PROGRAMACIÓN DE UN ROBOT AUTÓNOMO

Diseño, construcción, programación e integración de un robot autónomo en el entorno Pyro.

Memòria del Projecte Fi de Carrera

d'Enginyeria en Informàtica

realitzat per Antonio Jesús Dávila Molina

i dirigit per Ricardo Toledo Morales

Bellaterra, 12 de març de 2007

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, Ricardo Toledo Morales

Professor de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Antonio Jesús Dávila Molina

I per tal que consti firma la present.

Signat: .....

Bellaterra, 12 de març de 2007

"La suerte favorece sólo a la mente preparada."
Isaac Asimov.

Agradecimientos

Comenzaré con el clásico agradecimiento y sobradamente merecido a mi familia, dado que han sido extremadamente comprensivos con todo el tiempo que permanezco ausente, tanto mientras realizaba el proyecto como en mis estudios y profesión.

Quiero hacer mención especial a mi padre, que con su ejemplo motivó las ganas por conocer y descubrir el porque de las cosas, pero además como realizarlo de forma completamente autodidacta. Definitivamente, una de las lecciones mejor aprovechada de mi vida.

A Ricardo Toledo, tutor de este proyecto, agradecer su compresión, ayuda e ilusión. Ha sido un gran compañero en este viaje.

Una última gratitud la dirijo al trabajo realizado por los profesionales que han creado Pyro, Douglas Blank, Kurt Konolige, Deepak Kumar, Lisa Meeden, Holly Yanco, también a Richard Barry creador del FreeRTOS, y por último a Peter Fleury, por su librería de I2C.

Índice:

Introducción

Siempre ocurre que los títulos se eligen antes y los proyectos no acaban de la misma forma que se idean, porque un diseño inicial sirve principalmente para saber porque no se lleva a cabo tal y como se ideó, cada paso es un paso para poder cambiar muchas cosas, descubrir nuevas , o descartar ideas que parecían tremendamente sólidas.

A medida que este proyecto iba cogiendo forma, también lo hacia su objetivo, pero con una premisa que siempre ha estado presente, es que el trabajo realizado se pudiese utilizar en actividades de docencia e investigación, continuar y mejorar.

Objetivo

Construcción de un robot autónomo modular con distintos tipos de sensores/actuadores integrado en un entorno de programación apto para el desarrollo futuro de actividades de docencia e investigación.

La idea inicial era crear un interfaz para que cualquiera pudiese usar el robot desde su programa, y dotarla de una fuerte funcionalidad, además de crear un programa de demostración capaz de hacer que se desplazara en un entorno controlado evitando obstáculos.

Dado que los primeros experimentos fueron altamente positivos en cuanto a la factibilidad técnica de la construcción del robot, se decidió dotar a la plataforma de un nivel superior de prestaciones. Para ello se consideró importante la integración del robot dentro de un entorno de robótica que ofreciera mayores recursos.

Se escogió el entorno de robótica Pyro [6], que ofrece una correcta capa de abstracción y dispone de soporte para visión, lógica borrosa y redes neuronales entre otras funcionalidades. La integración del robot en el entorno Pyro se convirtió en uno de los principales hitos.

La integración en el entorno Pyro tiene dos consecuencias muy importantes. Desde el punto de vista de la docencia, facilita en aprendizaje debido a los diversos niveles de abstracción que se ofrecen. En investigación, permite abordar diversas áreas

sin perder tiempo en construir, diseñar o programar una base para ello.

Motivación

Desde hace mucho tiempo se ha soñado siempre con autómatas que ayudasen al hombre en el trabajo pesado o peligroso, incluso sustituirlo o hacer tareas que el hombre no esta capacitado para realizar. Autómatas guerreros, exploradores o muy inteligentes capaces de razonar.

La realidad es lejana de la ciencia ficción, algo sencillo e intuitivo como es para el hombre interactuar con el entorno, moverse, explorar, sentir o incluso algo tan obvio como ver, cuando intentamos realizarlo con un ordenador se convierte en una tarea complicada, de alto requerimiento computacional, o nuestro conocimiento no nos lo permite.

Personalmente siempre he sentido atracción por la robótica, la interacción de los ordenadores con el "mundo exterior", y esta curiosidad siempre me ha llevado a entrar en contacto directo con la electrónica, así que definitivamente este proyecto me atraía por encima de otros.

Análisis de Requerimientos

A continuación haremos un análisis de requerimientos jerárquico, es decir cual es el requerimiento principal y que requerimientos nos aparecen dependientes de este nivel ya sean funcionales, o físicos.

Requerimientos funcionales: robot autónomo

- **Ser capaz de moverse por un entorno controlado mostrando diferentes grados de "inteligencia".** Teniendo un entorno relativamente controlado donde se minimizan las interferencias que pudieran anular, mermar o engañar seriamente los sensores.
    - **Disponer de locomoción.** Disponer de algún mecanismo que permita al autómata desplazarse por el medio.
    - **Capacidad de percepción.** Es necesario que el autómata cuente con dispositivos capaces de detectar y obtener información del medio.
    - **Procesar la información obtenida.** Disponer de la suficiente potencia de cálculo como para poder desarrollar comportamientos inteligentes, con vistas al futuro y poder investigar en visión por computador.
- **Que el proyecto sea extensible y una posible herramienta de enseñanza e investigación.** Para este objetivo deberíamos alejar al alumno de los detalles complejos de implementación, y dar una interfaz sencilla que le permita solo dedicarse a aprender los detalles clave que realmente interesan.

- **Integración en el entorno de robótica Pyro.** El entorno Pyro[1] que permite abstraer el concepto de robot independientemente de cómo este construido o implementado, permite tener ese medio para enseñanza, además de incorporar muchas herramientas de inteligencia artificial ya casi preparadas su uso, como lógica difusa, redes neuronales o algoritmos genéticos entre otros.

- **Dispositivo que centralice todo el control del hardware.** Cada sensor o dispositivo hardware funciona de una manera diferente. Para poder desarrollar el módulo del Pyro, disponer de un solo camino para acceder a todos estos dispositivos de bajo caudal seria algo muy recomendable.

Requerimientos Físicos:

No podemos olvidar que el fin del proyecto es crear un robot, y esto tiene unas necesidades de materiales dispositivos, cada uno con de ellos con una serie de requisitos de funcionamiento, por ejemplo: para poder ejecutar el Pyro necesitamos un ordenador dentro del robot, pero para que este funcione necesita energía.

A continuación expondremos a grandes rasgos los principales requerimientos físicos derivados de los funcionales, ya entraremos con más detalle en capítulos posteriores.

- **Movimiento y Sentido:** el robot necesita unos dispositivos para moverse y otros para recibir información del medio, estos dispositivos por desgracia no son fáciles de conectar al ordenador directamente y necesitan de cierta adaptación tanto eléctrica como lógica. Para ello se ha creado la interfaz de

[1] acrónimo de **Py**thon **Ro**botics, software desarrollado principalmente por Douglas Blank y Deepak Kumar de la universidad de Bryn Mawr, Filadelfia, , Kurt Konolige universidad de Stanford en Palo Alto, California, Lisa Meeden de la universidad de Swarthmore en Filadelfia y Holly Yanco de la universidad de Massachussetts Lowell

control, que no solo controla, si no que también alimenta y adapta electrónicamente los sensores y dispositivos de actuación.

- **Interfaz de control:** Necesita una fuente de energía para poder funcionar Esta fuente debe estar separada de la alimentación de los motores dado que la electrónica de control es muy sensible al ruido electrónico.

- **Pyro:** para poder ejecutar el entorno de robótica Pyro necesitamos un ordenador PC compatible, su alimentación no es nada sencilla y requiere un cuidado especial.

Arquitectura del Robot

No existe un método ni consenso para definir cual es la mejor forma de diseñar un robot, muchas veces el hombre lo define como una copia de si mismo, y eso es porque nosotros mismos somos un ejemplo claro de autómata. Somos autosuficientes, además de ser capaces de realizar tareas realmente complejas.

El diseño de un autómata esta regido por los objetivos que deba cumplir. En nuestro caso el objetivo principal es que se pueda usar para investigar robótica y visión. Partiendo de la premisa anterior, el autómata deberá ser móvil y a la vez ha de poseer varios niveles de abstracción separados entre si de forma muy clara dado que también ha de ser un sistema para la enseñanza, y si dispusiera de multitud de partes interrelacionadas de forma poco clara complicaría su uso y el tiempo se malgastaría en comprender y hacer funcionar el sistema en lugar de focalizar en lo que realmente importe en cada actividad.

Diseño Lógico

Si nos miramos a nosotros mismos veremos un ejemplo de autómata muy avanzado que dispone de sus sensores, su sistema de locomoción, y su unidad de proceso, por lo que tomarnos de ejemplo nosotros mismos, es algo natural.

Así se hizo para organizar este robot. Si observamos la figura 1, como el diagrama indica, disponemos de una unidad de proceso que es un ordenador PC, que haría la función de cerebro, luego tenemos un circuito con un microcontrolador, que se encarga de unificar en una sola interfaz la gran mayoría del hardware de bajo nivel, si continuamos con nuestra comparación este sería nuestro cerebelo, descarga al cerebro de tareas rutinarias o de controlar al detalle otros dispositivos.

Y por último tenemos el hardware (nuestro cuerpo y sus sensores) que es lo que permite interactuar con el medio.

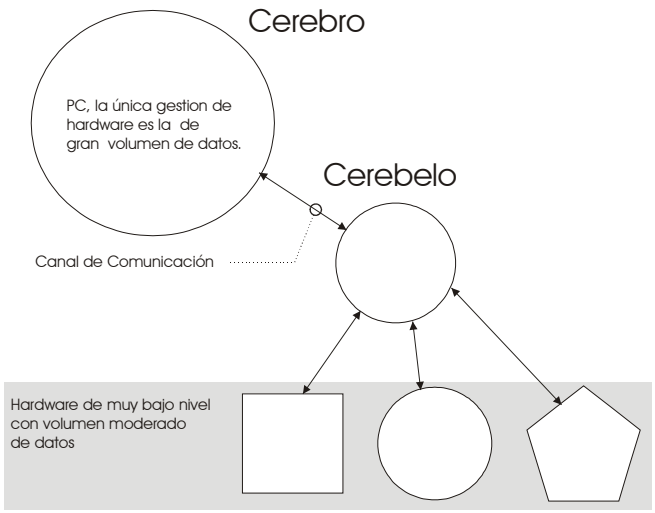


Figura 1: *Esquema Lógico.*

En el "cerebro" está el entorno Pyro que proporciona un medio eficaz para la construcción de comportamientos más o menos inteligente y su ejecución en nuestro autómata. En el "cerebelo" se sitúa nuestra unidad de control que nos permite gestionar todo el hardware de bajo volumen de datos, muy diferente entre si. El "cerebelo" tiene la capacidad de gobernar los dispositivos de actuación sobre el medio como pueden ser los motores y posee un sistema que tiene por misión interrogar a los sensores según se le ordene. .

El "cerebelo", (que a partir de ahora lo llamaremos interfaz de control), está controlado por un sistema operativo el cual da un soporte básico y nos permite aprovechar mucho mas el microcontrolador. Sobre este sistema operativo se ejecutan unas tareas (figura 2). Cada tarea o módulo, tiene ciertas responsabilidades. Por

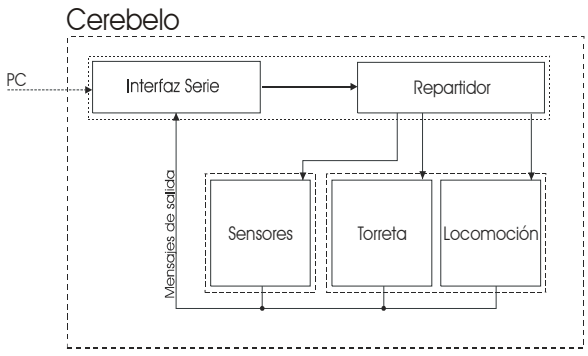


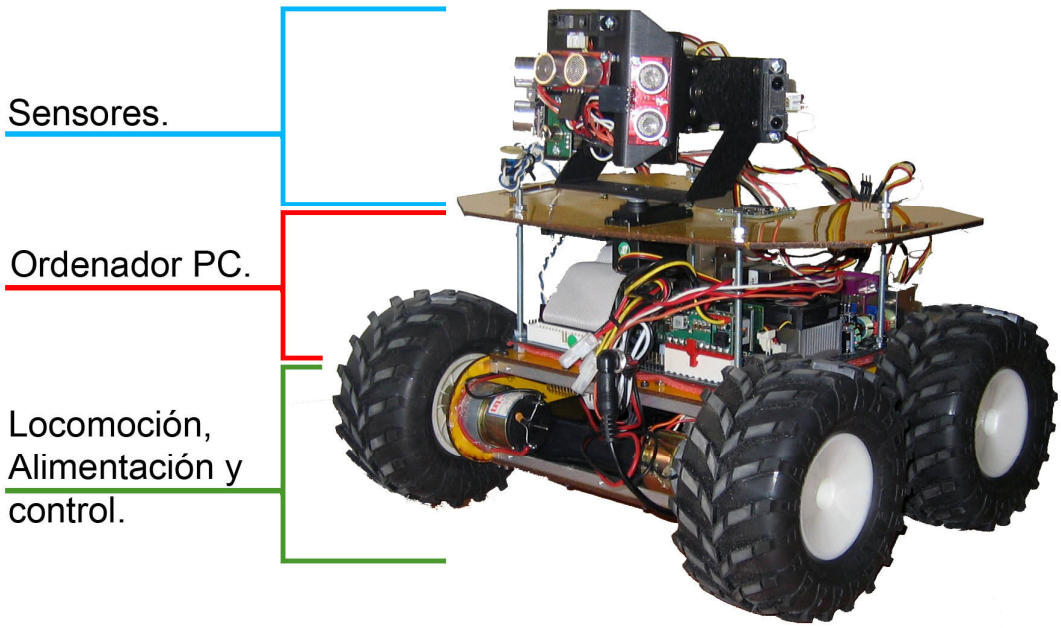
Figura 2: *Módulos del sistema de control.*

ejemplo: el módulo de interfaz serie se encarga de recibir y enviar comandos y datos desde y hacia el Ordenador.

El repartidor que forma parte de módulo interfaz reparte las órdenes según su naturaleza a los demás módulos, estos ejecutarán la orden y pueden o no devolver un resultado, también pueden enviar datos periódicamente.

Diseño Físico

A continuación haremos una descripción física del robot, nos basaremos en 3 apartados, según su posición física, comenzaremos de abajo hacia arriba



Locomoción

El diseño físico ha sido determinado por los materiales y herramientas empleados. Este robot es un vehiculo propulsado por ruedas, con una batalla muy corta que permite descartar el uso de ruedas directrices, y permite usar el mismo método para moverse que utilizan los carros propulsados por orugas. Es decir el robot es capaz de girar en función de la velocidad aplicada a cada par de ruedas de cada lado. Por ejemplo si las ruedas de la derecha giran a una cierta velocidad hacia delante y las de la izquierda a la misma velocidad hacia atrás el robot girará sobre su eje hacia la izquierda.

Para poder realizar tales movimientos detrás de cada rueda hay un pequeño motor de 7.2 voltios DC provisto de una caja reductora, los dos motores de cada lado son conectados en paralelo para trabajar como si fuesen uno, estos son vistos por el sistema como 2 motores uno para cada lado. Ver mas detalles en el Apéndice I

Sistema de energía (Alimentación)

Para alimentar el robot se utilizan dos baterías, una destinada a alimentar el ordenador, la electrónica de control y los sensores, y otra destinada a alimentar a los motores. Para alimentar el ordenador se utiliza una batería universal para portátil, es una batería de litio de gran capacidad que permite mantener encendido el ordenador y la electrónica de control entre 60 y 100 minutos.

Para alimentar los motores se ha optado por una batería de 6 células que da 7.2 voltios, este tipo de unidad es muy común en el hobby de automovilismo de radio control eléctrico. Permite alimentar los motores como mínimo tanto como la otra batería alimenta el ordenador.

Las baterías están alojadas en el compartimiento inferior junto con la placa de control, dado que son los elementos que más pesan, nos permite tener un centro de gravedad lo más bajo posible, que contribuye a la estabilidad del robot.

Control

Los motores y los sensores son gobernados por la unidad de control ("cerebelo"), la cual se comunica con el ordenador central mediante una comunicación serie. Hablaremos con mucho mas detalle de ella mas adelante.

Ordenador

El ordenador se encarga de dotar de la potencia suficiente al robot como para poder realizar un mínimo de comportamiento inteligente y poder incorporar cámaras para dotar al robot de visión por computador.

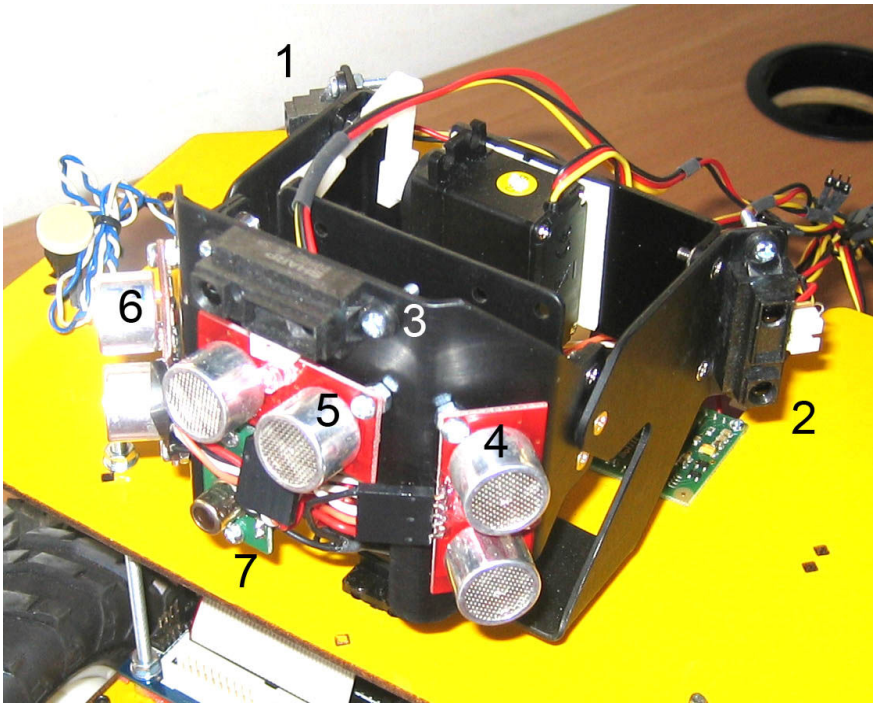
Sensores

El robot dispone de la mayoría sus sensores distribuidos encima de una torreta o unidad de *pan & tilt*[2], esta unidad es accionada por dos servomotores y gobernada por la

[2] Termino muy utilizado en robótica, que se refiere al conjunto mecánico que permite mover un sensor o grupo de estos como si de una cabeza se tratara, es decir horizontalmente y verticalmente.

unidad de control. En la torreta disponemos de varios tipos de sensores como: sonares, infrarrojos, una termopila que permite obtener imágenes térmicas muy sencillas. En una próxima etapa se incorporará a la torreta un par de cámaras.

Otros sensores se encuentran fuera de la torreta. Por ejemplo el compás magnético que nos proporciona información acerca de norte magnético. En apartados posteriores entraremos en más detalles con los sensores.



Los números 1, 2,3 son sensores de distancia mediante infrarrojos. 4,5 y 6 son sensores de distancia e iluminación mediante ultrasonidos (o sonares) y fotorresistencia, y el numero 7 es una termo pila que dispone de un vector vertical de 8 sensores de temperatura.

Dispositivos de Control

Unos de los módulos más importantes del proyecto es la Unidad de Control que permite unificar en una sola unidad todos los sensores de bajo volumen de datos y el control de los dispositivos de actuación.

Durante el desarrollo del proyecto se han probado dos opciones:

1. Utilizar una placa de propósito general ya creada y de sencilla programación
2. Construir nuestra propia placa. A continuación explicaremos los dos dispositivos por separado.

OOPic

La OOPic [7] *(del Ingles: Object Oriented PIC[3]),* es una placa desarrollada por la empresa americana Savage Innovations, es una placa de control diseñada específicamente para robótica, muy flexible en el conexionado y que se puede programar con sintaxis del estilo Basic, C o Java, pero solo la sintaxis, el código generado es convertido a un pseudo código que es interpretado en el hardware final.

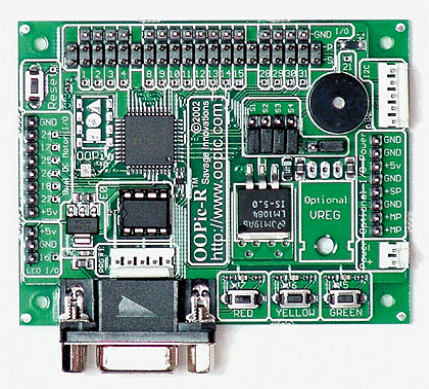


Figura 3: *OOPic-R*

La programación orientada a objetos del OOPic es muy fácil y muy rápida de aprender, toda su potencia reside en su biblioteca de objetos, en esta disponemos de una larga colección de objetos para controlar multitud de hardware diferente, desde sensores de infrarrojos o sonars incluido específicos para algunos fabricantes de sensores.

---

[3] PIC es un microcontrolador diseñado y construido por la empresa Microchip, tiene un rotundo éxito entre los aficionados a la robótica y electrónica por su fácil programación y la gran cantidad de recursos que existen en Internet.

En la figura 3 podemos apreciar como el OOPic es un entorno cerrado, muy simplificado y orientado al aficionado que no tiene excesivos conocimientos, esto hace que sus prestaciones no sean muy altas. Aunque el microcontrolador de 20 Mhz disponga de una potencia respetable, el programa del usuario en pseudo código se graba en una memoria externa (el interprete se encuentra grabado en la memoria interna del microcontrolador), con acceso mediante una comunicación serie (a como máximo 400 Khz.). Según las especificaciones de Savage se ejecutan 2000 instrucciones de pseudo código por segundo, con este primer dato ya podemos descartar cualquier intento de realizar tareas que sobrepasen ese nivel de prestaciones...

Aún así Savage Innovations propone superar este problema mediante los circuitos virtuales. En el programa se declaran unas puertas lógicas, que el usuario las interconecta entre si y con los objetos creando un circuito, esto se queda como un estado dentro del microcontrolador y es ejecutado a una velocidad muy superior que la del pseudo código.

La idea de los circuitos virtuales es buena, pero tropieza con otro problema que tiene el OOPic: los datos y los objetos comparten la memoria con el interprete, algo que en un principio debería ser hasta ventajoso pero en un microcontrolador esta memoria es muy limitada, en el caso del PIC 16F877[8] que es el que usa el interprete, es de 368 bytes de ram y de 256 bytes de eeprom, el OOPic nos permite usar los 256 bytes de eeprom como si fuese memoria, pero no nos permite usarla en circuitos virtuales (porque es de acceso lento), así que para usar circuitos virtuales solo nos quedan los 368 bytes compartidos, de los cuales 96 bytes son para objetos y otros 72 bytes para memoria del programa del usuario. Contando que cada objeto necesita entre 3 y 6 bytes, el tamaño de los programas (problemas a abordar) queda acotado a estos límites.

AVR

Debido al bajo rendimiento del OOPic se decidió buscar una alternativa, es decir buscar un nuevo microcontrolador. Luego de una revisión de varias alternativas se tomó la decisión de reemplazar el OOPic por un microcontrolador AVR.

El AVR es una familia de microcontroladores de 8 bits fabricada por Atmel. La principal diferencia con el entorno cerrado del OOPic, es que ahora no disponemos de un entorno que nos de soporte a multitud de hardware, es simplemente un microcontrolador.

El modelo de microcontrolador elegido es el Atmega128 [3,4], que es una máquina harvard con 128Kbytes para código y 4 Kbytes de memoria ram, dispone de multitud de dispositivos integrados como dos puertos serie, un bus I2C, generadores de pulso modulado por amplitud y conversores analógico – digital entre otras cosas.

Para el proyecto se decidió buscar una placa de propósito general dado que el integrado solo esta disponible en encapsulado SMD[4], y dificulta su montaje e instalación manual. La placa elegida es una fabricada por una empresa tailandesa llamada ETT. La placa (figura 4) no es más que el AVR ATMega128 de Atmel con todo lo imprescindible para funcionar, y unos conectores para hacer más fácil el acceso a las líneas del microcontrolador. Para información detallada de la placa del microcontrolador referirse al Apéndice I.

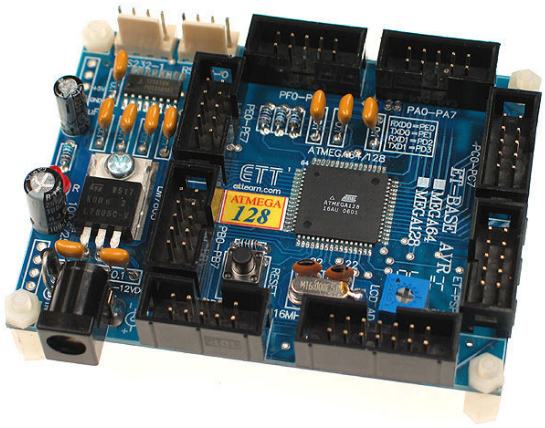


Figura 4: *Placa ET-Base AVR*

[4] **Surface Mount Device,** es un tipo de encapsulado el cual esta diseñado para ser soldado en la superficie del circuito impreso, este procedimiento normalmente lo realiza una máquina, tiene la ventaja de la que no es necesario taladrar el circuito y minimiza el espacio necesario.

Esta placa ha sido “expandida” con otra de creación propia para configurar y personalizar la placa anterior, esta expansión la veremos en detalle en el apartado de desarrollo.

El microcontrolador AVR puede ser programado tanto en Ensamblador como en C/C++, y es una orientación totalmente clásica a la programación de microcontroladores, veremos que para poder aprovechar mejor los recursos del microcontrolador AVR se optó por “instalar” un sistema operativo de tiempo real sobre el cual se programará el software de control.

Bus I2C

El bus I2C [9] *(Inter-Integrated Circuit)* fue diseñado por Philips en 1992, y su principal cometido es intercomunicar diversos circuitos integrados que se encuentran en el mismo circuito, normalmente microcontroladores con sus periféricos. Una de sus principales características es que es un bus serie multimaestro y solo necesita de dos líneas (masa aparte) para funcionar, permite direccionar 128 dispositivos y trabajar a frecuencias de entre 100 y 400 Khz.

El Bus I2C permite comunicar dispositivos realmente complejos con muy pocas líneas, y minimizar el número de conductores siempre conduce a minimizar el número de problemas.

En este proyecto el I2C cobra un importancia tanto estratégica como funcional, por una parte, nos permite conectar una considerable cantidad de dispositivos por el mismo canal, sin ocupar mas recursos externos del microcontrolador, y algo aun mas importante una forma común de acceso a estos, así que una gran parte de dispositivos del robot o soportan de alguna forma o solo son accesibles por I2C.

Placa Base

Como hemos comentado en los requerimientos es necesario un sistema compatible PC para poder integrar el entorno Pyro. Para ello se eligió una de la famosa serie Epia fabricada por Via Technologies Inc.

La placa base elegida ha sido una Via Epia MII 10000 [10] basada en un chipset y el procesador C3 del mismo fabricante, La Via Epia encapsula un ordenador

personal compatible PC, en una placa de 17x17 cm., con todo lo necesario desde tarjeta gráfica hasta tarjeta de sonido, hasta dispositivos USB, puertos serie y paralelos etc.

Dispositivos de entrada

Para poder interactuar con el medio es necesario tener capacidad de percepción, y para un autómata o robot esto se traduce en disponer de sensores Actualmente, existen sensores de diversos tipos y modos. Lo más importante es que suministren una cantidad de información adecuada para que se puedan extraer resultados con unos mínimos de fiabilidad y precisión preestablecidos

A continuación vamos a describir los sensores que se han incorporado en el prototipo del autómata del proyecto:

Ultrasonidos

El autómata dispone de sensores de ultrasonido, estos basan su funcionamiento en emitir un pulso de ultrasonido de una frecuencia determinada y "escuchar" el eco que se produce cuando la señal encuentra algún obstáculo en su camino y rebota, esto permite, contando el tiempo desde la emisión del pulso hasta la recepción del eco, calcular la distancia que nos dista el obstáculo.

En nuestro proyecto usamos el SRF08 fabricado por Devantech Ltd. Este sonar controlado por I2C permite detectar objetos a distancias entre tres centímetros y seis metros, pero no solo eso porque no se limita a escuchar un solo eco, sino que escucha hasta dieciséis ecos ordenados por distancia, algo que por ejemplo es muy útil para obtener conclusiones suministrándolos como entrada a redes neuronales.

El SRF08 a diferencia de otros dispositivos comerciales es autónomo, el microcontrolador maestro ordena una lectura (emisión de pulso y escucha), y este no debe hacer nada hasta el transcurso de un periodo de tiempo en el que el controlador del SRF08 ya tiene disponible todas las lecturas.

Además se su funcionalidad principal el SRF08 dispone de un sensor de luz LDR (Light Dependent Resistor), que puede ser leído con los demás datos.

Las ventajas del sonar, aparte de la gran cantidad de información que nos da , es que tiene una precisión aceptable para evitar obstáculos, y un rango de distancia bastante considerable, el SRF08 tiene precisión del orden del centímetro y un rango de 3

a 600 centímetros.

Los inconvenientes son:

1. No es de los sensores más rápidos

2. Ciertas superficies porosas pueden amortiguar la señal y dar lecturas erróneas o incluso no darlas.

Infrarrojos

Los dispositivos de infrarrojos GP2D12 desarrollados por Sharp, son dispositivos de medición de distancia que proporcionan una señal de medida analógica. Funcionan de manera parecida al sonar pero de forma continua, hay un diodo emisor de infrarrojos que emite una señal a una cierta frecuencia, esta es rebota en un objeto y recibida mediante un receptor de infrarrojos, la señal es procesada por un procesador que trata la señal recibida para descartar los reflejos y cambios ambientales que darían falsos valores y a continuación da un voltaje de salida.

Para obtener información detallada acerca del funcionamiento del sensor consultar el Apéndice I.

Compás

El CMPS03, desarrollado por Devantech, es un dispositivo electrónico que mediante dos sensores magnéticos uno rotado 90º respecto del otro , es capaz de dar la orientación del robot respecto al norte magnético, con una precisión de décima de grado y un margen de error de dos grados.

Es un dispositivo I2C y permite obtener el resultado en 8 y 16 bits, en 8 bits devuelve el resultado entre 0 y 255 unidad como Brads[5], si hacemos la lectura en 16 bits recibiremos un valor entre 0 y 3600 que da los grados con 1 décima de precisión.

---

[5] **Brads** (del ingles: *Binary Radians*) en robótica muchas veces los recursos son escasos y una forma de representar los 360º de la circunferencia en un solo byte es usar los Brads, su relación es: 256 Brads es igual a 360 grados.

Para información más detallada sobre este dispositivo consultar el apéndice I.

Termopila

La termopila TPA81 es un conjunto de 8 sensores térmicos de precisión dispuestos en línea formando un vector columna. Esta distribución permite mediante desplazamiento del sensor generar imágenes térmicas.

El TPA81 dispone además del vector columna de sensores de temperatura un noveno sensor de temperatura ambiente, que podría servir para fijar el centro de temperatura y poder discriminar todo aquello que se encuentra cerca de la temperatura ambiente, y por último también dispone de capacidad de control de un servo que automáticamente hace girar para generar una imagen de 32x8 píxeles que abarca unos 180º de giro. Para encontrar información detallada en el apéndice I.

Dispositivos de salida

Motores

El método de locomoción de este autómata es mediante ruedas, y con un funcionamiento parecido al de un vehiculo oruga, dispone de 4 motores uno para cada rueda que funcionan agrupados de dos en dos según el lado, es decir los dos de la izquierda funcionan como uno solo, y los dos de la derecha de la misma forma. Los motores son de corriente continua que funcionan a 7.2v y llevan incorporada una reductora 50:1. Para controlar los motores se utiliza un doble puente en "H", que es una construcción típica en electrónica para el control de motores, que permite el control de velocidad, aceleración y sentido del motor.

El doble puente en "H" utilizado es un producto de Devantech con el nombre de MD22, que permite el control de dos motores. La MD22 tiene como entrada una tensión de potencia destinada a los motores y de salida los contactos necesarios para conectar dos motores, permite ser controlada de diversas formas: Analógico, PWM[6] e I2C. Este

[6] **PWM** (del ingles: *Pulse Width Modulation*) técnica que se basa en trasmitir información mediante una señal pulsante en la que la información es el "ancho" (duración) del pulso transmitido.

último es el que se utiliza en nuestro proyecto. Para información mas detallada del MD22 refiérase al Apéndice I.

Servomotores

Los servomotores o sencillamente servos son mecanismo electromecánicos capaces de realizar movimientos de precisión, son dispositivos muy ampliamente usados en el hobby de radio control, como puede ser el automodelismo, aeromodelismo y modelismo naval entre otros, de servos hay muchos tipos según si lo que se requiere es fuerza o velocidad, son lineales o de rotación.

Nuestro proyecto incluye 2 servos, destinados a manejar la torreta en la que están montados todos los sensores. Se usan dos servos "estándares" (*figura 5)* que son de rotación capaces de girar 180º entre extremos y sostener un par de hasta 3 Kg/cm.

La gran mayoría de servos se controlan mediante una señal de pulso modulado (PWM), que ha de tener un periodo mínimo de 50ms, donde el pulso debe estar en alto entre 1 y 2 ms. Este tiempo en alto indica la posición del servo.

El servo compara esta señal de mando con la que generan sus mecanismos internos e intenta, mediante la actuación sobre el motor, que las dos señales sean iguales. Utilizando un control PID[7]. Más información detalla en el Apéndice I.

Figura 5: *Servo Hitec HS-311 empleado en el proyecto*

[7] **PID** (Proporcional Integral Derivativo) es un sistema de control con retroalimentación muy usado en sistemas de control industriales, inicialmente pueden ser mecánicos, o de construcción electrónica analógica o (últimamente) digitales.

Desarrollo

Ahora que conocemos los componentes del robot veremos las diferentes fases de desarrollo que ha tenido el proyecto, y entraremos en detalles en aquellas áreas donde se ha desarrollado el trabajo mas complejo y elaborado.

Aproximación Inicial

La idea inicial era básicamente crear una interfaz común para el robot basado en el OOPic dado que este disponía de puerto serie de comunicación y un protocolo de control. Esta idea se mantuvo durante todo el proyecto, aunque mas tarde se crea la interfaz para el entorno Pyro, esto no deja de ser una aplicación para la interfaz ya creada como veremos mas adelante.

Primer Obstáculo: el OOPic

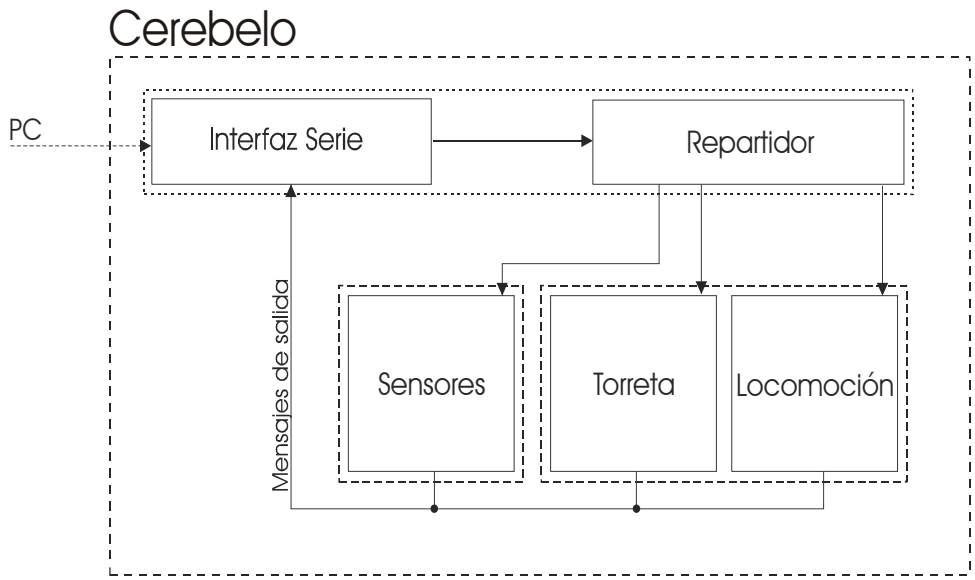
El OOPic ya estaba adquirido cuando comencé el proyecto, esto condiciona, sobretodo cuando la herramienta es desconocida, a usarla con total tranquilidad pensado que no habrá problema, pero todo depende del nivel que se quiera alcanzar.

Si el proyecto tratase de simplemente dotar al robot de un comportamiento sencillo solo hubiese hecho falta nada más que el OOPic. Pero el proyecto es mucho más ambicioso.

Hacerse con el dominio del OOPic no fue difícil, en 3 semanas tenia 3 de los 4 módulos prácticamente completados (con una funcionalidad mucho más limitada que la conseguida posteriormente con el AVR).

A continuación vamos a explicar el desarrollo máximo que se consiguió con el OOPic, lo orientaremos por módulos, se irá explicando los problemas encontrados, y al final expondremos unas pequeñas conclusiones.

A continuación refrescamos el gráfico de cómo esta diseñada la unidad de control:



Módulo Interfaz

El módulo interfaz trata de dialogar con el ordenador y repartir el trabajo. En el OOPic esto se realizaba a 9600 baudios, con la idea de aumentarlo más adelante a 31500[8] baudios que era la otra velocidad que el OOPic nos permitía seleccionar.

Este módulo recibe comandos de 5 caracteres, en función del primer carácter reparte la orden según su módulo, esto se hacia sencillamente llamando a una función de procesar por cada módulo, es un método sencillo, sí pero poco eficaz porque esto bloquea el programa hasta que esa orden ha sido procesada. Esta limitación no tiene solución, porque el OOPic no ofrece soporte directo a la multitarea.

Como explicamos secciones anteriores el OOPic tiene una herramienta llamada circuitos virtuales, estos permiten configurar acciones como si de un circuito lógico se tratase, pero fue imposible de utilizar para este cometido. Dado que para poder procesar los comandos se debía saltar entre los modos de circuitos virtuales y programa pseudo código, el problema está en la gran diferencia (órdenes de magnitud) de tiempo de ejecución entre un modo y el otro, mientras uno trabaja a 2000Hz el otro a 5Mhz, esto

[8] Esta velocidad no es un múltiplo lógico del puerto serie clásico, es la velocidad a al que trabaja el puerto MIDI (del ingles: *Musical Instrument Digital Interface)*, que es un puerto dedicado a conectar instrumentos musicales entre si.

hace que se colapse la pequeña pila del OOPic. Con apenas 2 comandos el OOPic termina en un estado incorrecto y consecuentemente bloqueado.

Así que la única forma para evitar este problema fue realizar el bucle de lectura de la interfaz serie en el pseudo código, cosa que limita a 2000 veces por segundo como máximo, la ejecución de este bucle

Módulo de Locomoción

Este módulo comunicaba por el bus I2C con la placa de control de motores, era un sistema sencillo en el que se ajustaba la velocidad y la aceleración, que son ajustes internos de la controladora MD22, se realizaban movimientos sencillos como avanzar, retroceder , y girar sobre su propio eje con la velocidad y aceleración previamente ajustada.

Módulo de Torreta.

En este módulo se manejan 2 servos para el control de la torreta, esto se realiza mediante pseudo código y un circuito virtual, para poder configurar y hacer funcionar un barrido automático.

Mediante comandos se puede ordenar la posición concreta a un servo, la información del ángulo viaja en 1 byte que era representado en brads, así que la precisión con la que se podía mover un servo era aproximadamente de 1.4 grados.

Mediante comandos también se puede activar o desactivar el circuito virtual que controla el barrido. Eran ajustables el arco en que se recorría cada paso, cuantos pasos se recorren de lado a lado y el tiempo de se espera entre cada paso, no me extenderé con esto aquí, dado que aprovecharemos para explicarlo e ilustrarlo con mas detalle más adelante en la versión final.

Aunque podamos barrer a 3 milisegundos por paso (calculo estimativo), no tenemos tiempo suficiente para leer los sensores, que necesitan mucho mas tiempo en una posición para poder estabilizar una lectura. Pero este no sería nuestro mayor problema. Ya que el OOPic no nos permitiría seguir el ritmo del sensor más lento, como puede ser un sonar que tarda 65ms.

Módulo de Sensores.

Este módulo no se llego a implementar ni en una versión elemental, es un módulo muy importante y crítico, pero aun optimizando los módulos anteriores, no teníamos recursos suficientes para implementar una versión mínima ¡Habíamos agotado todos los recursos del OOPic!

Conclusiones.

Antes de descartar el OOPic se valoró otras soluciones, como conectar otra OOPic en bus, el OOPic permite conectar hasta 128 placas iguales y poder interactuar entre ellas, la idea es que si el problema de recursos era solo espacial ¿porque no poner una segunda placa? para valorar correctamente se hizo una prueba.

Se creo un programa que comprobase a que ritmo podía absorber órdenes el sistema en el estado actual. Este programa dio un resultado desalentador, en una primera versión, solo era capaz de interpretar una orden por segundo.

Entonces se decidió optimizar y reducir al máximo incluso eliminar todos los mensajes escritos y código innecesario. Mejoró era 3 veces más rápido, pero insuficiente, porque la prueba era solo en un sentido, es decir era el cliente dando órdenes una tras otra tan rápido como permitía el OOPic, la unidad de control tiene mucho más que hacer que solo eso, debería controlar los servomotores e ir dando lectura de los sensores que se hubiesen pedido antes, y a un ritmo muy superior.

En definitiva el OOPic no era suficiente para los objetivos del proyecto.

Apuesta personal, el AVR

Personalmente veía imposible realizar un trabajo interesante, versátil y potente con el OOPic, así que me dedique a buscar un microcontrolador de mayores prestaciones. Además la elección no debía complicar excesivamente, el proyecto por lo que el microcontrolador se debería poder programar a un nivel de abstracción alto para poder terminar el proyecto en un tiempo razonable.

Viendo desde el lado del programador, es inevitable que se tendrán que programar registros, y a bajo nivel pero evitar el ensamblador era deseable y no muy difícil porque cualquier microcontrolador de cierto nivel ya se puede programar en lenguaje C, que es un lenguaje que se adapta perfectamente a la programación bajo nivel.

Debe disponer velocidad suficiente para poder procesar en tiempo real un mínimo de señales. Esto es confuso, porque no se trata de tener una elevada capacidad de cómputo, que en parte es deseable, es mucho más importante la frecuencia de instrucciones que el tamaño de las mismas. Vamos a ilustrarlo con un ejemplo: para nuestros objetivos es preferible hacer 16 millones de instrucciones en un procesador de 8 bits que 8 millones de instrucciones en un procesador de 32 bits. Siendo el rendimiento de cómputo del segundo muy superior, Nosotros necesitamos velocidad, cuantas más órdenes por segundo hagamos mejor dado que para empaquetar y enviar la información no necesitamos operaciones complejas ni de gran amplitud de palabra.

Desde el punto de vista electrónico el nuevo microcontrolador debería disponer de los recursos mínimos necesarios, un bus I2C, puerto serie, varias líneas analógicas, varias digitales, y encontrarse en un producto lo suficientemente terminado como para no tener que realizar trabajos importantes de electrónica.

La realidad es que la búsqueda fue más difícil de lo esperado, primero porque los microcontroladores de una cierta potencia no están disponibles en el encapsulado PDIP[9] dado su elevado número de pins, así que si no era en encapsulado PDIP debería ir en una placa que al menos llevase el equipamiento mínimo para su funcionamiento.

Después de buscar concienzudamente y descartar varios, fue seleccionada una placa diseñada y construida por la empresa tailandesa ETT, es una placa económica a la que se debe añadir otra placa "adaptadora" que nos ordenara las señales, conexiones y alimentación a nuestro gusto. En este caso la placa adaptadora se puede construir fácilmente.

---

[9] El encapsulado PDIP, es muy popular entre los aficionados, porque es fácil de manipular, tiene un tamaño fácil para soldar a mano y se puede insertar fácilmente en la parrilla de orificios de las placas de prototipado.

La placa se basa en un procesador de la familia AVR, fabricado por Atmel que hemos explicado en apartados anteriores.

En los siguientes apartados veremos la configuración física y lógica del AVR y su extensión como Unidad de Control.

Unidad de Control

Configuración Física.

Lo primero que vamos a ver es como está configurada físicamente la unidad de control y de que recursos dispone.

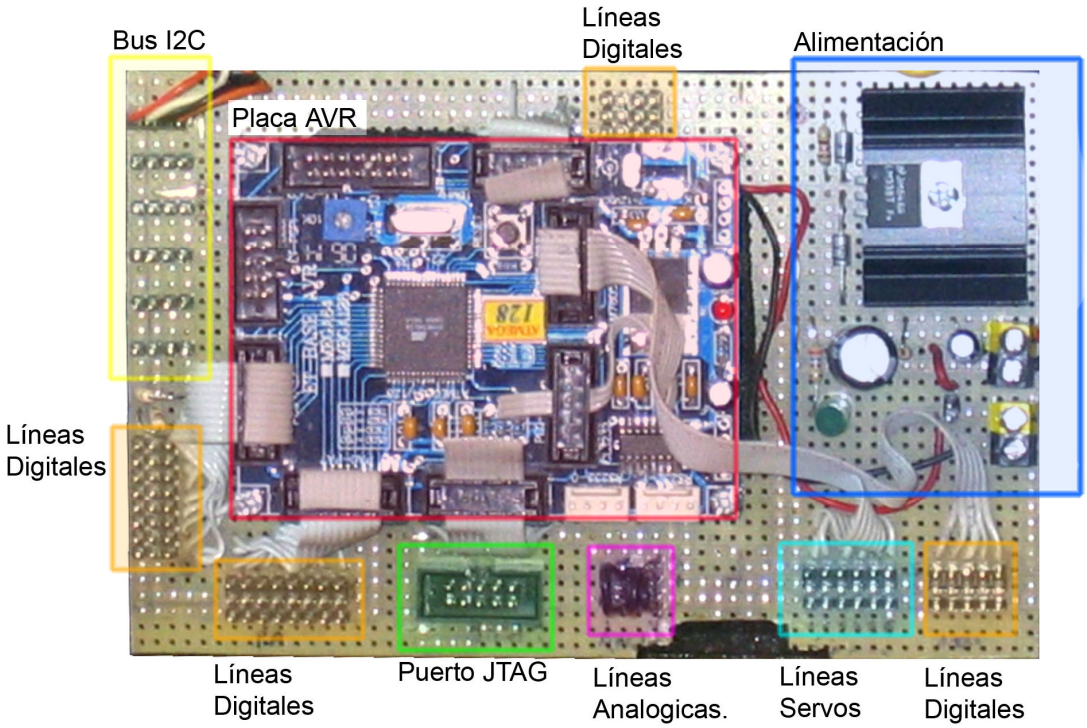


Figura 6: *Unidad de control*

Aquí podemos contemplar el primer prototipo de la placa de expansión sobre la que esta montada el AVR, dispone de varias partes bien diferenciadas. Podemos dos categorías de componentes principales; los activos y los pasivos. En el grupo activo tenemos la placa del AVR y la alimentación. En grupo pasivo tenemos todo lo demás: bus I2C, líneas digitales y analógicas, líneas de servos y el bus I2C, estos últimos

simplemente cumplen con el objetivo de distribuir las interconexiones con los demás dispositivos se puedan realizar de forma cómoda y práctica. Todas estas líneas están organizadas de tal forma que cada línea de control tiene agrupada su alimentación de forma que con un conector y cable de 3 hilos podemos conectar fácilmente un dispositivo. El orden es de arriba hacia abajo: señal, alimentación, tierra o masa.

La unidad activa AVR ya conocida es la que nos permitirá controlarlo todo, aunque no este indicado, de esta sale un cable con el conector serie que es el canal de comunicaciones con el ordenador.

Por ultimo comentar que el puerto JTAG es utilizado para grabar el programa del AVR, y además permite depuración en tiempo real.

Revisión 1.

Aprovechando el desarrollo del robot, en el departamento se decidió construir otro robot igual durante el curso de robótica, esto hizo que se emplearan más y mejores medios para el desarrollo del sistema, para ello se creo la placa de control con una máquina de control numérico. Aparte del cambio estético, la placa mejoró con la incorporación de una segunda fuente de alimentación separada para sensores y señales delicadas.

En el Apéndice I se podrá encontrar información más detallada acerca del esquema.

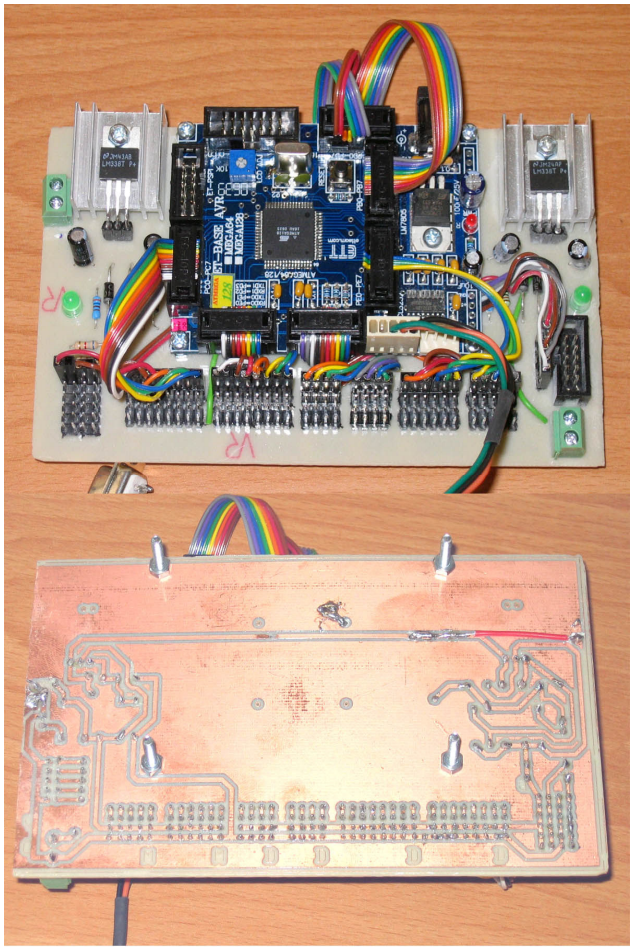


Figura 7: *Primera revisión*

Podemos apreciar la diferencia de acabado respecto al prototipo, además de que en el mismo tamaño entra una fuente de alimentación más.

FreeRTOS.

Contrariamente a lo que pasaba en el OOPic, en el AVR se dispone de una ya importante cantidad de recursos, no solo memoria, si no de contadores de tiempo, generadores de pulso modulado, puertos serie, bus I2C, líneas de entrada/salida entre otros. Al aumento de recursos hay que sumar la bajada del nivel en el que trabajamos, ahora no tenemos una biblioteca de objetos ya elaborados, lo cual complica considerablemente el trabajo.

Si atendemos al software que queremos desarrollar, es deseable modularizar bien y repartir equitativamente el tiempo entre varias tareas a realizar. Además de la necesidad de poder crear colas para poder absorber los picos de peticiones que se podrían generar. Incluso debemos poder arbitrar el uso del bus I2C dado que en el mismo disponemos de mismo sensores y control de motores.

Dadas estas necesidades, un salto crucial hacia delante fue el "no empezar de cero" y utilizar un sistema operativo. Se decidió el FreeRTOS [11].

El FreeRTOS es un proyecto de código libre que implementa un mini-kernel en tiempo real portable, esta escrito 99% en código C. Solo hay ensamblador en tareas que requieren conocer el hardware para poder llevarse a cabo, como la configuración del temporizador que dispara el planificador, cambio de contexto, etc.

El Núcleo.

El kernel puede ser configurado por el usuario, esta basado en un Round Robin con prioridades absolutas, cada proyecto dispone del fichero *FreeRTOSConfig.h*, que permite configurar el kernel. Parámetros importantes como decidir si el kernel es apropiativo o no, la velocidad de trabajo de la cpu, cada cuanto debe entrar el planificador en hertzios, o lo que es lo mismo la rebanada de tiempo mínima del planificador se pueden especificar en este fichero. La capacidad de parametrización se extiende a cuantos niveles de prioridades estarán disponibles, el tamaño mínimo de la pila para un proceso, el tamaño máximo del heap para el gestor de memoria y más.

Como un sistema operativo que es, el kernel permite a un tarea entrar en zona crítica, desactivar/activar interrupciones o forzar el cambio de contexto.

El Gestor de Memoria

El FreeRTOS nos permite elegir entre tres modelos de memoria, Este mini-kernel esta orientado a microcontroladores y estos suelen tener recursos muy escasos. Por ejemplo: si se va a reservar memoria dinámica pero no se va a liberar durante todo el funcionamiento, mantener en memoria, y ejecutar un gestor de memoria dinámica completo aparte de ser innecesario nos consumirá una valiosa cantidad de recursos, que en estos casos son limitados. Escogiendo un modelo se puede optimizar los recursos.

Actualmente existen tres modelos de memoria:

1. El *"Heap 1"* es el modelo más simple y una vez se reserva memoria no se puede liberar, este modo puede parecer bastante malo pero tiene aplicación en un gran número de aplicaciones, para poner una como ejemplo: este proyecto. Tiene restricciones importantes como que no se pueden crear tareas o colas en tiempo de ejecución, pero tiene también una ventaja importante es que es determinista siempre tarda lo mismo en devolver un bloque de memoria.

2. El *"Heap 2"* es como el anterior pero nos permite liberar memoria, con el importante detalle de que no combina los espacios libres en uno solo, eso puede traer problemas de "tener" espacio libre suficiente y no poder reservarlo por estar fragmentado.

3. El *"Heap 3"* se obtiene al directamente compilar las funciones de malloc y free de la librería estándar de C, esto da un gestor de memoria completo pero a cambio de incrementar la complejidad, el tamaño del kernel, y dejar de ser determinista. El determinismo es una propiedad se suele subestimar pero que puede ser de gran importancia. Este heap se usa normalmente en procesadores o microcontroladores muy potentes, en los ejemplos que trae el FreeRTOS este método de memoria se usa para la plataforma x86.

Dentro de un proyecto el heap se puede cambiar entre uno y otro modificando el makefile e incluyendo un modelo de heap u otro, el código fuente de cada modelo se encuentra en */portable/MenMang/heap_1.c , heap_2.c y heap_3.c*.

Procesos y Threads.

En el FreeRTOS los procesos y los threads tienen otros nombres, pero tienen la misma funcionalidad, Las tareas (Task) son procesos que tienen su propio espacio de pila y contexto. Las co-rutinas (co-rutine) son parecidas a los threads comparten pila y memoria con la tarea padre.

Al iniciarse el kernel crea una tarea "Idle" o de esperar. Esta tarea tiene la misión de liberar la memoria de las tareas borradas, siempre y cuando nuestro gestor de memoria nos permita hacerlo, asegurarse de que se dispondrá del tiempo necesario para que se pueda ejecutar, es decir hay que evitar la inanición de ésta.

Además el FreeRtos nos permite "colgar" o añadir una función al proceso Idle, así podríamos por ejemplo realizar tareas de mantenimiento mientras no haya trabajo que realizar, otra opción es crear una tarea con la misma prioridad que la tarea idle pero esto suponen un mayor uso de memoria.

Colas y Semáforos.

Como en cualquier operativo, tenemos disponibles colas y semáforos. Las colas tienen la limitación de que todo lo que es puesto en la cola es por copia y no por referencia, por eso se recomienda que ante estructuras complejas se haga una cola de apuntadores. Los semáforos trabajan del mismo modo que en cualquier otro sistema operativo.

Detalles y Desarrollo en el proyecto:

Instalación y puesta en marcha del FreeRTOS:

Una vez descomprimido el fuente del FreeRTOS, dentro de */source* tenemos los diversos ficheros que lo componen. Una carpeta importante es la llamada *portable* donde se encuentran los ficheros que son dependientes de la máquina en la que se va a utilizar.

Concretamente en nuestro proyecto se tuvo que crear una carpeta para el ATMega128, porque no estaba implementado, este trabajo fue más sencillo de lo esperado dado que ya había sido portado para el ATMega323 y en principio con unas modificaciones mínimas ya se consiguió la compatibilidad.

Un cambio obligado fue alterar casi por completo el temporizador que dispara el planificador, en la versión del ATMega323 se usa uno de los dos temporizadores de 16bits de los que dispone el AVR, ese contador, en nuestro proyecto, era imprescindible para manejar los servos por la resolución y por ciertos recursos asociados que permiten la generación de señales.

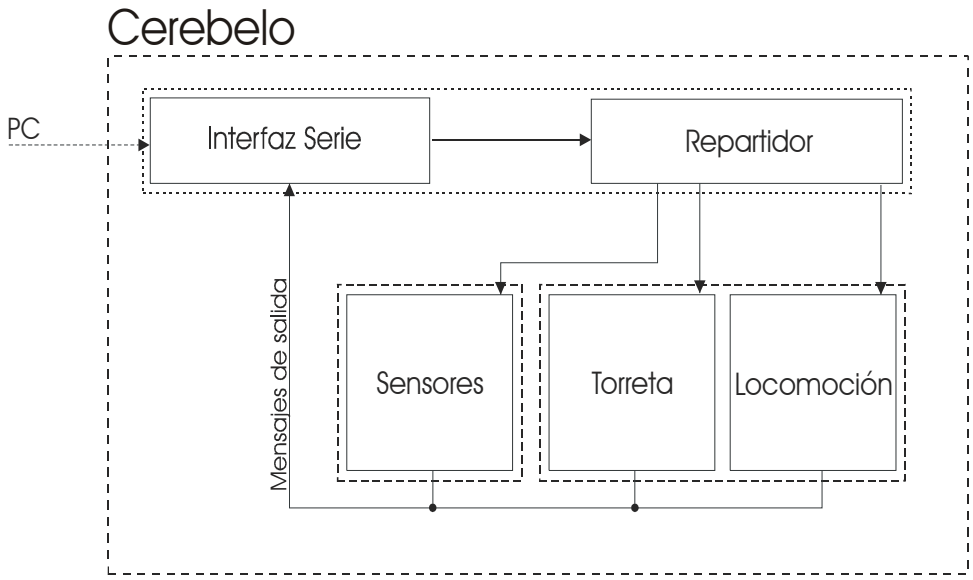
Así que se modificó el código dependiente de la maquina para que usase otro contador de 8 bits.

El FreeRTOS se ha revelado como una gran herramienta, se ha echado en falta algún sistema de control de errores, dado que esta diseñado para que se reinicie ante cualquier problema, y deja para la intuición personal el saber que lo ha provocado.

ARI.

El ARI es el programa desarrollado para gobernar los diferentes sistemas en el robot, ARI viene del inglés *Advanced Robot Interface*, y se bautizo sin pensarlo mucho.

A continuación se entrará en detalles sobre como esta construido el software, Como se ha visto varias veces el sistema esta dividido en varios módulos, a continuación recordaremos el esquema que los relaciona:

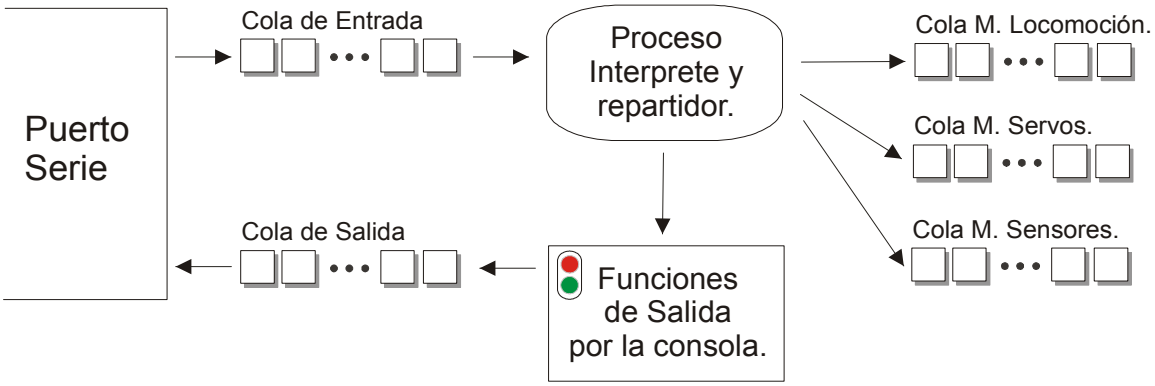


Cada módulo se traduce en un proceso en ejecución, así que el módulo de diálogo con el ordenador y que reparte las órdenes se llama Módulo Interfaz. El módulo que se encarga de controlar los motores Módulo de Locomoción, el módulo encargado de controlar los servos (en la figura Torreta) es llamado Módulo de Servos, y por último el módulo más importante el Módulo de Sensores que controla la percepción del robot.

El principal método de unión es mediante colas, éstas van muy bien para amortiguar los picos de trabajo.

Módulo Interfaz.

(modInteface.h y modInterface.c)



El módulo de interfaz controla las comunicaciones con el ordenador, recibe las órdenes por el puerto serie y reparte el trabajo, además dirige y gestiona el recurso de salida para todos los procesos.

En la cola de entrada se van acumulando los bytes recibidos por el puerto serie, estos son leídos en el proceso principal, hasta recibir un retorno de carro o superar los 10 caracteres que es la longitud máxima de de un comando. En caso de superar los 10 caracteres devolverá un mensaje de error "**E:too long**" y reiniciará la cuenta.

Si el mensaje recibido es de longitud correcta se interpreta el primer carácter, si este corresponde a un módulo conocido (**m**, = **m**otores, **s** = **s**ensores o **e** = s**e**rvo) se encolará en la cola del correspondiente módulo, si es una **i** (de interfaz), entonces se procesara el comando en ese momento. Si el comando no tiene un comienzo adecuado devolverá "**E:BadSec"** que indica sección incorrecta.

Los comandos disponibles para este módulo son, el reinicio, comprobar versión, activar/desactivar el echo, leer y escribir bytes en el bus I2C, ver la lista de tareas donde podemos consultar el estado, prioridad, uso de la pila e orden de inicialización, y el uso del heap que en nuestro proyecto es siempre fijo y se usa para tareas de depuración.

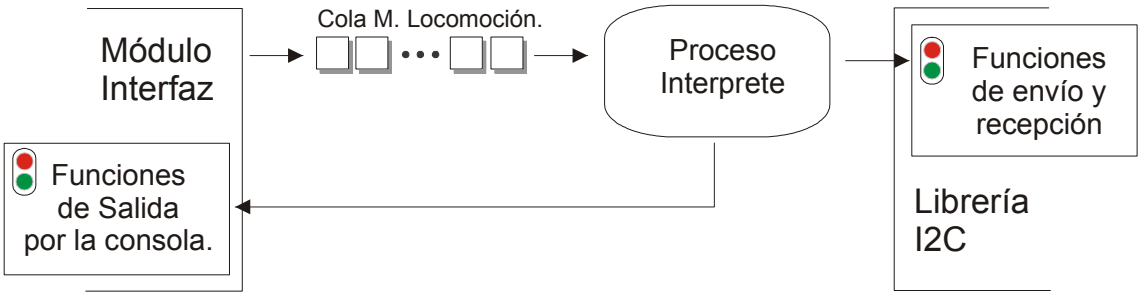
La Biblioteca del puerto serie estaba ya correctamente implementada en la versión AVR del FreeRTOS, solo hubo que retocar la posición de algunos registros para que

funcionara en el ATMega128.

Para mas detalles sobre los comandos del módulo interfaz ver el apéndice II.

Módulo de Locomoción

(modMotion.h y modMotion.c)



El módulo de locomoción se encarga de dar órdenes mediante el bus I2C al controlador de motores MD22, es un módulo sencillo que adapta los datos de entrada antes de enviarlos a la MD22.

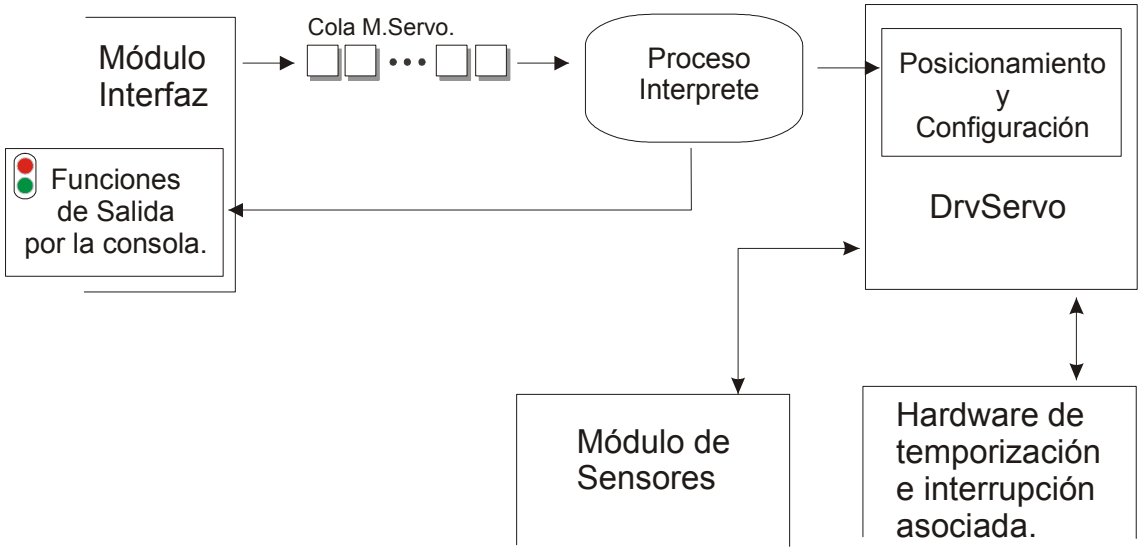
Durante el desarrollo se tuvieron que añadir dos funciones nuevas de control directo para el Pyro.

La Librería I2C se construyó a partir de una ya creada para el AVR, trabajo de Peter Fleury [12], que se ha ido expandiendo a lo largo de todo el proyecto. Una primera modificación fue la inclusión de un arbitraje por semáforos para poder utilizar la librería entre dos o más procesos. Peter creó esta librería para que fuera sencilla, para usarla en programas que no tuviesen el nivel de multitarea que tiene este proyecto.

También se han creado nuevas versiones de la función de lectura del bus para poder en una misma transacción realizar lecturas múltiples y así recuperar un tiempo valioso que se emplea en iniciar y finalizar la comunicación cada vez. Así que a la inicialmente creada función de lectura de un byte se añadió la de leer una palabra, y también la de leer bloques de bytes consecutivos.

Módulo de Servomotores

(drvservo.c, drvservo.h, modServo.h y modServo.c)



El módulo de Servos es capaz de controlar hasta seis servos. Todo el control del tiempo para generar la señal necesaria para poder usar los servos se encuentra en el controlador del Servo (drvServo). Gracias al hardware asociado a los dos temporizadores de 16 bits de los que dispone el AVR la necesidad de interrupciones para generar la señal se reduce significativamente.

Cada temporizador se puede ajustar para que termine en un tiempo determinado. Además llevan 3 registros hardware asociados que en cada incremento del temporizador son comparados para comprobar si el temporizador es mayor o igual que su valor. En cuanto el resultado de la comparación sea positivo, se dispara una opción que realiza un cambio de valor de 0 a 1 o de 1 a 0 en un pin del microcontrolador.

Principio de funcionamiento de los servomotores.

La principal diferencia entre un motor convencional de corriente continua y un servomotor es que el servomotor tiene la capacidad de ubicarse y mantenerse estable en una posición concretada.

Los principales elementos de los que se compone son: Motor, reductora, y electrónica de control. El motor es un motor de corriente continua convencional, la reductora permite al motor ganar fuerza pero a la vez lo hace más lento.

La electrónica dispone de un potenciómetro o resistencia variable que es un dispositivo con un eje y en función de su posición ofrece una resistencia u otra, esta señal es usada por la electrónica del servo para saber en que posición se encuentra,

La señal es comparada con la señal de entrada, el control del servo motor esta diseñado para minimizar la diferencia entre las dos señales, y así situar el servo.

La señal de entrada es una señal periódica de entre 11 y 20 milisegundos, la cual debe permanecer un mínimo de 0,5 milisegundos y un máximo de 2,1 milisegundos en valor positivo, y el resto en valor negativo, en función de la duración del pulso de entrada es indicada una posición, un pulso de 1ms indica un extremo normalmente el extremo derecho, y un pulso de 2ms indica el extremo contrario. Un ejemplo gráfico mas claro se puede ver en la figura 8.

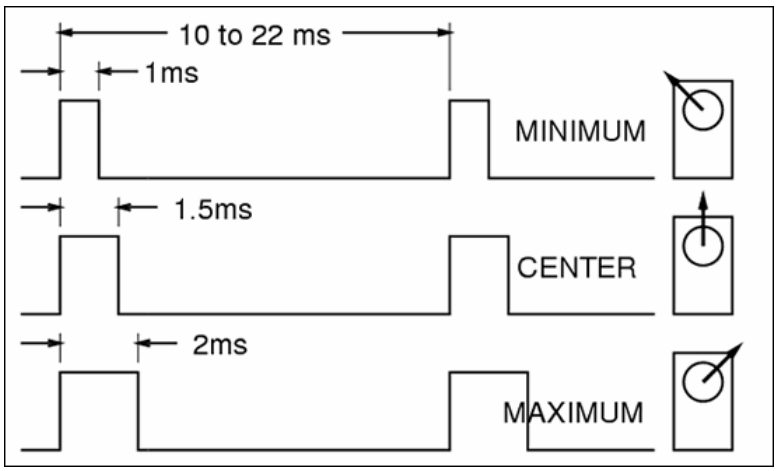
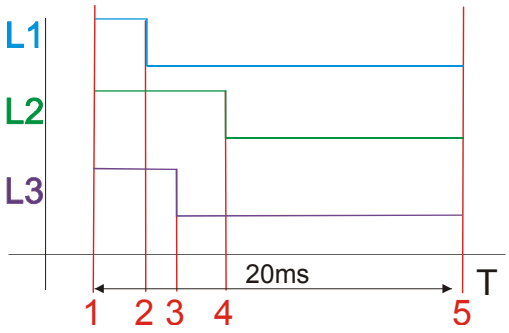


Figura 8: *Relación entre la señal de entrada de un servomotor y su posición.*

Ilustremos como el módulo de servomotores genera estas señales: Supongamos que un servo necesita una señal periódica de periodo 20ms, la cual debe empezar con un pulso alto hasta 1ms y se debe detener antes de llegar a los 2ms. Una señal de 1ms representa un desplazamiento del servo de -90 grados y una señal de 2ms representa un desplazamiento de +90 grados. En el gráfico podemos interpretar la L1 como la primera, la L2 como la segunda y la L3 un lugar intermedio entre las 2.



Las tres señales representan los tres registros asociados al temporizador que son comparados de un temporizador, en el paso 1, salta la interrupción que señala que el temporizador ha llegado a su fin, entonces lo que hacemos es poner a 1 todas las líneas de salida, y cargamos en los registros de cada comparador el valor correspondiente al tiempo deseado para marcar una posición, termina la interrupción. En el paso 2 ocurre que el registro de comparación de la L1 es menor que el del temporizador, por eso automáticamente y sin que el procesador intervenga la línea cambia de estado (de 1 a 0), lo mismo ocurre en 3 con la L3, y en 4 con la L2. En el paso 5 vuelve a entrar la interrupción.

Explotando estas opciones del hardware solo necesitamos una interrupción cada 20ms, que además es deseable para cambiar el registro de tiempo de las líneas y consecuentemente la posición de los servos.

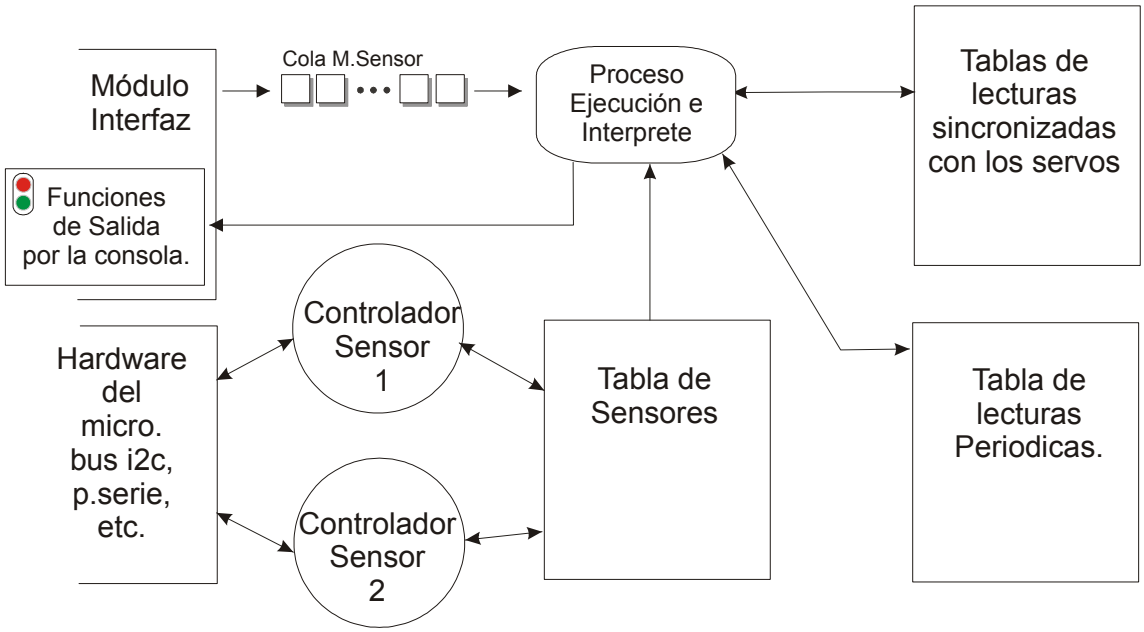
Una funcionalidad incorporada a los servos es la de poder configurar barridos automáticos, estos barridos van actualizando automáticamente la posición de uno o varios servos, la idea es poder apuntar los sensores a mas de una posición y realizar esto periódicamente, es decir que si hacemos un barrido de tres posiciones, izquierda, luego al centro y luego a la derecha, esto se repetirá hasta que el cliente ordene parar. Además este sistema esta sincronizado con el módulo de sensores para poder efectuar las

lecturas.

El proceso interprete se encarga de preparar los datos necesarios según las órdenes que recibe. Convirtiendo el valor de entrada al valor adecuado y configurar los parámetros de los barridos.

Módulo de Sensores.

(modSensor.h y modSensor.c)



El módulo de sensores, es de los más complejos y vitales, dado que es el encargado de recoger los datos de los diferentes tipos de sensores y mostrarlo al cliente como un interfaz unificado.

El proceso, además de la tarea básica de recoger e interpretar los comandos del usuario(o sistema de mas alto nivel como en nuestro caso es el PC-Pyro), se encarga de realizar las tareas de mantenimiento y ejecución asociadas con las tablas de lecturas sincronizadas y las lecturas periódicas.

Vamos a comenzar explicando como se añaden sensores al sistema, y como es el tratamiento de los mismos. Los sensores se definen mediante una o dos funciones, estas tienen un prototipo de declaración fija de tal forma que al añadir un sensor nuevo simplemente se debe implementar una o dos funciones.

Hay muchos tipos de sensores, En algunos se puede hacer una lectura directa porque, o bien se actualizan mucho mas rápido de lo que nosotros podemos interrogarlos, o porque tienen un buffer y siempre podemos acceder al último valor, pero también hay sensores que necesitan una orden para iniciar y realizar una fase de lectura, como pueden ser un sonar, y además este deja de responder durante la obtención de los datos, que tiene una duración considerable.

La primera función tiene el prototipo void *iniciar(unsigned short t, char i)* y está pensada para dispositivos que necesitan orden para comenzar una lectura, esta función recibe dos parámetros: una marca de tiempo en milisegundos, y un número usado como identificador que se define al agregar la función al módulo de sensores.

La marca de tiempo es para poder controlar el tiempo que transcurre desde que se ordena el inicio de la lectura hasta que se lee, para evitar abusar de un recurso compartido o gastar tiempo en trabajo innecesario. El segundo parámetro se usa a criterio del programador, un ejemplo práctico sería para poder usar las mismas funciones con varios sensores idénticos, por ejemplo si tuviésemos tres sonars, esta variable se puede utilizar para diferenciarlos, dado que al asignarlos al módulo de sensores, estos usaran las mismas funciones.

La otra función es la de leer, que tiene tres parámetros, el primero es, nuevamente, la marca de tiempo, el segundo es un puntero a un byte, y el tercero es el parámetro de uso a criterio del programador.

Cuando un sensor es “cargado” en el módulo de sensores se guardan dos punteros a estas funciones. Además, se guarda un nombre de cuatro caracteres, y un modo de lectura.

En el modo mas simple o modo entero, la función leer retorna un entero de 16 bits, que es el valor de la lectura. En el otro modo se retorna el puntero a una cadena de caracteres, en la que no puede haber caracteres de control, y la longitud de la cadena se devuelve a través de aquel segundo parámetro puntero a byte.

El cliente puede consultar cuantos sensores existen, y sus nombres, realizar una petición de lectura del último valor conocido, o asignarlo a algún servicio, como la lectura periódica, o la sincronización con los servos.

Una vez el sistema esta en marcha la tabla de sensores es fija, el índice de la tabla se usa como denominador principal de los sensores. Se usará ese número para referirse al sensor tanto dentro de la aplicación como por parte del usuario. Es decir el usuario puede ordenar leer el número cuatro y el programa devolverá el último valor conocido del sensor número cuatro.

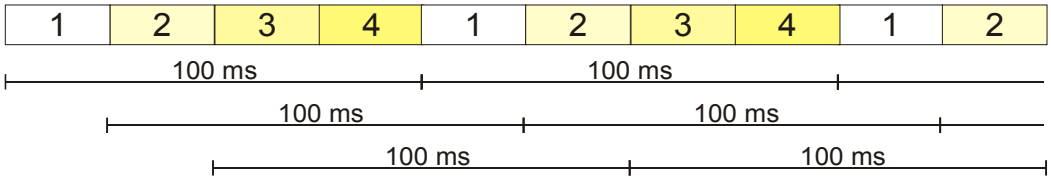
Una de las funcionalidades más potentes, que libera de mucha carga tanto al ordenador como a la unidad de control, son las tablas de lecturas, estas tablas inicialmente están vacías, y el cliente las puede alterar en tiempo de ejecución, la primera de ellas que es la tabla de lectura periódica, realiza una lectura periódica de cada sensor que se encuentre en ella, esta tabla permite tener hasta 25 sensores, la periodicidad con la que esta tabla se puede leer también es ajustable y varía entre una décima de segundo y mas de 2 segundos.

¿Porqué 25 sensores máximo? ¿Por qué una décima de segundo?

La idea común y que primero se presenta es la de utilizar una interrupción cada décima de segundo y leerlo todo. Quizás nuestro problema no es, ni mucho menos, que no nos de tiempo. El inconveniente es otro: si estamos todo el tiempo ejecutando interrupciones seguramente se acabarán anidando una encima de otra y terminen colapsando el sistema. De hecho esto llegó a ocurrir en nuestro sistema y nos condujo al diseño de esta alternativa.

La idea fue repartir equitativamente en el tiempo las lecturas, pero además no hacerlo mediante interrupción.

Es evidente que si el proceso consume muchos recursos y no permite que el bucle se ejecute lo suficientemente rápido, la alternativa no nos será de utilidad. Dado que la demanda de recursos del proyecto lo permite, lo que se hace, controlado por tiempo, es ejecutar 1 lectura cada bucle (si toca por tiempo). Si queremos mantener la cadencia de 100ms de 25 sensores a la vez, tenemos que leer 1 sensor cada 4ms, y así repartimos el trabajo a lo largo del tiempo sin generar cargas puntuales, esto aporta estabilidad al sistema. Además, si el sistema tiene un exceso de trabajo, no se colapsará por exceso de anidamiento de interrupciones, solo irá mas lento, algo preferible al colapso.



Este ejemplo simplificado a 4 sensores en lugar de 25, muestra como se distribuye la carga a lo largo del tiempo para conseguir que cada sensor sea interrogado con una cadencia de 100 ms.

La tabla de lectura sincronizada permite la adquisición de datos sincronizada con los movimientos de barrido de los servos, funciona igual que la tabla periódica, pero aquí si toca leer, no tenemos más remedio que hacerlo todo, dado que el servo nos espera para cambiar a la siguiente posición.

En cada ejecución del bucle que contiene el proceso de los sensores se realizan las siguientes tareas:

Primero comprueba que no halla ningún servo en modo de barrido y que este se encuentre detenido esperando la lectura de los sensores asociados, en el caso afirmativo, leerá los sensores correspondientes.

El segundo paso, es atender las lecturas periódicas de los sensores, esto es mucho menos costoso computacionalmente que el paso anterior, el tiempo se divide en 25

secciones, la duración de cada sección depende de la velocidad a la que se ajuste la lectura periódica, esto variara entre 4 ms para lecturas periódicas de una décima de segundo a mas 40ms para lecturas periódicas de mas de 1 segundo. Pero cada sección tiene correspondencia directa con una de las 25 posiciones de la tabla de lectura periódica, gracias a este detalle, leer un sensor se convierte en algo tan sencillo como utilizar la función de llamada de la entrada actual de la tabla e incrementar el contador que indexa la tabla.

El tercer y último paso es atender a los comandos recibidos del usuario.

Configuración e Inicialización.

Hasta ahora hemos hablado de cómo trabaja todo una vez en marcha, a continuación vamos a comentar como si inicializa.

En el fichero *main.c* se realiza parte de la configuración y la inicialización. Como configuración destacar que es aquí donde se incluyen todos los controladores de los sensores. La primera tarea es crear y configurar el proceso de comunicaciones y repartidor, aunque aun no comenzará a ejecutarse hasta que no arranquemos el planificador.

El siguiente paso es inicializar el Bus I2C, muchos otros procesos utilizan el bus I2C en la inicialización para fijar el hardware a un estado conocido.

A continuación se crean las tres colas para los módulos de locomoción, servos y sensores. Se continua con la creación e inicialización de las tareas correspondientes, como ya disponemos del módulo de sensores configurado, el siguiente paso es inicializar los controladores de los sensores, el orden en que estos son inicializados influye en qué números de sensor tendrán, pero no en su identificador alfanumérico, que es el que debería usar cualquier software cliente para identificar los dispositivos conectados.

El último paso es activar el planificador del FreeRtos, una vez realizada la llamada a *vTaskStartScheduler();* el planificador comenzará y el sistema comenzará a funcionar.

Integración en Pyro

Si hablamos del trabajo sobre el Pyro, puedo comentar que es una de las tareas más duras, pero a la vez una de las mejores decisiones. Si uno de los objetivos era la docencia, que mejor que un entorno elaborado con multitud de recursos, y abstracción, para aislar al usuario de detalles técnicos que desviarían el aprendizaje a áreas que no son el motivo principal del curso o la clase.

Entorno Pyro.

El Pyro esta desarrollado en el lenguaje python, de los más rápidos entre los lenguajes interpretados. En el entorno Pyro hay cuatro áreas principales: Mundo, robot, dispositivos y cerebro, como podemos observar en la siguiente figura.

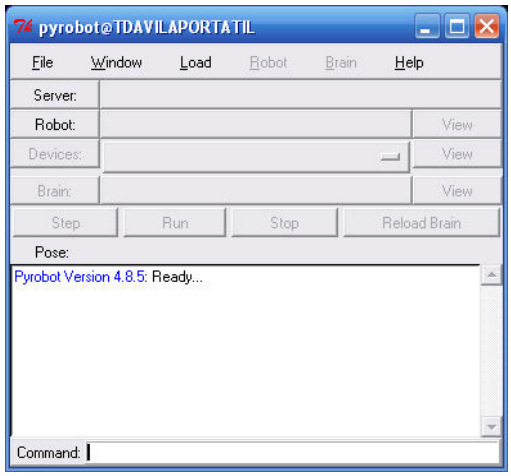


Figura 8: *Pantalla principal del entorno Pyro, se pueden observar las cuatro secciones principales : "Server","Robot", "Devices" y "Brain".*

El apartado Servidor (Server) es un programa python con un formato de clases especial que implementa un mundo utilizando un simulador, si este programa no es cargado el Pyro asumirá que se trabaja sobre el robot real.

El Robot es un grupo de clases que implementan un grupo mínimo de funciones básicas y parte o todos sus dispositivos. Los robots pueden ser dispositivos físicos y/o software, que pueden rodar en el mundo real o sobre un servidor (simulación) o sobre los dos.

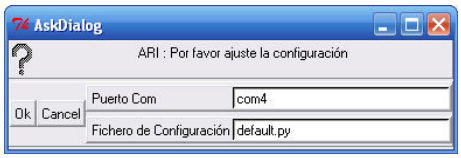
La pestaña "Devices" contendrá una vez cargado el robot todos los dispositivos de los que éste dispone, desde unidades de actuación como puede ser el control de la unidad de pan & tilt, hasta los dispositivos de ultrasonidos.

En la pestaña “Brain” se carga el comportamiento del robot. Un “cerebro” puede basarse exclusivamente en la abstracción de Pyro por lo que será completamente compatible con cualquier robot implementado, como también puede hacer uso de las características mas avanzadas de un robot determinado.

En este proyecto se ha implementado en Pyro un robot, desde las funciones básicas requeridas por el Pyro, como las funcionalidades más avanzadas de nuestro robot. , hasta las funcionalidades más avanzadas de nuestro robot.

El Robot según Pyro

El Robot del proyecto se cargar (una vez instalado sobre el entorno) abriendo desde el botón de “Robot” el fichero que se encuentra en “/pyrobot/plugins/robots/Ari.py”, lo primero que nos saldrá es una ventana pidiéndonos dos parámetros, uno el puerto al que esta conectado el robot, un puerto serie, y otro la configuración del robot.



La pantalla inicial de configuración del robot.

La configuración es un fichero python, en el que se declara una lista global de tuplas, que determinan que dispositivos serán cargados, las tuplas contienen los datos de configuración de cada dispositivo, entre los que se encuentran el nombre a mostrar en el entorno, y el código alfanumérico correspondiente de la unidad de control, así como la distribución física de estos, requisito necesario del Pyro.

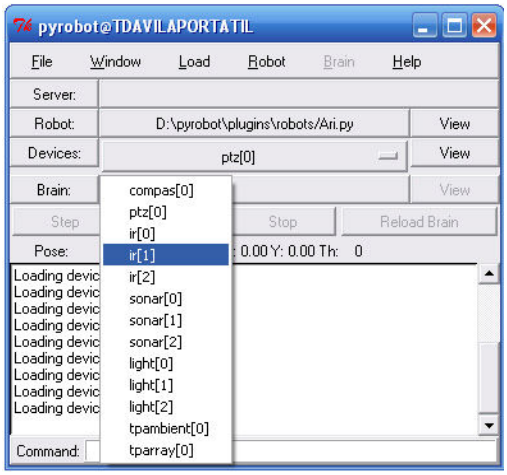
El enlace entre los dispositivos de la unidad de control y el entorno Pyro se realiza mediante nombre. De esta forma el cambio de orden de los dispositivos en la unidad de control no afecta al módulo del Pyro.

La abstracción realizada por el Pyro es muy sencilla, para el movimiento simplemente se deben implementar tres métodos de la clase robot, “move”, “rotate” y

“translate”, estos métodos permiten desplazar el robot independientemente de cómo este construido y su medio de locomoción.

De los sensores la única abstracción disponible es el dato miembro “Range” donde se espera una colección de sensores de distancia, que independientemente de cómo estén construidos, o su método de trabajo, su abstracción devuelve la distancia en metros.

Podemos ver la lista de dispositivos disponibles haciendo clic en “devices”. Cada dispositivo tiene asociada una ventana. Dependiendo del mismo, su contenido puede ser, desde un sencillo campo de texto que indica el valor actual, hasta ventanas elaboradas que muestran gráficamente la información o botones para la configuración.

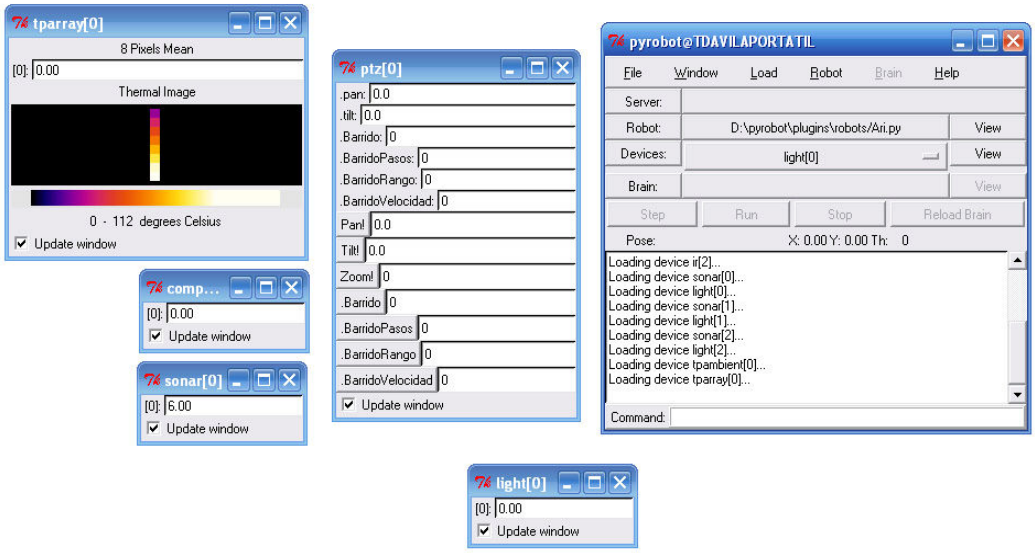


Lista de dispositivos del robot.

Los Dispositivos disponibles son los que vimos anteriormente, aunque la información mostrada por estos es la misma que se obtiene de la unidad de control, tenemos que hacer mención especial al dispositivo “ptz”, este dispositivo controla la unidad de “pan & tilt” o torreta.

Para poder explicar como de importante es la funcionalidad de este nuevo dispositivo vamos a explicar como trata el Pyro los dispositivos.

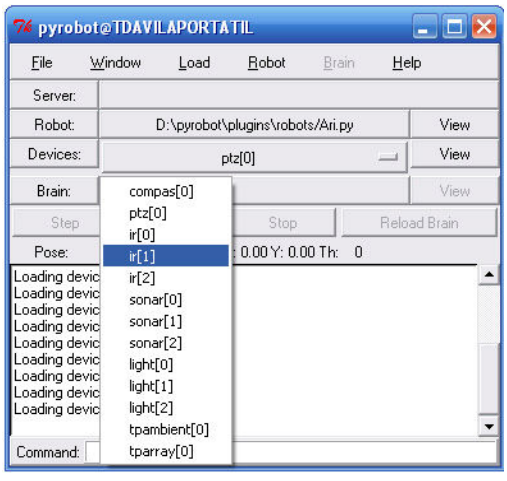
Los dispositivos tienen una información física asociada, y normalmente se agrupan por tipo, es decir, si un robot dispone de 8 dispositivos sonar, estos estarán en un solo dispositivo, además Pyro permite interrogar a estos 8 dispositivos según su localización en el robot, si delante, detrás, delante a la izquierda, o todo lo de delante entre muchas más opciones. De esta forma permite al programador del comportamiento inteligente, y al comportamiento mismo, abstraerse de la forma y localización exacta de los sensores



Cada dispositivo dispone de su propia ventana,
en la que podemos observar los datos y configurar detalles.

De los sensores la única abstracción disponible es el dato miembro "Range" donde se espera una colección de sensores de distancia, que independientemente de cómo estén construidos, o su método de trabajo, su abstracción devuelve la distancia en metros.

Podemos ver la lista de dispositivos disponibles haciendo clic en "devices". Cada dispositivo tiene asociada una ventana. Dependiendo del mismo, su contenido puede ser, desde un sencillo campo de texto que indica el valor actual, hasta ventanas elaboradas que muestran gráficamente la información o botones para la configuración.



Lista de dispositivos del robot.

Los Dispositivos disponibles son los que vimos anteriormente, aunque la información mostrada por estos es la misma que se obtiene de la unidad de control, tenemos que hacer mención especial al dispositivo "ptz", este dispositivo controla la unidad de "pan & tilt" o torreta.

Para poder explicar como de importante es la funcionalidad de este nuevo dispositivo vamos a explicar como trata el Pyro los dispositivos.

Los dispositivos tienen una información física asociada, y normalmente se agrupan por tipo, es decir, si un robot dispone de 8 dispositivos sonar, estos estarán en un solo dispositivo, además Pyro permite interrogar a estos 8 dispositivos según su localización en el robot, si delante, detrás, delante a la izquierda, o todo lo de delante entre muchas más opciones. De esta forma permite al programador del comportamiento inteligente, y al comportamiento mismo, abstraerse de la forma y localización exacta de los sensores.

Desarrollo

El entorno Pyro como ya se ha comentado funciona sobre Python, que es un lenguaje orientado a objetos, a continuación veremos como esta construido el módulo del robot en el Pyro.

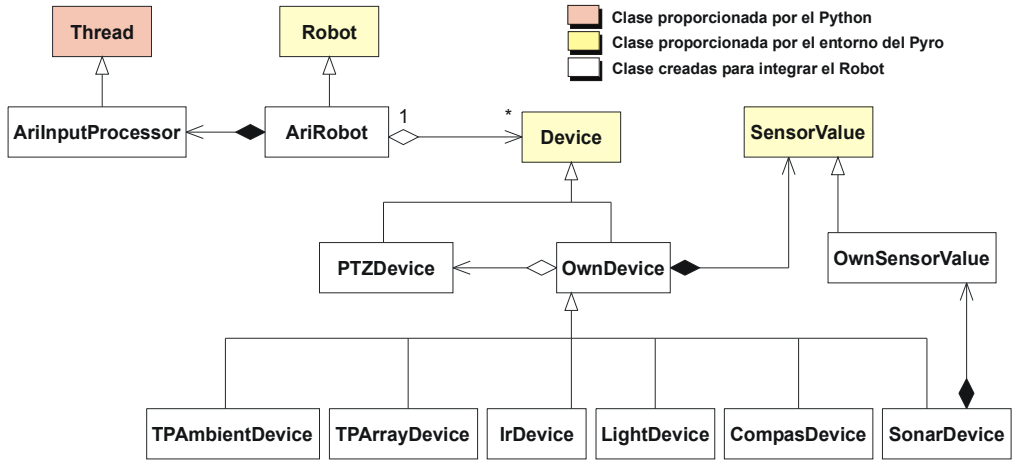


Diagrama de Clases del módulo del robot desarrollado para el Pyro. En el diagrama de clases la flecha indica relación de herencia el diamante vacío relación de uso en la interfaz y el diamante rellenado relación de contenido.

La clase principal del módulo es AriRobot que hereda de la clase Robot del Pyro, e implementa las funciones mínimas requeridas por esta última.

La clase principal del módulo es **AriRobot** que hereda de la clase **Robot** del Pyro, e implementa las funciones mínimas requeridas por esta última. Contiene un objeto de la clase **AriInputProcessor** y usa objetos de la clase **Device**

La clase **AriInputProcessor** hereda de la clase **thread y** se encarga de procesar todos los mensajes recibidos del puerto serie, normalmente los datos sensibles son dejados en un diccionario[10], que será accedido por los sensores.

[10] Un Diccionario es un tipo de datos básico de Python que es como una lista pero esta indexada por una tabla hash, y permite indexar por cualquier cosa, cadenas, objetos y mas.

La clase **Device** ha sido ampliada en **OwnDevice,** puede llevar asociado un dispositivo de pan & tilt o **PTZDevice,** en este último en cada cambio de configuración del barrido se generará una lista con las nuevas posiciones. La lista pasa a la clase **OwnDevice,** que se encarga de mantenerla con los valores obtenidos en cada posición.

La clase **SensorValue**, es la clase de objetos que se devuelve de una lectura (el usuario recibe listas de esta clase instanciada con datos), la instancia de **SensorValue** contiene el valor de lectura asociado, posición y orientación.

Todas las clases: **TPAmbientDevice, TPArrayDevice, IrDevice, LightDevice, CompasDevice y SonarDevice** implementan los dispositivos. Los hay sencillos como **CompasDevice,** que simplemente sobrecarga el miembro **get_val()** para transformar los datos a un formato adecuado, en este caso el ángulo que entra respecto del norte magnético, viene en grados multiplicado por diez, y lo que hace simplemente es dividir entre diez.

En Cambio **TPArrayDevice** muestra un grafico con los valores recibidos de la termopila, o el **SonarDevice** modifica un par de métodos más para poder devolver en lugar de **SensorValue, OwnSensorValue**. Se ha tomado la decisión de ampliar la clase **SensorValue** porque los Sonars no devuelven solo un valor sino hasta dieciséis valores en una lectura.

Para una descripción mas detallada consulte el apartado correspondiente en el Apéndice II.

Resultados, Conclusiones y Mejoras

En apartados anteriores se pudo ver porque se descartó el OOPic, pero ahora toca evaluar la alternativa. En un primer paso evaluaremos la capacidad de trabajo de la unidad de control:

La unidad de control es capaz de leer hasta 25 sensores de forma periódica a una cadencia de una décima de segundo, además de recibir más de 30 órdenes por segundo desde el cliente, se puede extraer que es un sistema considerablemente más ágil, como mínimo comparado con el rendimiento anteriormente obtenido en el OOPic. Con esta mejora sustancial permite un movimiento relativamente dinámico del autómata.

También se han programado y probado con éxito comportamientos inteligentes basados en lógica difusa, otros ejemplos son comportamientos fotofóbicos y fotofílicos aprovechando los sensores de luz incorporados en los dispositivos de ultrasonidos.

En la vida real, el coste tiene una gran importancia en el desarrollo de cualquier producto, dispositivo o software y minimizar este es como en cualquier negocio, importante para aumentar las ganancias.

En este proyecto el coste ha ascendido aproximadamente a 1400 euros, que es poco menos que el coste de un Aibo para un cliente con fines académicos. Pero comparado, es mucho más potente en capacidad de cómputo, mucho más versátil dado que se conoce como esta construido, podremos modificarlo para una tarea concreta.

Otra ventaja muy subestimada hoy en día es la capacidad de mantenimiento interno, actualmente los mecanismos y sistemas son tan sofisticados, y existe tanto secreto industrial que prácticamente muy pocos productos pueden ser mantenidos enteramente por el usuario final. Así que poder disponer de esta característica es un valor añadido

Mejoras

Desde la premisa que la perfección no existe, este último razonamiento permite inducir que todo es mejorable. Partiendo del anterior razonamiento vamos a ver que partes del robot son susceptibles de mejora.

Si en la unidad de control disponemos de un microcontrolador más rápido podríamos obtener lecturas de los sensores a mas frecuencia o poder leer mas sensores, si cambiamos la comunicación serie por USB obtendremos mayor velocidad a la hora de transferir los datos a la unidad central, la velocidad de la comunicación aumentaría de aproximadamente 7kbytes/s a 23kbytes/s, en una primera y sencilla aproximación al USB.

Respecto al software de control, se podría crear funcionalidad para cargar desde el puerto serie nuevos sensores sin tener que volver a compilar y grabar el microcontrolador otra vez.

Otra funcionalidad sería dotar de algún tipo de reflejos al robot, imitar el hecho de que el cerebelo animal reacciona automáticamente a ciertos estímulos básicos. En el robot por ejemplo: Se detendría justo antes de impactar y avisaría a la unidad principal del problema encontrado.

Adoptar una distribución Linux en lugar de Windows XP en el ordenador incorporado nos proporcionaría funcionalidades de Pyro que no están disponibles en los sistemas de Microsoft, no habría problema con el software desarrollado dado que esta escrito en python, que es portable a multitud de sistemas operativos.

Bibliografía y Referencias

Libros:

1. GADRE, Dhananjay V. *Programming and customizing the AVR microcontroller*. Nueva York: McGraw-Hill, 2001. 339p. ISBN 007134666X.

2. CLARK, Dennis. *Programming and Customizing the OOPic Microcontroller: The Official OOPic Handbook*. 1º Edición. McGraw-Hill/TAB Electronics; 2003. 352p. ISBN 0071420843.

Referencias Electrónicas:

3. Atmel Corporation. *ATMEGA128 (L) Summary*. [En línea]. 2002. Atmel Corporation, 2002, Última revisión: 2006, [2007].
http://www.atmel.com/dyn/resources/prod_documents/2467S.pdf

4. Atmel Corporation. *ATMEGA128 (L) Reference* [En línea]. 2002. Atmel Corporation, 2002, Última revisión: 2006, [2007].
http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

5. Python Software Foundation. *Python Documentation* [En línea]. 2007. Python Software Foundation, 2007, [2007].
http://docs.python.org

6. Pyro, Python Robotics. *Pyro Homepage* [En línea]. 2007. Blank, D.S., Yanco, H., Kumar, D., and Meeden L, 2007, [2007].
http://pyrorobotics.org

7. Savage Innovations. *OOPic* [En línea]. 1999-2007. Savage Innovations, 1999-2007, [2006-2007].
http://www.oopic.com

8. Microchip Inc. *Pic 16F873/874/876/877 Datasheet* [En línea]. 1998-2005. Microchip Inc, 1998-2005, [2006-2007].
http://ww1.microchip.com/downloads/en/DeviceDoc/30292c.pdf

9. NXP Semiconductors. *I2C Documentation* [En línea]. 2000-2007. NXP Semiconductors (fundada por Philips), 2000-2007, [2006-2007]. ***http://www.nxp.com/products/interface_control/i2c/***

10. Via Technologies Inc. *Via Epia MII-10000* [En línea]. 2000-2007. Via Technologies Inc., 2000-2007, [2006-2007]. ***http://www.via.com.tw/en/products/mainboards/motherboards.jsp?motherboard_id=202***

11. Richard Barry. *FreeRTOS* [En línea]. 2000-2007. Richard Barry., 2003-2007, [2006-2007]. ***http://www.freertos.org***

12. Peter Fleury. *AVR-GCC Software* [En línea]. 2006. Peter Fleury., 2006-2007, [2006-2007]. ***http://homepage.hispeed.ch/peterfleury/avr-software.html***

Apéndice I

Especificaciones Hardware

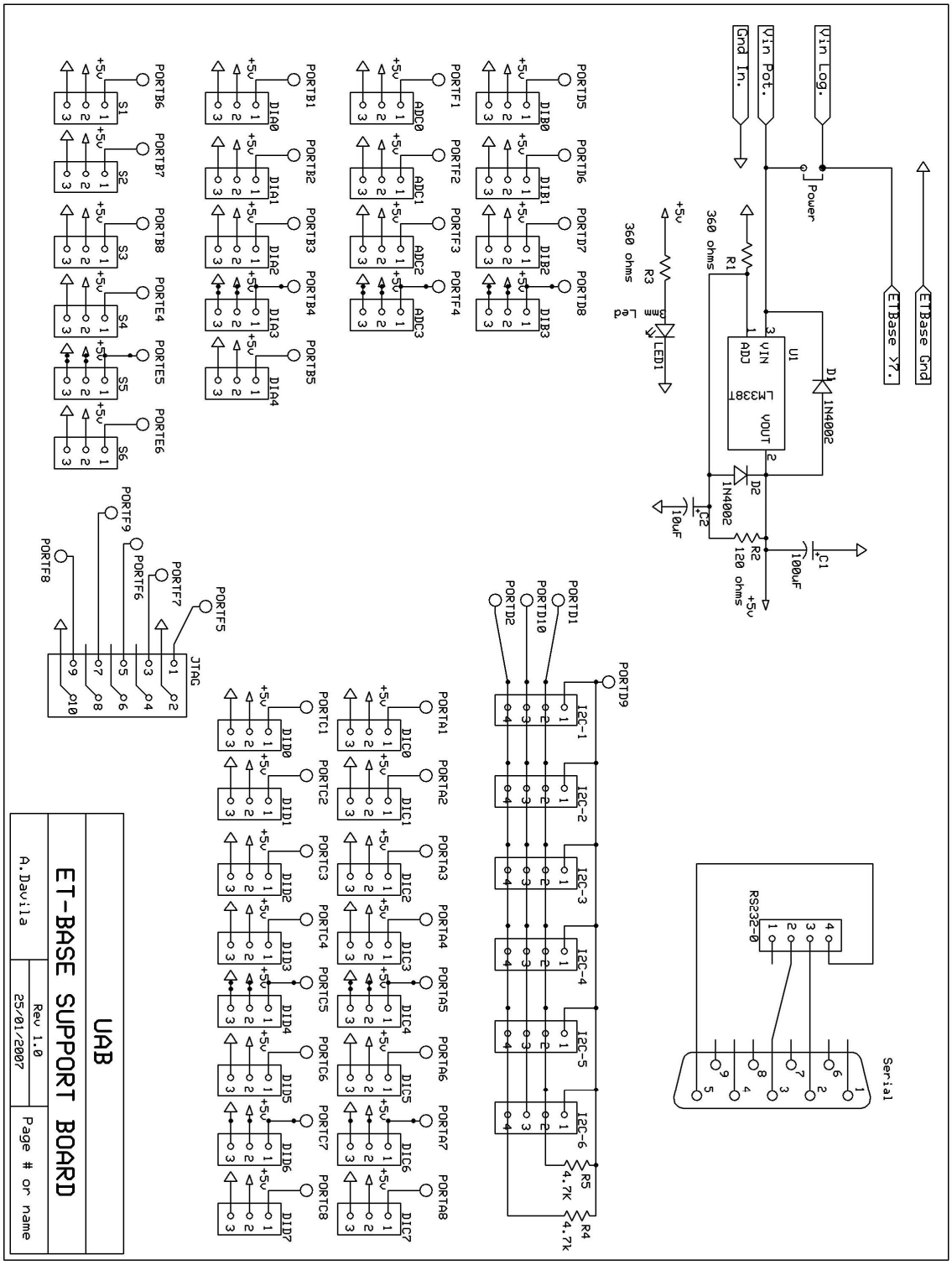
Detalles de la placa de soporte

Prototipo:

Listado de Componentes

Cant.	Id.	Descripción.
1	U1	Regulador de Voltaje LM338T
1	-	Radiador para LM338T
2	R1,R3	Resistencias 360 Ohms.
1	R2	Resistencia 120 Ohms.
2	R4,R5	Resistencias 4,7 Ohms.
1	LED1	Diodo led.
1	C1	Condensador electrolítico de 100uF
1	C2	Condensador electrolítico de 10uF
2	D1,D2	Diodo 1N4002.
1	Serial	Conector Sub D 9 Pins Hembra
1	RS232-0	3 Pin .100” Polarized Header Connector
1	JTAG	IDC 10 Pin Shrouded Polarized Male Headers
2	Vin, EttBaseIn	Clemas
23	Conectores	Tiras de 3 pins.
1	Conectores	Tira de 2 pins.
6	Conectores	Tira de 4 pins.

Esquema de conexionado:

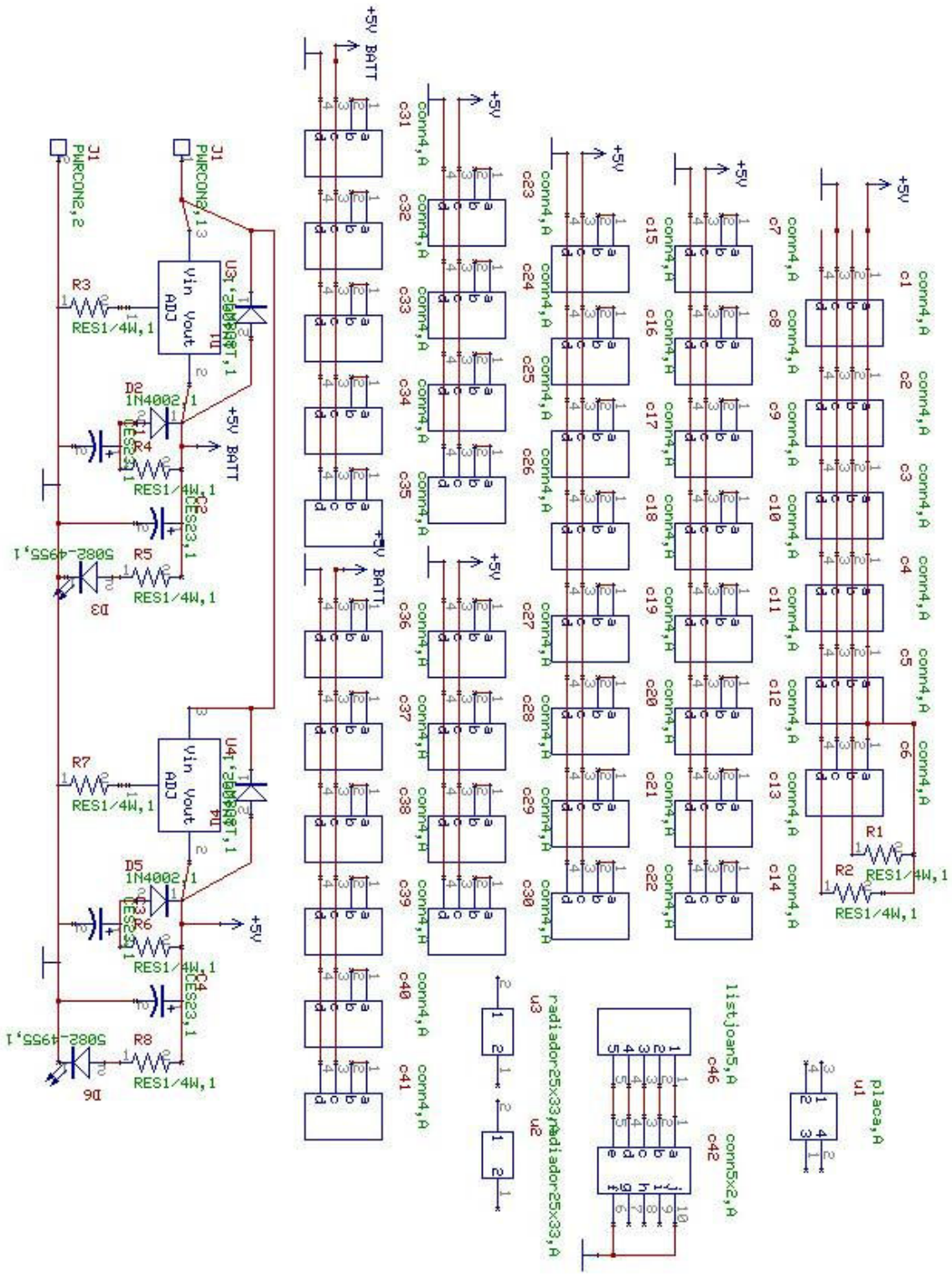


Revisión 1

Listado de Componentes:

Cant.	Id.	Descripción.
2	U1	Regulador de Voltaje LM338T
2	-	Radiador para LM338T
4	R1,R3	Resistencias 360 Ohms.
2	R2	Resistencia 120 Ohms.
2	R4,R5	Resistencias 4,7 Ohms.
2	LED1	Diodo led.
2	C1	Condensador electrolítico de 100uF
2	C2	Condensador electrolítico de 10uF
4	D1,D2	Diodo 1N4002.
1	Serial	Conector Sub-D 9 Pins Hembra
1	RS232-0	3 Pin .100” Polarized Header Connector
1	JTAG	IDC 10 Pin Shrouded Polarized Male Headers
2	Vin, EttBaseIn	Clemas
29	Conectores	Tiras de 4 pins.

Esquema de conexionado:



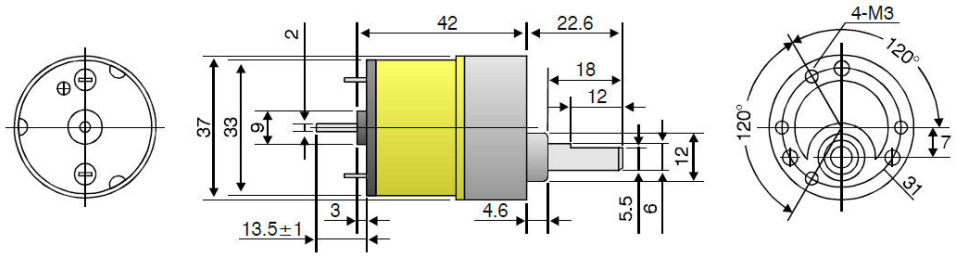
Especificaciones Técnicas:

Motor 7.2v DC con reductora 50:1: HN-GH7.2-2414T

Data sheet for:
GHM-04
7.2vdc 50:1 175rpm
6mm shaft

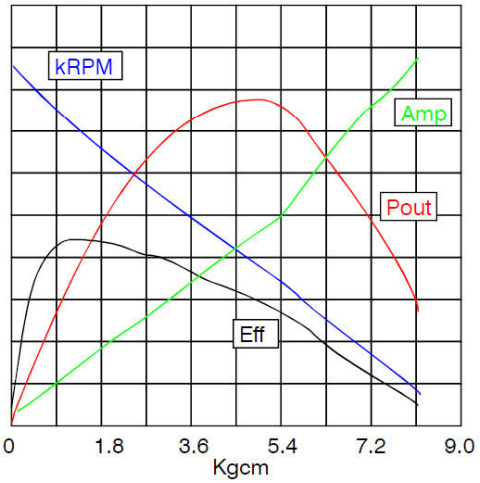
www.lynxmotion.com

I. OUTER DIMENSIONS



II. DRAWING OF CURVES

Pout	Amp	Eff	kRPM
5.0	4.0	1.0	0.20
4.5	3.6	0.9	0.18
4.0	3.2	0.8	0.16
3.5	2.8	0.7	0.14
3.0	2.4	0.6	0.12
2.5	2.0	0.5	0.10
2.0	1.6	0.4	0.08
1.5	1.2	0.3	0.06
1.0	0.8	0.2	0.04
0.5	0.4	0.1	0.02
0.0	0.0	0.0	0.00



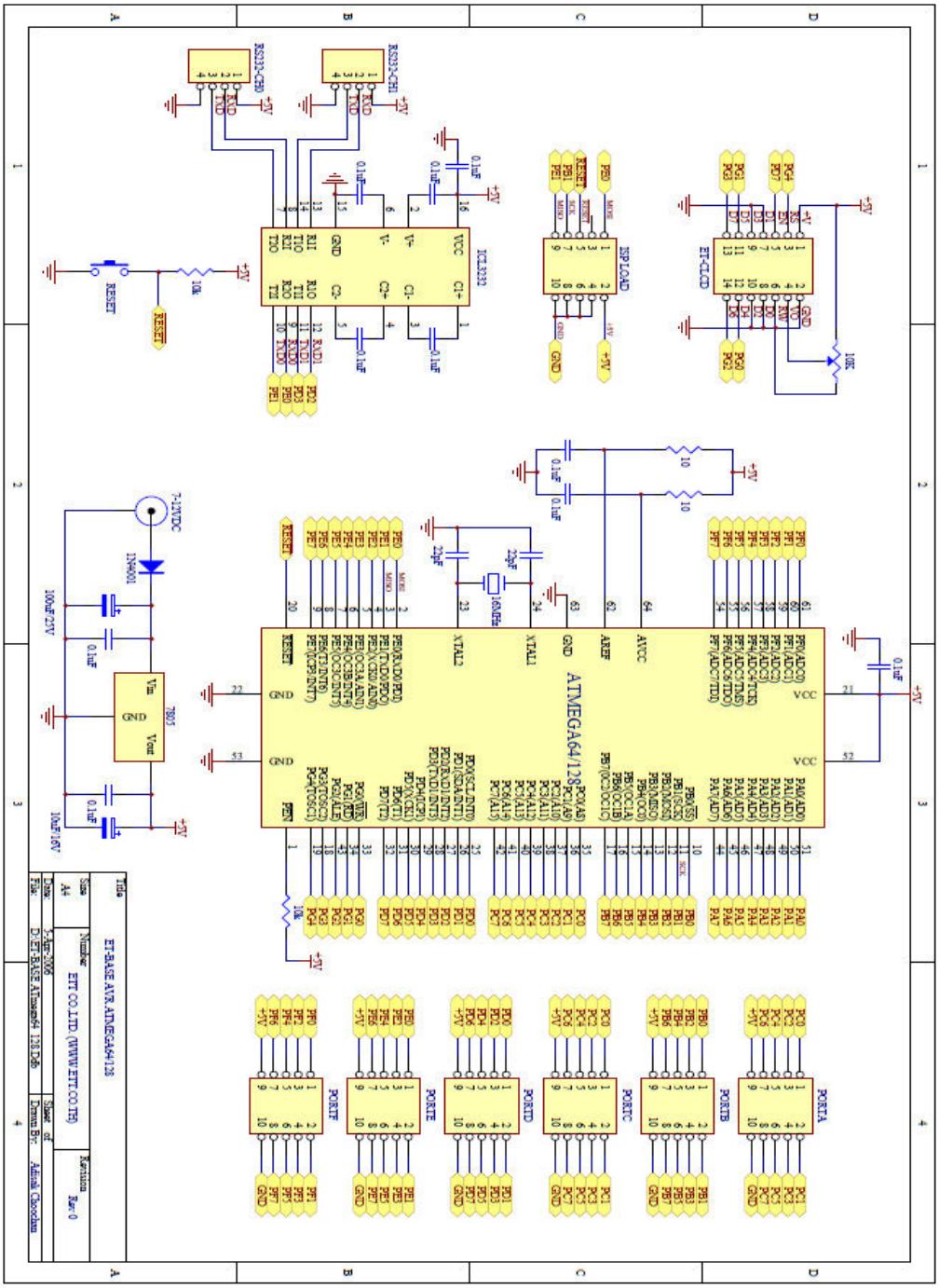
III. SPECIFICATIONS

Type: HN-GH35GMB
Model: HN-GH7.2-2414T - 50:1

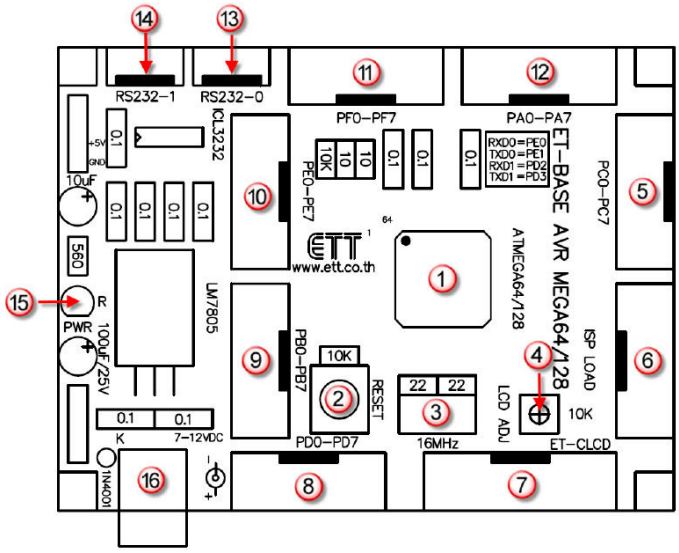
1. Testing Conditions:
   Temp: 25° Celsius
   Humidity: 60%
   Motor Orientation: Horizontal
2. Rated Voltage: 7.2vdc
3. Voltage Operating Range: 6-7.2vdc
4. Rated Load at 7.2vdc: 1.0Kg-cm
   Do not exceed rated load. Damage may occur!
5. No Load Speed at 7.2vdc: 175 RPM +/- 10%
6. Speed at Rated Load (1.0Kg-cm): 146 RPM +/- 10%
7. No Load Current at 7.2vdc: < 221mA
8. Current at Rated Load (1.0Kg-cm): < 556mA
9. Shaft End-Play: Maximum 0.8m/m
10. Insulation Resistance: 10M ohm at 300vdc
11. Withstand Voltage: 300vdc for 1 Second
12. The gear motor is not intended for instant reverse. The gear motor must be stopped before reversing.
13. The gear motor does not include protection from water or dust etc.

Placa ETT-BASE AVR MEGA128

Esquema:



Estructura:



1. Microcontrolador Atmel AVR ATMEGA 128.
2. Botón de reset
3. Cristal oscilador de 16Mhz.
4. Resistencia para el control del contraste del LCD
5. Puerto C del microcontrolador
6. Puerto ISP (In circuit Serial Programming).
7. Puerto pantalla LCD HD44780 compatible en modo 4 bits.
8. Puerto D
9. Puerto B
10. Puerto E
11. Puerto F
12. Puerto A
13. Puerto serie con señales adaptadas
14. Puerto serie con señales adaptadas.
15. Led testigo de alimentación.
16. Conector de alimentación.

Controlador de motores MD22

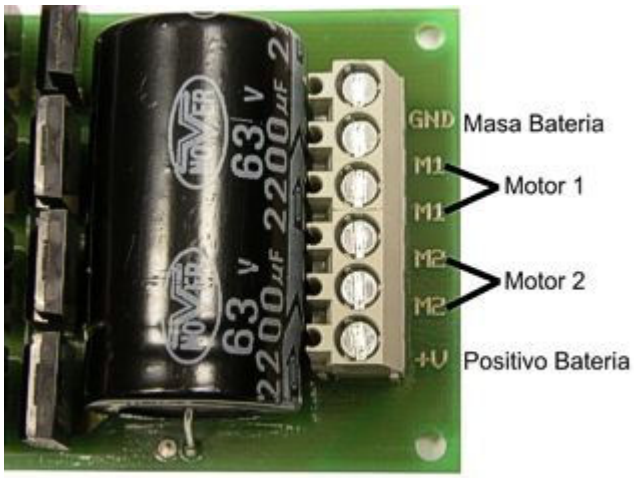
Fuente: ***www.superrobotica.com***

El MD22 es un controlador para dos motores de corriente continua de mediana potencia, diseñado para proporcionar mas potencia que los controladores basados en un único circuito integrado. Las principales características son la facilidad de uso y la flexibilidad.

Los 15V de la tensión de control del MOSFET se genera en el mismo circuito mediante una bomba de carga, por lo que solo se requieren 5V a 50 mA para la alimentación del circuito, además de la alimentación del motor que esta comprendida entre los 5 y los 50V dependiendo de los requerimientos del motor.

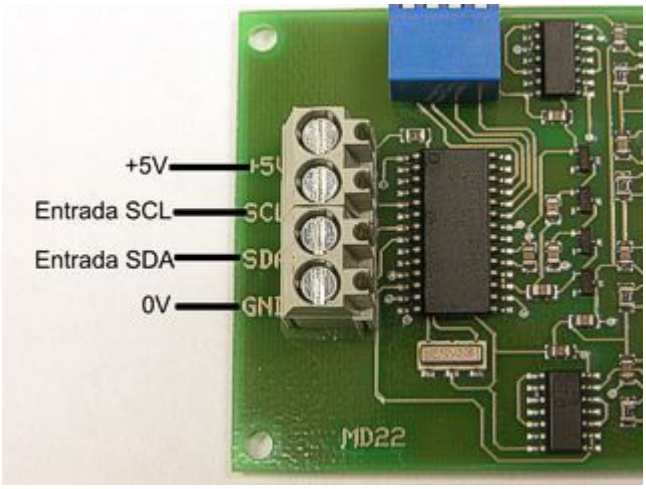
Conexiones del Motor

Recuerde utilizar siempre cables con una sección adecuada a la potencia del motor para hacer las conexiones del motor y de la batería. Igualmente es imprescindible intercalar entre la entrada de la batería y la MD22 un fusible de de 10 amperios.



Conexiones del Control

Las conexiones indicadas en el circuito impreso son las correspondientes al modo I2C del controlador MD22. Para las conexiones de los otros modos de funcionamiento, lea el texto mas adelante.



Modos de Funcionamiento

El circuito controlador de motores MD22 tiene 5 modos diferentes de funcionamiento que son:

Modo Diferencial

Además de controlar dos motores de manera independiente, el controlador MD22 es capaz de controlar dos motores que están colocados de forma opuesta uno respecto al otro y que por lo tanto controlan el movimiento hacia la izquierda o la derecha dependiendo del motor que este girando. En este modo, el MD22 puede utilizar un canal para controlar la velocidad y el otro para controlar la dirección.

Modo Analógico 0V - 2,5V - 5V

En este modo, los motores son controlados independientemente por dos señales analógica de 0 a 5 voltios en la entrada SCL para el motor 1 y la entrada SDA para el motor 2. En este modo 0V es el máximo en un sentido, 2,5V es la posición central de reposo o posición de parada y 5V corresponde al máximo en el otro sentido.

Hay una pequeña zona muerta de un 2,7% de ancho en la zona central, para proporcionar una zona de apagado estable. La impedancia de entrada es de 47K.

Modo Analógico 0V - 2,5V - 5V con control de Dirección.

En este modo el motor la velocidad de ambos motores es controlada por una señal analógica de entre 0V y 5V conectada a la entrada SCL y la dirección se controla mediante una entrada de =V y 5V en la entrada SDA. El control de la velocidad es igual que en el modo anterior, mientras que el control de giro corresponde a 2,5V para ir recto y 0v y 5v para girar a izquierda y derecha respectivamente. También una zona muerta de un 2,7% de ancho en la zona central, para proporcionar una zona neutral estable en ambos canales.

Modo RC (radio control)

Este modo permite la conexión directa a un receptor de radio control estándar, es decir que se controla como si fuera un servo. La mayoría de los receptores comerciales trabajan con un paquete de baterías de 4,8 a 6V y pueden ser alimentados por los mismos 5V empleados para alimentar el circuito controlador de motores MD22. El pulso de control (amarillo en muchos servos) del receptor debe conectarse al terminal SCL para el motor 1 y al terminal SDA para el motor 2. Conecte el cable de alimentación (rojo) del receptor al terminal de +5V y el cable de tierra (negro) al terminal de tierra del módulo controlador.

La salida del receptor es un pulso alto de 1,5 ms de ancho cuando el joystick esta en el centro. El rango de trabajo del MD22 proporciona un control total en el rango desde 1mS hasta los 2mS siendo 1,5mS la zona central correspondiente a la parada. Hay una zona muerta de 7uS entorno a la zona central para facilitar la posición de parada. El control de centrado del radio transmisor deberá ajustarse de forma que el motor este parado cuando la palanca de control este en su posición de reposo.

Modo RC con Control de Dirección

Este modo es similar al Modo RC, pero en esta ocasión se utiliza un canal conectado a la entrada SCL para controlar la velocidad y el sentido de giro de ambos motores simultáneamente, mientras que el otro canal se conecta a la entrada SDA para controlar la dirección mediante el control diferencial de ambos motores.

Modo I2C

El modo I2C permite conectar el circuito controlador MD22 a controladores como el Basicx24, el OOPic y el Basic Stamp por citar solo unos cuantos o bien a microcontroladores como PIC, 8051, H8, etc.

El protocolo de comunicación I2C del circuito controlador MD22 es el mismo que el empleado en las conocidas eeprom como la 24C04. Para leer uno o más registros del MD22, primero se envía un bit de comienzo seguido de la dirección del módulo (0XB0 es la dirección base del módulo) con el bit de lectura/escritura puesto a cero. Después se manda el numero del registro que desea leer seguido de nuevo de un bit de comienzo y otra vez la dirección del módulo con el bit lectura/escritura puesto a 1(0Xb1). Ahora puede leer uno o más registros.

Registros del Modo I2C

El MD22 tiene 8 registros numerados del 0 al 7 tal y como se muestran en la siguiente tabla.

Dirección Registro	Nombre	LecturaR / EscrituraW	Descripción
0	Modo	R/W	Modo de funcionamiento (ver mas adelante)
1	Velocidad1	R/W	Velocidad Motor Izquierdo (modo 0,1) o Velocidad (modo 2,3)
2	Velocidad2/Giro	R/W	Velocidad Motor Derecho(modo 0,1) o Giro (Modo 2,3)
3	Aceleración	R/W	Aceleración para I2C
4	Sin Usar	Lectura	Devuelve 0
5	Sin Usar	Lectura	Devuelve 0
6	Sin Usar	Lectura	Devuelve 0
7	Versión	Lectura	Número de Revisión del Software

Registro de Modo

El registro de modo tiene por defecto un valor 0 y selecciona el modo de funcionamiento de acuerdo a los siguientes valores:

0 Cuando el registro tiene un valor 0 el significado de los registros de velocidad 1 y 2 tienen un significado literal de 0 para atrás todo, 128 para parada y 255 para adelante todo.

1 Este modo es similar al modo 0 pero en este caso el valor de los registros de velocidad se tratan como valores con signo, es decir que -128 corresponde a atrás todo, 0 es parado y 128 es todo adelante.

2 Escribiendo un 2 en el registro de modo se consigue que el registro velocidad 1 controle la velocidad de ambos motores y el registro velocidad 2 controle el giro. El rango se encuentra entre 0 en un extremo, 128 que es el valor central y 255 que es el otro extremo.

3 El modo 3 es similar al modo 2, pero los valores son considerados como valores con signo, por lo que el rango va desde -128 para un extremo, 0 para el centro y 128 para el otro extremo.

Registro de Velocidad 1

Dependiendo del modo de funcionamiento, este registro afecta a la velocidad de uno o ambos motores. Cuanto mayor sea el valor escrito, mas potencia se aplica al motor.

Registro de Velocidad 2 / Giro

En los modos 0 y 1 este registro funciona de forma similar al Registro de Velocidad 1, mientras que en los modos 2 y 3 se convierte en el registro de control del giro y entonces el valor del registro de velocidad 1 es combinado con el contenido de este registro para realizar el giro.

Registro de Aceleración

Este registro establece el ratio a la que el motor acelera o desacelera desde su velocidad actual hasta la velocidad indicada en el Registro de Velocidad. Los valores admitidos son de 0 a 255 y cuanto mayor sea este valor menos tiempo tardara el motor en alcanzar la velocidad indicada. Escribiendo un 255 en este registro, se consigue la máxima aceleración. El valor introducido controla la velocidad a la que el controlador varía la velocidad entre la velocidad actual y la introducida en el registro de velocidad. Con el valor de 0 se cambia la velocidad (acelera) a su mínima velocidad tomando cada paso 16,4 ms. Cuando el registro de aceleración esta en su máximo valor de 255 (valor por defecto) el controlador cambia la velocidad cada 64 uS. El cálculo del tiempo en

segundos se hace mediante la siguiente formula:

Tiempo = (256-Registro de aceleración) * 256

Por ejemplo:

Reg Acel	Tiempo/paso	Velocidad Actual	Nueva Velocidad	Pasos	Tiempo de Aceleración
0	16.4ms	0	255	255	4.18s
20	15.1ms	127	255	128	1.93s
50	13.2ms	80	0	80	1.06s
100	10ms	45	7	38	0.38s
150	6.8ms	255	5	250	1.7s
200	3.6ms	127	0	127	0.46s
255	64us	65	150	85	5.4ms

Registro de Versión del Software

La lectura de este registro devuelve el valor del software del microcontrolador PIC16F873 que lleva el MD22. La versión actual a la fecha del 22 de Abril del 2004 es la 1.

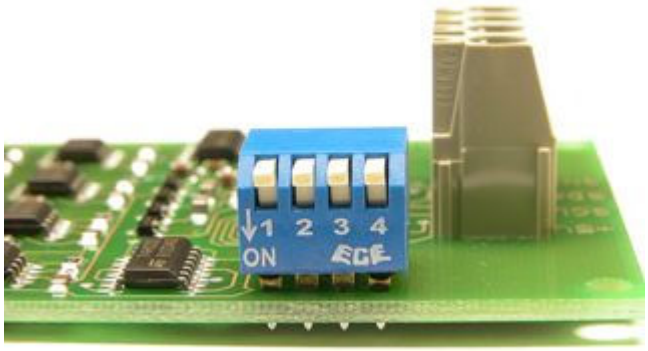
Modos de Funcionamiento

Hay cuatro micro interruptores numerados del 1 al 4 que se utilizan para establecer el modo de funcionamiento y en su caso la dirección del módulo dentro del bus I2C según se muestra en la siguiente tabla.

Modo	1	2	3	4
Dirección. Bus I2C 0xB0	On	On	On	On
Dirección. Bus I2C 0xB2	Off	On	On	On
Dirección. Bus I2C 0xB4	On	Off	On	On
Dirección. Bus I2C 0xB6	Off	Off	On	On
Dirección. Bus I2C 0xB8	On	On	Off	On
Dirección. Bus I2C 0xBA	Off	On	Off	On
Dirección. Bus I2C 0xBC	On	Off	Off	On
Dirección. Bus I2C 0xBE	Off	Off	Off	On
0v - 2.5v - 5v Analógico	On	On	On	Off
0v - 2.5v - 5v Analógico + Dirección	Off	On	On	Off
Radio Control	On	Off	On	Off
Radio Control + Dirección	Off	Off	On	Off

Las demás combinaciones no son validas y si se establecieran el led parpadeará y no ocurrirá nada más. Fíjese que las direcciones I2C corresponden a los 7 bits superiores. El bit 0 se emplea como bit de lectura / Escritura (R/W). De esta forma las direcciones 0XB0/0XB1 corresponden a la misma dirección. Pero de escritura y lectura respectivamente. El rango de direcciones es el mismo utilizado en el controlador MD22.

Los interruptores solo se leen durante la puesta en marcha, por lo que no deben realizarse cambios en los mismos mientras este conectado, ya que tampoco tendría ningún efecto.



Utilización del circuito controlador de motores MD22

El MD22 puede manejar grandes corrientes, por lo que es necesario tomar algunas precauciones al hacer el cableado. Es muy importante evitar que la corriente del motor retorne al circuito lógico a través de la masa. Esto quiere decir que no asuma que por el hecho de que se alimenta con una batería, las masas del circuito y del motor van juntas. Siempre que sea posible utilice dos baterías distintas para alimentar el motor y la electrónica. Tampoco conecte la masa de las baterías juntas, ya que esto se hace en el propio circuito impreso del controlador MD22, y si lo hiciera crearía un bucle de retorno de tierra con los consiguientes problemas.

Una de las formas más sencillas de conectar el circuito MD22 con un controlador programable en Basic como el Basicx24, es utilizar el Modo RC, para controlarlo como si fuera un servo. Para ello lo primero que tiene que hacer es seleccionar este modo de funcionamiento antes de conectar la alimentación, con la ayuda de los micro interruptores que se encuentran en el lateral del circuito. Ahora todo lo que queda por

hacer es mandar la orden pulse out para simular los pulsos de control de los servos y controlar los motores. El pulso tiene que variar entre 1ms y 2mS, siendo 1,5 ms la posición central. A diferencia de los servos que necesitan que los pulsos sean repetidos cada 20 ms, el circuito MD22 solo necesita un nuevo pulso cuando se quiere cambiar la velocidad, por lo que si no se envían más pulsos, simplemente se continúa con la misma velocidad. Los valores de los tiempos varían en función de los microcontroladores por lo que se muestra en la siguiente tabla algunos de los valores correspondientes a los microcontroladores mas conocidos.

Controlador	Resolución de Pulse out	Atrás Todo	Parado	Adelante Todo	Orden de ejemplo para Parado
BS2	2uS	500	750	1000	pulsout mot1, 750
BS2e	2uS	500	750	1000	pulsout mot1, 750
BS2sx	0.8uS	1250	1875	2500	pulsout mot1, 1875
BS2p	0.8uS *	1250	1875	2500	pulsout mot1, 1875
Atom	1uS	1000	1500	2000	pulsout mot1, 1500
BX-24	1.085uS	922	1382	1843	call pulseout(mot1, 1382, 1)

Sonar SRF08

Fuente: www.superrobotica.com

SRF08 es un medidor ultrasónico de distancias para robots que representa la ultima generación en sistemas de medidas de distancias por sonar, consiguiendo niveles de precisión y alcance únicos e impensables hasta ahora con esta tecnología. El sensor es capaz de detectar objetos a una distancia de 6 m con facilidad además de conectarse al microcontrolador mediante un bus I2C, por lo que se pueden conectar cuantos sensores sean necesarios en el mismo bus. Con una alimentación única de 5V, solo requiere 15 mA, para funcionar y 3mA mientras esta en reposo, lo que representa una gran ventaja para robots alimentados por pilas. El sensor SRF08 Incluye además un sensor de luz que permite conocer el nivel de luminosidad usando igualmente el bus I2C y sin necesidad de recursos adicionales.

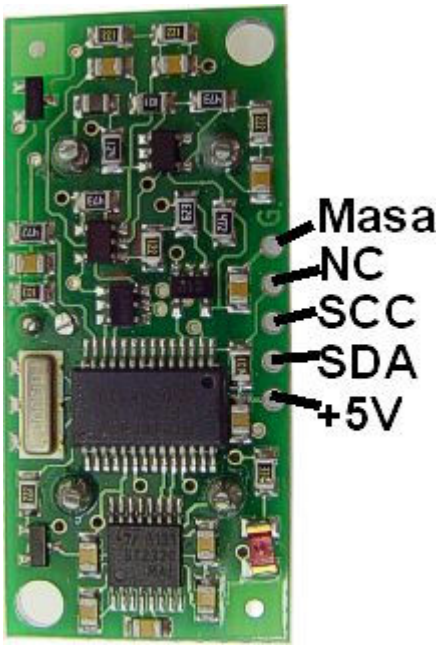
Controlando el sensor de distancias ultrasónico SRF08

La comunicación con el sensor ultrasónico SRF08 se realiza a través del bus I2C. Este está disponible en la mayoría de los controladores del mercado como BasicX-24, OOPic y Basic Stamp 2P, así como en una amplia gama de microcontroladores. Para el programador, el sensor SRF08 se comporta de la misma manera que las EEPROM de las series 24xx, con la excepción de que la dirección I2C es diferente. La dirección por defecto de fábrica del SRF08 es 0xE0. El usuario puede cambiar esta dirección con 16 direcciones diferentes: E0, E2, E4, E6, E8, EA, EC, EE, F0, F2, F4, F6, F8, FA, FC o FE, por lo que es posible utilizar hasta 16 sensores sobre un mismo bus I2C. Además de las direcciones anteriores, todos los sonares conectados al bus I2C responderán a la

dirección 0 -al ser la dirección de atención general. Esto significa que escribir un comando de medición de la distancia para la dirección 0 de I2C (0x00) iniciará las mediciones en todos los sensores al mismo tiempo. Esto debería ser útil en el modo ANN (Véase a continuación). Los resultados deben leerse de manera individual de cada uno de las direcciones reales de los sensores. Disponemos de ejemplos del uso de un módulo SRF08 con una amplia gama de controladores del mercado.

Conexiones

El pin señalado como “Do Not Connect” (No conectar) debería permanecer sin conexión. En realidad, se trata de la línea MCLR de la CPU y se utiliza solamente en la fábrica para programar el PIC16F872 después del montaje, dispone de una resistencia interna de tipo pull-up. Las líneas SCL y SDA deberían tener cada una de ellas una resistencia pull-up de +5v en el bus I2C. Sólo necesita un par de resistencias en todo el bus, no un par por cada módulo o circuito conectado al bus I2C. Normalmente se ubican en el bus maestro en vez de en los buses esclavos. El sensor SRF08 es siempre un bus esclavo - y nunca un bus maestro. Un valor apropiado seria el de 1,8 K en caso de que las necesitase. Algunos módulos como el OOPic ya disponen de resistencias pull-up por lo que no es necesario añadir ninguna más.



Registros

El sensor SRF08 tiene un conjunto de 36 registros.

Ubicación	Lectura	Escritura
0	Revisión de Software	Registro de comando
1	Sensor de luz	Registro de ganancia máx. (por defecto 31)
2	Byte alto de 1º eco	Registro de alcance de distancia (por defecto 255)
3	Byte alto de 2º eco	No disponible
----	----	----
34	Byte alto de 17º eco	No disponible
35	Byte bajo de 17º eco	No disponible

Solamente se puede escribir en las ubicaciones 0, 1 y 2. La ubicación 0 es el registro de comandos y se utiliza para iniciar la sesión de cálculo de la distancia. No puede leerse. La lectura de la ubicación da como resultado la revisión del software de SRF08. Por defecto, la medición dura 65mS, aunque puede cambiarse modificando el registro de alcance de la ubicación 2. Si lo hace, tendrá que cambiar la ganancia analógica en la ubicación 1. Consulte las secciones siguientes relacionadas con el cambio de medición y ganancia analógica.

La ubicación 1 es el sensor de luz en placa. Este dato se actualiza cada vez que se ejecuta un comando de medición de distancia y se puede leer cuando se leen los datos de la medición. Las dos ubicaciones siguientes, 2 y 3, son resultados sin signo de 16 bits de la última medición - el nivel lógico alto en primer lugar. El significado de este valor depende del comando utilizado, y puede estar expresado en pulgadas, o en centímetros, o bien el tiempo de vuelo del ping expresado en uS. Un valor cero indica que no se ha detectado objeto alguno. Hay hasta 16 resultados adicionales que indican los ecos de objetos más lejanos.

Comandos

Existen tres comandos para iniciar una medición de distancia (desde 80 hasta 82), que devuelve el resultado en pulgadas, centímetros o microsegundos. Asimismo, también existe un modo ANN (Artificial Neural Network) que se describe a continuación y un grupo de comandos para modificar la dirección de I2C del srf08.

COMANDOS		ACCIÓN
Decimal	Hexadecimal	
80	0X50	Modo cálculo distancia - Resultado en pulgadas
81	0X51	Modo cálculo distancia - Resultado en centímetros
82	0X52	Modo cálculo distancia - Resultado en microsegundos
83	0X53	Modo ANN - Resultado en pulgadas
84	0X54	Modo ANN - Resultado en centímetros
85	0X55	Modo ANN - Resultado en micro-segundos
160	0XA0	1º en la secuencia para cambiar la dirección I2C
165	0XA5	3º en la secuencia para cambiar la dirección I2C
170	0XAA	2º en la secuencia para cambiar la dirección I2C

Modo de cálculo de distancia con el SRF08

Para iniciar la medición de la distancia, deberá escribir uno de los comandos anteriores en el registro de comando (registro 0) y esperar el tiempo necesario para la ejecución de la operación. A continuación, deberá leer el resultado en el formato que desee (pulgadas, centímetros, etc.). El búfer de eco se pone a cero al comienzo de cada medición. La primera medición del eco se coloca en las ubicaciones 2 y 3, la segunda en 4 y 5, etc. Si una ubicación (niveles altos o bajos de bytes) es 0, entonces no se encontrará ningún otro valor en el resto de los registros. El tiempo recomendado y establecido por defecto para realizar la operación es de 65mS, sin embargo es posible acortar este periodo escribiendo en el registro de alcance antes de lanzar el comando de medición. Los datos del sensor de luz de la ubicación 1 se actualizarán también después del comando de medición.

Modo ANN

El modo ANN (Artificial Neural Network) ha sido diseñado para proporcionar datos múltiples de un modo en el que es más fácil entrar en una red neural, o al menos eso es lo que se pretende - aunque aún no se ha hecho. El modo ANN ofrece un búfer de 32 bytes (ubicaciones de 4 a 35 inclusive) en el que cada byte representa el tiempo

máximo de vuelo 65536uS dividido por 32 tramos de 2048uS cada uno - equivalente a aproximadamente 352mm de alcance. Si se recibe un eco en uno de los espacios de tiempo de bytes, a continuación se fijará en un valor diferente a cero, para que no sea cero. Por lo tanto si se recibe un eco desde los primeros 352mm, la ubicación 4 será diferente a cero. Si se detecta un objeto a 3 metros de distancia, la ubicación 12 será diferentes de cero (3000/352 = 8) (8+4=12). Organizar los datos de esta manera sería mejor para una red neural que para otros formatos. La entrada a su red debería ser 0 si el byte es cero y 1 si es diferente de cero. En el futuro, se pretende organizar un mapa SOFM (Self Organizing Feature Map) para la red neural, aunque se espera que sea aplicable para cualquier tipo de red.

Ubicación 4	Ubicación 5	Ubicación 6	Ubicación 7	Ubicación 8
0-352mm	353-705mm	706-1057mm	1058-1410mm	En adelante

Cómo comprobar que una medición ha finalizado

No es necesario utilizar un temporizador en su propio controlador para saber que la medición ha terminado. Puede aprovechar la ventaja que le ofrece el hecho de que el sensor SRF08 no responde a ninguna otra actividad I2C mientras está realizando la medición. Por lo tanto, si intenta leer el valor en el sensor SRF08 (utilizamos el número de revisión de software en la ubicación 0) por lo que recibirá 255 (0xFF) durante la medición. Esto se debe a que la línea de datos I2C (SDA) se eleva si nada lo está controlando. Tan pronto como finaliza la medición el sensor SRF08 responderá de nuevo al bus I2C, por lo que deberá esperar a que desaparezca el valor 255 (0xFF) en el registro. A continuación, podrá leer los datos del sensor. El controlador puede aprovechar esta ventaja para realizar otras tareas mientras el SRF08 está realizando la medición.

Cómo cambiar el rango de alcance

El alcance máximo del sensor SRF08 está controlado por el temporizador interno. Por defecto, este es 65mS o el equivalente a 11 metros de alcance. Esto supera los 6 metros de los que el SRF08 es realmente capaz de ofrecer. Es posible reducir el tiempo que espera el sensor SRF08 a escuchar un eco, y por lo tanto el alcance, modificando el registro range en la ubicación 2. El alcance puede regularse en pasos de aproximadamente 43mm (0,043 metros o 1,68 pulgadas) hasta llegar a los 11 metros. El

alcance es ((Range Register x 43mm) + 43mm) por lo que fijar este registro (Range Register) en el valor 0 (0x00) ofrece un alcance máximo de 43mm. Fijar el registro Range Register en el valor 1 (0x01) ofrece un alcance máximo de 86mm. En un ejemplo más útil, el valor 24 (0x18) ofrece un alcance de 1 metro mientras que el valor 140 (0x8C) da 6 metros. El valor 255 (0xFF) ofrece los 11 metros originales (255 x 43 + 43 es 11008mm).

Existen dos razones por las que es positivo reducir el tiempo de medición.

1. Para obtener la información sobre el alcance en menos tiempo
2. Para poder realizar mediciones con el sensor SRF08 a una tasa más rápida.

Si lo único que desee en recibir en menos tiempo, la información sobre el alcance y pretende realizar las mediciones a una tasa de 65ms o más lento, todo funcionará de manera correcta. Sin embargo, si desea lanzar el sensor SRF08 a una tasa ligeramente más alta de 65mS, deberá reducir la ganancia - consulte la siguiente sección.

El alcance está fijado en el valor máximo cada vez que se pone en marcha el sensor SRF08. Si necesita un alcance diferente, cámbielo al principio como parte del código de iniciación del sistema.

Ganancia analógica

En el registro de la ganancia analógica, se configura la ganancia máxima de las etapas analógicas. Para configurar la ganancia máxima del srf08, simplemente deberá escribir uno de estos valores en el registro de ganancia de la ubicación 1. Durante la medición, la ganancia analógica empieza con su valor mínimo de 94. Este valor se incrementa en intervalos de aproximadamente 70uS hasta llegar al valor de ganancia máxima, configurada en el registro 1. La ganancia máxima posible se alcanza después de aproximadamente 390mm de alcance. La finalidad de poder limitar la ganancia máxima es permitirle iniciar mediciones a una frecuencia mayor de 65mS. Dado que la medición puede ser muy corta, es posible iniciar una nueva medición tan pronto como se hayan leído los datos de la medición previa. Un riesgo potencial de esto es que la segunda medición podría captar un retorno de un eco distante del “ping” anterior, dando un resultado falso referente a un objeto cercano cuando en realidad no hay ninguno. Para reducir esta posibilidad, la ganancia máxima puede reducirse para limitar la

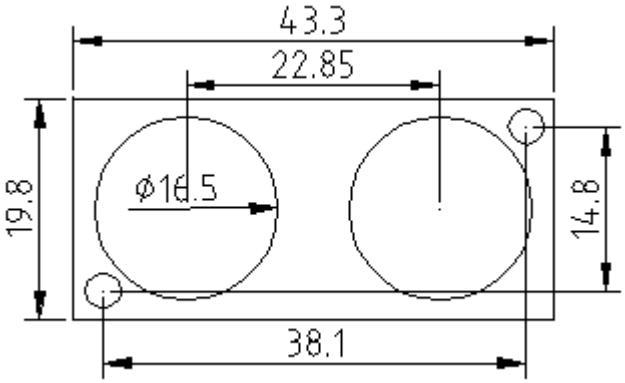
sensibilidad de los módulos al eco distante más débil, mientras que al mismo tiempo sigue siendo capaz de detectar la proximidad de objetos. La configuración de la ganancia máxima se almacena sólo en la memoria RAM del CPU y se inicia con el encendido del equipo, por lo que si sólo desea realizar las mediciones cada 65mS, o más, puede ignorar los registros Range y Gain.

Nota - Es efectivo sólo en Modo de cálculo de distancia, en el Modo ANN, la ganancia se controla automáticamente.

Registro de ganancia		Ganancia analógica máxima
Decimal	Hexadecimal	
0	0X00	Fija la ganancia analógica máxima en 94
1	0X01	Fija la ganancia analógica máxima en 97
2	0X02	Fija la ganancia analógica máxima en 100
3	0X03	Fija la ganancia analógica máxima en 103
4	0X04	Fija la ganancia analógica máxima en 107
5	0X05	Fija la ganancia analógica máxima en 110
6	0X06	Fija la ganancia analógica máxima en 114
7	0X07	Fija la ganancia analógica máxima en 118
8	0X08	Fija la ganancia analógica máxima en 123
9	0X09	Fija la ganancia analógica máxima en 128
10	0X10	Fija la ganancia analógica máxima en 133
11	0X11	Fija la ganancia analógica máxima en 139
12	0X12	Fija la ganancia analógica máxima en 145
13	0X13	Fija la ganancia analógica máxima en 152
14	0X14	Fija la ganancia analógica máxima en 159
15	0X15	Fija la ganancia analógica máxima en 168
16	0X16	Fija la ganancia analógica máxima en 177
17	0X17	Fija la ganancia analógica máxima en 187
18	0X18	Fija la ganancia analógica máxima en 199
19	0X19	Fija la ganancia analógica máxima en 212
20	0X20	Fija la ganancia analógica máxima en 227
21	0X21	Fija la ganancia analógica máxima en 245
22	0X22	Fija la ganancia analógica máxima en 265
23	0X23	Fija la ganancia analógica máxima en 288
24	0X24	Fija la ganancia analógica máxima en 317
25	0X25	Fija la ganancia analógica máxima en 352
26	0X26	Fija la ganancia analógica máxima en 395
27	0X27	Fija la ganancia analógica máxima en 450
28	0X28	Fija la ganancia analógica máxima en 524
29	0X29	Fija la ganancia analógica máxima en 626
30	0X30	Fija la ganancia analógica máxima en 777
31	0X31	Fija la ganancia analógica máxima en 1025

Tenga en cuanta que la relación entre el registro de ganancia y la ganancia real no es una relación lineal. No existe una fórmula mágica que diga “si utiliza este valor de ganancia, el alcance será exactamente este”. Depende del tamaño, forma, y material del objeto y de los elementos restantes de la habitación. Lo recomendable es experimentar con diferentes valores hasta obtener los resultados deseados. Si obtiene lecturas falsas, puede que sean los ecos de los “pings” anteriores, vuelva a lanzar el sensor SRF08 cada 65mS o más (menos tasa).

Si tiene alguna duda acerca de los registros Range y Gain, recuerde que en el sensor SRF08 se fijan los valores por defecto automáticamente cuando se inicia el sistema. Es más, puede olvidarse de esta configuración y utilizar los valores por defecto y el sensor funcionará correctamente, detectando objetos a 6 metros cada 65mS o menos.



Sensor de luz

El medidor ultrasónico SRF08 dispone de un sensor fotoeléctrico en la propia placa. Este medidor realiza una lectura de la intensidad de la luz cada vez que se calcula la distancia en los modos Ranging o ANN (La conversión analógica/digital se realiza realmente justo antes de que se lance el “ping” mientras el generador de 10v +/- se encuentra en fase de estabilización). EL valor de la lectura va aumentando a medida que aumenta la intensidad de la luz, por lo que valor máximo lo obtendrá con una luz brillante y el valor mínimo en total oscuridad. La lectura debería acercarse a 2-3 en total oscuridad y aproximadamente a 248 (0xF8) en luz diurna. La intensidad de la luz puede leerse en el registro del sensor de luz en la ubicación 1 al mismo tiempo que puede leer los datos del alcance.

LED

EL indicador LED rojo se utiliza para indicar el código de la dirección I2C del sensor en el encendido (ver abajo). Así mismo, también emite un breve destello durante el “ping” en el cálculo de la distancia.

Cambio de la dirección del bus I2C del SRF08

Para modificar la dirección I2C del sensor SRF08 sólo podrá tener un sensor conectado al bus. Escriba los 3 comandos de secuencias en el orden correcto seguidos de la dirección. Ejemplo; para cambiar la dirección de un sensor que tiene actualmente la dirección 0xE0 (la dirección de fábrica por defecto) a la dirección 0xF2, escriba lo siguiente en la dirección 0xE0; (0xA0, 0xAA, 0xA5, 0xF2). Se deberían enviar estos comandos con el orden secuencial correcto para modificar la dirección I2C. Además, no es posible emitir cualquier otro comando en medio de la secuencia. La secuencia debe enviarse al registro de comandos de la ubicación 0, lo que implica que se escribirán 4 transacciones independientes en el bus I2C. Una vez realizado todo esto, deberá etiquetar el sensor con su dirección. No obstante, si olvida hacerlo, cuando lo encienda, no se enviará ningún comando. El sensor SRF08 indicará su dirección mediante el LED. Un destello largo seguido de un número de destellos cortos indicará la dirección. Los destellos terminarán inmediatamente después de enviar un comando al sensor SRF08.

Dirección			
Decimal	Hexadecimal	Destello Largo	Destellos cortos
224	E0	1	0
226	E2	1	1
228	E4	1	2
230	E6	1	3
232	E8	1	4
234	EA	1	5
236	EC	1	6
238	EE	1	7
240	F0	1	8
242	F2	1	9
244	F4	1	10
246	F6	1	11
248	F8	1	12
250	FA	1	13

252	FC	1	14
254	FE	1	15

Asegúrese de no configurar más de un sensor con la misma dirección, ya que se produciría una colisión en el bus, con resultados totalmente imprevisibles.

Consumo de corriente

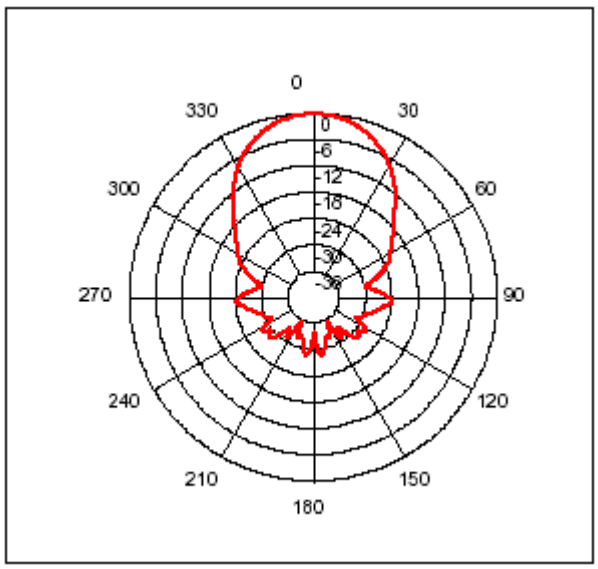
El consumo medio de corriente se calcula que es aproximadamente 12mA durante el cálculo de la distancia, y 3mA en modo de espera. El módulo entrará automáticamente en modo de espera después de terminar la medición, mientras espera al siguiente comando del bus I2C. El perfil real de consumo de corriente del srf08 es el siguiente:

Tipo de operación realizada	Corriente	Duración
Comando de medición de la distancia recibido -Encendido	275mA	3uS
Estabilización del generador de +/- 10v	25mA	600uS
8 ciclos de “ping” 40kHz	40mA	200uS
Medición	1mA	65mS máx.
Modo de espera (Stand-by)	3mA	Indefinido

Los valores de la tabla anterior se ofrecen sólo a modo orientativos, no se han comprobado en unidades de producción.

Cambio del ángulo de detección

El ángulo de detección no se puede cambiar. Esta es una pregunta que se hace muy frecuentemente y cuya respuesta es que no se puede alterar. El foco de trabajo del SRF08 es un cono cuyo ancho depende del propio traductor y esta es fija. La forma del área de trabajo del traductor ultrasónico empleado en el SRF08 es la de la siguiente figura, tomada de la hoja de características del fabricante.



Infrarrojos GP2D12

Fuente: Documento técnico oficial de Sharp.

SHARP

GP2D12/GP2D15 Distance Measuring Sensors

General Purpose Type Distance Measuring Sensors

■ General Description

SHARP's **GP2D12/GP2D15** are general purpose type distance measuring sensors which consist of PSD* and infrared emitting diode and signal processing circuit. It enables to detect objects without any influence on the color of reflective objects, reflectivity, the lights of surroundings.

*PSD:Position Sensitive Detector

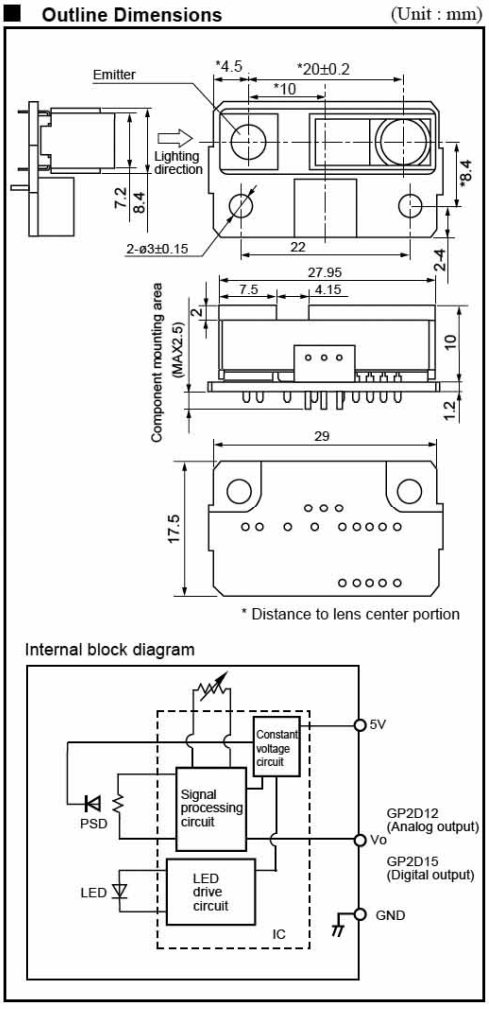
■ Features

(1) Less influence on the color of reflective objects, reflectivity
(2) Line-up of distance output/distance judgement type
Distance output type(analog voltage) : **GP2D12**
Detecting distance : 10 to 80cm
Distance judgement type : **GP2D15**
Judgement distance : 24cm (Adjustable within the range of 10 to 80cm)
(3) External control circuit is unnecessary.
(4) Low cost

■ Applications

(1) TVs
(2) Personal computers
(3) Cars
(4) Copiers

■ Outline Dimensions (Unit : mm)



Internal block diagram

(Notice) • In the absence of device specification sheets, SHARP takes no responsibility for any defects that may occur in equipment using any SHARP device shown in catalogs, data books, etc. Contact SHARP in order to obtain the latest device specification sheets before using any SHARP device.
• Specifications are subject to change without notice for improvement.

(Internet) • Data for SHARP's optoelectronic/power device is provided on internet. (Address http://www.sharp.co.jp/ecg/)

SHARP

Tec.U970501

■ Specifications

GP2D12 (Ta=25˚C)

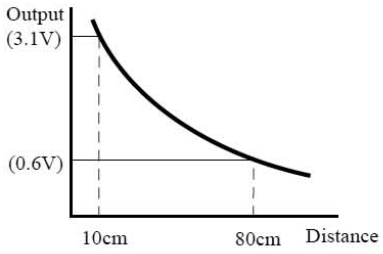
Parameter	Symbol	Rating
Supply voltage	Vcc	4.5 to 5.5V
Dissipation current	Icc	MAX.35mA
Measuring range	L	10 to 80cm
Output type	—	Analog output
Operating temperature	Topr	-10 to +60˚C

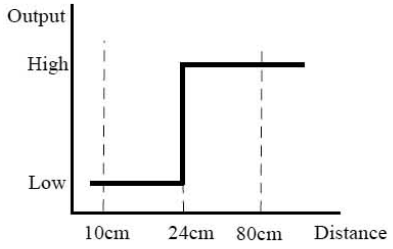
GP2D15 (Ta=25˚C)

Parameter	Symbol	Rating
Supply voltage	Vcc	4.5 to 5.5V
Dissipation current	Icc	MAX.35mA
*Judgement distance	L	TYP.24cm
Output type	—	Digital output
Operating temperature	Topr	-10 to +60˚C

* Adjustable within the range of 10 to 80cm.<Custom products>

■ Output pattern





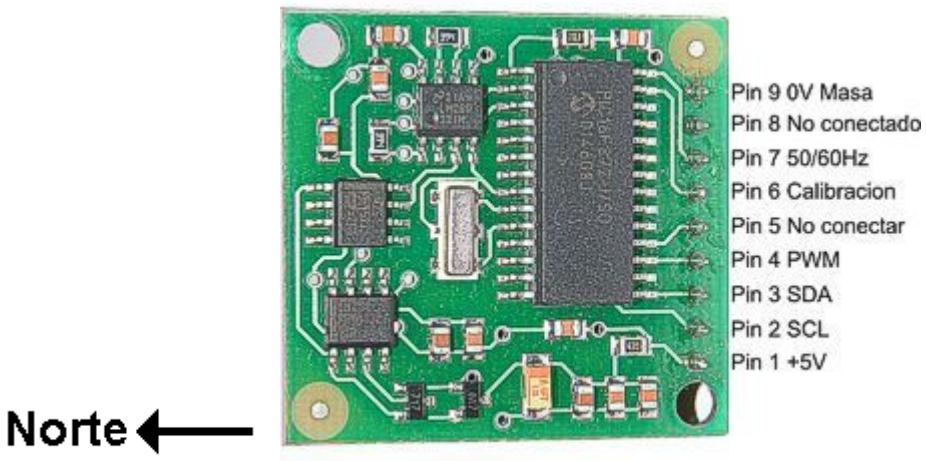
Compás CMPS03

Fuente: ***www.superrobotica.com***

La brújula digital CMPS03 es un sensor de campos magnéticos que una vez calibrado ofrece una precisión de 3-4 grados y una resolución de décimas. Tiene dos interfaces, mediante pulsos temporizados (modulación en anchura), o bien por medio de un bus I2C, lo que facilita su comunicación con una amplia gama de microcontroladores, incluyendo los Basic Stamp, Basic X, OOPic y otros lenguajes compilados. Este sensor magnético esta específicamente diseñado como sistema de navegación para robots. La brújula esta basada en los sensores KMZ51 de Philips que son lo suficientemente sensibles como para captar el campo magnético de la tierra. Usando dos de estos sensores colocados en ángulo de 90 grados, permite al microprocesador calcular la dirección de la componente horizontal del campo magnético natural.

Conexionado y Funcionamiento

En la siguiente imagen se puede observar las conexiones de los diferentes pines del CMPS03, entre los que destaca la patilla 4 y la 2 - 3, que corresponden a los dos posibles interfaces que incorpora: el PWM (Pulse Witdh Modulation - Modulación por anchura de pulso) y el bus I2C formado por las señales SDA (señal de datos) y SCL (señal de reloj) y que es un interfaz serie bidireccional sincrónico.



En la salida 4, se obtiene una señal PWM en la que el pulso positivo representa el ángulo de la brújula. El pulso varía en duración desde 1mS (0º) hasta 36,99 ms

(359,9 º), o dicho de otra forma, el pulso es igual a 100 uS por cada grado más 1ms de tara. La señal permanece a cero durante 65 ms entre pulsos, por lo que el periodo de trabajo es de 65mS + la anchura del pulso. El pulso es generado por un contador de 16 bits del propio procesador, con una resolución de 1 uS, aunque en la práctica no es recomendable hacer mediciones con una resolución de más de 0,1º (10uS). Cuando use el interfaz PWM, es necesario conectar a +5V mediante 2 resistencias de 47 Kohm, los pines 2 y 3 (SCL - SDA) del interfaz I2C, ya que no se incluye resistencias de pull-up en el circuito.

La otra posibilidad es la de usar el interfaz I2C formado por los pines 2 y 3, que nos permite una lectura directa del valor en grados de la dirección. El protocolo de comunicación I2C empleado en el módulo, es el mismo que se emplea con la populares eeprom como la 24C04. Primero se envía un bit de comienzo, la dirección del módulo (0XC0) con el bit de lectura a cero, y a continuación el numero del registro que se desea leer. Después se vuelve a mandar el bit de comienzo y la dirección del módulo con el bit de lectura a uno (0XC1). Ahora se puede leer uno, o los dos bytes correspondientes a los registros de 8 y 16 bits respectivamente. (El byte de mayor peso se lee primero en los registros de 16 bits).

Registros

La brújula tiene un total de 16 bytes de registros, algunos de los cuales forman registros de 2 bytes tal y como puede verse en la siguiente tabla:

Registro	Función
0	Numero de Revisión del Software
1	Dirección en 1 byte 0-255 para 0 - 360º
2,3	Dirección en 2 bytes 0-3599 para 0 - 359,9º
4,5	Test interno señal diferencial sensor 1
6,7	Test interno señal diferencial sensor 2
8,9	Test interno, valor de calibración 1
10,11	Test interno, valor de calibración 2
12	Sin usar, devuelve 0
13	Sin usar, devuelve 0
14	Sin usar, devuelve 0
15	Comando de calibración, escribir 255 para calibrar

El registro 0 es la Revisión del software que actualmente es el 8. El registro 1 es la dirección en grados convertida en un valor entre 0 y 255 y que puede ser muy útil en ciertas aplicaciones donde resulta complicado utilizar la escala de 0 a 360 grados que requiere dos bytes y que esta disponible en los registros 2 y 3 (el 2 es el mas significativo) con un valor que va entre 0 y 3599 que equivale a 0 -359,9º. Los registros 4 a 11 son de uso interno y del 12 al 14 no se usan, por lo que no deberán leerse con el fin de no consumir el ancho de banda del bus I2C. El registro 15 se usa para calibrar la brújula tal y como se especifica más adelante.

El bus I2C del circuito no incorpora las necesarias resistencias de pull-up, por lo que será necesaria su implementación en el mismo, para ello es recomendable utilizar dos resistencias de 1K8 en caso de utilizar el bus a 400 KHz y de 1K2 o 1K si se utiliza a una frecuencia de 1Mhz. Solo son necesarias 2 resistencias en total para todo el bus, no por cada circuito que este conectado al mismo. El sensor de brújula digital esta diseñado para ser compatible con la velocidad estándar de reloj de 100 Khz, aunque esta pueda aumentarse si tiene en cuenta lo siguiente:

A velocidades superiores a los 160 KHz, la CPU no puede responder lo suficientemente rápido como para leer los datos, por lo que hay que incorporar un retardo de 50 us al finalizar la escritura del registro de dirección. Si se hace esto de

forma correcta, es posible comunicar con el módulo a velocidades superiores a 1 MHz. Esto solo afecta a programas escritos en lenguajes de alta velocidad y bajo nivel como es el ensamblador, y no afecta a las aplicaciones escritas para los compiladores internos como son el Basic stamp, el OOPic o el Basic X o similares. El módulo de sensor de brújula siempre actúa como un esclavo, nunca como un master del bus I2C.

El pin 7 se utiliza para seleccionar entre 50 Hz (puesta a cero) o 60 Hz (puesta a uno). Esto es debido a una desviación errónea de unos 1,5º causada por el campo generado por la red eléctrica. Sincronizando la conversión con la frecuencia en hertzios de la red, se consigue disminuir el error a tan solo 0,2º. El pin si tiene una resistencia interna de pull up, por lo que si se deja sin conectar, funcionara a 60 Hz. El circuito realiza una conversión interna cada 40ms (50 Hz) o cada 33,3 ms (60Hz) de acuerdo con la conexión de esta entrada. No hay ningún tipo de sincronismo entre la realización de la conversión y la salida de los datos, ya que cuando estos son leídos se devuelven el valor mas reciente que este almacenado en su respectivo registro.

El pin 6 se usa para calibrar el sensor magnético. Esta entrada tiene su propia resistencia de polarización (pull up) y puede dejarse sin conectar una vez realizada la conversión.

Los pines 5 y 8 están marcados como no conectados, aunque el pin 8 es en realidad el reset del microprocesador, con el fin de poder programarlo una vez soldado al circuito impreso. Esta entrada no tiene resistencia de pull up.

Calibración

ATENCION: Antes de realizar la calibración, el módulo deberá mantenerse perfectamente horizontal con los componentes hacia arriba y los dos sensores en la cara inferior. Mantener el módulo alejado de objetos metálicos y muy especialmente de objetos magnéticos como imanes y altavoces. También es necesario conocer con precisión la dirección en la que se encuentran los cuatro puntos cardinales, por lo que es **absolutamente necesario** comprobarlo con una brújula magnética.

La calibración de la brújula digital puede hacerse por cualquiera de los siguientes dos métodos:

El Método I2C

Ese Método consiste en escribir 255 en el registro 15 del módulo por cada uno de los cuatro puntos cardinales. El valor 255 es borrado internamente cada vez que se completa la calibración Los puntos de calibración pueden hacerse en cualquier orden, pero siempre es necesario calibrar los 4 puntos. Por ejemplo:

1 Apunte el circuito hacia el Norte. Escriba 255 en el registro 15
2 Apunte el circuito hacia el Este. Escriba 255 en el registro 15
3 Apunte el circuito hacia el Sur. Escriba 255 en el registro 15
4 Apunte el circuito hacia el Oeste. Escriba 255 en el registro 15

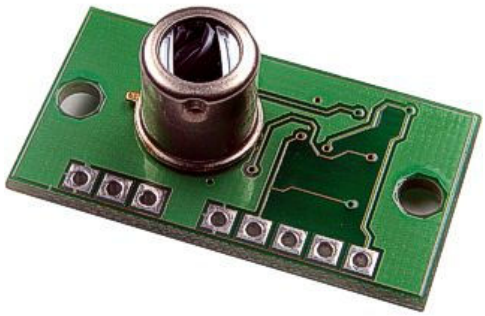
El Método del pulsador.

Consiste en utilizar un pulsador entre masa y el pin 6 del circuito, con el fin de iniciar la calibración Tenga en cuenta que este pin tiene una resistencia de polarización interna y puede dejarse sin conectar una vez realizada la calibración Para realizar la calibración, bastara con poner a masa el pin 6 momentáneamente por cada uno de los puntos cardinales. De igual forma que con el otro Método, los puntos pueden calibrarse en cualquier orden, pero siempre es necesario calibrar los 4 puntos cardinales. Ejemplo:

1 Apunte el circuito hacia el Norte. Pulse momentáneamente en pulsador.
2 Apunte el circuito hacia el Este. Pulse momentáneamente en pulsador.
3 Apunte el circuito hacia el Oeste. Pulse momentáneamente en pulsador.
4 Apunte el circuito hacia el Sur. Pulse momentáneamente en pulsador.

Termopila TPA81

Fuente: ***www.superrobotica.com***



TPA81 es un sensor térmico de 8 píxeles capaz de medir la temperatura de un objeto a distancia. Este sensor esta formado en realidad por una matriz de 8 sensores colocados linealmente de forma que puede medir 8 puntos adyacentes simultáneamente. A diferencia de los sensores pir utilizados en sistemas de alarmas y detectores para encender luces, el sensor térmico no necesita que haya movimiento para detectar el calor, por lo que su aplicación en el campo de la robótica, abre gran cantidad de aplicaciones no disponibles hasta ahora. El sensor se conecta por bus I2C y además se le puede conectar un servo estándar que es controlado por el propio sensor para hacer un barrido y tomar 32 mediciones diferentes, obteniéndose un mapa térmico de 180 grados. El TPA81 es capaz de detectar la llama de una vela a 2 metros de distancia y además no le afecta la luz ambiental.

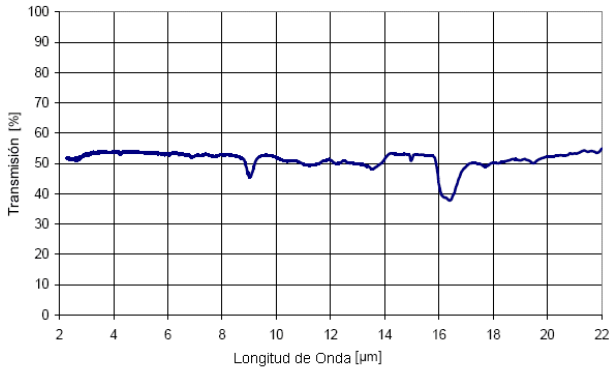
Introducción

El sensor de temperatura TPA81 es un módulo térmico de detección por infrarrojos en un rango de 2µm a 22µm que es la longitud de onda del calor radiante. Los sensores pir que se utilizan generalmente en las alarmas antirrobo y para el encendido/apagado de las luces exteriores, detectan señales infrarrojas en la misma banda de onda, sin embargo estos sensores sólo detectan cambios en la temperatura, no la temperatura en si. Aunque son útiles en robótica, sus aplicaciones son limitadas, ya que no pueden detectar y medir la temperatura de una fuente de calor estática. El otro tipo de sensor es el sensor termopila, que se utilizan en los termómetros por infrarrojos sin contacto. Tienen un ángulo de detección o campo de visión (FOV) muy amplio, de aproximadamente 100° y requieren una carcasa con una lente para obtener un campo de visión reducido de aproximadamente 12°. Actualmente es difícil encontrar sensores con módulo de termopilas, electrónica y lente de silicona integrada con el caso del TPA81. Este sensor tiene en realidad una formación de ocho termopilas organizadas en una hilera. El Sensor de temperatura TPA81 puede medir la temperatura de 8 puntos

adyacentes de manera simultánea. El TPA81 también puede controlar un servo para manejar el módulo y generar una imagen térmica.

Respuesta de espectro

La repuesta TPA81 es normalmente de 2µm a 22µm como se muestra a continuación:



Campo de visión (FOV)

El campo de visión normal de TPA81 es de 41° por 6° convirtiendo cada uno de los ocho píxeles en 5.12° por 6°. El conjunto de ocho píxeles está orientado a lo largo de la placa de circuito impreso (PCB) - como indica el siguiente diagrama de arriba a abajo. El número de píxel es el más cercano a la pestaña del sensor - o en la parte inferior del diagrama siguiente.

Sensibilidad

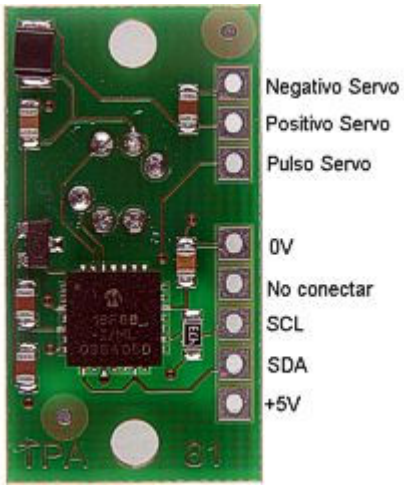
Estos son algunos de los números resultantes de uno de nuestros módulos de prueba:
para una vela, los números para cada uno de los ocho píxeles a una distancia de 1 metro en una habitación fresca a 12° son los siguientes:
11 10 11 12 12 29 15 13 (Todos en grados centígrados)
Puede ver que la vela de muestra aparece como la lectura de 29° C. A una distancia de 2 metros, ésta se reduce a 20° C, aproximadamente unos 8° C por encima de la temperatura ambiente, por lo que todavía es fácilmente detectable. A 0,6 metros da aproximadamente 64° C. A 0,3 metros da aproximadamente 100° C.

En una habitación más cálida a 18° C, la llama mide 27° C a 2 metros de distancia. Esto se debe a que la vela sólo ocupa una pequeña parte del campo de visión y

la fuente de calor de la vela se suma a la temperatura ambiente del aire no la superpone por completo. Un cuerpo humano a 2 metros aparecerá como 29° C, con una temperatura ambiente de 20° C.

Conexiones

Toda la comunicación con el sensor térmico TPA81 se realiza a través del bus I2C. El sensor TPA81 utiliza una conexión I2C de 5 pines. El pin marcado con el mensaje "No conectar" se debería dejar sin conexión. En realidad se trata de la línea MCLR de la CPU y se utiliza solamente para programar el PIC16F88 en la propia placa después del montaje, tiene una resistencia de polarización positiva interna. Las líneas SCL y SDA deberían disponer las dos de una resistencia de polarización a +5v en algún punto del bus I2C. Lo único que necesitará es un par de resistencias en todo el bus, y no un par para cada módulo. Normalmente están ubicadas en el circuito del bus maestro. El sensor térmico TPA81 actúa siempre como un módulo esclavo dentro del bus I2C. Si necesita utilizar resistencias de polarización, le recomendamos resistencias de 1K8. Algunos módulos como los OOPic ya disponen de resistencias de polarización por lo que no necesitará conectar unas nuevas. El TPA81 incluye un conector para un servo estándar que se alimenta desde los 5v de la alimentación del módulo. Se pueden enviar comandos de posicionamiento al TPA81 para colocar el servo en cualquiera de las 32 posiciones disponibles, los pulsos de control del servo los genera el propio módulo TPA81.



Registros

El TPA81 aparece como un conjunto de 10 registros.

Registro	Lectura	Escritura
0	Revisión de Software	Registro de comando
1	Temperatura ambiente ° C	Utilizado para la calibración- no escribir
2	Temperatura Píxel 1° C	Utilizado para la calibración- no escribir
3	Píxel 2	Utilizado para la calibración- no escribir
4	Píxel 3	N/A
5	Píxel 4	N/A
6	Píxel 5	N/A
7	Píxel 6	N/A
8	Píxel 7	N/A
9	Píxel 8	N/A

Sólo se pueden escribir los registros 0, 1, 2 y 3. El Registro 0 se trata de un comando de registro y se utiliza para colocar la posición del servo y cambiar la dirección I2C de TPA81. No se puede leer. La lectura del registro 0 devuelve la revisión de software de TPA81. Los Registros 1, 2 y 3, se utilizan para calibrar el sensor. No escriba en estos registros ya que se pueden eliminar los datos de calibración de los sensores. (Existe protección para ello. Debe proporcionarse una secuencia de comandos específica de 3 bytes similar a la secuencia de cambio de dirección I2C para habilitar el modo de calibración). La calibración requiere el uso de dos fuentes de calor de cuerpos negros. Sólo podrá calibrar el módulo si dispone de estos cuerpos. Todos los módulos están calibrados en nuestro taller, como parte de nuestros procesos de prueba.

Hay 9 lecturas de temperatura disponibles, todas expresadas en grados centígrados. El registro 1 se trata de la temperatura medida por el sensor. Los registros 2-9 son las temperaturas de 8 píxeles. La adquisición de temperatura se realiza de manera continua y las lecturas serán correctas 40mS después de que el sensor apunte a una nueva posición.

Posición de Servo

Los comandos 0 a 31 establecen la posición del servo. Hay 32 pasos (0-31) que representan los 180 grados de rotación en un servo Hitec HS311. El cálculo es SERVO_POS*60+540uS. Por lo que el alcance del pulso del servo es de 0.54mS a 2.4mS en pasos de 60uS. Si se escribe cualquier otro valor al registro del comando se detendrán los pulsos del servo.

Comando		Acción
Decimal	Hexadecimal	
0	0x00	Establece la posición del servo al mínimo
nn	nn	Establece la posición del servo
31	0x1F	Establece la posición del servo al máximo
160	0xA0	1º en la secuencia para cambiar la dirección I2C
165	0xA5	3º en la secuencia para cambiar la dirección I2C
170	0xAA	2º en la secuencia para cambiar la dirección I2C

Cambio de la dirección I2C del Bus

Para cambiar la dirección I2C de TPA81 debe tener sólo un módulo en el bus. Escriba las 3 secuencias de comandos en el orden correcto seguido de la dirección. Por ejemplo, para cambiar la dirección de un TPA81 actualmente en la dirección 0xD0 (dirección predeterminada de fábrica) a 0xD2, escriba lo siguiente para la dirección 0xD0; (0xA0, 0xAA, 0xA5, 0xD2). Estos comandos deben enviarse en la secuencia correcta para cambiar la dirección I2C, además, no se puede enviar otro comando en medio de la secuencia. La secuencia debe enviarse al registro de comandos en la ubicación 0, lo que significa que se crearán 4 transacciones de escritura en el bus I2C. Además, DEBE haber un retardo de al menos 50mS entre la escritura de cada byte de la secuencia de los cambios de dirección. Cuando lo haya realizado, debería etiquetar el sensor con sus direcciones, si pierde las direcciones del módulo, la única manera de averiguarlas es buscar todas las direcciones una a una hasta averiguar cuál es la que responde. TPA81 puede tener hasta ocho direcciones I2C- 0xD0, 0xD2, 0xD4, 0xD6, 0xD8, 0xDA, 0xDC, 0xDE. La dirección predeterminada de fábrica es 0xD0.

Lista del Coste detallada

Descripción :	Cantidad	Coste	Total parcial
Chasis Lynxmotion 4WD3	1	158,70	158,70
Plataforma	1	7,3462	7,3462
Torreta Avanzada	1	55,37	55,37
MD22 - Control Motores I2C	1	59,10	59,10
ATBASE 128 (futulec.com)	1	22,07	22,07
Via EPIA MII 10000 Placa Base	1	166,25	166,25
Adaptador CF - IDE	1	10	10
Compact Flash 4Gb	1	108,25	108,25
SIM DDR 1 Gb	1	87,21	87,21
Fuente ATX DC - DC 60W	1	73	73
Wifi USB	1	36	36
BATERIA Zaapa (65W/H)	1	69	69
Batería 7.2V RControl 4000mA	1	41	41
Ultrasonidos SRF08	3	39	117
Compás CMPS03	1	38,15	38,15
Sharp GP2D12	3	15,82	47,46
Termo Pila TPA81	1	56,81	56,81
Acelerómetro LIS3V02DQ	1	27,90	27,90
Servos HS311	2	19,19	38,39
Grabador JTAG AVR	1	24,29	24,29
Cargador RC	1	50	50
Tornillería	1	16	16
cable	1	10	10
Componentes electrónicos	1	80	80
		Total:	1399,32 €

Apéndice II

Especificaciones Software

Comandos de control

A continuación se detalla todos los comandos posibles para gobernar ARI desde el puerto serie. ARI trabaja de forma comprobación perezosa, la desventaja de este sistema es que un comando mal escrito tiene un comportamiento imprevisible, pero el código para interpretar los comandos es mucho más reducido y rápido.

Interficie:

Todos los comandos que empiezan por **i** son los referidos a la interfaz serie y el controlador en general.

iv

El comando "iv" demanda a la unidad principal que devuelva la versión actual del firmware. Recibiremos una salida como esta:

```
iv
DARI v0.5Beta.
```

ie

El comando **ie** permite desactivar el eco, algo muy práctico para cuando el firmware es controlado por otro programa.

```
ie
echo off
ie
echo on
```

En la salida vemos un **ie** en gris eso significa que ha sido introducido pero no saldría en la vista de la consola porque el echo estaba desactivado.

iw:aarrvv

aa Dirección hexadecimal del dispositivo I2C al que se desea enviar el dato.

rr Número de registro al que se desea acceder. (también en hexadecimal)

vv Byte a enviar en hexadecimal..

El comando **iw** nos permite enviar un byte de datos a un dispositivo I2C, esta pensado para mantenimiento y diagnostico, por ejemplo cambiar la dirección I2C de un dispositivo.

Ejemplo: cambiar la dirección I2C de un sonar en la posición E0 a la posición F2:

```
iw:e000a0
iw:e000aa
iw:e000a5
iw:e000f2      "Aquí el dispositivo ya tiene la nueva dirección,
iw:e000a0      si intentamos escribir otra vez fallara "

E:I2CError
```

il:aarr

aa Dirección hexadecimal del dispositivo I2C de donde se desea leer el dato.

rr Número de registro al que se desea acceder. (también en hexadecimal)

El comando **il** nos permite leer un byte de un dispositivo I2C, este comando igual que el anterior solo debe ser usado para tareas de mantenimiento.

```
ir:e001
a0
```

ir

El comando **ir** realizada un reinicio completo de la unidad de control, cuando el reinicio se complete recibiremos la versión de la unidad.

```
ir
DARI v0.5Beta.
```

ih

El comando **ih** devuelve el total de memoria utilizada del heap de memoria dinámica.

```
ih
Heap Usado:2020 bytes.
```

it

El comando **it** devuelve un listado con la lista de tareas y su estado, donde podemos comprobar el uso de la pila de cada proceso. , cual esta en ejecución, o parada., la prioridades de estas y el estado de inicialización.

```
it
Tareas:     Estado.  Prio.   Stack.  Inic.
MSerial       R      3       109     0
IDLE          R      0       66      4
MServo        B      2       87      1
MMotion       B      2       98      2
MSensor       B      3       56      3
```

Locomoción

Los comandos relacionados con el control de motores comienzan con **m**.

mtddd y muddd

ddd valor que indica dirección y velocidad del motor izquierdo(**mt**) o el motor derecho(**mu**) , un 0 será hacia delante a toda velocidad, 128 es estado de reposo y 255 es a toda velocidad hacia delante.

El comando **mt** permite controlar los motores del lado izquierdo tanto en sentido como en velocidad, este comando se añadió para dar un mejor soporte al Pyro.

mt100

La ejecución del comando **mt** puede dar error ("**E:I2CDevNotFound**") si no se encuentra la placa de control de motores MD22.

mf

El comando **mf** ordena al robot avanzar a la velocidad previamente establecida por el comando **ms**, inicialmente el ARI inicializa esta velocidad a 0.

mh

El comando **mh** ordena al robot detener cualquier movimiento que este realizando. Esto no inicializa los registros de velocidad ni aceleración, estos se encontraran tal y como estaban previamente a la llamada al comando **mh**.

mb

El comando **mb** ordena al robot retroceder a la velocidad previamente establecida por el comando **ms**, inicialmente el ARI inicializa esta velocidad a 0.

ml y mr

Estos comandos ordenan al robot girar sobre su mismo eje hacia la izquierda (**ml)** o la derecha (**mr)**, esto se realiza a la velocidad previamente establecida por el comando **ms**, inicialmente el ARI inicializa esta velocidad a 0.

ma*ddd*

ddd valor entre 0 y 255 en el que 0 es aceleración lenta y 255 es aceleración inmediata, para más detalles de los tiempos de aceleración consulte la documentación del MD22.

El comando **ma** ajusta el parámetro de aceleración del MD22.

ms*ddd*

ddd valor entre 0 y 127 que indica la velocidad a la que se mueve el robot, 127 es el máximo.

El comando **ms** ajusta la velocidad para los comandos **mf**, **mb**, **ml** y **mr**, los comandos **mt** y **mu** son independientes.

Control Servos

Los comandos que comienzan con **e** controlan los servos de diversas maneras.

e*x*m*ddds*

x Número de servo a mover

ddd Ángulo en grados x 10 de la nueva posición absoluta.

s Es o **r** o **l** e indica todo a la izquierda o todo a la derecha, si es omitido se toma **l** por omisión

Este comando realiza un movimiento absoluto, define el 0 como la posición central del servo y permiten girar 90 grados a cada lado, con precisión de décima de grado (no todos los servos tienen esta precisión).

Ejemplos:

e0m900r	servo 0, 90 grados a la derecha
e0m900l	servo 0, 90 grados a la izquierda
e1m000	servo 1, centrado a 0 grados (izquierda).
e4m001r	servo 4, 0.1 grados a la derecha.

e*x*i*ddds*

x Número de servo a mover

ddd Ángulo en grados x 10 del nuevo incremento.

s Es o **r** o **l** e indica todo a la izquierda o todo a la derecha, si es omitido se toma **l** por omisión

Este comando realiza un movimiento incremental e incrementa el valor de ddd en el sentido indicado por s, con precisión de décima de grado (no todos los servos tienen esta precisión).

Ejemplos:

e0i455r	servo 0, incrementa 45.5 grados a la derecha

e0i900l	servo 0, incrementa 90 grados a la izquierda
e4i001r	servo 4, incrementa 0.1 grados a la derecha.

e*x*c

x Número de servo.

Este comando guarda la posición actual como el centro del servo. el dato se guarda en la eeprom del microcontrolador, así que aunque se desconecte el parámetro queda guardado.

Ejemplos:

```
e0c
I:CtrSaved
```

e*x*p*d*

x Número de servo.

d Un cero desactivar la función de barrido , un uno la activa.

Este comando activa/desactiva la función de barrido.

Ejemplos:

e0p1	Activar el barrido en el servo 0
e3p0	Desactivar el barrido en el servo 1

e*x*t*ddd*

x Número de servo.

ddd valor que ajusta la velocidad de barrido.

Este comando ajusta la velocidad de barrido, una valor de 25 equivale a 1 segundo, es el tiempo que espera entre movimientos del servo. Por defecto esta ajustado a 50 (2 segundos).

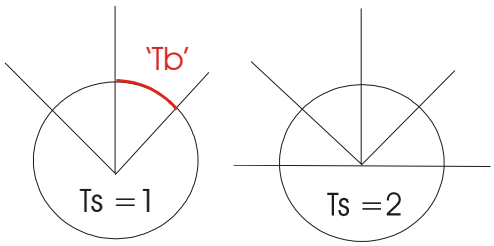
Ejemplos:

e3t025	Ajustar el tiempo a 1 segundo.

e*x*r*ddd*

x Número de servo.

ddd Ángulo entre pasos en el gráfico esta indicado como Tb.



Este comando ajusta cuanto se moverá el servo entre pasos, es muy recomendable detener el barrido y centrar el servo antes de cambiar este parámetro.

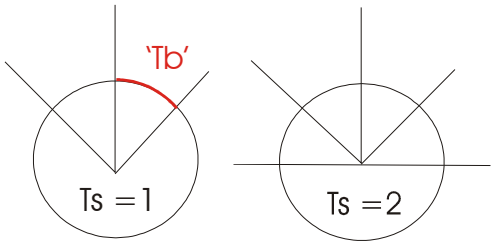
Ejemplos:

e0r450	servo 0, ajustar a 45 grados el ángulo recorrido entre pasos.
e1r060	servo 1, desactivar el barrido en el servo 1

e*x*s*ddd*

x Número de servo.

ddd Número de pasos hacia un lado. En el gráfico es Ts., fíjese en las líneas que seccionan la circunferencia., mínimo debe ser 1.



Este comando ajusta cuantos pasos se va a mover, el número introducido indica cuantos pasos a cada lado se mueve, siempre se para en el centro, es muy recomendable detener el barrido y centrar el servo antes de cambiar este parámetro.

Ejemplos:

e0s001	servo 0, ajustado a 1 paso a cada lado es decir 3 paradas.
e1s032	servo 1, ajustado a 32 pasos a cada lado así que en total son 65.

e*x*e

x Número de servo.

Este comando devuelve la configuración actual del sistema de barridos, los números devueltos en base 36, representa: Ex:P,SS,RR,TT , donde P es si esta activo o no (0/1), SS el número de pasos, RR es el ángulo de paso y TT es el tiempo entre pasos.

Ejemplos:

e0e
E0:0,01,19,0S

Sensores.

La **s** inicial indica que el comando es del módulo de sensores.

sf

Este comando vacía por completo la lista de lecturas periódicas.

Ejemplos:

sf TimeListClear

sd*ddtx*

dd Número de sensor.

t Selecciona de donde será eliminado el sensor las opciones son “t” para la lista de lecturas periódicas, y “e” para los servos.

x Selecciona de que servo será eliminado. (sólo para “e”).

Este comando permite eliminar de las listas un sensor, tanto puede ser la lista periódica como una de las 6 listas sincronizadas con los servos

Ejemplos:

sd03t	Elimina de la lista de lectura periodica el sensor 03.
sd05e2	Elimina de la lista de la lista sincronizada el servo 02.

sa*ddtx*

dd Número de sensor.

t Selecciona de donde será añadido el sensor las opciones son “t” para la lista de lecturas periodicas, y “e” para los servos.

x Selecciona en que servo será añadido. (sólo para “e”).

Este comando permite añadir a las listas un sensor, tanto puede ser la lista periodica como una de las 6 listas sincronizadas con los servos

Ejemplos:

sa03t	añade a la lista de lectura periodica el sensor 03.
sa05e2	añade a la lista de la lista sincronizada el servo 02.

sc

Devuelve el número de sensores disponibles en el sistema.

Ejemplos:

sc	
Sc:10	Hay 10 sensores disponibles.

st:*ddd*

ddd Número que indica el periodo de escaneo de los sensores de la lista, a menor número, más rápido.

Este comando permite ajustar la cadencia de escaneo de los sensores.

sndd

dd Número de sensor que se desea consultar.

Este comando permite consultar el nombre de un sensor , el nombre es un código alfanumérico de 4 cifras.

Ejemplos:

```
sn
Sn01:ADC1
```

srdd

dd Número de sensor que se desea consultar.

Este comando consulta el último valor valido del sensor, en caso de ser un sensor que necesita ser “disparado”, posteriormente a la consulta realiza un disparo, se puede realizar dos lecturas para obtener un resultado rápido, este comando no es recomendado usarlo para peticiones periódicas. Para la consulta periódica y continua del sistema es recomendable usar las funciones pertinentes.

Detalles I2C y librería

Principio de funcionamiento:

El bus I2C, es un bus multimaestro, que es capaz de direccionar hasta 128 dispositivos, la dirección se envía en un byte, los 7 bits más altos son la dirección y el bit más bajo indica si lectura o escritura, esto hace que todas las direcciones de dispositivos sean números pares.

Para realizar una lectura, después de conseguir el bus, se envía un byte con la dirección y el bit de escritura a 0, luego se pone otro byte que contiene la dirección, se espera que el dispositivo ponga el dato en el bus, si el master responde con ACK, el dispositivo responderá con el siguiente dato, si el maestro responde con NO ACK se finalizará la comunicación.

Para la escritura es muy parecido, ponemos el bit de escritura a 1, y el último byte también es escrito por nosotros. , después de cada byte va una marca de ACK.

Librería:

La librería I2C basada en la creada por Peter Flury [12] varia en dos detalles:

1. Ha sido alterada para soportar concurrencia, dado que Peter la ideó para su uso en programas sencillos y mono tareas. En este aspecto no hay nada que tener en cuenta, dado que son las propias funciones las que se encargan de ello.

2. Ampliada con nuevas funciones para lectura en bloques con la finalidad de optimizar el acceso al bus, permitiendo leer más de un byte por acceso.

Funciones:

- unsigned char **i2c_readbyte** (char addr, char reg), Esta es la función básica, recibe la dirección y el registro, y devuelve el byte correspondiente.

- unsigned short **i2c_readword** (char addr, char reg), Recibe la dirección, y el registro y lee el indicado y el registro siguiente, y lo almacena en un registro de 16bits, en modo big Endian.

- unsigned char **i2c_readblock** (char addr, char reg, char* block, short count), Esta función permite leer **count** registros, estos son almacenados en **block,** desde el final hasta el principio.

- char **i2c_writebyte** (char addr, char reg, char value), Permite enviar un byte de valor **value,** al registro **reg,** de la dirección **addr.**

Añadir un sensor nuevo a ARI

Unas de las funcionalidades del software de control ARI es permitir desarrollar controladores para nuevos sensores de forma modular a nivel de código, así que añadir un dispositivo nuevo requiere compilación y grabación otra vez del microcontrolador.

Además de la modificación del código *python* que sirve de interfaz al Pyro, se ha intentado que sea lo mas sencillo posible, y a continuación se expondrá los conceptos básicos y un par de ejemplos prácticos, para llevar esta tarea a buen puerto.

Soporte en el ARI para sensores.

Como se ha explicado antes el software del ARI esta dividido en varios módulos según su función uno de estos es *El Módulo de Sensores* que permite añadir hasta 25 sensores. La forma de implementar estos sensores es crear una pareja de ficheros *.c* y *.h* que usando una pequeña API permite integrar un sensor nuevo con solo escribir 3 funciones:

Una función de inicialización que se encargará de dejar configurado y listo el dispositivo, esta función no tiene ningún formato obligado, pero se recomienda que se llamé xxxxStart( ) donde xxxx es el un nombre de dispositivo. También es la que se encargar de añadir las siguiente funciones al módulo de sensores.

La función de inicio de lectura, esta función aísla en sensores en los que la interrogación y la lectura tiene que ser tratada de forma diferente, por ejemplo un sonar, en el que debemos esperar cierto tiempo entre que hemos "disparado" y podemos leer el resultado.

La función de Lectura, esta función es básica y vital dado que es la que debe devolver la lectura del sensor.

Incluyendo el fichero **ModSensor.h** tendremos acceso a la siguiente función que nos permitirá asignar las funciones anteriores como un único sensor.

```
void sensorAttach( short (*iniciar)(unsigned short,char),short (*leer)(unsigned short, char* len,char param), char* nombre,char param ,short mode);
```

El primer parámetro es un puntero a una función del tipo: *short (*iniciar) (unsigned short, char)* que es nuestra función de iniciar, puede ser nulo que indicara que este sensor se puede leer en cualquier momento, el primer parámetro es un valor short que indica tiempo, y nos puede servir para no sobrecargar recursos compartido o evitar leer el sensor cuando realmente no tiene un valor nuevo. El segundo parámetro es un parámetro que nosotros mismo podemos asignar cuando anexamos la función, lo explicaremos mas adelante.

El segundo parámetro es un puntero a otra función en este caso la de lectura que tiene una estructura de esta forma : *short (*leer)(unsigned short, char* len,char param)* donde el primer parámetro es una marca de tiempo como en la función iniciar, la segunda es un puntero donde se debe devolver la longitud de la cadena resultante según el modo de trabajo (ver mas adelante), y el tercer parámetro es el parámetro igual que la función anterior que será de uso según el controlador.

El tercer parámetro es un nombre de 4 letras para el sensor. El cuarto es un parámetro que se pasará a las funciones iniciar y leer cuando sean llamadas.

El quinto y último parámetro es el modo de este sensor, es posible elegir entre devolver un valor binario entero, este se convertirá a ASCII y será enviado o podemos devolver una cadena de texto de longitud variable, se ajusta este parámetro usando las constantes definidas en **ModSensor.h.**

```
#define SENSOR_INTEGER 0
```

En el modo Entero es tan sencillo como devolver un dato del tipo short en lo que devuelve la función, en cambio en el modo “BIN8” debemos devolver por el retorno de la función un puntero a la cadena, y por el puntero len de los parámetros la longitud de esta.

Si resumimos en una lista los pasos a realizar para crear el controlador de un sensor para ARI son estas:

- Crear los ficheros .c y .h
- Si es necesario realizar una función de iniciar lectura que cumpla con esta definición : short nombre(unsigned short, char)
- Realizar una función que realice la lectura del sensor y tenga este prototipo: short lectura (unsigned short timestamp_n,char* len,char param)
- Realizar función de inicialización que incluya como mínimo una llamada a *sensorAttach.*
- Añadir la llamada a la función de Inicialización en el main.c
- Incluir el .c en el makefile de proyecto para que sea compilado.
- Mas adelante hay unos ejemplos prácticos comentados que mejoraran la comprensión.

Ejemplos Prácticos:

Añadir 16 líneas digitales

En este ejemplo veremos un caso muy sencillo en el que se trata de dar salida a 16 líneas digitales.

Los primeros pasos es crear los ficheros **drvdigital.h** y **drvdigital.c**, como siempre en el fichero .h declararemos nuestras funciones y constantes, para el controlador digital tiene un aspecto como este:

```
#ifndef DRVDIGITAL_H
#define DRVDIGITAL_H   1
#include "modSensor.h"
// Controlador De 16 entradas digitales
 void DigitalStart(void);

// Funciones de lectura del compas

short Digital_leer (unsigned short
,char*,char);
#endif
```

Como podemos observar tenemos la declaración de una función de inicialización y la declaración de la función de lectura que sigue el formato ya comentado.

A continuación veremos la definición de estas funciones.

```
#include "drvdigital.h"

void DigitalStart(void){
// Inicialización
  DDRA = 0;
  PORTA = 0;
  DDRC = 0;
  PORTC = 0;

  // Añadir el sensor
sensorAttach((void*)0,
&Digital_leer,"DIN01",0,SENSOR_INTEGER);
   sensorAttach((void*)0,
&Digital_leer,"DIN02",1,SENSOR_INTEGER);
  }

short Digital_leer (unsigned short
timestamp_n,char* len,char param)  {

(void)len;
(void)timestamp_n;
   if (param == 0) {
 return PINA;
  }
  else {
    return PINC;
  }
 }
```

En la definición vemos como se inicializa el hardware (en este caso es muy sencillo dado que es simplemente devolver un registro del hardware del microcontrolador).

Luego vemos como se añade 2 veces el sensor, sin la función de iniciar lectura ((void*)0), a los se les asigna la misma función de lectura pero con diferentes parámetros, esto permite escribir una sola función para diversos dispositivos.

En la función de lectura según el parámetro devolvemos un valor u otro. Que corresponde a un dispositivo u otro, este ejemplo es muy sencillo dado que no tenemos que inicializar ningún hardware especial.

Añadir un grupo de Sonars.

El Sonar fabricado por devantech requiere un control más complejo, inicialmente se debe mandar la orden de iniciar lectura, esperar 65 milisegundos y después leer.

```
#ifndef DRVSONAR_H
#define DRVSONAR_H   1

#include "modSensor.h"
#include "i2cmaster.h"
#include "i2cdevices.h"

#define numReads 16

// Configura el Hardware y adjunta las funciones al módulo de
sensores.

 void SonarStart(void);

 // Funciones de lectura de los sonars
 short Sonar_iniciar(unsigned short timestamp_n,char);
 short Sonar_leer (unsigned short,char* len,char);
 short SoLuz_leer (unsigned short,char* len,char);

#endif
```

Como podemos observar en el fragmento anterior se declaran todas las funciones disponibles para implementar un sensor, al ser un dispositivo I2C necesitamos acceso a la librería de funciones del I2C. Esto se consigue mediante la inclusión de ***"i2cmaster.h"***, para un mejor orden, las direcciones de los dispositivos se definen en constantes en ***"i2cdevices.h"***. La inclusión de ***"modSensor.h"*** es obligatoria ya que contiene todas las herramientas para comunicar con el módulo de sensores.

La constante **numReads** nos permite configurar cuantos resultados serán leídos de cada sonar, estos son capaces de proporcionar hasta 16 resultados en cada lectura, si se desea limitar la cantidad leída podemos hacerlo modificando esta constante.

La función **iniciar,** que debe ser declarada y definida tal y como vemos en el siguiente fragmento, dado que es el prototipo que espera el módulo de sensores. **Iniciar** se encarga de enviar la orden de iniciar lectura al sonar.

```
short Sonar_iniciar(unsigned short timestamp_n,char num) {

 temp = timestamp_n - sonartstamp_ch[(short)num];

 if ( temp > 68) {
i2c_writebyte (sonaraddr_ch[(short)num],0,81); // modo distancia cm
   sonartstamp_ch[(short)num] = timestamp_n; // nuevo timestamp
}
return 0;

}
```

Como podemos observar, el primer paso es comparar la marca de tiempo del sonar en cuestión, para reducir código redundante, se hace uso del parámetro "num." que esta para utilización a discreción del controlador, en este caso lo usamos para distinguir entre diferentes sonars.

Así que la primera instrucción, calcula la diferencia entre la "fecha" del último acceso y la "fecha" actual, si la diferencia es de 68 milisegundos o más entonces, se enviara el comando vía el bus I2C. Que se traduce a escribir en la dirección **sonaraddr_ch[(short)num],** registro 0, el valor 81, que índica según la documentación del sonar, inicio de lectura.

Existen 2 funciones de lectura una para leer el resultado de la exploración y otra para leer el sensor de luz incorporado en el sonar, es casi la misma función, solo cambia el valor devuelto, las dos aprovechan para leer los datos de la otra función de esta forma se minimiza el acceso al bus I2C.

Los primeros pasos es recoger la dirección I2C, y la marca de tiempo del dispositivo que toca en función de **num.,** si el tiempo es el adecuado, se lee en bloque el valor del sensor de luz y después el numero de registros especificados en numReads multiplicado por 2 dado que el bus I2C solo lee bytes, y cada registro es de 2 bytes.

La siguiente línea guarda el valor del sensor de luz que es un byte, el siguiente bucle for prepara una cadena con los datos, los cambia base 36 de esta forma minimizamos los datos a mandar por el puerto serie, los valores recibidos varían entre 3 y 600 eso en base 36 es como máximo 2 cifras, el bucle concatena un 0 por delante a los datos que no tengan las 2 cifras, de esta forma se envía una cadena de hasta 32 bytes, esta cadena es almacenada con los datos locales.

Para pasar esta cadena al módulo de sensores, se devuelve un puntero al arreglo con la cadena de valores ya construida, y en el parámetro len se devuelve la longitud de esta,

El módulo espera este método de trabajo porque ha sido así determinado en la inicialización.

```
short Sonar_leer (unsigned short timestamp_n,char* len,char num)  {


  char bloque[(numReads*2)+1];
  char i;
  short * a;
  char addr;

  addr = sonaraddr_ch[(short)num];

  temp = timestamp_n - sonartstamp_ch[(short)num];

  if ( temp > 68) {
 // ha transcurrido el tiempo suficiente para leer

  i2c_readblock(addr,1,bloque, (numReads*2)+1);

  sonarvalue_ch[(short)num][0] = bloque[(numReads*2)];


  for (i=0;i < numReads;i++) {

          sonarvalue_ch[(short)num][(short)(i<<1)+1] = '0';

          a = (short*)&bloque[((numReads*2)-2)-(i<<1)];
          if ( *a > 36 )  itoa(*a,&sonarvalue_ch[(short)num][(i<<1)+1],36);
          else itoa(*a,&sonarvalue_ch[(short)num][(i<<1)+2],36);
  }


  }
  else {
  //comprobar que no sea el cambio.. de 32767 a 0
    if (temp < 0) sonartstamp_ch[(short)num] = timestamp - 60;

  }

  *len = (numReads*2);
  return (short)&sonarvalue_ch[(short)num][1];

}
```

La función de leer el sensor de luz es prácticamente igual solo que en lugar de devolver el puntero y la dirección, devuelve un número entero de 16bits con el valor, este cambio radical de funcionamiento para el mismo prototipo de función se configura en la inicialización.

```
short SoLuz_leer(unsigned short timestamp_n,char* len,char num) {

  (void) len;
  char bloque[(numReads*2)+1];
  char i;
  short * a;
  char addr = sonaraddr_ch[(short)num];
  temp = timestamp_n - sonartstamp_ch[0];

  if ( temp > 68) {
 // ha transcurrido el tiempo suficiente para leer

 i2c_readblock(addr,1,bloque, (numReads*2)+1);

  sonarvalue_ch[(short)num][0] = bloque[(numReads*2)];


  for (i=0;i < numReads;i++) {
         sonarvalue_ch[(short)num][(short)(i<<1)+1] = '0';
         a = (short*)&bloque[((numReads*2)-2)-(i<<1)];
         if ( *a > 36 )  itoa(*a,&sonarvalue_ch[(short)num][(i<<1)+1],36);
         else itoa(*a,&sonarvalue_ch[(short)num][(i<<1)+2],36);

        }
  }
  else {
  //comprobar que no sea el cambio.. de 32767 a 0
    if (temp < 0) sonartstamp_ch[(short)num] = timestamp - 60;
  }
   return 0x00FF & sonarvalue_ch[(short)num][0];
}
```

El siguiente trozo de código contiene la inicialización, se reserva memoria para albergar las direcciones de los dispositivos (en este caso 3), luego espació para el buffer de datos leídos, y por ultimo tres enteros para guardar la marca de tiempo.

La función de inicialización comienza estableciendo los datos iniciales, poniendo los búferes en un valor conocido, y recogiendo las direcciones de los dispositivos de las constantes. , después envía una primera petición de lectura para que los sensores se inicialicen.

Y por último y mas importante añade los sensores al modulo de sensores, podemos observar como los sensores de luz son añadidos como sensores completamente independientes de los de distancia.

```
#include "drvsonar.h"

char sonaraddr_ch[3];
char sonarvalue_ch[3][(numReads*2)+2];
short sonartstamp_ch[3];

short temp;

 void SonarStart(void){


 for(temp = 0;temp < (numReads*2)+1;temp++){
 sonarvalue_ch[0][temp]=0;
 sonarvalue_ch[1][temp]=0;
 sonarvalue_ch[2][temp]=0;
 }


 sonaraddr_ch[0] = DevSRF08_1;
 sonaraddr_ch[1] = DevSRF08_2;
 sonaraddr_ch[2] = DevSRF08_3;

 // Una primera lectura:

 // inicializar a modo de distancia en centimetros.
 i2c_writebyte (sonaraddr_ch[0],0,81);
 i2c_writebyte (sonaraddr_ch[1],0,81);
 i2c_writebyte (sonaraddr_ch[2],0,81);

 // Attach to sensor list
 sensorAttach(&Sonar_iniciar, &Sonar_leer,"SO1R",0,SENSOR_BIN8);
 sensorAttach(&Sonar_iniciar, &SoLuz_leer,"SO1L",0,SENSOR_INTEGER);
 sensorAttach(&Sonar_iniciar, &Sonar_leer,"SO2R",1,SENSOR_BIN8);
 sensorAttach(&Sonar_iniciar, &SoLuz_leer,"SO2L",1,SENSOR_INTEGER);
 sensorAttach(&Sonar_iniciar, &Sonar_leer,"SO3R",2,SENSOR_BIN8);
 sensorAttach(&Sonar_iniciar, &SoLuz_leer,"SO3L",2,SENSOR_INTEGER);

 }
```

El primer parámetro de sensorAttach, es la dirección de la función para iniciar la lectura, el siguiente otro puntero esta vez a la función de lectura, como podemos ver esta es la que diferencia el sensor de luz del de distancia. El tercer parámetro es un nombre, estos son códigos alfanuméricos de 4 cifras que posteriormente es utilizado por el módulo del Pyro para identificar los dispositivos. El número que ocupa el cuarto parámetro es el parámetro **num** que hemos visto en las funciones anteriores, y mediante esta diferenciamos entre los diferentes sensores.

El último parámetro define el comportamiento del dispositivo, mientras que el modo SENSOR_INTEGER simplemente espera un número entero devuelto por la función de lectura, SENSOR_BIN8, espera un puntero a un arreglo, de longitud el valor puesto en **len**.

Información detallada del módulo Pyro

Uso y configuración

El módulo del robot consta de los siguientes ficheros:

- **/pyrobot/robot/device.py,** Aquí se han realizado modificaciones, para poder crear la imagen que muestra la termopila.
- **/pyrobot/robot/ari.py,** El fichero principal, que implementa el robot.
- **/pyrobot/robot/ARIConfig/** En esta carpeta se encuentran los ficheros de configuración que permiten diferentes distribuciones físicas y configuraciones de sensores.
- **/pyrobot/plugins/robots/ARI.py** Este es el fichero de carga, este nos mostrara una ventana para elegir el puerto serie y el fichero de configuración. Debemos arrancar el módulo desde aquí.

Los ficheros de configuración simplemente crean una lista de tuplas global, estas tuplas consta de 4 elementos que dan la configuración a los diferentes dispositivos.

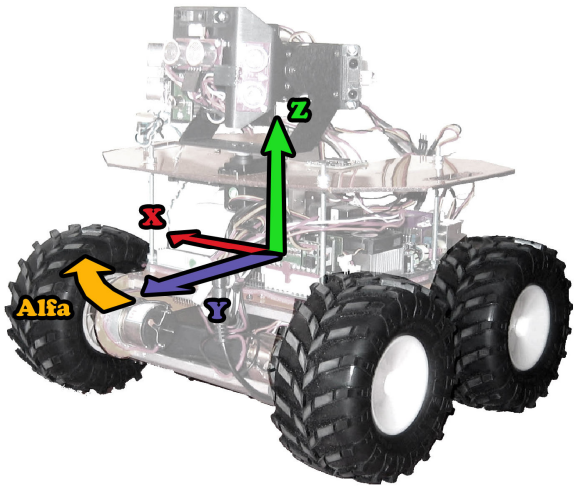
```
configuracion.append(("compas","CO16",(0,40,15,0),0))
configuracion.append(("ptz","",(0,50,70,0),(1,0)))
configuracion.append(("ir","ADC3",(-30,-30,30,0),0))
configuracion.append(("sonar","SO1R",(0,30,30,0),0))
configuracion.append(("light","SO1L",(0,30,30,0),0))
configuracion.append(("sonar","SO2R",(30,30,30,-45),0))
configuracion.append(("light","SO2L",(30,30,30,-45),0))
configuracion.append(("tpambient","TPAB",(0,30,15,0),0))
configuracion.append(("tparray","TPDA",(0,30,15,0),0))
```

El primer parámetro contiene la clase de dispositivo a la que va asociada la información, el segundo parámetro es el dispositivo de la unidad de control a la que va asociado, el tercer parámetro indica la posición desde el centro del robot o el ultimo dispositivo ptz, y el cuarto parámetro debe ser siempre 0 excepto para el dispositivo ptz.

Por ejemplo el primer dispositivo es de la clase **compás** , tiene asignado los datos del sensor de la unidad de control llamado CO16, esta posicionado según la tupla del tercer parámetro, que indica lo siguiente (x, y, z, ,alfa) son las posiciones en milímetros

respecto del centro del robot , y alfa es la orientación respecto el frente en grados, estos datos son importantes, porque en función de ellos se calcula la abstracción de “front” , “front-left” , “back” , que es la más usada en el pyro.

Hemos de hacer un inciso especial en el dispositivo “ptz”, este dispositivo controla 2 servomotores, y además explota las funcionalidades ya explicada del barrido dentro de Pyro, en lo que afecta a la configuración, hace que cualquier otro dispositivo declarado después de el se trate como si estuviese montado en el y su posición será relativa a la posición del ptz en lugar del centro del robot.



Base de coordenadas para la orientación de los sensores en la configuración.

Si encontramos otra unidad de ptz esta no será montada encima de la anterior, si no que será tratada como de otra unidad aparte, y todo sensor declarado posteriormente pertenecerá solo a esta unidad ptz.

El cuarto parámetro de la unidad ptz son los dos servomotores de los que depende la unidad para realizar su movimiento, el primer número es el servo asignado para “Pan”, y el segundo número indica el servo asignado para tilt.

Clases AriRobot y AriInputProcessor

Las clases **AriRobot** y la clase **AriInputProcessor** son de las clases más importantes dentro del módulo del robot.

La Clase **AriRobot** deriva de la clase robot, y sobre carga de la anterior los siguientes métodos: __init__ , setup, __getitem__ , __del__, addDevice, translate, rotate , move, startDeviceBuiltin, update.

En el constructor "__init__" simplemente guarda los parámetros de configuración como variables locales al objeto, luego posteriormente Pyro invoca setup que es quien busca y configura el sistema.

El primer paso es intentar conectar con la unidad de control, si no la encuentra abortará la carga del módulo, el siguiente paso es cargar los dispositivos indicados en el fichero de configuración, se asigna todos los dispositivos de distancia al parámetro range, y por último se inicializa **AriInputProcessor** que es un thread que se encarga de recibir y procesar toda la información recibida del la unidad de control.

La sobrecarga de __getItem__ permite fusionar en la propiedad range del robot diversos tipos de dispositivos de distancia, **range,** junto a **move, translate** y **rotate**, son los métodos más elementales que utiliza el pyro para recibir información de distancia y moverse.

Otros métodos y propiedades permiten la generación de datos intermedios , como por ejemplo : la propiedad **groupsbydegrees** guarda una relación entre los grupos de posiciones y ángulos de orientación, esta propiedad es principalmente usada por **getGroupsByDegreesDephase,** para calcular la nueva tabla desfasada los grados que se le pasen, esto es utilizado por los dispositivos y sensores para poder calcular el grupo al que pertenecen.

Los métodos: **readservo, timeattach, timedeattach,** y **read** sirven a los dispositivos para asignarse a una lista u otra , con **timeattach** y **timedeattach** , se asignan o no a la lectura periódica, con **readservo** se asigna a la lectura sincronizada con el servomotor que se le indique.

Por ultimo la función **execute** permite en envío de cualquier comando, es la única función que realizar salida real sobre el puerto serie, y esta arbitrada por semáforo para evitar salidas incorrectas.

La clase **AriInputProcessor**, recibe todo tipo de respuestas de la unidad de control, este **thread** trata de mantener actualizados al último instante los siguientes diccionarios :**sensorbyname**, **sensornamenumber**, y las listas **sensorbynumber**, **sensornumbername**. Los elementos **sensornamenumber** y **sensornumbername** son simplemente relaciones entre los nombres y el número de sensor, donde realmente se guardan los valores es en **sensorbyname** y **sensorbynumber.**

Clase PTZDevice

La clase **PTZDevice** es muy especial como hemos visto anteriormente, es la única clase dispositivo final que no deriva de **OwnDevice,** esto se debe a su especial funcionalidad.

Cuando los dispositivos son inicializados, estos reciben por su constructor un parámetro que indica a que unidad **PTZDevice** pertenece, si encuentran nulo este parámetro quiere decir que no pertenecen a ninguna, el siguiente paso es invocar ***attachSensor*** del dispositivo ptz, de esta forma el dispositivo ptz puede mantener una lista de los dispositivos que dependen de el.

Los métodos mas importantes en **PTZDevice** son : **updateDevice** y **generateGroups.** En **updateDevice** se lleva a cabo la comprobación de la configuración de barrido, si todo sigue como hasta ahora no se realizara acción ninguna, si no se invocará el método **PTZupt** de cada dispositivo en la lista de dependencia. Estos invocaran **generateGroups,** con su ángulo de desfase respecto de la unidad ptz, **generateGroups** genera una tupla de 5 elementos, con listas y diccionarios con los datos necesarios para poder "multiplicar" el sensor y que aparezca como un numero mayor de sensores según los pasos.

El primer parámetro devuelto por **generateGroups** es los ángulos en que se realizan las paradas del barrido por ejemplo: [-90, -45, 0, 45, 90],estos se utiliza para saber donde almacenar la lectura actual en función de la posición del servomotor.

El segundo parámetro, es un diccionario que como índice tiene los ángulos anteriores, y como datos tiene el índice de posición.

El tercer y cuarto parámetro, son dos listas que contienen las coordenadas x e y del sensor en cada paso, por último el quinto parámetro son los grupos “front”, “front-left” etc.… con las posiciones según se encuentren en el robot.

Clase OwnDevice

Como hemos visto hasta ahora en la clase anterior, hay una fuerte relación entre los dispositivos y la clase **PTZDevice,** para poder centralizar toda esta funcionalidad se heredó la clase **OwnDevice** de la clase **Device** y se agregó la funcionalidad adicional, de esta forma cualquier dispositivo nuevo heredará de **OwnDevice** toda la nueva funcionalidad.

El método **updateDevice** mantiene actualizada la lista values, si no esta activado el modo barrido o no se encuentra encima de una unidad ptz, es una lista donde solo la posición 0 será actualizada, en caso contrario mantendrá una lista de valores según la posición del servomotor correspondiente. Hasta aquí los valores no son tratados ni alterados, tal cual llegan se almacenan.

El método **PTZupt** actualiza los datos según la nueva configuración del dispositivo **PTZupt.**

El método **getSensorValue** es el que invoca el Pyro para leer el sensor, este devuelve una clase llamada **SensorValue** esta clase aparte del valor de la lectura lleva asociada toda la información geométrica de esta.

getSensorValue realiza una llamada a **_getVal** que es quien se encarga según la posición de devolver el valor adecuado.

Dispositivos

Todos los dispositivos heredan de **OwnDevice.** Para implementar un dispositivo simple, es tan sencillo como sobrecargar el constructor y el método **_getVal** , así lo hace por ejemplo el dispositivo **CompasDevice,** sobrecarga el constructor **__init__** para establecer valores de geometría relacionados con el sensor.

```
    def __init__(self, dev,ptzdevice=None,sensorname="CO16",posxyza = (0,0,0,0),param =
""):

    if ptzdevice != None :
      raise AttributeError, "El Compas digital no soporta estar en una unidad pan & tilt"

    # Llamada al constructor de la clase heredada  para inicializar.
    OwnDevice.__init__(self, dev,ptzdevice,sensorname,posxyza,"compas")

    self.arc = 180.0 * PIOVER180 # radians
    self.units    = "RAW" # current report units
    self.radius = dev.radius # universally in METERS
    # ox, oy, oz in METERS as well
    # ---------------------------------------------
    # natural units (not alterable):
    self.rawunits = "RAW"
    self.maxvalueraw = 360.0 # in rawunits
    # ---------------------------------------------
```

La sobrecarga del método **_getVal** es también muy sencilla, por ejemplo en el compás debemos dividir entre 10 el valor recibido, dado que para tener precisión de grado y evitar trabajar con números flotantes a bajo nivel los grados son representados de 0 a 3600.

```
def _getVal(self, pos):
    x =  OwnDevice._getVal(self,pos)

    if x :
       return float(x)/10
    else :
       return 0
```

El caso especial lo marcan aquellos dispositivos que devuelven resultados más elaborados, y complejos que un simple valor, para ello se creó la clase **OwnSensorValue**.

OwnSensorValue, solo extiende a **SensorValue**, con la propiedad **rawValueList,** donde se encuentra la lista de valores leídos, por ejemplo como ya se ha comentado varias veces los sonars son capaces de responder con 16 resultados , pues para mayor compatibilidad , seguimos pudiendo leer el valor mas cercano en **value** pero también podemos leer los 16 en la propiedad **valueList.**

Apéndice III

Herramientas

Instalación y configuración del entorno AVR Studio y FreeRTOS

Instalación del AVR Studio.

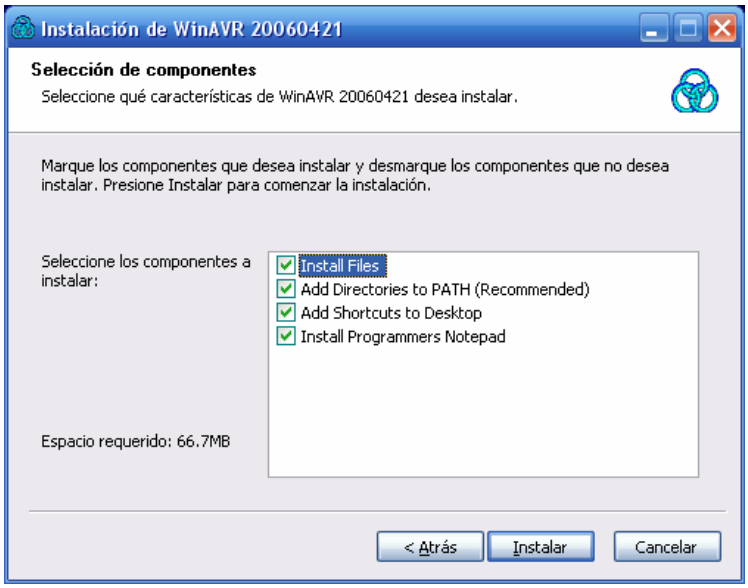
Para instalar el Atmel AVR Studio, al menos en la versión 4.12 que es la empleada en el proyecto se necesitan 3 ficheros:

- **aStudio4b460.exe** Avr Studio 4.12 (build 460).
- **aStudio412SP4b498.exe** Avr Studio 4.12 Service Pack 4 (build 498).
- **WinAVR-20060421-install.exe** WinAVR (Compilador Gcc para la plataforma AVR).

Estos ficheros se encuentran en: /software/avr/ del disco del proyecto.

Instalar este software es muy sencillo primero instalamos el AVR Studio 4 y su "Service Pack 4", son unos pasos sencillos que simplemente escogemos el directorio de destino en el primero.

Instalara el WinAVR también es sencillo pero hemos de tener en cuenta que se encuentren marcados todos los ítems de la siguiente ventana, se podría omitir el último si se quisiera, el WinAVR detecta la instalación del Avr Studio y toma las medidas pertinentes.



Instalación del FreeRTOS.

El FreeRTOS viene en un paquete comprimido, simplemente es una estructura de directorios que deberemos descomprimir en una carpeta, y anotar la ruta porque más adelante la necesitaremos.

La versión utilizada del FreeRTOS es la 4.1.2, en el disco del proyecto puede encontrarse en **/software/freeRtos/FreeRTOSV4.1.2.exe**.

Una vez descomprimido deberemos aplicar la modificación para poder utilizar el FreeRTOS con el AVR ATMega128, esta se encuentra en **/proyecto/FreeRTOS-ATMEGA128/FreeRTOS –ATMEGA128.zip.** Este fichero simplemente se ha de descomprimir en la misma carpeta que el FreeRTOS.

Instalación y Configuración de ARI.

El código fuente de ARI se encuentra en **/proyecto/avr/ARI ATMEGA 128.zip** .Para instalarlo se debe descomprimir en una carpeta, una vez hecho esto abrirlo con el AVR Studio 4 (fichero del proyecto **RTOS.aps**).

Este proyecto no usa el generador de makefile incorporado en Avr Studio 4, así que para cada fichero que creemos nuevo debemos agregarlo al makefile, después antes de poder ponernos a trabajar, necesitamos indicar en el makefile la ruta del FreeRTOS.

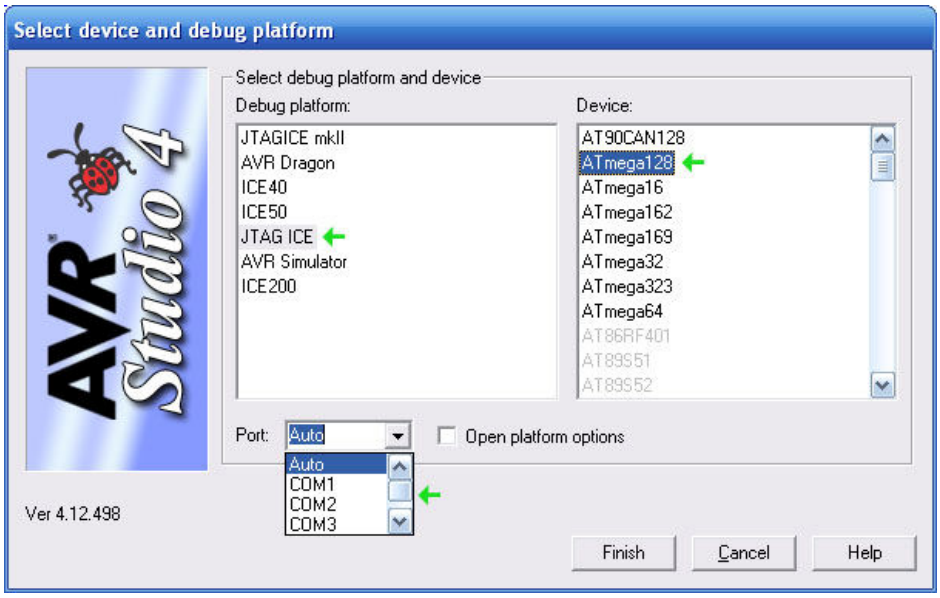
```
…

# Optimization level, can be [0, 1, 2, 3, s]. 0 turns off optimization.
# (Note: 3 is not always the best optimization level. See avr-libc FAQ.)
OPT = 3


# List C source files here. (C dependencies are automatically generated.)
ABSOLUTO = D:/Proyecto-Robot/FreeRTOS
DEMO_DIR = $(ABSOLUTO)/demo/Common/Minimal
SOURCE_DIR = $(ABSOLUTO)/Source
# Modificacion personal para liberar timer1.
PORT_DIR = $(ABSOLUTO)/Source/portable/GCC/ATMega128

SRC     = \
main.c \
…
```

Debemos sustituir el texto en negrita, por la ruta donde hemos instalado FreeRTOS. Ahora el proyecto debería compilar sin problemas.

Configuración del ETT-AVR JTAG como Depurador

Una vez configurado el proyecto y seamos capaces de generarlo si errores, para poder depurar y programar el dispositivo necesitamos configurar el AVR-JTAG.

Para utilizar el **ET-AVR-JTAG** en modo depuración, debemos ir al menú ***Debug*****,** hacer clic en ***"Select Platform and Device",*** nos aparecerá una ventana para seleccionar una herramienta para programar y depurar



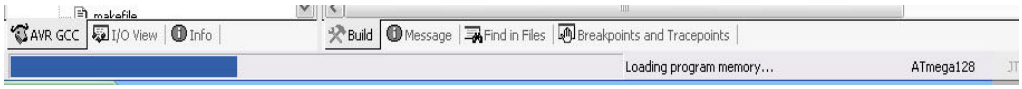
Deberemos seleccionar la herramienta ***"JTAG ICE"***, y el dispositivo final en cuestión, en el caso de este proyecto el ***"ATmega128",*** seleccionamos el puerto serie donde tengamos el **ET-AVR-JTAG** conectado y finalmente hacemos clic en ***"Finish"***.

Para poder comprobar que todo sea correcto podemos probar en generar el proyecto e iniciar la depuración, para compilar, seleccionar la opción ***"Rebuild all"*** del menú ***"Build"*** y para iniciar la depuración, la opción ***"Start Debugging",*** del menú ***"Debug"***. Si nos saliese esta ventana:

Quiere decir que no encuentra el dispositivo, el típico fallo es que dado que el **ET-**

AVR-JTAG se alimenta de la placa de destino, no haber conectado la alimentación.

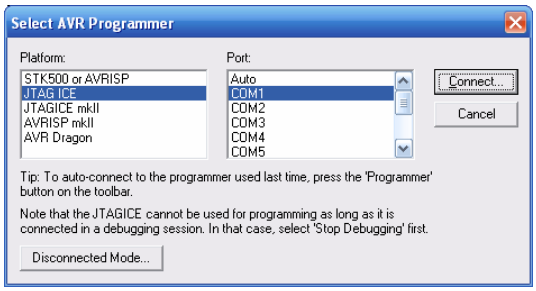
En caso de que el funcionamiento sea correcto veremos en la barra de estado de la aplicación **AVR-Studio** el progreso de la operación:



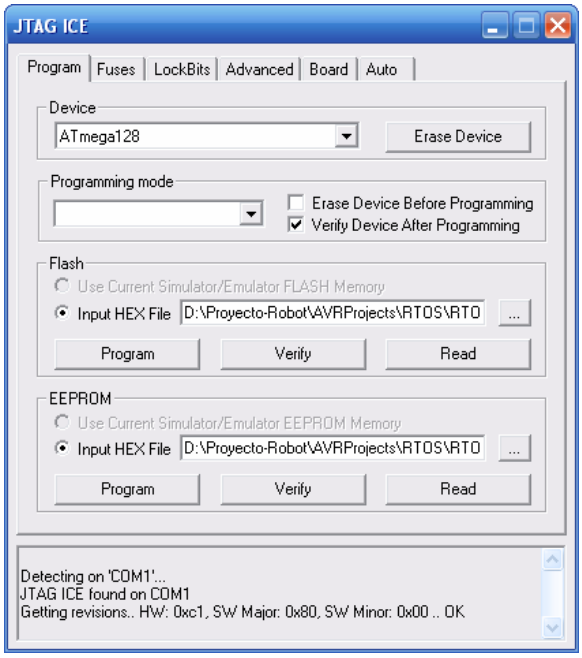
Configuración del ETT-AVR JTAG como Programador.

Aunque normalmente una vez en modo depuración el programa ya es grabado, y con solo abortar dicho modo, es suficiente para grabar el microcontrolador, los siguientes pasos muestran como usar el **ET-AVR-JTAG,** como solo programador.

Seleccionando del menú ***"Tools"*** las opciones ***"Program AVR > Connect"*** , nos mostrará la siguiente ventana que seleccionaremos las opciones igual que en modo depuración.



La siguiente ventana muestra todas las opciones disponibles del dispositivo seleccionado, normalmente para programar el dispositivo, pulsando el boton ***"Program"*** del panel ***"Flash"*** se grabará la ultima versión del programa.



Para mas información de las configuraciones disponibles en las otras pestañas consultar la documentación del microcontrolador [4].

Instalación de Python, Pyro y módulo de ARI.

Podría instalar versiones mas actualizadas de Python y Pyro pero por seguridad en el disco del proyecto se incluyen las versiones utilizadas durante el desarrollo, esta están totalmente probadas.

Tanto Python como Pyro funcionan en entornos Linux, mejor dicho, es donde mejor funcionan, la versión de Windows de Pyro tiene gran parte de su funcionalidad limitada o desactivada, aun así se desarrollo en Windows por ser la herramienta había en su momento disponible, de todas formas gracias a la portabilidad de Python, el módulo ARI del robot debería funcionar a la perfección.

Instalación de Python 2.4 y Módulos Accesorios

En **/software/pyro/Windows/** se encuentran los siguientes ficheros.:

- **Python-2.4.1.msi** , Entorno Python 2.4.
- **PIL-1.1.5.win32-py2.4.exe** Python Image Library para Win32.
- **Numeric.23.8.win32-py2.4.exe**, Python Numeric Extensions para Win32.
- **pyWin32-210.win32-py2.4.exe,** Python Extensions for Microsoft Windows.

La instalación esta muy simplificada y no reviste mayor gravedad que elegir el directorio de destino e ir pulsando con el ratón en siguiente. Eso si el entorno Python debe ser instalado antes que los demás.

Instalación del entorno Pyro

La instalación de Pyro es simplemente descomprimir el fichero del disco: **/software/pyro/Windows/pyrobot-windows-4.8.5.zip**, en una carpeta.

Una vez realizado si esta Python correctamente instalado, Pyro debería arrancar ejecutando simplemente /bin/pyrobot.py.

Instalar ARI.

El módulo Pyro de ARI se encuentra localizado en : **/proyecto/pyro/pyrobot.zip .** Instalarlo es tan sencillo como descomprimirlo en la misma carpeta donde se encuentra Pyro.

Para añadir una funcionalidad extra se modificó /robot/device.py que es parte de Pyro, es posible que en versiones mas actualizadas de Pyro este fichero reciba modificaciones, recomendable ante una posible actualización respetar el /robot/device.py y simplemente añadir el siguiente código en negrita:

```
def addButton(self, name, text, command):
        """Adds a button to the device view window."""
        self.widgets[name] = Tkinter.Button(self, text=text, command=command)
        self.widgets[name].pack(fill="both", expand="y")
    def addCheckbox(self, name, text, variable, command):
        """Adds a checkbox to the device view window."""
        self.widgets[name] = Tkinter.Checkbutton(self, text=text, variable=variable,
command=command)
        self.widgets[name].pack(anchor="w")
    def addLabel(self, name, text):
        """Adds a label to the device view window."""
        self.widgets[name] = Tkinter.Label(self, text=text)
        self.widgets[name].pack(fill="both", expand="y")
    def addCanvas(self,name,mwidth,mheight,mbg):
        """Adds a canvas to the device view window."""
        self.widgets[name] = Tkinter.Canvas(self,bg=mbg,width = mwidth,height=mheight)
        self.widgets[name].pack()
        return self.widgets[name]
    def updateWidget(self, name, value):
        """Updates the device view window."""
        try:
            self.widgets[name+".entry"].delete(0,'end')
            self.widgets[name+".entry"].insert(0,value)
        except: pass
    def addData(self, name, text, value):
        """Adds a data field to the device view window."""
        self.visibleData = 1
        frame = Tkinter.Frame(self)
        frame.pack(fill="both", expand="y")
```

RESUM:

En aquest projecte, s'ha dissenyat, construït y programat un robot autònom, dotat de sistema de locomoció i sensors que li permeten navegar sense impactar en un entorn controlat. Per a assolir aquests objectius s'ha dissenyat i programat una unitat de control que gestiona el hardware de baix volum de dades amb diferents modes de operació, abstraient-lo en una única interfície. Posteriorment s'ha integrat aquest sistema en el entorn de robòtica Pyro. Aquest entorn permet usar i adaptar, segons es necessiti, eines d'intel·ligència artificial ja desenvolupades.

RESUMEN:

En este proyecto, se ha diseñado, construido y programado un robot autónomo, dotado de sistema de locomoción y sensores que le permiten navegar sin impactar en un entorno controlado. Para alcanzar los objetivos se ha diseñado y programado una unidad de control que gestiona el hardware de bajo volumen de datos en diferentes modos de funcionamiento, abstrayéndolo en una única interfaz. Posteriormente se ha integrado este sistema en el entorno de robótica Pyro. Este entorno que permite usar y adaptar según sea necesario, herramientas de inteligencia artificial ya desarrolladas.

ABSTRACT:

In this project, an autonomous robot has been designed, built and programmed. It is equipped with a locomotion system and sensors that allows it to navigate without crash on a controlled environment. To achieve this goal a control unit was designed and programmed, it is able to manage hardware with low data rate and has different working modes, it abstract all these thing on a unique interface. Later the platform was integrated into a robotic environment called Pyro. Such environment allows the use and customization of already done artificial intelligence tools.