

Laboratorio de Robótica
Curso 2008-2009
(2º semestre)

Universidad Autónoma de Madrid
Escuela Politécnica Superior

1. Presentación
2. Normas del laboratorio
3. GPBOT: Kit Básico de Robótica
4. Prácticas

5. Programación básica (Motores y Sensores)
6. COP y Frecuencia Interna
7. Programación avanzada (serie y timer)

Andrés Prieto-Moreno Torres

andres@iearobotics.com

Tutorías:

Por mail en cualquier momento

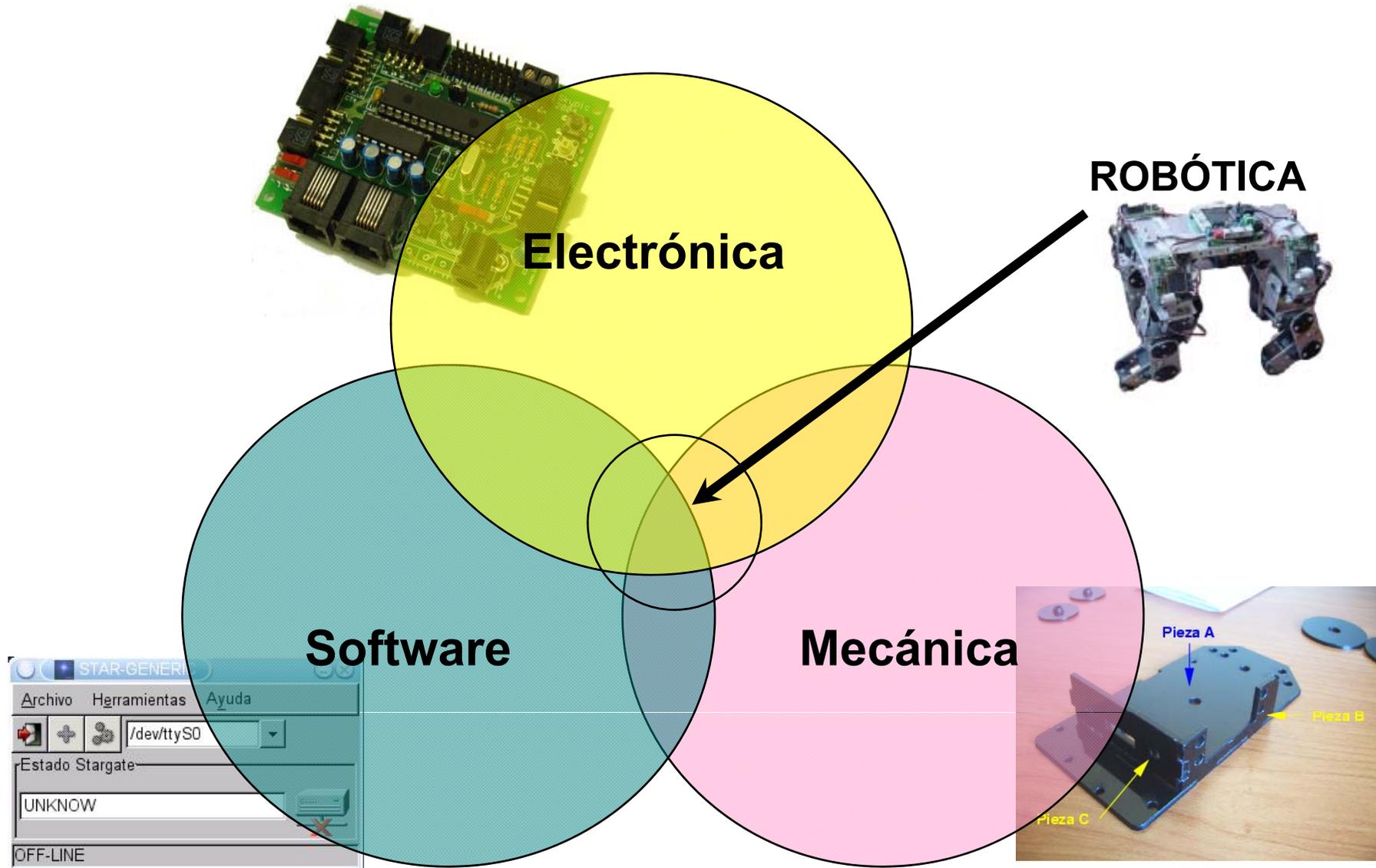
Presenciales:

Miércoles (grupo A): 18:00 – 20:00

Viernes (grupo XX): 16:00 – 18:00

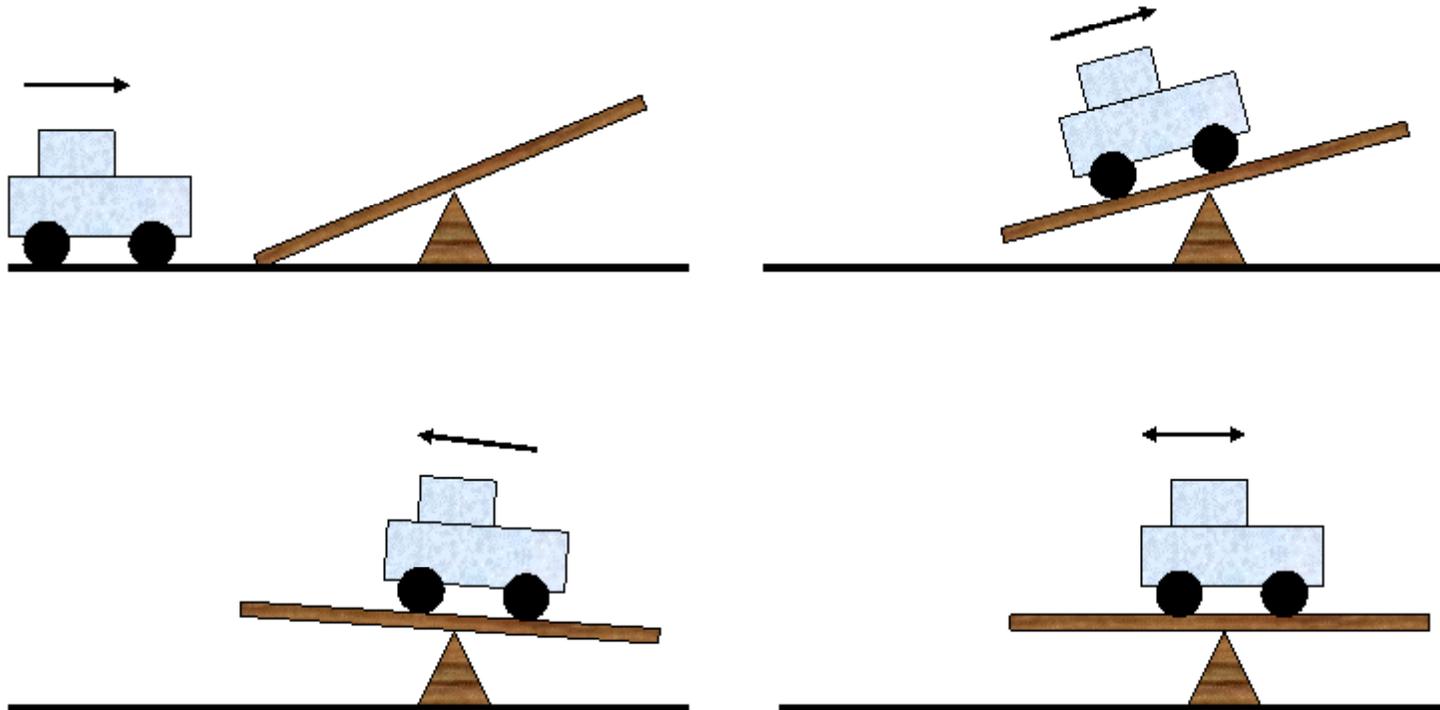
Web asignatura: Guillermo de Rivera (coordinador Laboratorio)

<http://www.ii.uam.es/~gdrivera/robotica/robotica.htm>



- Se aprueba asistiendo a la P0 y aprobando la práctica final con nota >5
- Las prácticas se entregan según el calendario de la web
- Las prácticas se hacen en grupos de hasta 3 personas:
 - Se evalúa independientemente a cada alumno
 - Pueden tener distinta nota.
- Habrá material de la UAM que se dará en préstamo

- La asistencia no es obligatoria pero si recomendable
- Cada grupo hace las prácticas en el horario asignado
- No hay cambios de grupo -> Guillermo de Rivera
- **Las prácticas se evalúan en el laboratorio**
- Copia:
 - Copiado y copiator suspensos
 - Cuidado con dejar la práctica en el ordenador
 - Cuidado con el código de internet



1. Puesta en marcha del entorno (2 semanas)

- Conseguir la placa entrenadora GPBOT
- Fabricar el cable de conexión a la Fuente de alimentación.
- Fabricar cable de conexión serie. (Opcional pero recomendable)
- Instalar el software y probar con algún ejemplo básico

2. Control de motores y sensores (4 semanas) (13-17 Abril)

- Conectar los motores y hacer el control.
- Poner en marcha los encoders.
- Poner en marcha el inclinómetro.

3. Construcción del robot (4 semanas) (18-23 Mayo)

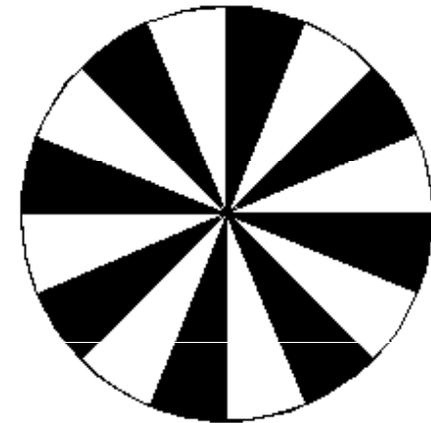
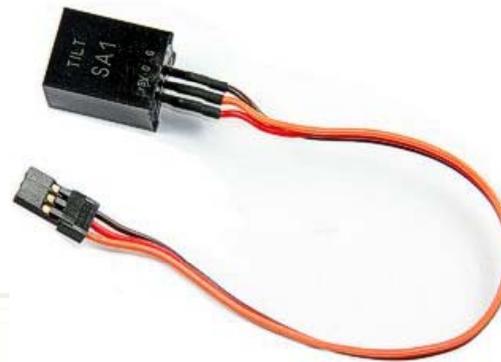
- Construir el robot
- Programar en C un software que realice la estabilización.

Elección motores para la práctica:**Motor Paso a paso:**

Consumen más, van más lento, circuito de encoder separado.

Motor continua:

Consumen menos, Control sencillo. Encoders incorporados al motor.



Trucaje Futaba:

<http://www.iearobotics.com/proyectos/cuadernos/ct2/ct2.html>

Conexión CNY70 y Motor a la GPFAZ

<http://arantxa.ii.uam.es/~gdrivera/robotica/curso0607/p1/p1-doc.html>

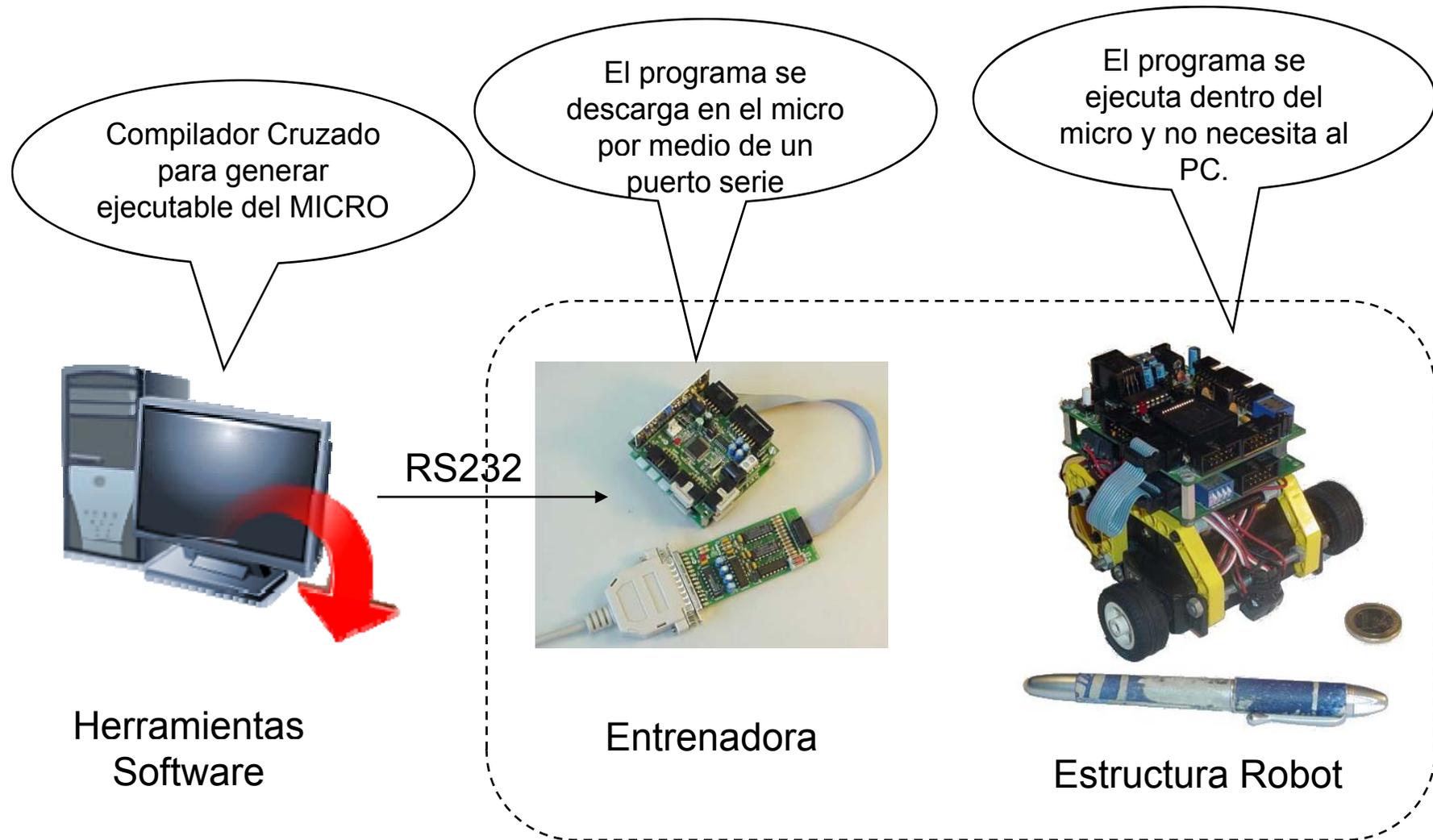
Construcción cable serie

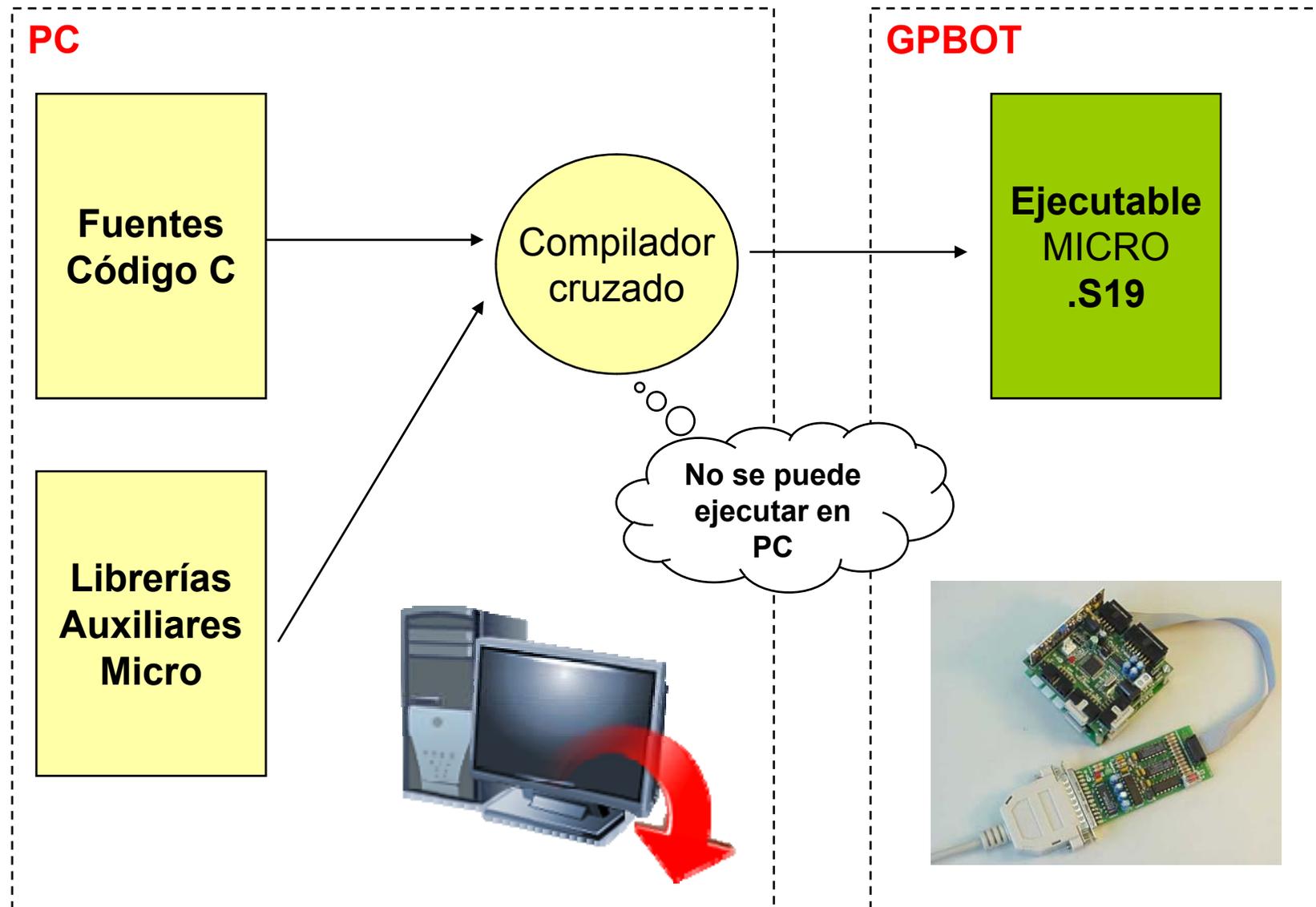
<http://arantxa.ii.uam.es/~gdrivera/robotica/curso0506/p2/p2-doc.html>

Ejemplos de Programación

<http://www.iearobotics.com/personal/juan/proyectos/gpbot/gpbot.html>







Windows

a) IDE: Todo en uno comercial

CodeWarrior: Editor, compilador, simulador y descarga



b) SDCC compilador Free.

Editor **Programer's Notepad**

Compilador **SDCC** (compilador Cruzado de C)

Descarga con **Prog08sz** (Pemicro)

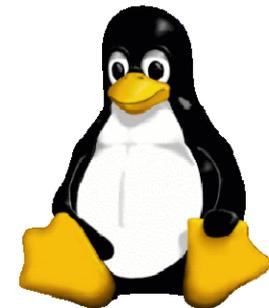
Linux

Editor Anjuta, Vim, emacs

Compilador **SDCC** (make)

Descarga con **GPDOWN** (learobotics)

<http://arantxa.ii.uam.es/~gdrivera/robotica/cuadernos/cl0/cl0.html>



The screenshot displays the Programmer's Notepad 2 interface with the following components:

- Main Editor:** Shows the source code for `ledp.c`. The code includes comments in Spanish and a function `delay_init` that configures a timer. A red arrow points to the parameter `-mhc08 --stack-loc 0x023f %f` in the code.
- Options Dialog:** A dialog box titled "Options" with a "Tools" tab. It lists the tool `sdcc GPBOT` with its command and parameters.
- Properties Dialog:** A dialog box titled "Propiedades de Edit Tool" (Edit Tool Properties) for `sdcc GPBOT`. The "Parameters" field is circled in red and contains the text `-mhc08 --stack-loc 0x023f %f`. A red arrow points from this field to the corresponding parameter in the main editor.
- Output Window:** Shows the command executed: `"C:\Archivos de Programa\SDCC\bin\sdcc.exe" -mhc08 --stack-loc 0x023f ledp.c` and the result: `Process Exit Code: 0`.

PROG08SZ Programmer - Version 2.22.00.00

File Device Program Verify Upload Windows Help

Choose Programming Function

BM Blank check module	NOT ACTIVE
BR Blank check range	NOT ACTIVE
EB Erase byte range	NOT ACTIVE
EW Erase word range	NOT ACTIVE
EM Erase module	NOT ACTIVE
PB Program bytes	NOT ACTIVE
PW Program words	NOT ACTIVE
PM Program module	NOT ACTIVE
CM Choose module .08P	
UM Verify module	NOT ACTIVE
UR Verify range	NOT ACTIVE
UM Upload module	NOT ACTIVE
UR Upload range	NOT ACTIVE
SS Specify Object File	
SM Show module	NOT ACTIVE
HE Help	
QU Quit	
RE Reset chip	
SA Show Algorithm Src	NOT ACTIVE
CS Choose Serial File	NOT ACTIVE

Configuration

Module = none
 Object File = none
 Base =
 Un-Secured =
 Serial File = none
 Next Serial =

Status Window

```
Contact with HC08 Monitor established.
Monitor Code Version:4
Trying Security $FFFFFFFFFFFFFFFF Success (Secu
(68HC08 Device responded properly, but ROM
(Successful Status: 0=Y,1=Y,2=Y,3=Y,4=Y,5=Y
```

Specify Programming Algorithm to Use!

Buscar en: algorithms

- 908_GP20.08P
- 908_GP32.08P
- 908_GP32_HighSpeed.08P**
- 908_GT8.08P
- 908_GT8_HighSpeed.08P
- 908_GT8_trim\$dfff_HighSpeed.08P
- 908_GT8_trim\$ff80.08P
- 908_GT8_trim\$ff80_HighSpeed.08P
- 908_GT8A.08P
- 908_GT8A_HighSpeed.08P
- 908_GT16.08P
- 908_GT16_HighSpeed.08P
- 908_GT16_trim\$dfff_HighSpeed.08P
- 908_GT16_trim\$ff80.08P
- 908_GT16_trim\$ff80_HighSpeed.08P
- 908_GT16A.08P
- 908_GT16A_HighSpeed.08P

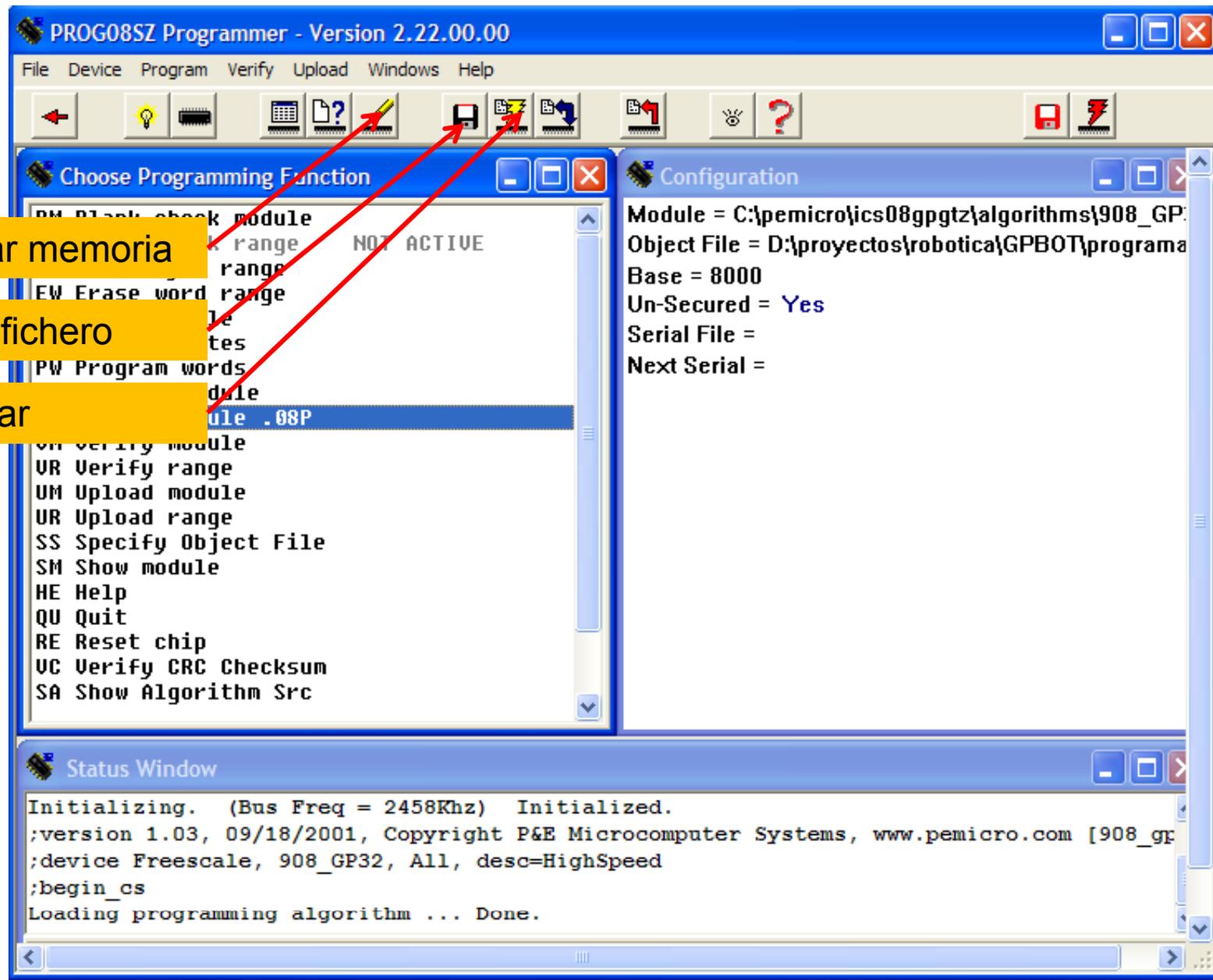
Nombre: 908_GP32_HighSpeed.08P
 Tipo: P&E Prog Algorithm (*.08P)

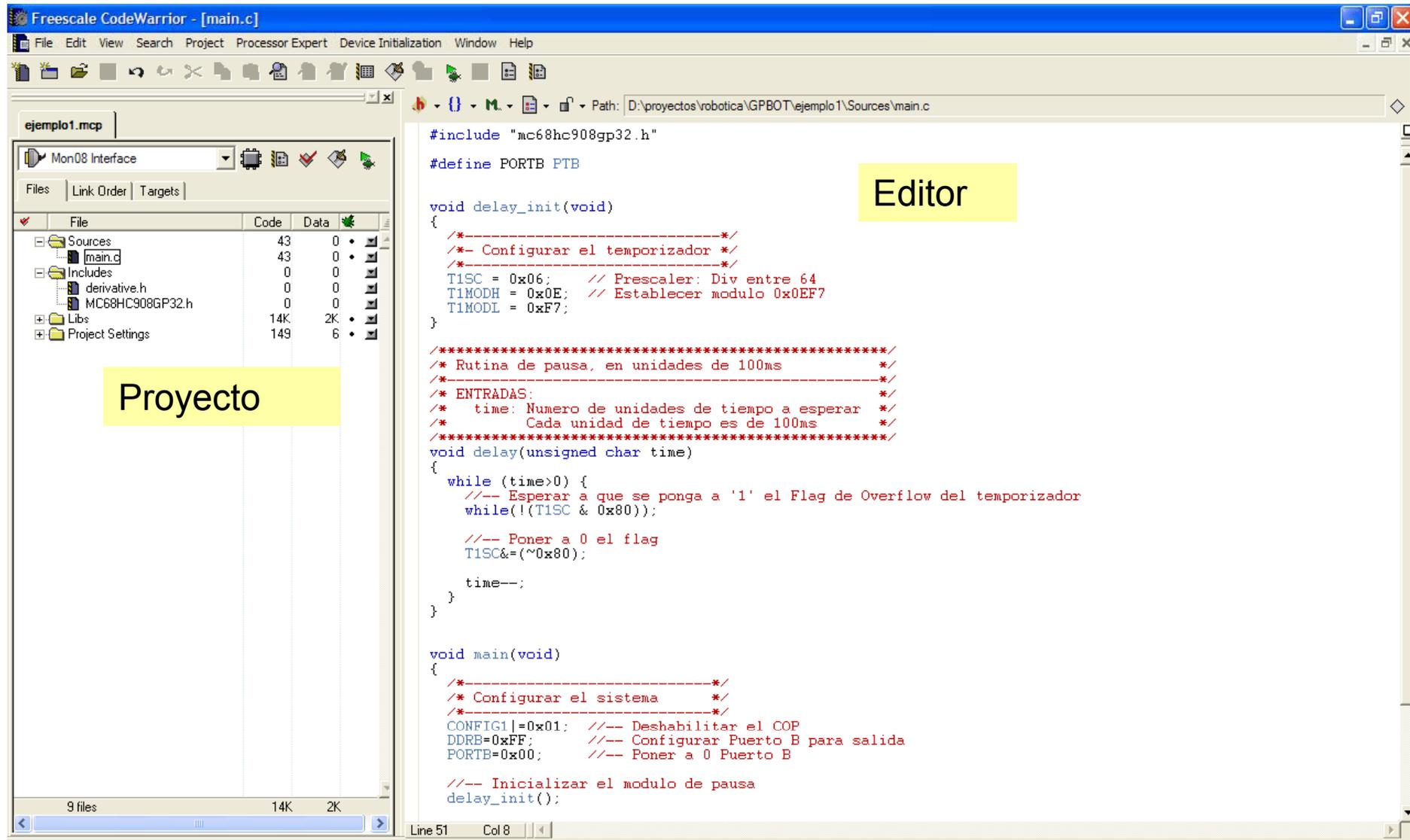
Abrir Cancelar

1. Borrar memoria

2. Abrir fichero

3. Grabar





Microcontrolador = mc68hc908gp32

The screenshot displays the CodeWarrior II IDE interface. On the left, the 'ejemplo1.mcp' project is open, showing a file tree with 'Sources', 'Includes', and 'Libs' folders. A yellow box labeled 'Mon08 Interface' points to the hardware icon in the toolbar. The main workspace is divided into several panes:

- Source:** Shows the C code for 'main.c' with a loop that toggles bit 0 of port B every 500ms. A yellow box labeled 'Simulador en circuito' points to the 'True-Time Simulator & Real' window title.
- Assembly:** Displays the assembly code for the 'main' function, including instructions like BSET, MOV, CLR, BSR, COM, LDA, and BSR.
- Register:** Shows the current state of registers: A=5, HX=4D, SP=14F, SR=7B, Status=VHINZC, and PC=80C8.
- Memory:** Shows a memory dump starting at address 0080.
- Command:** Shows the status 'STOPPED STEPPED OVER' and a prompt 'in>'.
- Data:** Shows variable declarations for _T1SC, _T1MOD, _CONFIG1, and _DDRB.

The status bar at the bottom indicates the target device is 'HC908GP32' and the simulation is in a 'STOPPED' state.

SDCC:

- No añades estructura base. Empiezas desde cero.
- Añadir fichero:
- Los puertos se llaman PORTB, PORTC, PORTD...
- Por defecto el Watchdog está activado y nosotros debemos pararlo

CodeWarrior:

- Añade estructura base (include, MCU_Init, ...)
- Usa librería de definición intermedia
- Los puertos se llaman PTB, PTC, ...
- Por defecto deja activado el Watchdog pero lo gestiona.

Mi recomendación es borrar el Código del CodeWarrior y partir de cero.
Y para que sea compatible con los ejemplos de SDCC poner:

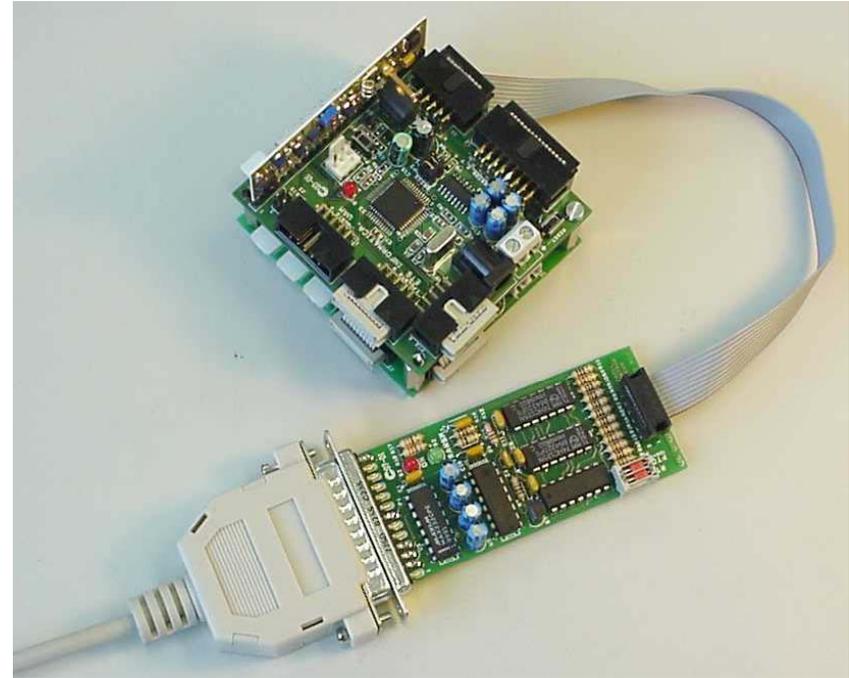
```
#include <mc68hc908gp32.h>  
#define PORTX PTX (donde X puede ser A, B, C, D ...)
```

Micro Motorola MC68HC908GP32

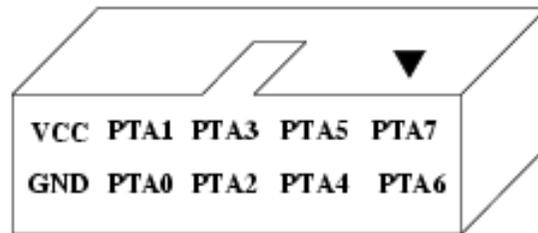
- 32Kb de memoria FLASH
- 512 bytes memoria RAM
- Comunicaciones SPI, SERIE
- 2 temporizadores de 16 bits
- 8 conversores AD
- Pines IO
- Adaptado para programar en C

GPBOT

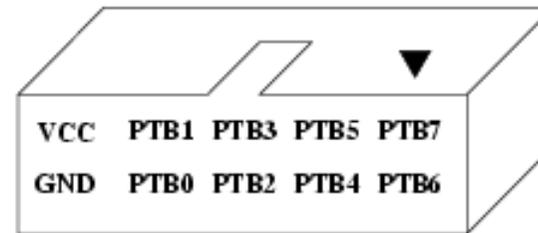
- Recomendable 6.0v a 1A máximo
- Reloj Externo a 9.8 Mhz
- Fbus interna a 2,45 Mhz



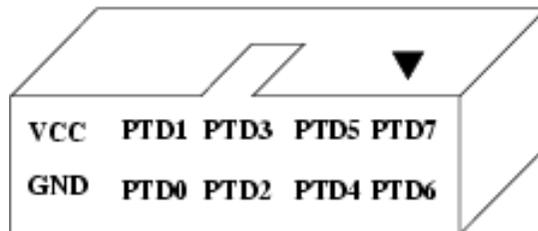
PUERTO A



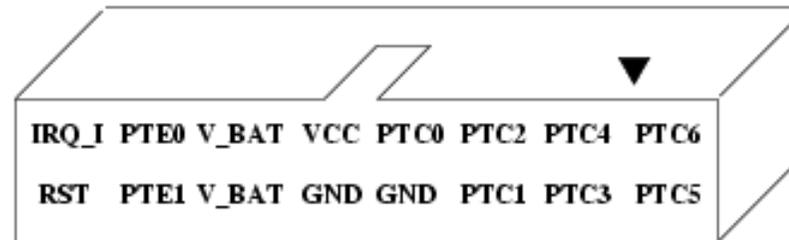
PUERTO B



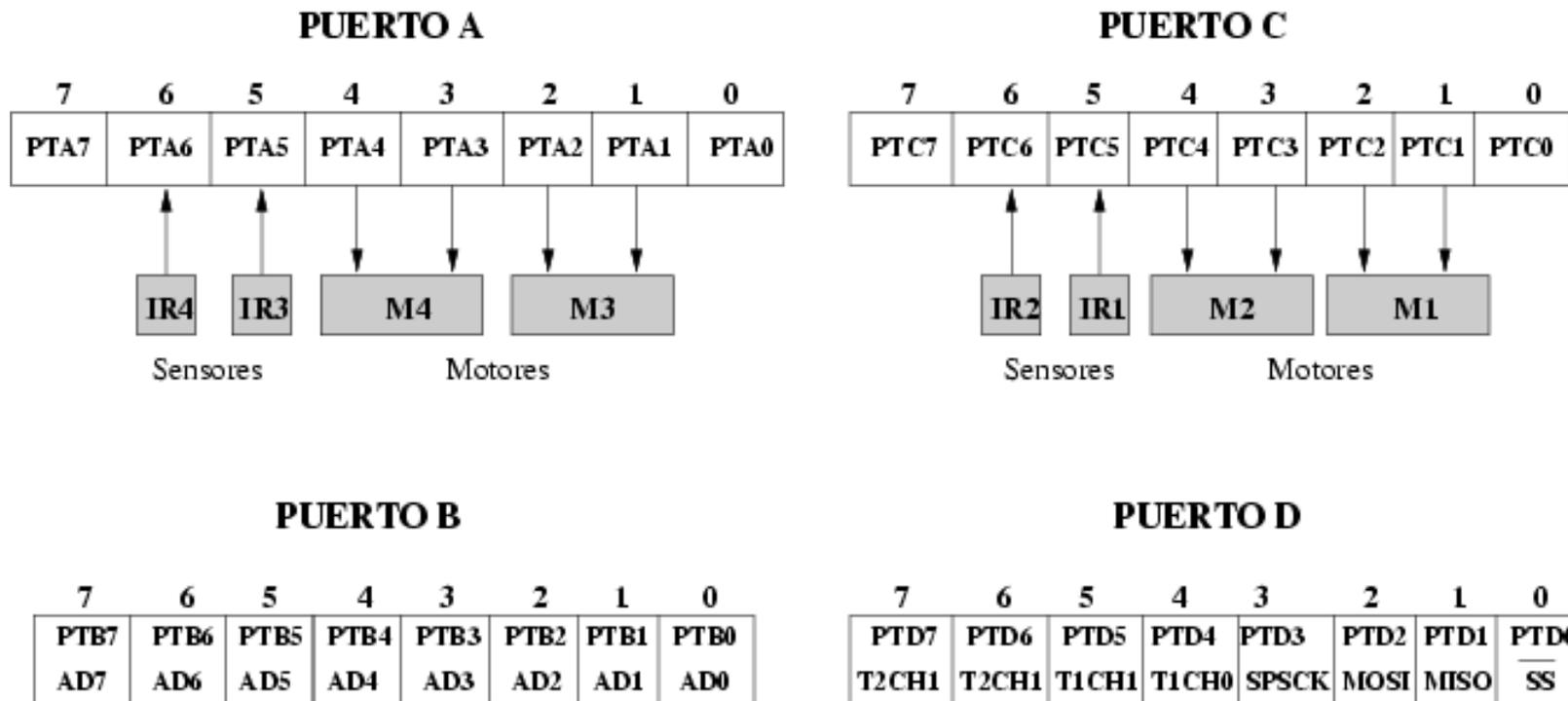
PUERTO D



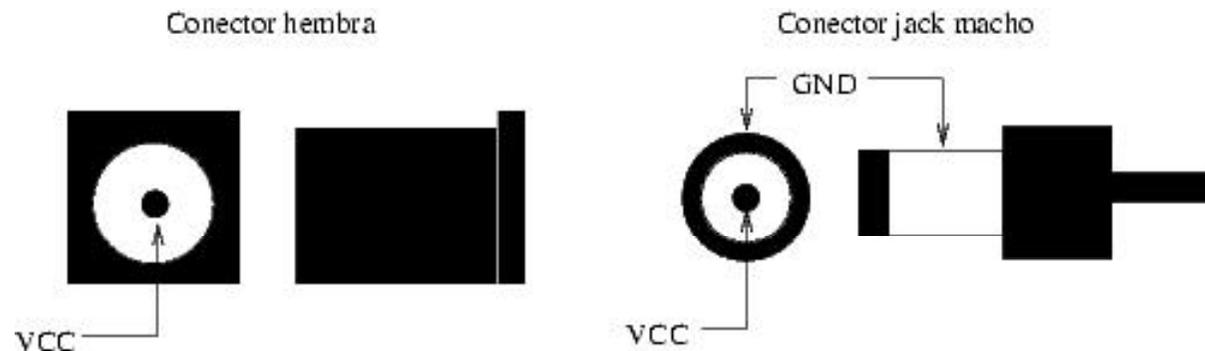
PUERTO C y E



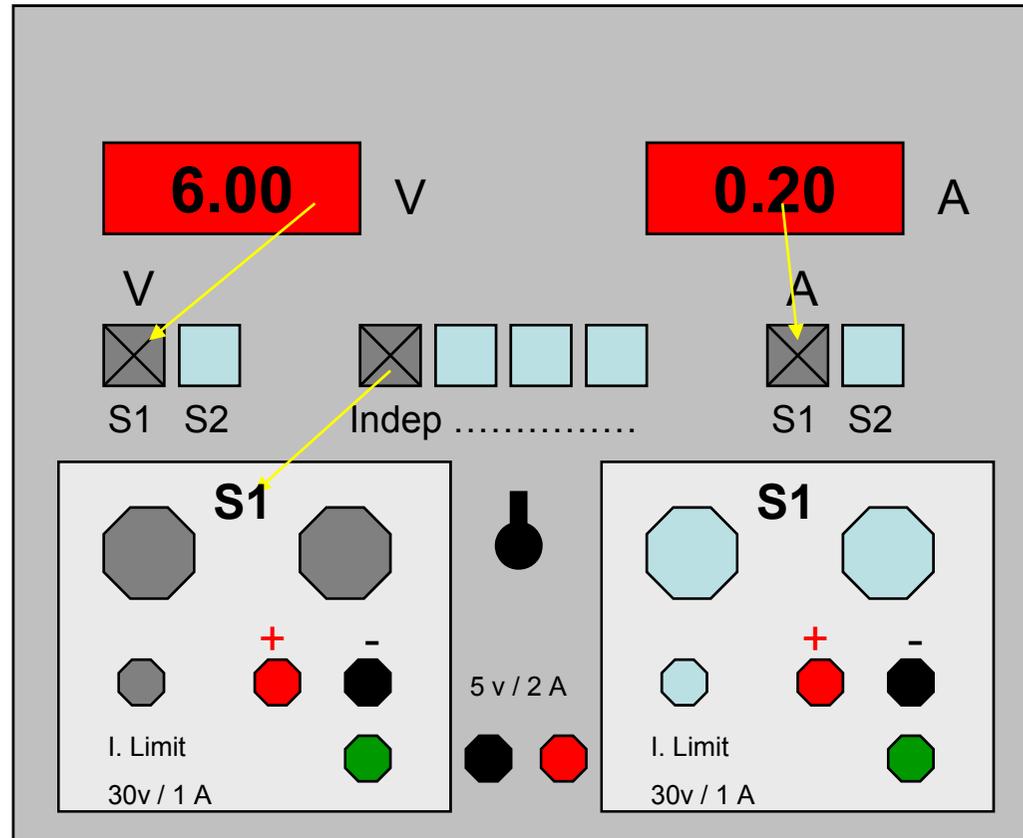
- PTA** Registro de entrada y salida de 8 bits. (PORTA)
DDRA Configura cada bit del Puerto A como entrada (1) o salida (0)
PTAPUE Activa (1) las resistencias de Pull-Up del Puerto A.
- PTB** Registro de entrada y salida de 8 bits. (PORTB)
DDRB Configura cada bit del Puerto B como entrada (1) o salida (0)
- PTC** Registro de entrada y salida de 8 bits. (PORTC)
DDRC Configura cada bit del Puerto C como entrada (1) o salida (0)
PTCPUE Activa (1) las resistencias de Pull-Up del Puerto C.
- PTD** Registro de entrada y salida de 8 bits. (PORTD)
DDRD Configura cada bit del Puerto D como entrada (1) o salida (0)
PTDPUE Activa (1) las resistencias de Pull-Up del Puerto D.
- PTE** Registro de entrada y salida de 2 bits. (Solape con TX y RX)
DDRE Configura cada bit del Puerto E como entrada (1) o salida (0)



Antes de enchufar verificar la tensión de la fuente



Alimentación entre 6 y 9 voltios. Recomendable 7.5 v



Cortocircuito -> $A = \infty$!!! -> $V = 0$

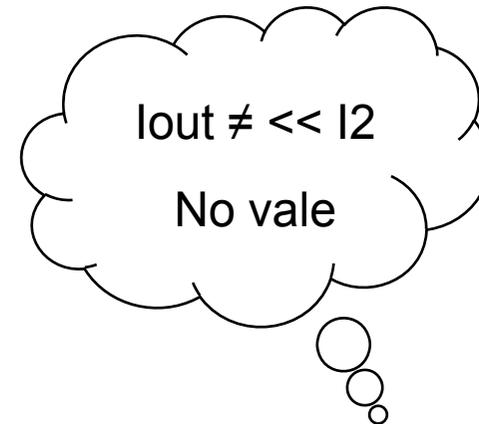
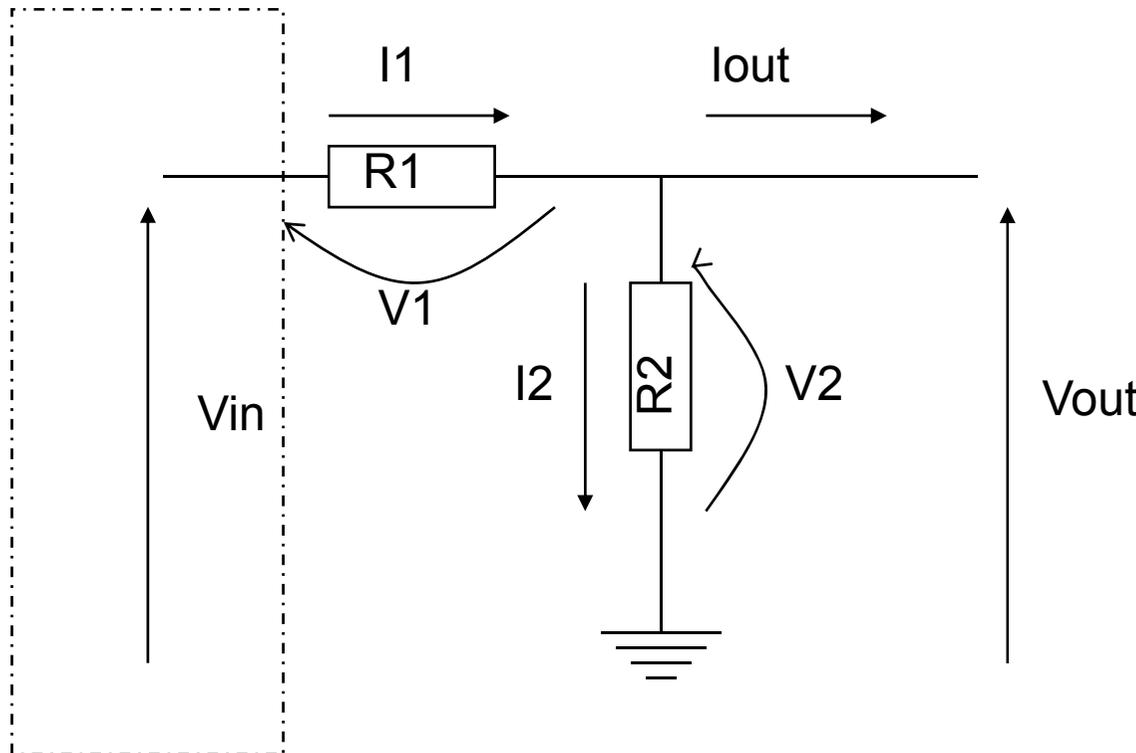
Abierto -> $A = 0$ y $V = 6\text{vdc}$

Operación normal -> $V = 6.5\text{ vdc}$ y $A \leq 1\text{A}$

$$V_{in} = V_1 + V_2$$

$$I_1 = I_2 + I_{out}$$

$$V = I \cdot R$$

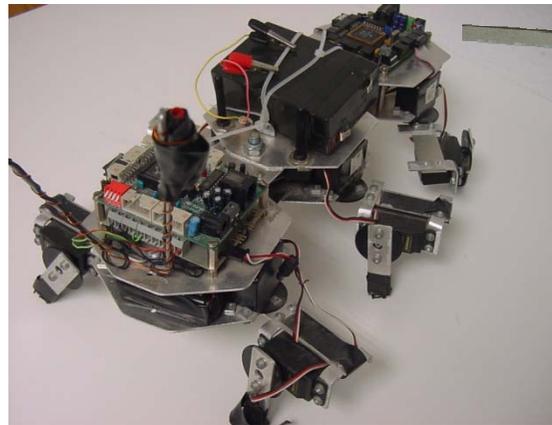
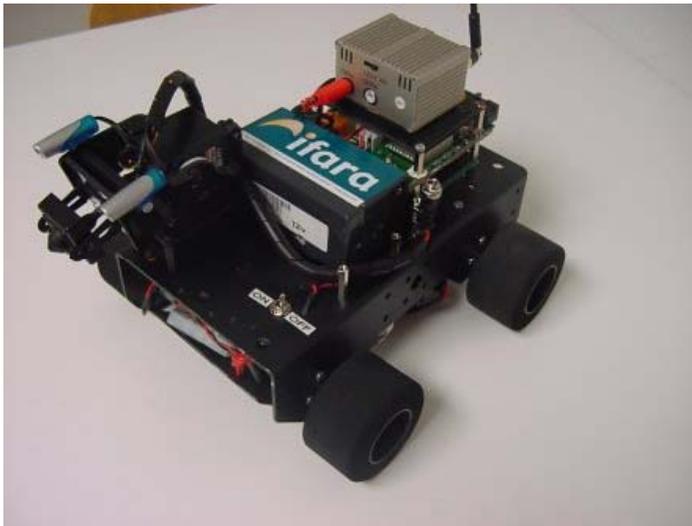
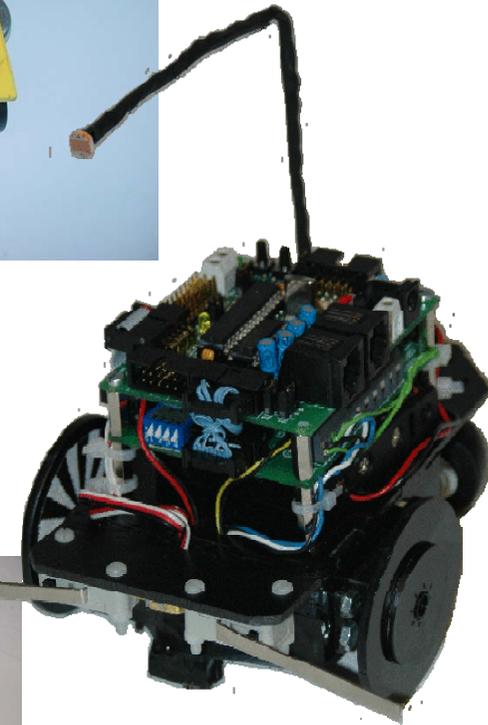
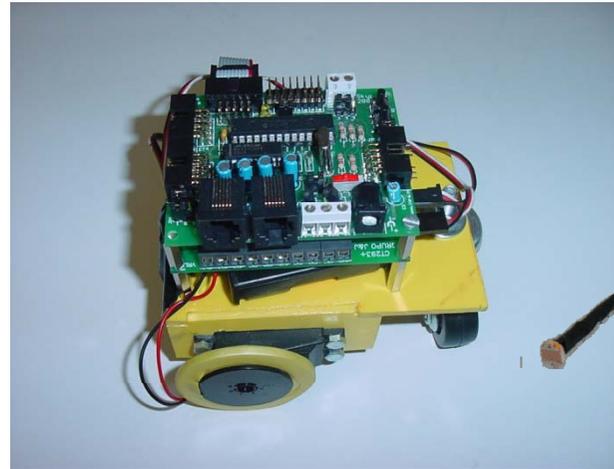


$$I_1 = I_2 + I_{out} = I_2 + 0 \approx I_2 = I$$

$$V_{out} = I \cdot R_2$$

$$V_{in} = (R_1 + R_2) \cdot I$$

$$V_{out} = V_{in} \cdot R_2 / (R_1 + R_2)$$



Trucaje Futaba:

<http://www.iearobotics.com/proyectos/cuadernos/ct2/ct2.html>

Conexión CNY70 y Motor a la GPFAZ

<http://arantxa.ii.uam.es/~gdrivera/robotica/curso0607/p1/p1-doc.html>

Construcción cable serie

<http://arantxa.ii.uam.es/~gdrivera/robotica/curso0506/p2/p2-doc.html>

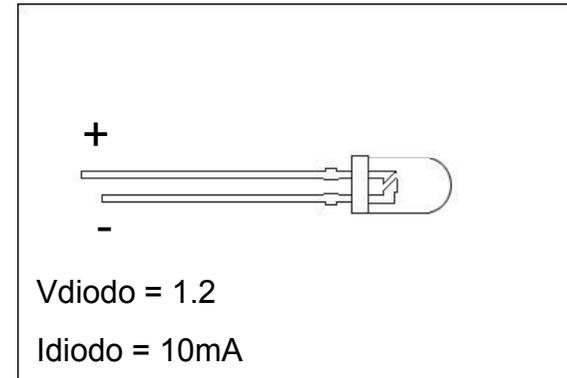
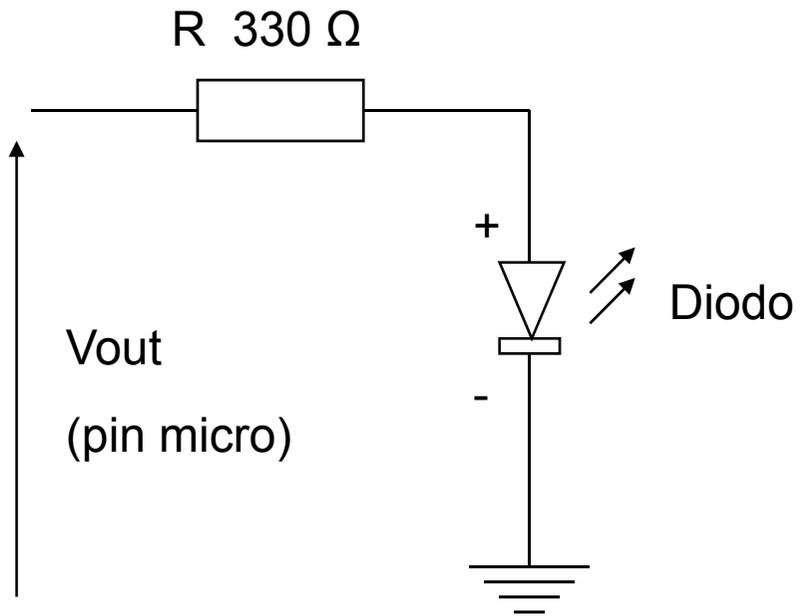
Ejemplos de Programación

<http://www.iearobotics.com/personal/juan/proyectos/gpbot/gpbot.html>



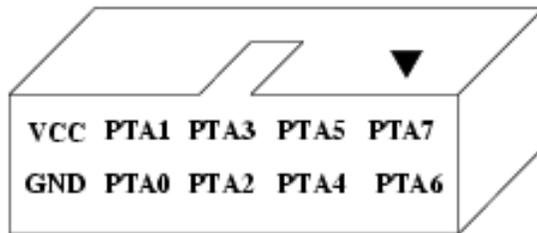
1. Presentación
2. Normas del laboratorio
3. Prácticas
4. GPBOT: Kit Básico de Robótica

5. Programación básica (Motores y Sensores)
6. COP y Frecuencia Interna
7. Programación avanzada (serie y timer)

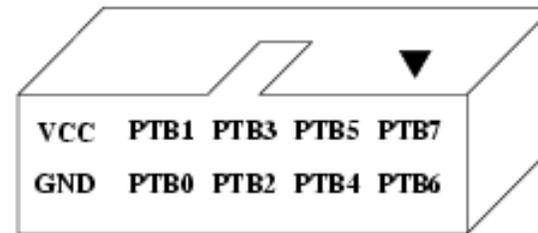


$$R = \frac{V_{out} - V_{diodo}}{I_{diodo}} = 330 \Omega$$

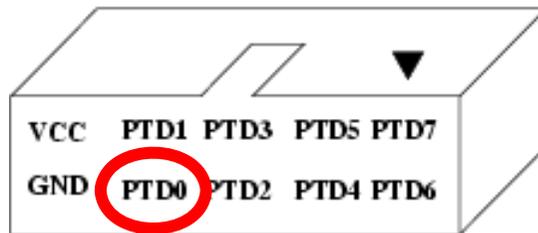
PUERTO A



PUERTO B

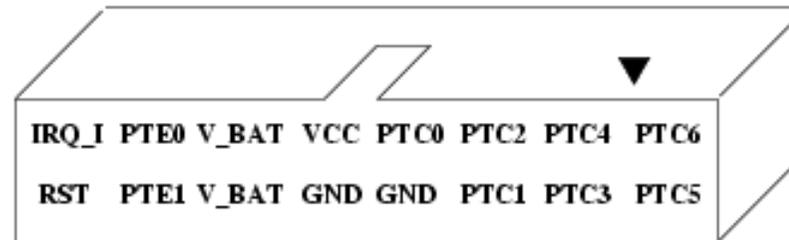


PUERTO D



LED

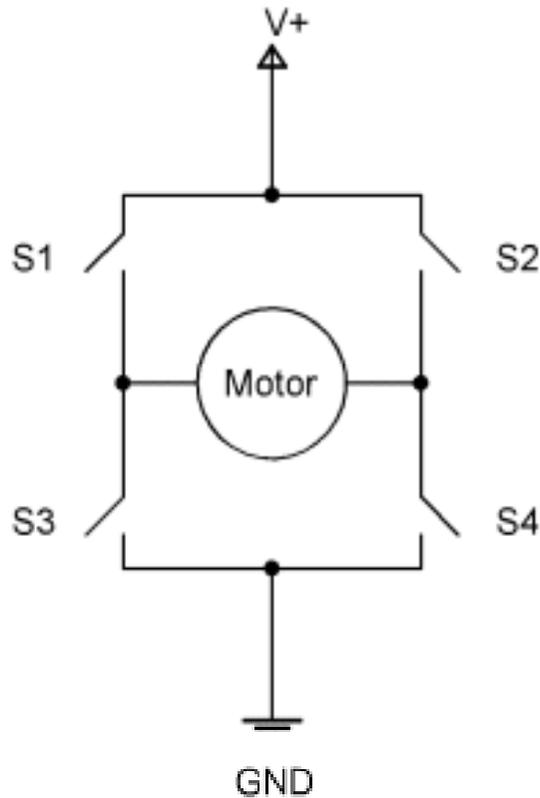
PUERTO C y E



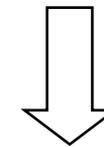
```
/******  
/* portb-sal.c (c) Juan Gonzalez. Marzo 2004 */  
/*-----*/  
/* Ejemplo para el PUERTO B. */  
/* Se configura para salida y se envia un dato */  
/*-----*/  
/* LICENCIA GPL */  
/******  
  
#include <mc68hc908gp32.h>  
  
void main(void) {  
    DDRD = 0xFF; //-- Configurar Puerto B para salida  
    PORTD = 0x01; //-- Enviar un dato  
}
```

Software de prueba `portb_sal.c`

Puente en H: Controlador 1 motor

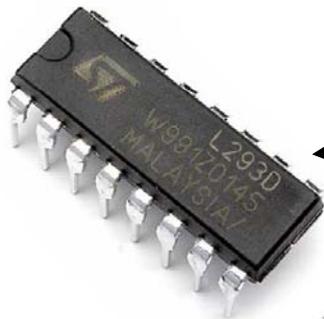


S1	S3	S2	S4	
On	Off	Off	On	Izq
Off	On	On	Off	Der
On	On	X	X	Error
X	X	On	Off	Error
On	Off	On	Off	Stop



Simplificando

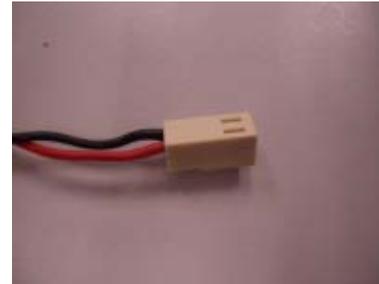
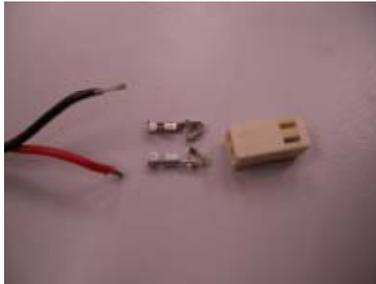
A	B	
on	Off	Izq
off	on	Der
Off	Off	Stop



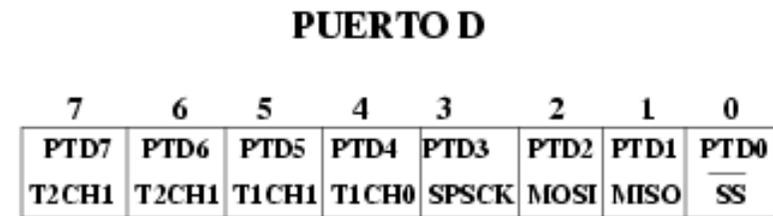
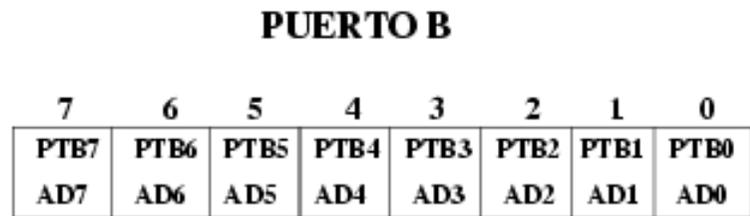
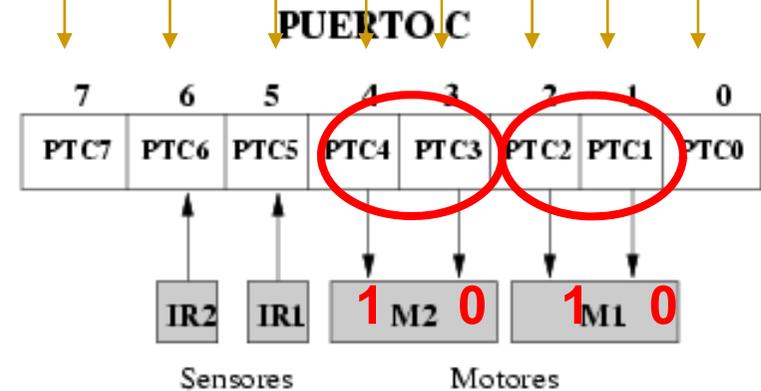
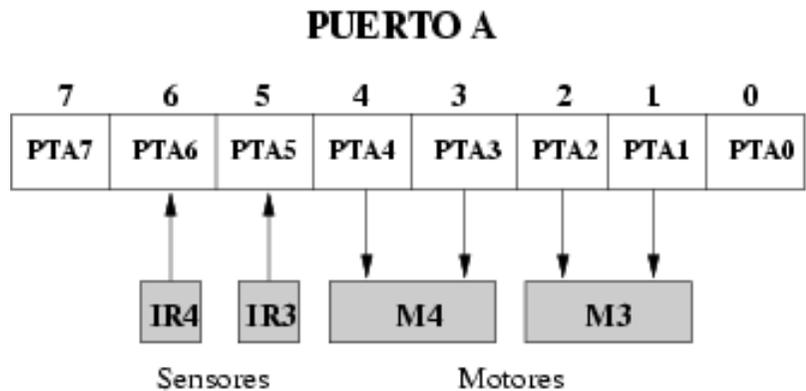
x2

L293D ofrece 2 P-H para dos motores

<http://www.learobotics.com/proyectos/cuadernos/ct2/ct2.html>



DDRC = 0 0 0 1 - 1 1 1 0 = 0x1E



```
#include <mc68hc908gp32.h>
```

```
void main(void){
```

```
    //-- Configurar pines PTC1, PTC2, PTC3 y PTC4 para salida  
    DDRC=0x1E;
```

```
    //-- Activar el motor 1 y 2  
    PORTC=0x14;
```

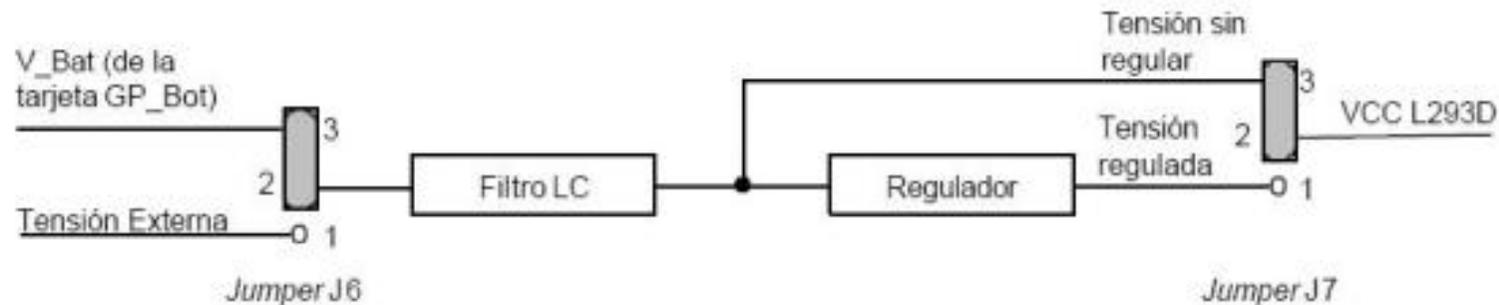
```
    //-- Bucle infinito  
    for (;;);
```

```
}
```

Software de prueba `motor_on.c`

La tarjeta GP_IFAZ permite alimentar los motores con una tensión diferente a la utilizada para el resto de circuitos (+5v DC)

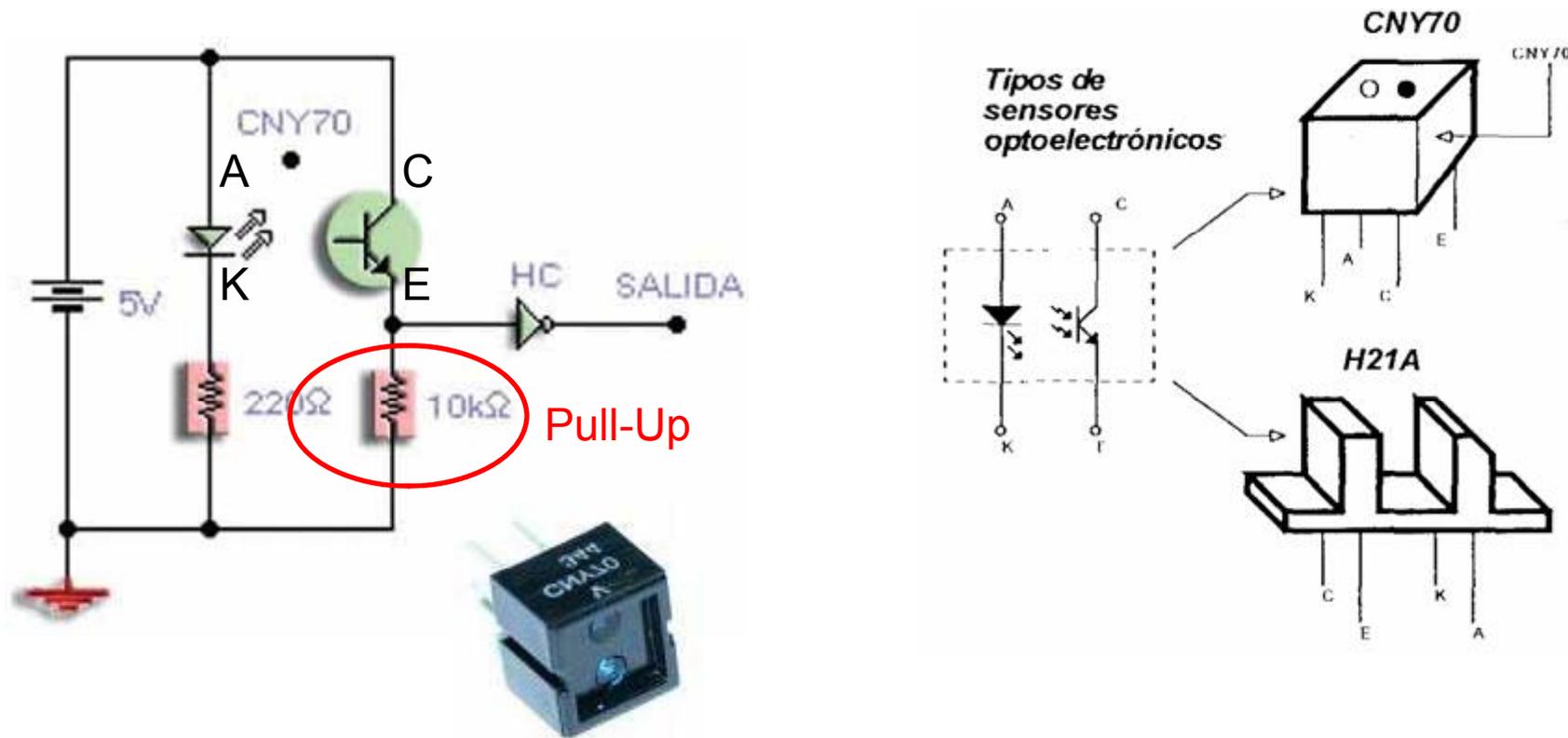
Se usará la Clema Doble J4 con una configuración especial de J6 y J7



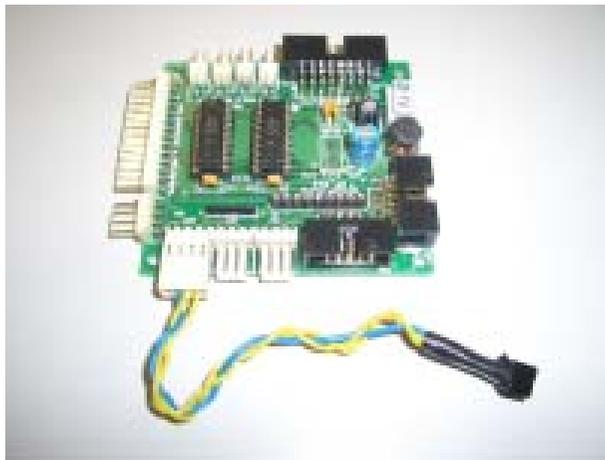
JUMPER J6 tiene que colocarse en la posición 1-2

JUMPER J7 tiene que colocarse en la posición 2-3

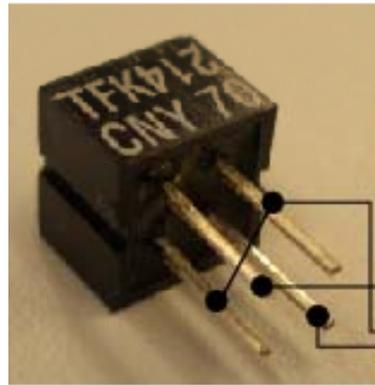
Ejemplo de lectura de "Pin de entrada" usando un CNY70



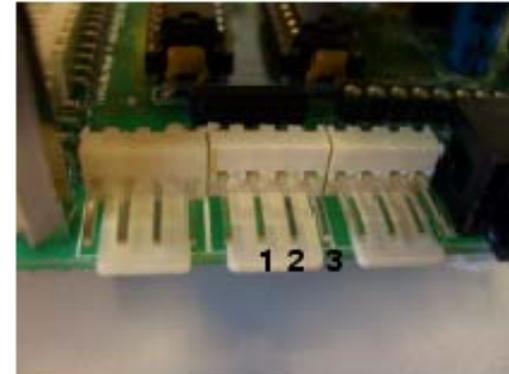
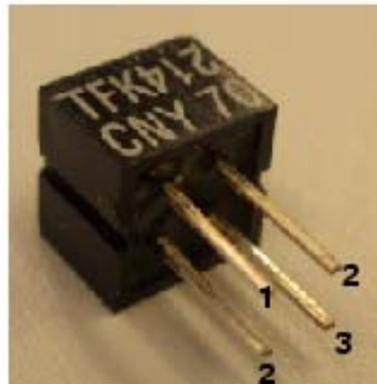
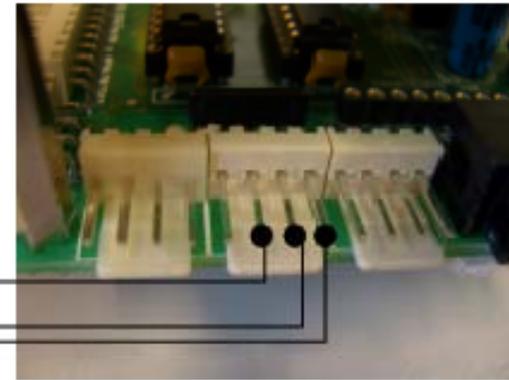
<http://arantxa.ii.uam.es/~gdrivera/robotica/curso0607/p1/p1-doc.html>

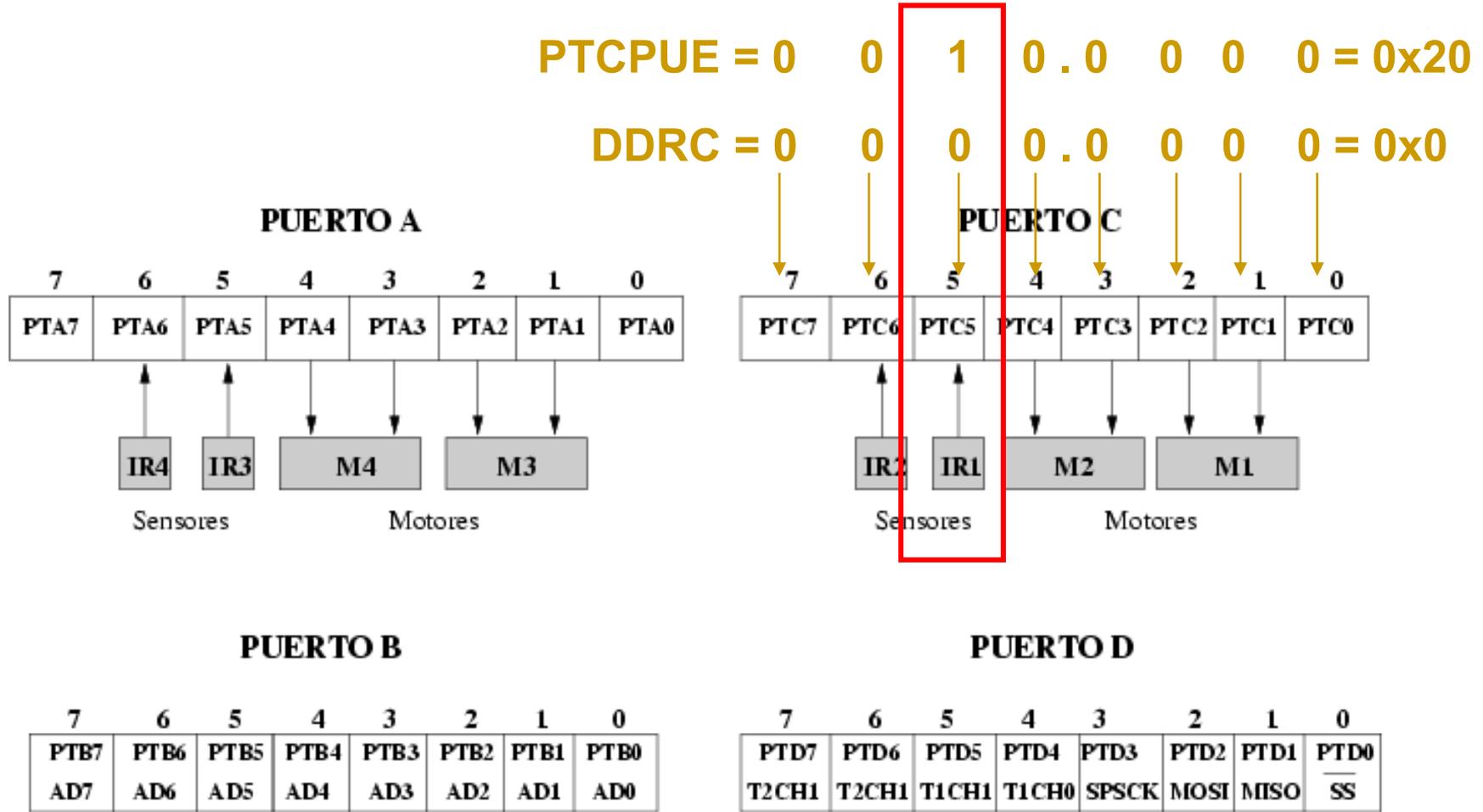


Sensor CNY70



GP_IFAZ





```
/* Sensor IR1: Pin PTC5          */
/* Valores devueltos por el sensor: */
/* Negro --> 1                  */
/* Blanco --> 0                 */
#include <mc68hc908gp32.h>
```

Software de prueba **sensor1.c**

```
unsigned char sensor;
```

```
void main(void){
    CONFIG1|=0x01; //-- Deshabilitar el COP
    //-- Configurar puerto B para salida
    DDRB=0xFF;
    //-- Configuración para utilizar el sensor en modo digital
    //-- Configurar puerto C para entrada
    DDRC=0x00;
    //-- Configurar pull-up
    PTCPU=0x20;
    for(;;) {
        //-- Leer sensor IR1
        sensor=(PORTC & 0x20);
        //-- Enviar el valor al puerto B para visualizarlo
        PORTB=sensor;
    }
}
```



1. Presentación
2. Normas del laboratorio
3. GPBOT: Kit Básico de Robótica
4. Prácticas

5. Programación básica (Motores y Sensores)
6. COP y Frecuencia Interna
7. Programación avanzada (serie y timer)

COP: Computer Operating Properly

Sistema de seguridad basado en un contador que provoca un Reset Interno del Microcontrolador cuando se produzca un OVERFLOW en la cuenta.

En programas con esperas activas largas puede provocar que la aplicación se resetee sola !!!! ya que se permite que el contador haga OVERFLOW.

Para desactivarlo hay que poner el CPOD a 1 en el Config Register.

```
CONFIG1|=0x01;
```

O a cada cierto tiempo escribir en el registro **COPCTL**.

LVI: Low Voltage Inhibit

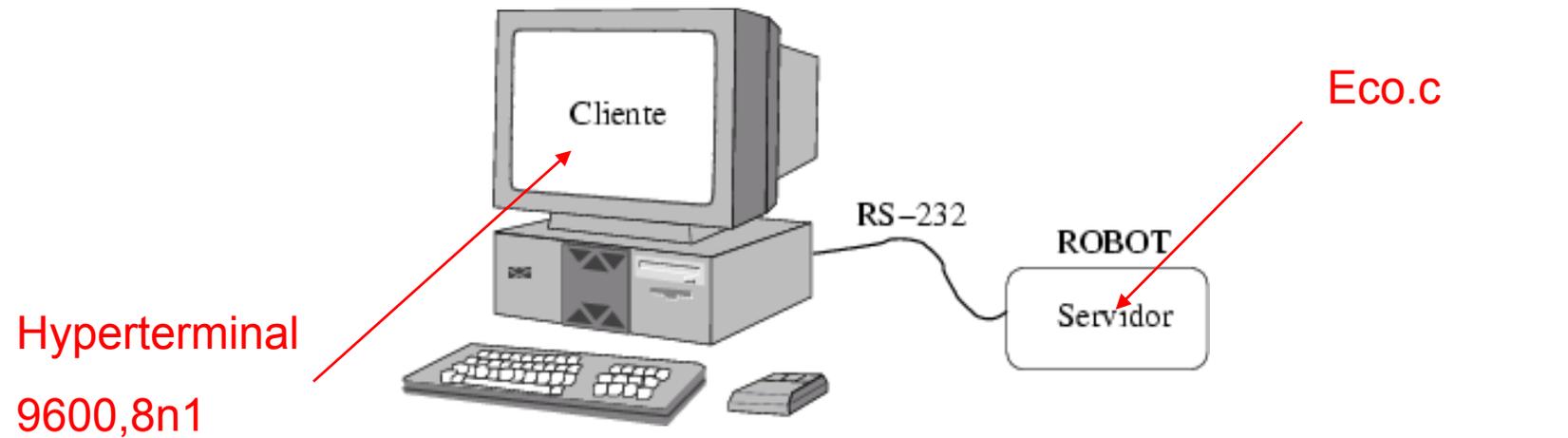
Sistema de seguridad que hace un reset del micro cuando la tensión de alimentación es menor de 3v DC.

Al invertir el sentido de giro de un motor se produce un pico de corriente que puede provocar una caída de tensión por debajo de 3v. !!!!

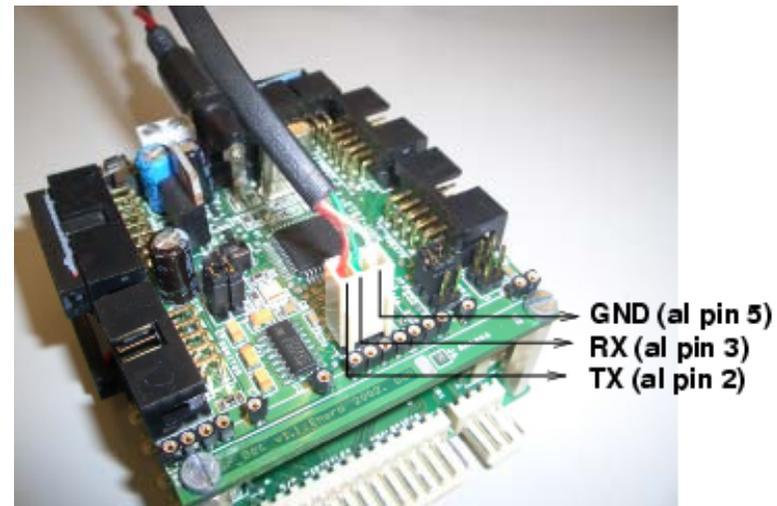
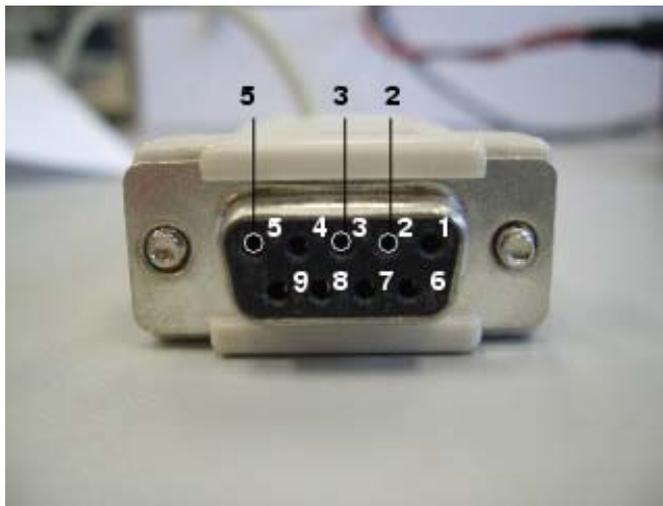
Para desactivarlo hay que poner el LVIPWRD a 1 en el Config Register.

CONFIG1|=0x10;

O utilizar fuentes de alimentación capaces de absorber los picos de corriente. Por ejemplo las pilas y las baterías lo hacen.



<http://arantxa.ii.uam.es/~gdrivera/robotica/curso0506/p2/p2-doc.html>



Serial Communications Interface Module (SCI)

Addr.	Register Name	Bit 7	6	5	4	3	2	1	Bit 0	
\$0013	SCI Control Register 1 (SCC1)	Read:	LODPS	ENSCI	TXINV	M	WAKE	ILTY	PEN	PTY
		Write:								
		Reset:	0	0	0	0	0	0	0	0
\$0014	SCI Control Register 2 (SCC2)	Read:	SCTIE	TCIE	SCRIE	ILIE	TE	RE	RWU	SBK
		Write:								
		Reset:	0	0	0	0	0	0	0	0
\$0015	SCI Control Register 3 (SCC3)	Read:	R8	T8	DMARE	DMATE	ORIE	NEIE	FEIE	PEIE
		Write:								
		Reset:	U	U	0	0	0	0	0	0
\$0016	SCI Status Register 1 (SCS1)	Read:	SCTE	TC	SCRF	IDLE	OR	NF	FE	PE
		Write:								
		Reset:	1	1	0	0	0	0	0	0
\$0017	SCI Status Register 2 (SCS2)	Read:							BKF	RPF
		Write:								
		Reset:	0	0	0	0	0	0	0	0
\$0018	SCI Data Register (SCDR)	Read:	R7	R6	R5	R4	R3	R2	R1	R0
		Write:	T7	T6	T5	T4	T3	T2	T1	T0
		Reset:	Unaffected by reset							
\$0019	SCI Baud Rate Register (SCBR)	Read:			SCP1	SCP0	R	SCR2	SCR1	SCR0
		Write:								
		Reset:	0	0	0	0	0	0	0	0

Figure 18-2. SCI I/O Register Summary

- ENSCI** Activar el Puerto Serie. Bit6 del SCC1 puesto a 1
- TE** Activa el Trasmisor del puerto serie. Bit3 del SCC2 puesto a 1
- RE** Activa el Receptor del puerto serie. Bit2 del SCC2 puesto a 1
- SCTE** Se activa cuando esta libre el registro de envío. Bit7 del SCS1
- SCRF** Se activa cada vez que llega un carácter. Bit 5 del SCS1
- SCDR** Registro para enviar/leer un dato por el puerto serie
- SCBR** Configuración del puerto serie (velocidad en baudios)

```
#include <mc68hc908gp32.h>

void sci_init(void) {
    SCBR = 0x22;  //-- 9600 Baudios
    SCC1 = 0x40;  //-- Habilitar SCI
    SCC2 = 0x0C;  //-- Habilitar Transmisor y receptor
}

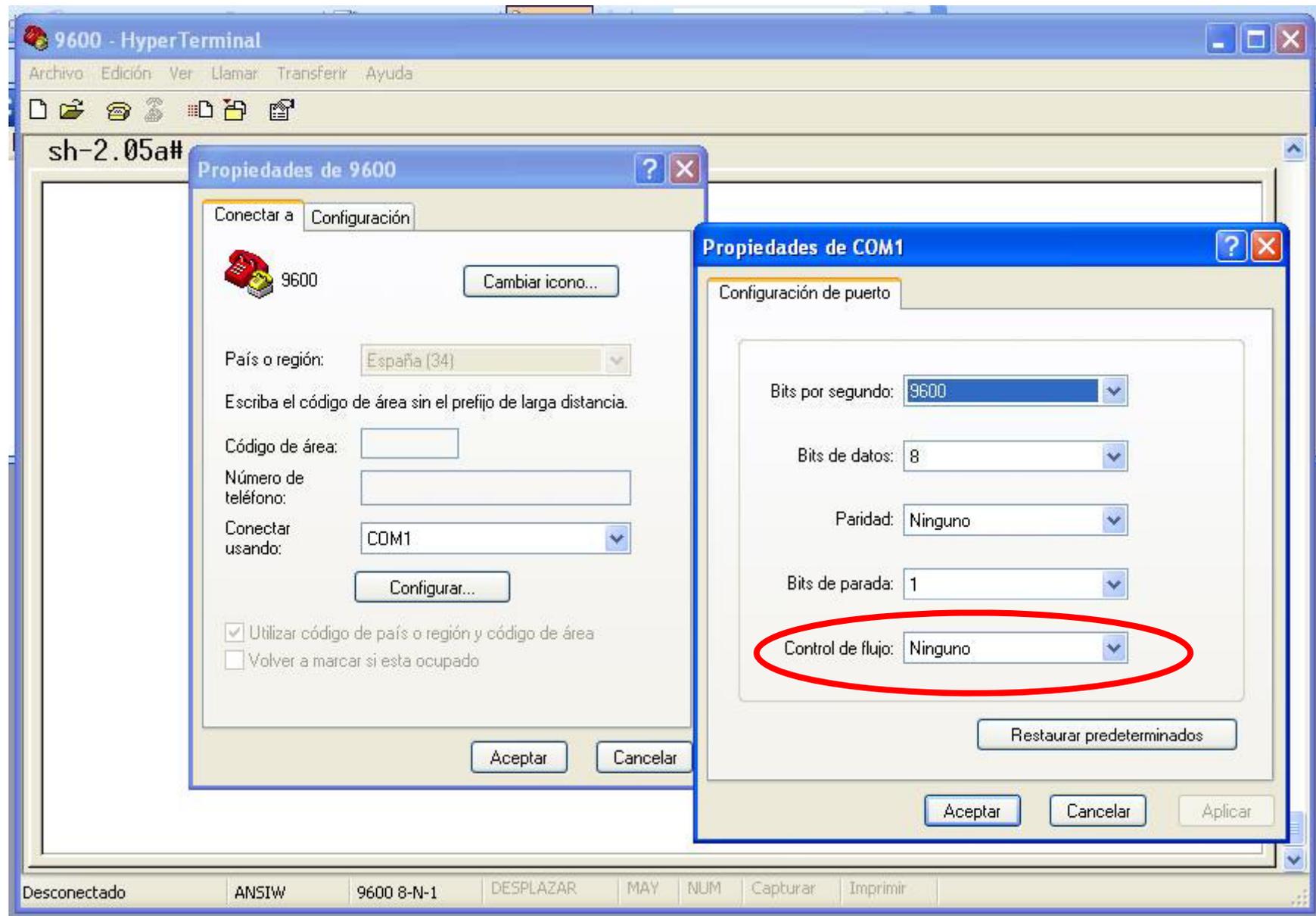
unsigned char sci_leer_car(void){
    //-- Esperar a que se ponga a '1' el Flag (Bit 5)
    while(!(SCS1 & 0x20));
    return SCDR;
}

void sci_enviar_car(const char c){
    /* Esperar hasta que se pueda enviar algun caracter */
    /* Se tiene que activar el bit SCTE del registro SCS1 */
    while(!(SCS1 & 0x80));
    /* Enviar el caracter */
    SCDR = c;
}
```

```
void main(void){
    unsigned char car;

    CONFIG1|=0x01; //-- Deshabilitar el COP
    // Configuramos el puerto serie
    sci_init();

    for (;;) {
        // Esperamos a recibir un caracter
        car=sci_leer_car();
        // Lo enviamos por el puerto serie
        sci_enviar_car(car);
    }
}
```



FRECUENCIA DE RELOJ

Para usar el TIMER correctamente es necesario conocer la frecuencia de funcionamiento interna del microcontrolador.

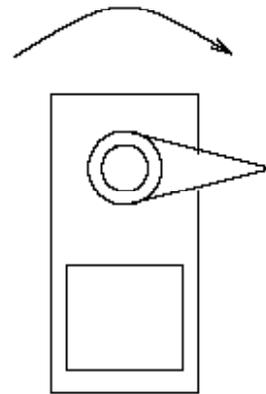
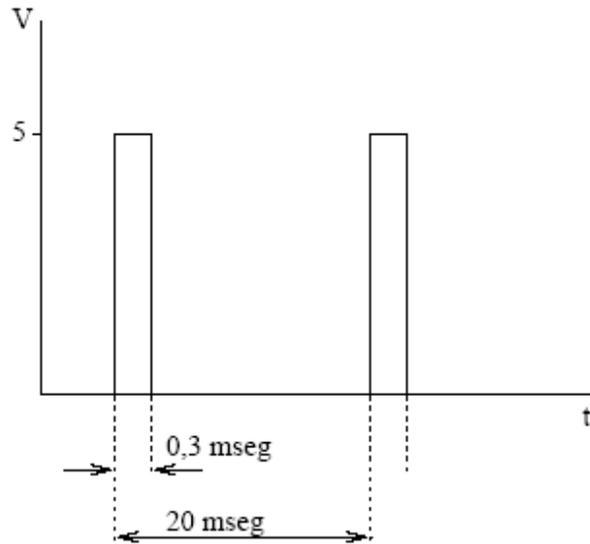
CMGMX_CLK : Señal equivalente al reloj externo = 9.8304 Mhz

CGMV_CLK: Señal generada en el PLL (**No se usa** el PLL)

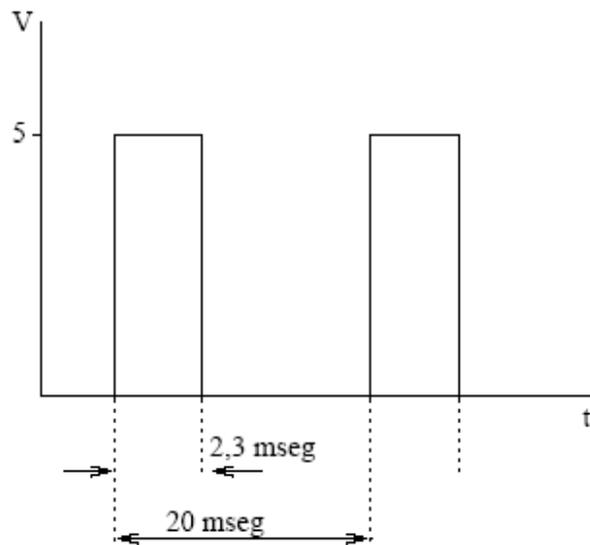
CGMOUT: Señal de reloj Base = CMGMX_CLK / 2

Reloj interno (Fbus) = CGMOUT / 2 = CMGMX_CLK / 4

Reloj Externo 9.8304 Mhz -> Fbus= 2.45 MHz

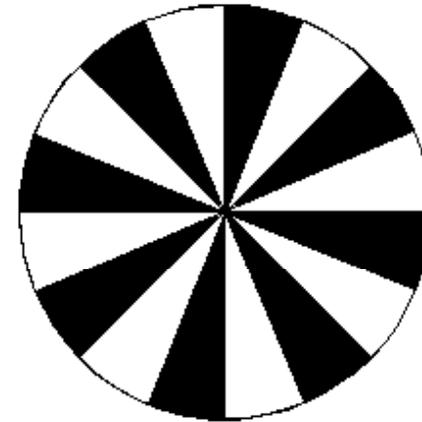


Futaba S3003 Analógico



	Futaba 3003
Velocidad	0.23 seg/60°
Par	3.2 Kg-cm
Tamaño	40 x 20 x 36 mm
Peso	37 g

Software por interrupciones

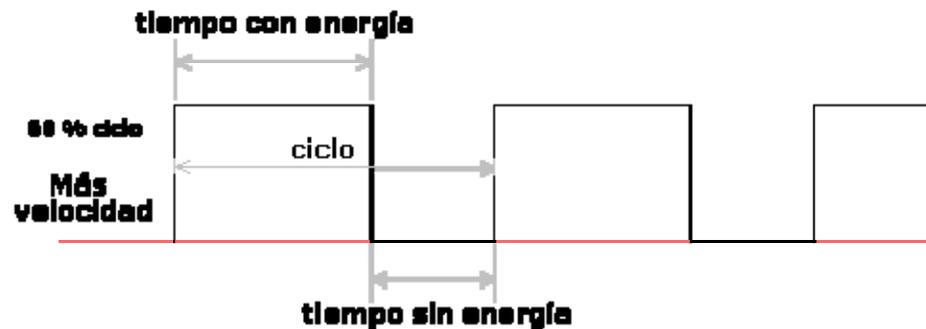


PWM = Modulación por ancho de pulso

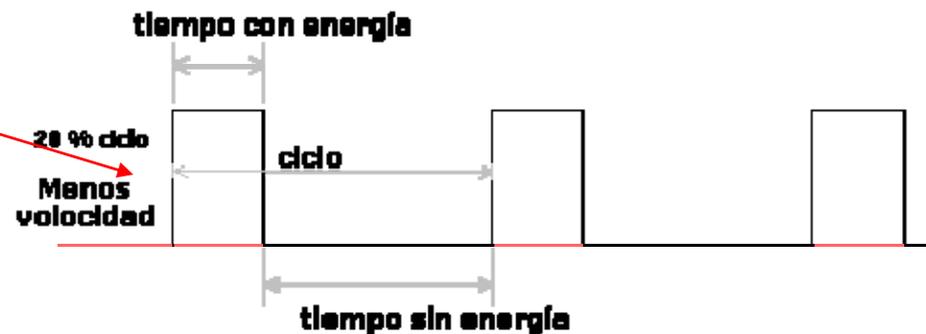
Nos va a permitir controlar la velocidad de un motor

V → Velocidad

I → Potencia



El motor responde al valor medio de la señal aplicada.



<http://homepages.which.net/~paul.hills/SpeedControl/SpeedControllersBody.html>

Algoritmo por espera activa.

Algoritmo por interrupciones

Algoritmo Capturador de pulsos