frecuentemente usadas.

Los archivos setup.c y setup.h se encargan de la configuración general del sistema.

Las funciones can.c y can.h se encargan de la gestión del CAN, funciones de envio de datos por el bus CAN.

Los archivos funciones_i2c.c y funciones_i2c.h se encargan de la gestión de las funciones de lectura y escritura de datos en los sensores de ultrasonidos.

A continuación se detallan los códigos de cada uno de los archivos creados, y las diversas funciones de cada uno de ellos.

```
#*******************************************************************************//
#*******************************************************************************//
# ARCHIVO: MAIN.C                                                                //
# AUTOR: Jose Basterrechea                                                      //
# Tarjeta de sensores                                                            //
#                                                                                //
#*******************************************************************************//
#*******************************************************************************//



#include <LPC21xx.H>
#include "LPC_FullCAN_SW.h"
#include "setup.h"
#include "driver.h"
#include "funciones_i2c.h"
#include "can.h"



#*******************************************************************************//
# NOMBRE: main.c                                                                 //
# AUTOR: Jose Basterrechea                                                      //
# FUNCIÓN: Ejecución global del nodo de los motores                              //
# PARAMETROS RECIBIDOS: No tiene                                                 //
# PARAMETROS DEVUELTOS: No tiene                                                 //
#*******************************************************************************//



unsigned char ultra[]={0xE6,0xE2,0xE4,0xE6,0xE8,0xEA,0xEC,0xEE,0xF0};
int i=0,k=0;
unsigned int medida[9];
int main(void){
        int j=0,aux1=0,aux2=0;
        int ejex,ejey,ejez;
        for (i=0;i<10;i++)
                medida[i]=0;
        i=0;
        int mensaje1=0,mensaje2=0,etiqueta;
        config();
        config_I2C();

        while(1){

                if(T1==1)
                {
                        if (j==1){
                                for(k=0;k<4;k++){
                                aux1=leo_I2C(ultra[k],0x02);
                                aux2=leo_I2C(ultra[k],0x03);
                                medida[k]=((aux1<<8)|aux2);
                                }
                                for(i=4;i<9;i++){
                                        i2c_send(ultra[i], 0x51);
                                }
                        mensaje2=(((medida[0]<<16)&0xFFFF0000)|(medida[1]&0x0000FFFF));
                        mensaje1=(((medida[2]<<16)&0xFFFF0000)|(medida[3]&0x0000FFFF));
                        etiqueta=0x00080202L;
                        envio_can(mensaje1,mensaje2,etiqueta);
                        j=0;
                        }
                        else if (j==0){
                                ejex=adquiero_acel(0x012E0401);
                                ejey=adquiero_acel(0x012E0402);
                                ejez=adquiero_acel(0x012E0404);
                                mensaje2=(((ejex<<16)&0xFFFF0000)|(ejey&0xFFFF));
                                mensaje1=(((ejez<<16)&0xFFFF0000)|(medida[8]&0xFFFF));
                                etiqueta=0x00080203L;
                                envio_can(mensaje1,mensaje2,etiqueta);
                                for(k=4;k<9;k++){
                                        aux1=leo_I2C(ultra[k],0x02);
                                        aux2=leo_I2C(ultra[k],0x03);
                                        medida[k]=((aux1<<8)|aux2);
                                }
                                for(i=0;i<4;i++){
                                        i2c_send(ultra[i], 0x51);
```

```
                        }
                mensaje2=(((medida[4]<<16)&0xFFFF0000)|(medida[5]&0x0000FFFF));
                mensaje1=(((medida[6]<<16)&0xFFFF0000)|(medida[7]&0x0000FFFF));
                        etiqueta=0x00080201L;
                        envio_can(mensaje1,mensaje2,etiqueta);
                        j=1;
                }
                Timer1=700;
                Active_T1=1;
                T1=0;
                }
        }
        return 0;
}
```

```
#*******************************************************************************//
#*******************************************************************************//
# ARCHIVO: SETUP.C                                                              //
# AUTOR: Jose Basterrechea                                                      //
# Tarjeta de  sensores                                                          //
#                                                                               //
#*******************************************************************************//
#*******************************************************************************//



#*******************************************************************************//
# NOMBRE: config                                                                //
# AUTOR: Jose Basterrechea                                                    //
# FUNCION: Configuración del sistema                                            //
# PARAMETROS RECIBIDOS: No tiene                                              //
# PARAMETROS DEVUELTOS: T0 = 2000; Timer1 = 700                               //
#                       Active_T0 = 1; Active_T1 = 1                            //
#                       T0 = 0; T1 = 0;                                         //
#*******************************************************************************//



void config(void){

        Timer0=2000;
        Active_T0=1;
        T0=0;

        Timer1=700;
        Active_T1=1;
        T1=0;


        VPBDIV = 1;

        VICIntEnClr = 0xFFFFFFFFL;
        VICIntSelect = 0x00000000L;

        VICDefVectAddr = (unsigned long) DefaultISR;

        // Inicialización del bus CAN

        FullCAN_Init(1,0,CANBitrate001m_12MHz);
        FullCAN_SetErrIRQ(2);

        T0MR0 = 5999; // 100 microsegundos = 6.000-1 cuentas
        T0MCR = 3;
        T0TCR = 1;
        VICVectAddr3 = (unsigned long) Timer0ISR;
        VICVectCntl3 = 0x20 | 4;
        VICIntEnable = 0x00000010;

        // Espera de 10 milisegundos para una correcta inicialización
        SlowDown(100);
}
```

```
#*****************************************************************************//
# NOMBRE: Timer0ISR                                                            //
# AUTOR: Jose Basterrechea                                                    //
# FUNCION: Control de los timers                                               //
# PARAMETROS RECIBIDOS: Timer0, Timer1                                        //
#                       Active_T0, Active_T1                                  //
# PARAMETROS DEVUELTOS: T0 = 1, Active_T0 = 0;                                //
#                       T1 = 1, Active_T1 = 0;                                //
#*****************************************************************************//


void Timer0ISR (void)
{

        gTimerTick++;
        cleanISR();//

        if (Active_T0)
                Timer0--;
        if (Timer0==0){
                T0=1;
                Active_T0=0;
        }

        if (Active_T1)
                Timer1--;
        if (Timer1==0){
                T1=1;
                Active_T1=0;
        }

        VICVectAddr = 0xFFFFFFFF;

}
```

```
#*******************************************************************************//
#*******************************************************************************//
# ARCHIVO: CAN.C                                                                //
# AUTOR: Jose Basterrechea                                                      //
# Tarjeta de sensores                                                           //
#                                                                               //
#*******************************************************************************//
#*******************************************************************************//



#*******************************************************************************//
# NOMBRE: envio_can                                                             //
# AUTOR: Jose Basterrechea                                                    //
# FUNCION: Enviar por el bus CAN los mensajes creados                           //
# PARAMETROS RECIBIDOS: datoA (distancia en cm ó acelerometro),               //
#                       datoB (distancia en cm ó acelerometro), etiqueta      //
# PARAMETROS DEVUELTOS: Mensaje con los datos                                  //
#*******************************************************************************//



void envio_can(int datoA, int datoB,int etiqueta){

        FULLCAN_MSG MsgBuf;
        MsgBuf.Dat1 = etiqueta;
        MsgBuf.DatA = datoA; //
        MsgBuf.DatB = datoB; //

        // Transmision del mensaje
        FullCAN_PushMessage(1,&MsgBuf);

}
```

```
#*****************************************************************************//
#*****************************************************************************//
# ARCHIVO: FUNCIONES_I2C.C                                                    //
# AUTOR: Jose Basterrechea                                                    //
# Tarjeta de sensores                                                         //
#                                                                             //
#*****************************************************************************//
#*****************************************************************************//



#*****************************************************************************//
# NOMBRE: config_I2C                                                          //
# AUTOR: Jose Basterrechea                                                  //
# FUNCION: Configurar el bus I2C                                              //
# PARAMETROS RECIBIDOS: No tiene                                             //
# PARAMETROS DEVUELTOS: No tiene                                             //
#*****************************************************************************//


void config_I2C(void){        // Configuración del bus I2C

        // Activación de los pines del bus I2C
        PINSEL0 &= 0xffffff0f;
        PINSEL0 |= 0x00000050;

        I2SCLH=255;
        I2SCLL=255;


        I2CONSET=0x40;
        I2CONCLR = 0x28;
}



#*****************************************************************************//
# NOMBRE: adquiero_acel                                                       //
# AUTOR: Jose Basterrechea                                                  //
# FUNCION: Adquirir los datos del acelerometro                                //
# PARAMETROS RECIBIDOS: No tiene                                             //
# PARAMETROS DEVUELTOS: valor2 (0 - 800)                                     //
#*****************************************************************************//



int adquiero_acel(int adcr){
        int valor2;
        ADCR = adcr;   // Comienzo de la adquisición del acelerometro
        do
        {
                valor2 = ADDR;
        } while ((valor2 & 0x80000000) == 0);        // Espera fin de la adquisición

        ADCR &= ~0x01000000;
        valor2 = (valor2 >> 6) & 0x03FF;     // Obetención del dato

        return valor2;


}
```

```
#*******************************************************************************//
# NOMBRE: i2c_send                                                              //
# AUTOR: Jose Basterrechea                                                      //
# FUNCION: Enviar los datos a los sensores                                      //
# PARAMETROS RECIBIDOS: addr = Dirección del sensor; data = Tipo de medida      //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//



void i2c_send(unsigned char addr, unsigned char data)
{

        I2CONSET = 0x20;
        while(I2STAT != 0x08); // START, comienzo de la transmisión
        I2DAT = addr;
        I2CONCLR = 0x28;
        while(I2STAT != 0x18); // Espera : SLA+W transmitido; ACK recibido
        I2DAT = 0x00;
        I2CONCLR = 0x08;
        while(I2STAT != 0x28); // Espera : Dato byte en I2DAT
        //transmitido; ACK recibido
        I2DAT = data;
        I2CONCLR = 0x08;
        while(I2STAT != 0x28); // Espera : Dato byte en I2DAT
        //transmitido; ACK recibido
        I2CONSET = 0x10;
        I2CONCLR = 0x08;

}



#*******************************************************************************//
# NOMBRE: leo_I2C                                                               //
# AUTOR: Jose Basterrechea                                                      //
# FUNCION: Leer los datos de los sensores                                       //
# PARAMETROS RECIBIDOS: addr = dirección del sensor; reg = registro a leer      //
# PARAMETROS DEVUELTOS: i = medida realizada                                    //
#*******************************************************************************//



int leo_I2C(unsigned char addr,unsigned char reg)
{
        int i;

        I2CONSET = 0x20;
        while(I2STAT != 0x08); // Espera : START transmitido
        I2DAT = addr;
        I2CONCLR = 0x28;
        while(I2STAT != 0x18); // Espera : SLA+W transmitido; ACK recibido
        I2DAT = reg;
        I2CONCLR = 0x08;
        while(I2STAT != 0x28); // Espera : Dato byte en I2DAT
        //transmitido; ACK recebido
        I2CONSET = 0x20;
        I2CONCLR = 0x08;
        while(I2STAT != 0x10); // Espera : Repetición de START transmitido
        I2DAT = (addr|0x01);
        I2CONCLR = 0x28;
        while(I2STAT != 0x40); // Espera : SLA+R transmitido; ACK recibido
        I2CONCLR = 0x0C;
        while(I2STAT != 0x58); // Espera : Dato byte recibido; NAK retornado
        i = I2DAT;
        I2CONSET = 0x10;
        I2CONCLR = 0x08;
        while(I2STAT != 0xf8); // Espera a fin de lectura
        I2CONSET=0x40;
        I2CONCLR = 0x28;

        return i;
}
```

```
#*****************************************************************************//
#*****************************************************************************//
# ARCHIVO: DRIVER.C                                                           //
# AUTOR: Jose Basterrechea                                                  //
# Tarjeta de sensores                                                         //
#                                                                             //
#*****************************************************************************//
#*****************************************************************************//



void dir_io(int direccion);
void set_io(int pines);
void clear_io(int pines);

void cleanISR(void);

void init_timer(void);

void activoISR(void);
void desactivoISR(void);



#*****************************************************************************//
#*****************************************************************************//
# ARCHIVO: DRIVER.C                                                           //
# AUTOR: Jose Basterrechea                                                  //
# Tarjeta de sensores                                                         //
#                                                                             //
#*****************************************************************************//
#*****************************************************************************//


#*****************************************************************************//
# NOMBRE: dir_io                                                              //
# AUTOR: Jose Basterrechea                                                  //
# FUNCION: Declarar los puertos como entradas                                 //
# PARAMETROS RECIBIDOS: Direccion de los puertos                            //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//


void dir_io(int direccion){
        IODIR0 = direccion;
}



#*****************************************************************************//
# NOMBRE: set_io                                                              //
# AUTOR: Jose Basterrechea                                                  //
# FUNCION: Poner a '1' los pines                                              //
# PARAMETROS RECIBIDOS:   Pines a modificar                                 //
# PARAMETROS DEVUELTOS: No tiene                                              //
#*****************************************************************************//



void set_io(int pines){
        IOSET0 = pines;
}
```

```
#*******************************************************************************//
# NOMBRE: clear_io                                                              //
# AUTOR: Jose Basterrechea                                                      //
# FUNCION: Poner a '0' los pines                                                //
# PARAMETROS RECIBIDOS: Pines a modificar                                       //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//


void clear_io(int pines){
	IOCLR0 = pines;
}


#*******************************************************************************//
# NOMBRE: init_timer                                                            //
# AUTOR: Jose Basterrechea                                                      //
# FUNCION: Activar el timer                                                     //
# PARAMETROS RECIBIDOS: No tiene                                                //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//


void init_timer(void){
	VICIntEnable = 0x00000010;
}


#*******************************************************************************//
# NOMBRE: cleanISR                                                              //
# AUTOR: Jose Basterrechea                                                      //
# FUNCION: Limpiar el flag de atención a la interrupción                        //
# PARAMETROS RECIBIDOS: No tiene                                                //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//


void cleanISR(void){
	T0IR = 1;
}


#*******************************************************************************//
# NOMBRE: activoISR                                                             //
# AUTOR: Jose Basterrechea                                                      //
# FUNCION: Activar de las interrupciones                                        //
# PARAMETROS RECIBIDOS: No tiene                                                //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//


void activoISR(void){
	VICIntEnable = 0x00000010;
}


#*******************************************************************************//
# NOMBRE: desactivoISR                                                          //
# AUTOR: Jose Basterrechea                                                      //
# FUNCION: Desactivar las interrupciones                                        //
# PARAMETROS RECIBIDOS: No tiene                                                //
# PARAMETROS DEVUELTOS: No tiene                                                //
#*******************************************************************************//


void desactivoISR(void){
	VICIntEnable = 0x00000000;
}
```

VII. BIBLIOGRAFÍA

- http://www.keil.com/dd/chip/3698.htm
- http://www.keil.com/support/man/docs/uv3/uv3_overview.htm
- http://www.roboteq.com/files_n_images/files/manuals/ax3500quickstart19b-060107.pdf
- http://www.canbus.galeon.com/electronica/canbus.htm
- http://www.kyheingenieria.com/info/Kyhe_Lawicel_CANUSB_manual.pdf
- http://www.embeddedartists.com/products/boards/lpc2129_can.php
- http://webs.ono.com/valleverde/8582.htm

- http://www.iesjuandelacierva.es/~fremiro/Transparencias%20DPPE/Bus%20I2C%20PCF8574A.pdf
- http://alcabot.org/seminario2006/Trabajos/JoseManuelMurciaBarba.pdf
- http://es.wikipedia.org/wiki/Puerto_serial
- http://es.wikipedia.org/wiki/COM1
- http://www.monografias.com/trabajos33/puertos-de-comunicacion/puertos-de-comunicacion.shtml
- http://es.viatech.com/es/products/mainboards/motherboards.jsp?motherboard_id=81
- http://www.psicofxp.com/forums/electronica.149/464598-controlador-pid.html
- http://es.geocities.com/jeeesusmeeerino/procesos/teoriapid/teoriapid.html
- http://www.superrobotica.com/S320122.htm
- http://www.sparkfun.com/commerce/product_info.php?products_id=252
- "Robot telecomandado bajo entorno GNU/LINUX"
- "Guiado semiautomático de una silla de ruedas"
- Paulo F. S. Amaral, Juan Carlos G. García, Teodiano F. Bastos Filho, Manuel Mazo, "Ambient Assisted Route Planner Based on XML Files with Accessibility Information, pp. 147 – 152.