

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

Universidad Autónoma de Madrid  
Escuela Politécnica Superior

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)

**Andrés Prieto-Moreno Torres**

[andres@iearobotics.com](mailto:andres@iearobotics.com)

**Tutorías:**

Por mail en cualquier momento

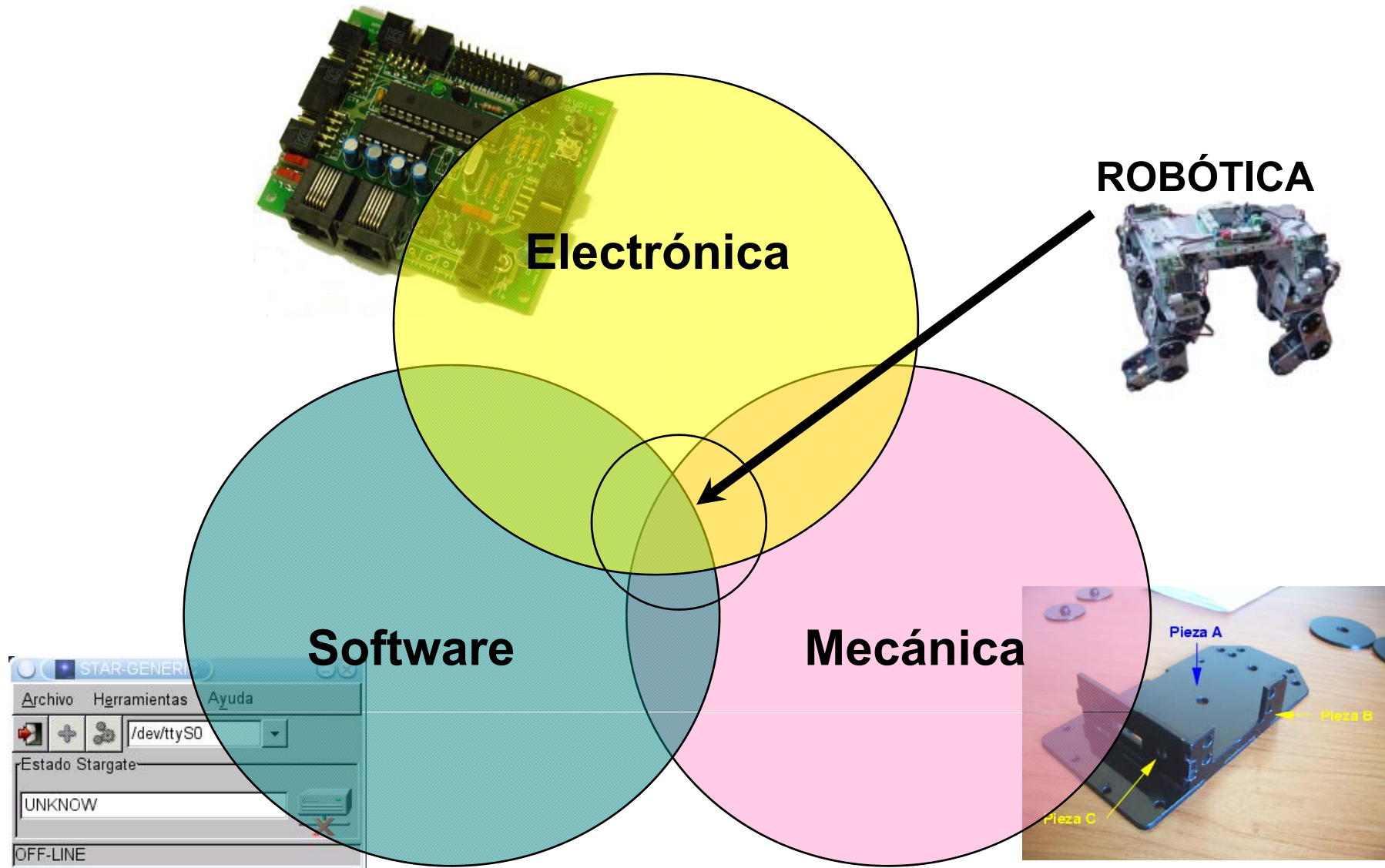
Presenciales:

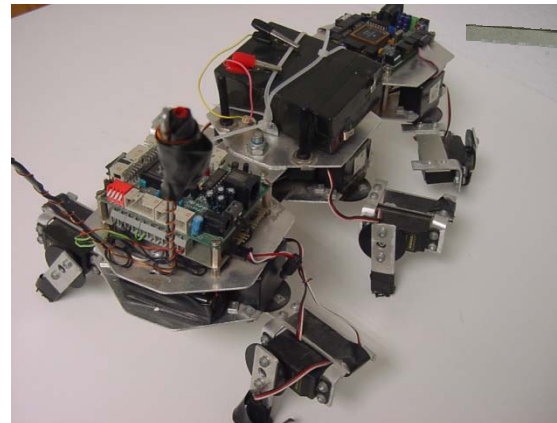
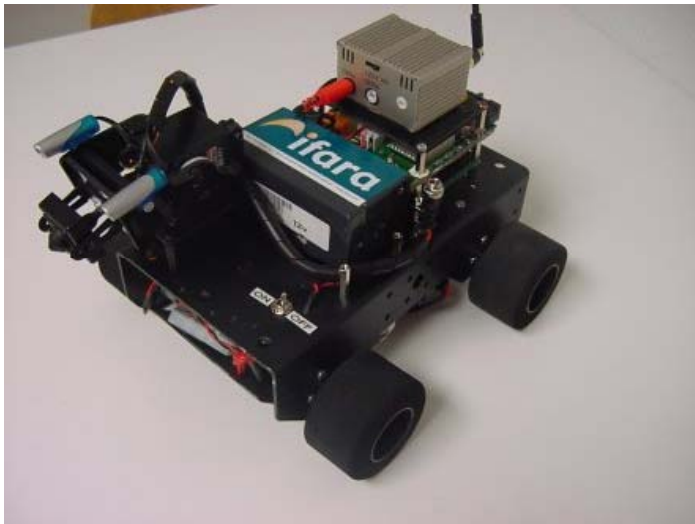
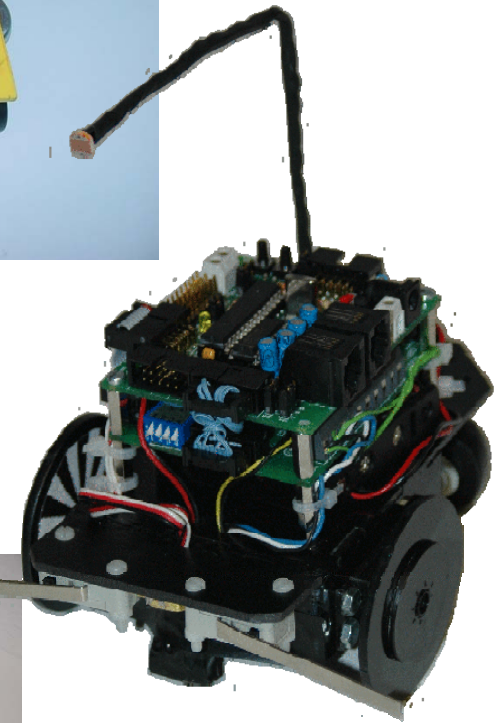
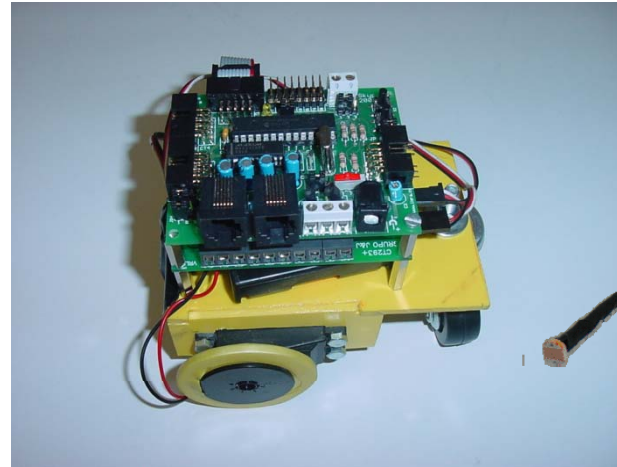
Miércoles (grupo A): 18:00 – 20:00

Viernes (grupo XX): 16:00 – 18:00

**Web asignatura: Guillermo de Rivera (coordinador Laboratorio)**

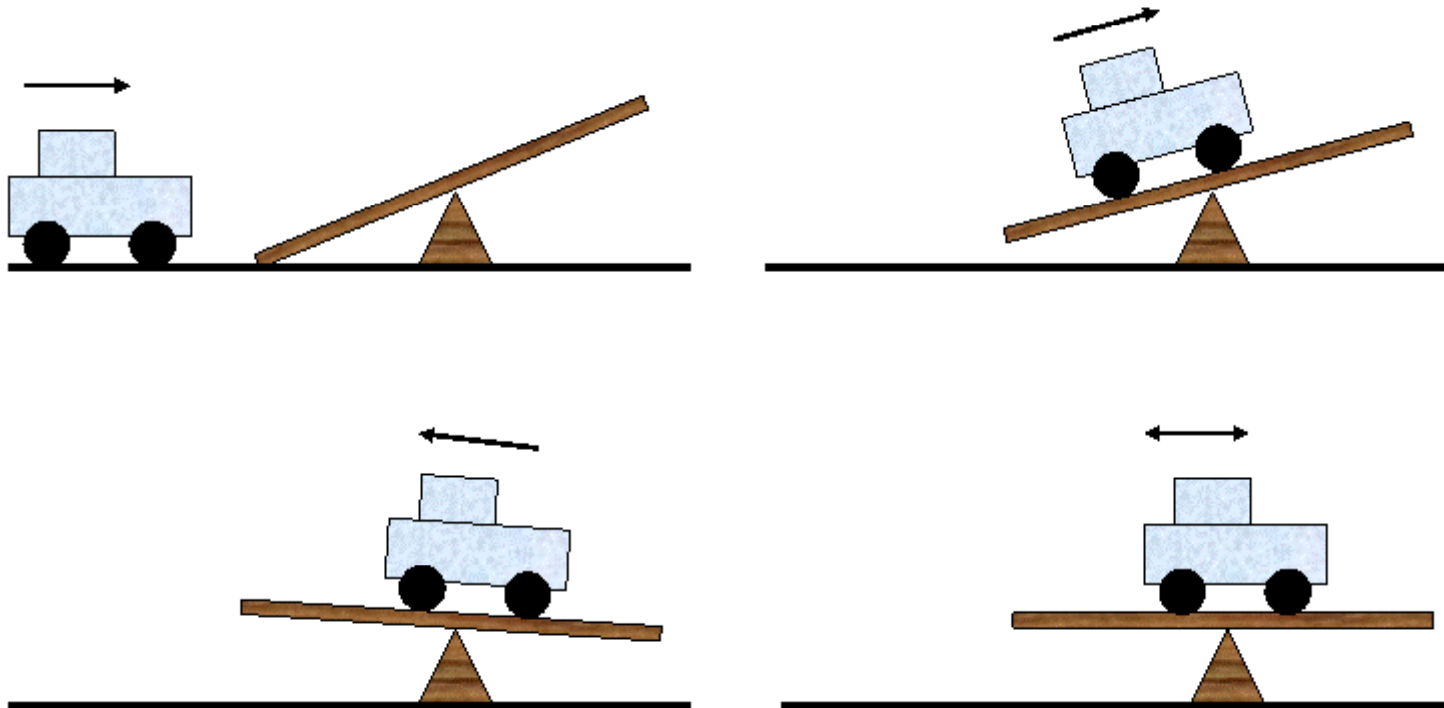
<http://www.ii.uam.es/~gdrivera/robotica/robotica.htm>





- 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 copiador 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)**

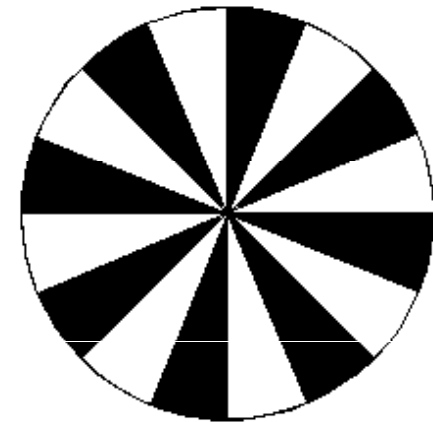
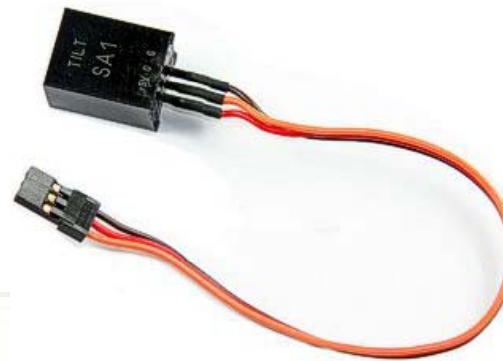
- Construir el robot
- Programar en C un software que realice la estabilización.

**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

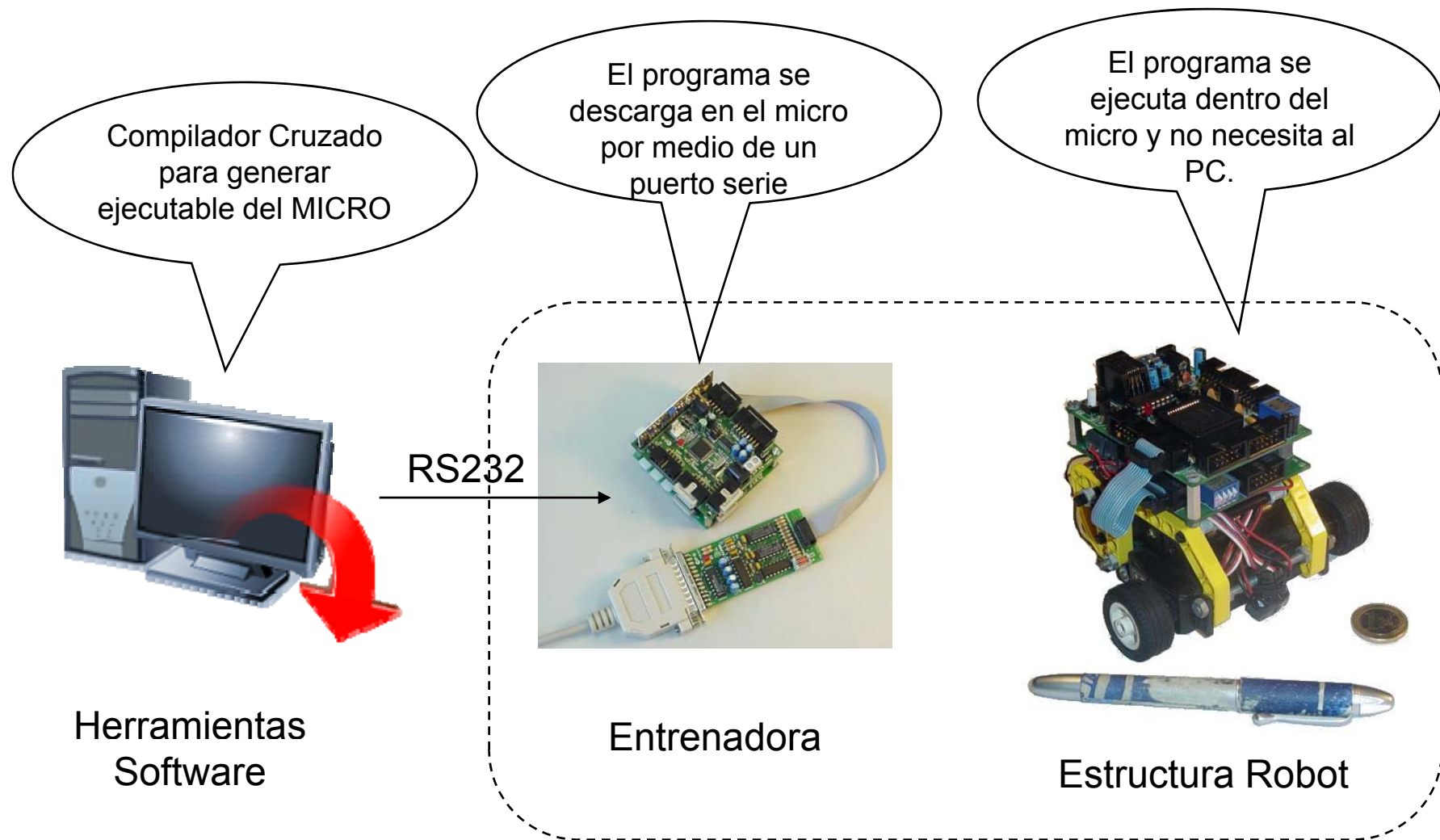
<http://arantxa.ii.uam.es/~gdrivera/robotica/curso0607/p1/p1-doc.html>

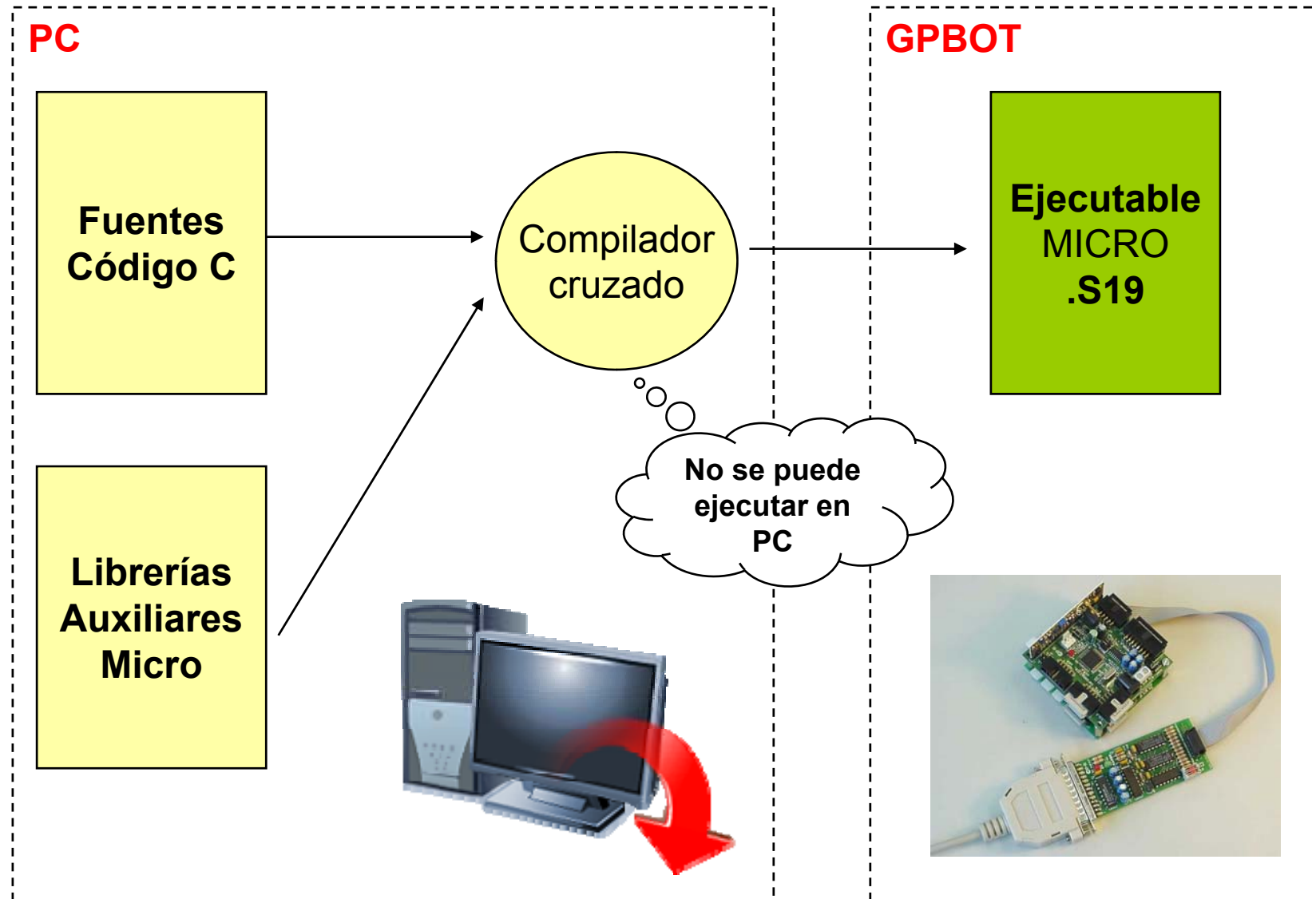
Construcción cable serie

<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)

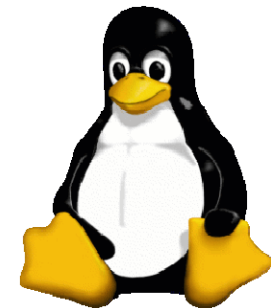
## Linux

Editor Anjuta, Vim, emacs

Compilador **SDCC** (make)

Descarga con **GPDOWN** (learobotics)

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



The image shows a screenshot of the Programmer's Notepad 2 application. The main window displays a C program named 'ledp.c' with the following code:

```

/* ledp.c (c) Juan Gonzalez. Marzo 2004 */
/* Ejemplo que hace parpadear un led conectado */
/* al bit 0 del puerto B */
/* Se cambia el estado del led cada 0.5 segs */
/* LICENCIA GPL */
#include "mc68hc908gp32.h"

/* Inicializacion del temporizador. Se debe */
/* llamar esta funcion antes de hacer cualquier */
/* pausa. */
/* Se configura el TIM1 para active el flag de */
/* overflow cada 100ms */
void delay_init(void)
{
    /*-----*/
    /*- Configurar el temporizador */
    /*-----*/
    T1SC = 0x06; // Prescaler: Div entre 64
    T1MODH = 0x0E; // Establecer modulo 0x0EF7
    T1MODL = 0xF7;
}

/* Rutina de pausa, en unidades de 100ms */
/* ----- */

```

The 'Options' dialog is open, showing the 'Tools' tab with a list of tools:

Name	Command	Params
sdcc	C:\Archivos de Pro...	-mpic14 -p16f8...
sdcc con libreria auxilia...	%d\crear_ejecuta...	%n "%d"
sdcc GPBOT	C:\Archivos de Pro...	-mhc08 --stack-...

The 'Propiedades de Edit Tool' dialog is also open, showing the 'Console I/O' tab. The 'Parameters' field is highlighted with a red circle and contains the text: `-mhc08 --stack-loc 0x023f %f`. A red arrow points from this field to the corresponding line in the code below.

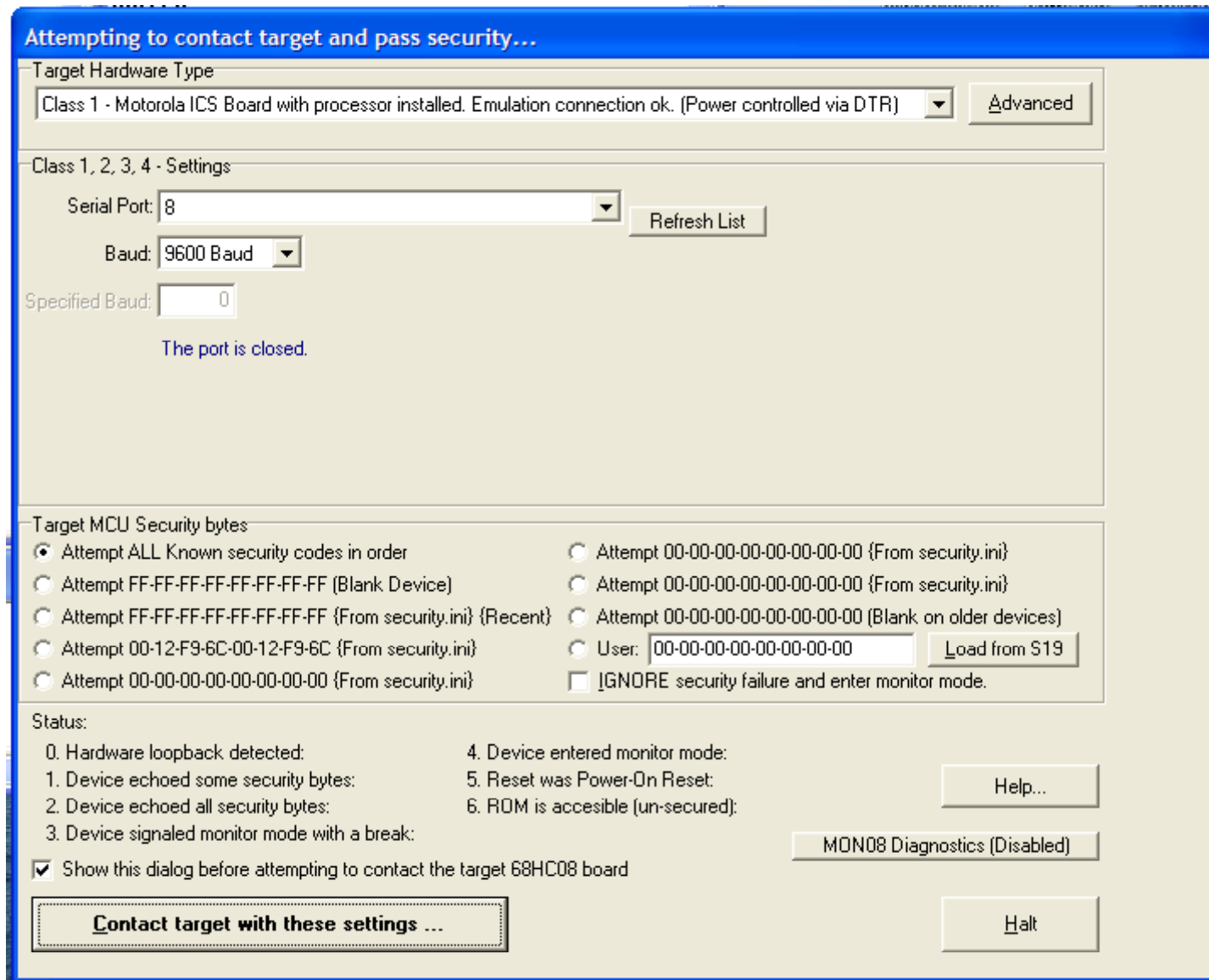
The 'Output' window shows the command executed:

```

> "C:\Archivos de Programa\SDCC\bin\sdcc.exe" -mhc08 --stack-loc 0x023f ledp.c
> Process Exit Code: 0

```

At the bottom of the code editor, the command `-mhc08 --stack-loc 0x023f %f` is highlighted in red, with a red arrow pointing to it from the 'Parameters' field in the dialog.





The screenshot displays the PROG08SZ Programmer interface with several windows open:

- Choose Programming Function:** A list of functions with their status. The 'CM Choose module .08P' option is selected.
- Configuration:** Shows settings for the current module, which is currently set to 'none'.
- Status Window:** Displays a message: 'Contact with HC08 Monitor established. Monitor Code Version:4. Trying Security \$FFFFFFFFFFFFFFFF Success (Security OK) (68HC08 Device responded properly, but ROM not programmed) (Successful Status: 0=Y,1=Y,2=Y,3=Y,4=Y,5=Y)'. The status bar at the bottom shows 'ROM not programmed'.
- Specify Programming Algorithm to Use!:** A file explorer window showing a list of algorithm files in the 'algorithms' folder. The file '908\_GP32\_HighSpeed.08P' is selected.

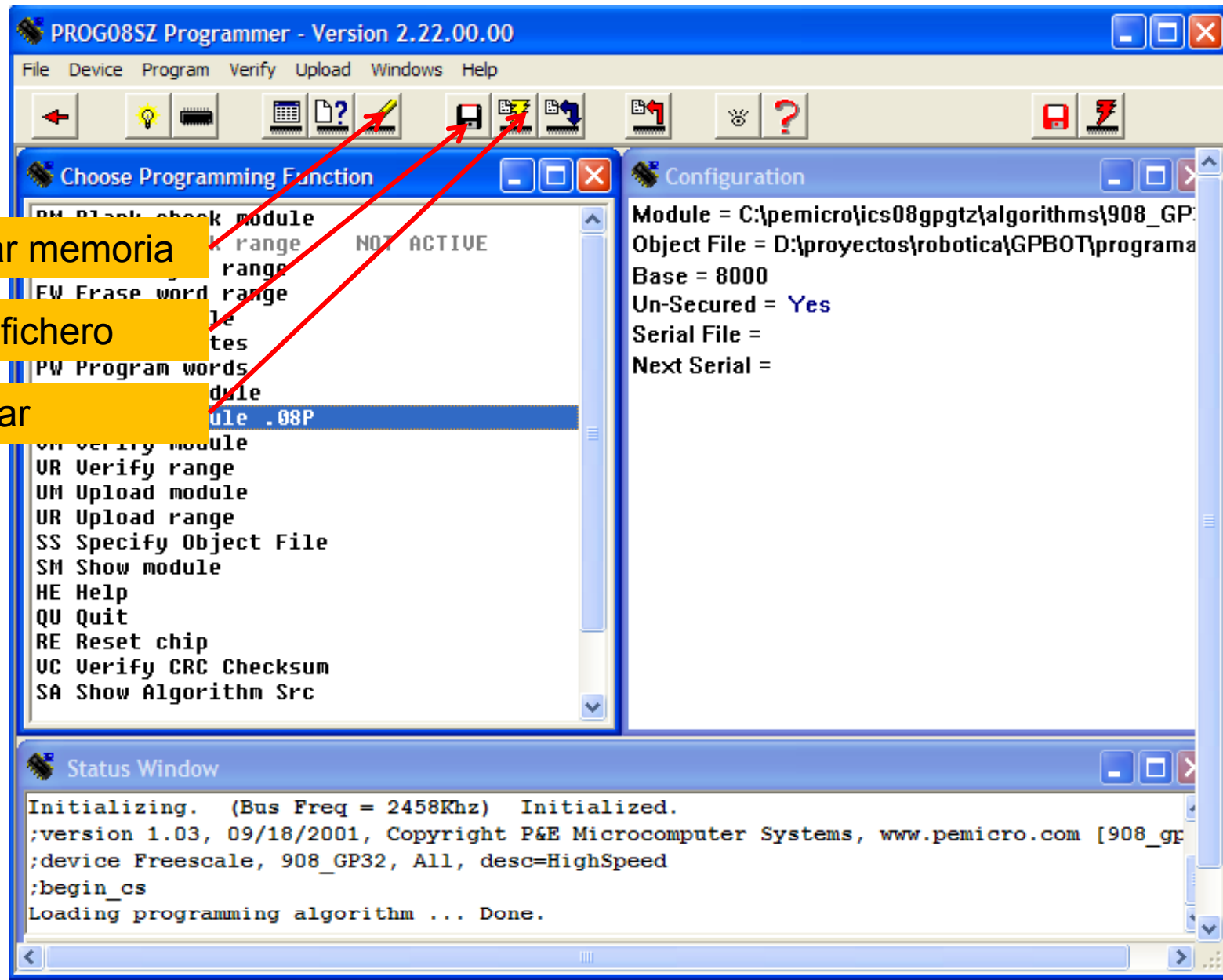
Function	Status
BM Blank check module	NOT ACTIVE
BR Blank check range	NOT ACTIVE
EB Erase byte range	NOT ACTIVE
EW Erase word range	NOT ACTIVE
EM Erase module	NOT ACTIVE
PB Program bytes	NOT ACTIVE
PW Program words	NOT ACTIVE
PM Program module	NOT ACTIVE
<b>CM Choose module .08P</b>	
UM Verify module	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

File Name
908_GP20.08P
908_GP32.08P
<b>908_GP32_HighSpeed.08P</b>
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

1. Borrar memoria

2. Abrir fichero

3. Grabar





The screenshot displays the CodeWarrior IDE interface. On the left, the 'ejemplo1.mcp' project is open, showing a file tree with 'Sources', 'Includes', 'Libs', and 'Project Settings'. A yellow box labeled 'Mon08 Interface' points to the hardware configuration icon in the toolbar. The main window is titled 'True-Time Simulator & Real' and contains several panes:

- Source:** Shows the C code for 'main.c' at line 48:
 

```

      //-- Bucle principal
      for (;;) {
          PTB^=0xff; //-- Cambiar de estado bit0 del puerto B
          delay(5); //-- Pausa de 500ms
      }
      
```
- Assembly:** Shows the assembly code for 'main':
 

```

      80BD BSET 0,0x1F
      80BF MOV #0xFF,0x05
      80C2 CLR 0x01
      80C4 BSR *-33 ;abs = 0x80A3
      80C6 COM 0x01
      80C8 LDA #0x05
      80CA BSR *-29 ;abs = 0x80AD
      
```
- Register:** Shows the register values for HC08:
 

A	5
HX	4D SP 14F
SR	7B Status VHINZC
PC	80C8
- Memory:** Shows memory dump for addresses 0080 to 0098.
- Command:** Shows the command prompt with 'STOPPED STEPPED OVER' and 'in>'.

At the bottom of the IDE, the status bar indicates 'For Help, press F1', 'HC908GP32', and 'ICD STEPPED OVER'.

**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:**

- Añades 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:

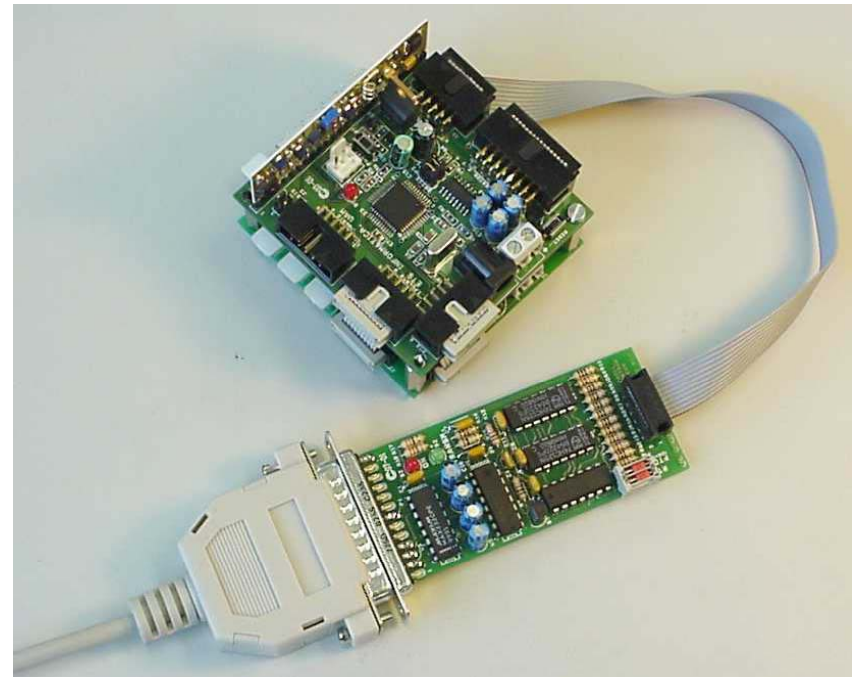
```
#include <mc68hc908gp32.h>  
#define PORTX PTX (donde X puede ser A, B, C, D ...)
```

Micro Motorola MC68HC908GP32

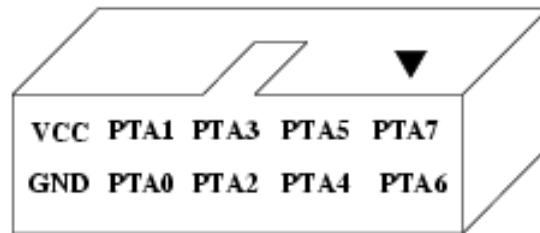
- 32Kb de memoria FLASH
- 512 bytes memoria RAM
- Comunicaciones SPI, SERIE
- 2 temporizadores de 16 bits
- 8 conversores AD
- Pines IO
- Adaptado para programar en C

GPBOT

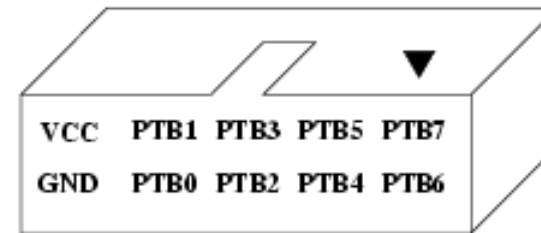
- Recomendable 6.0v a 1A máximo
- Reloj Externo a 9.8 Mhz
- Fbus interna a 2,45 Mhz



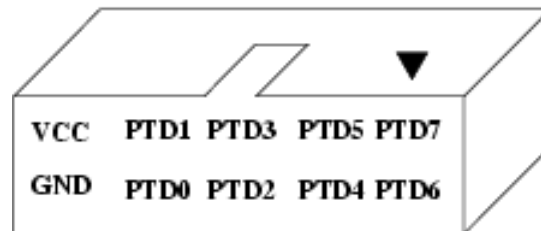
**PUERTO A**



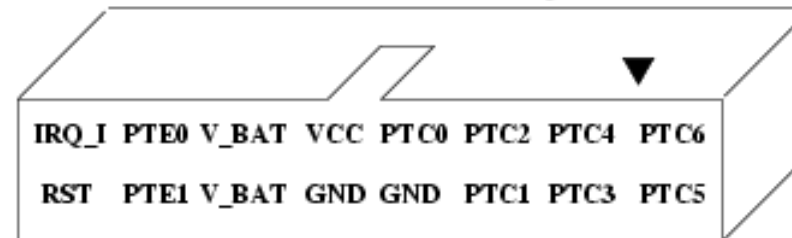
**PUERTO B**



**PUERTO D**

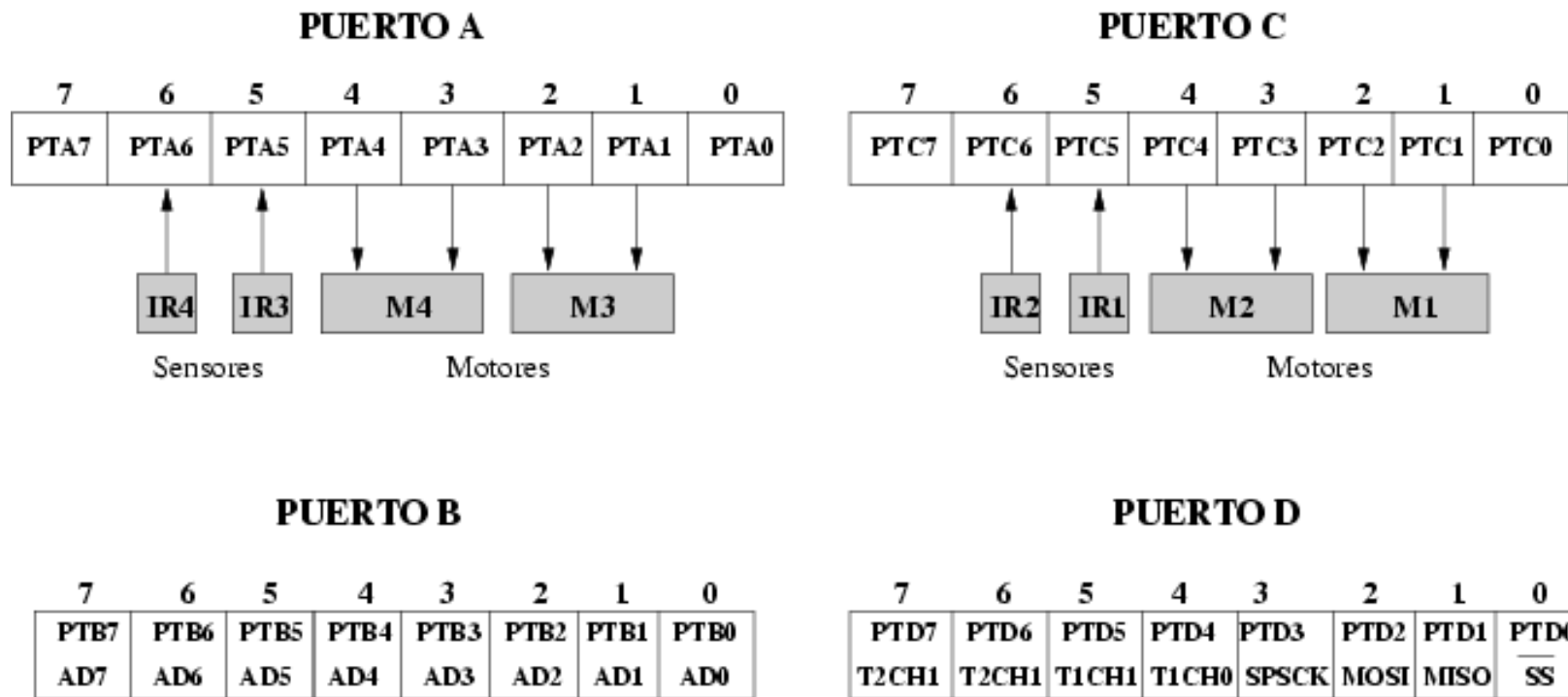


**PUERTO C y E**

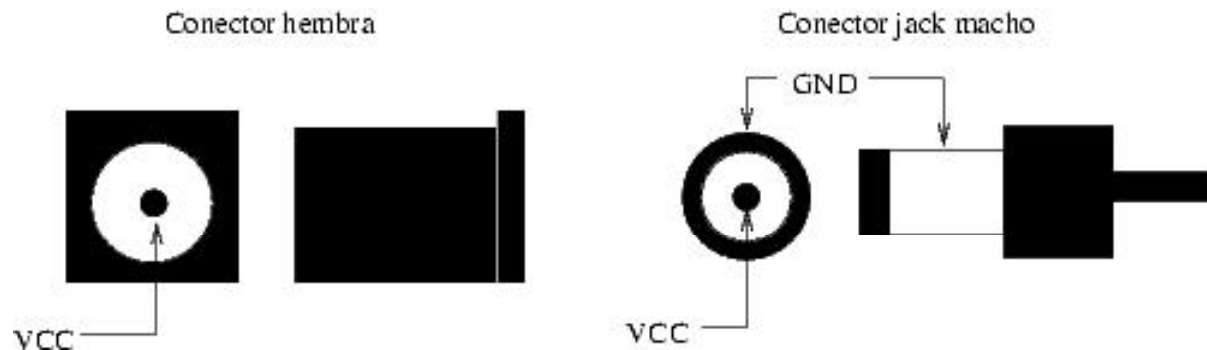


- PTA** Registro de entrada y salida de 8 bits. (PORTA)  
**DDRA** Configura cada bit del Puerto A como entrada (1) o salida (0)  
**PTAPUE** Activa (1) las resistencias de Pull-Up del Puerto A.
- PTB** Registro de entrada y salida de 8 bits. (PORTB)  
**DDRB** Configura cada bit del Puerto B como entrada (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)

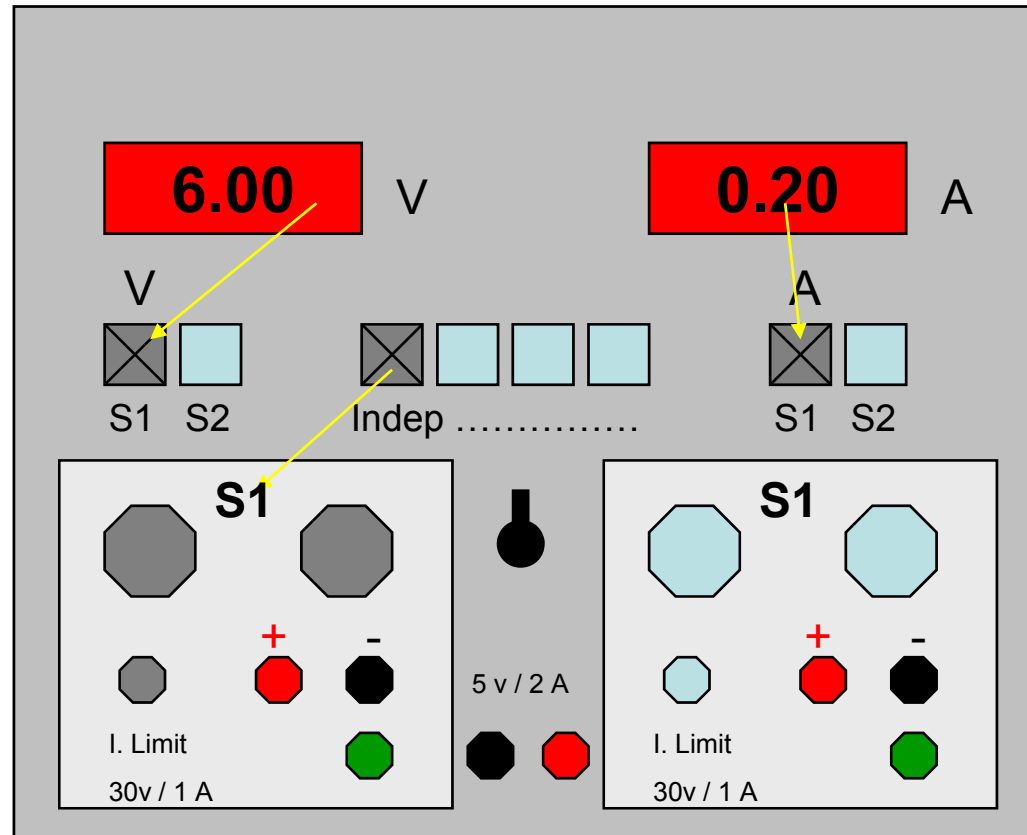




**Antes de enchufar verificar la tensión de la fuente**



Alimentación entre 6 y 9 voltios. Recomendable 7.5 v



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

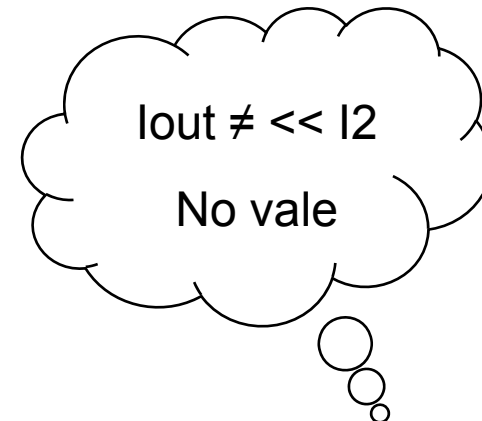
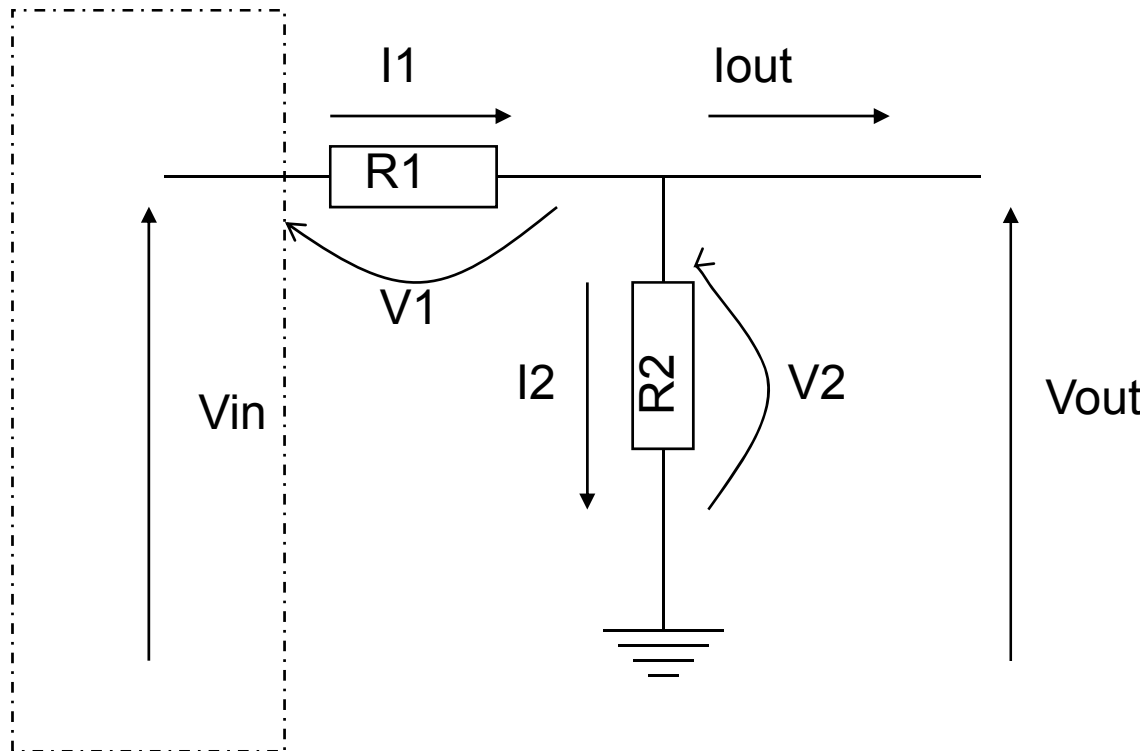
Abierto ->  $A = 0$  y  $V = 6\text{vdc}$

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>

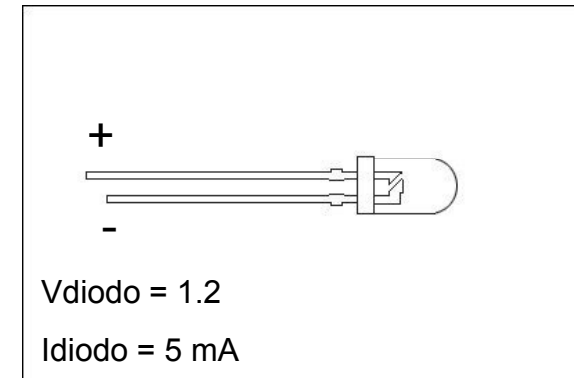
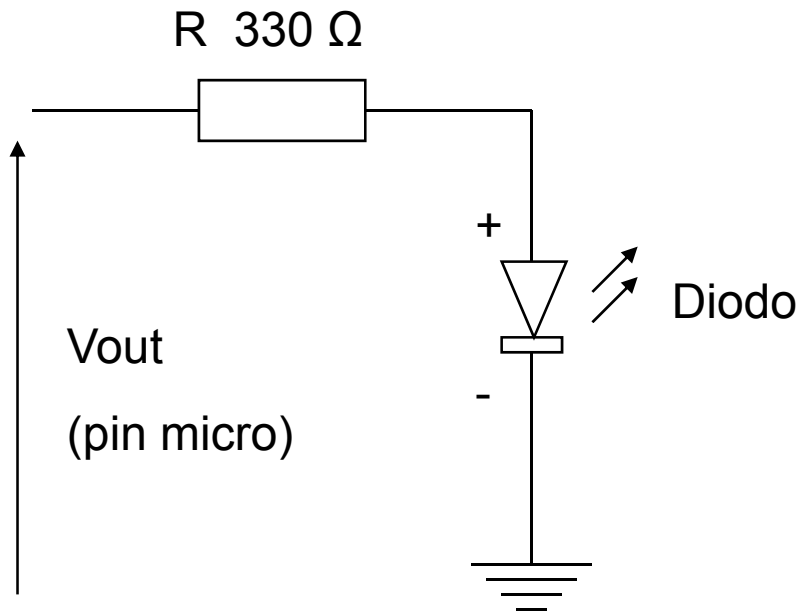
Construcción cable serie

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

Ejemplos de Programación

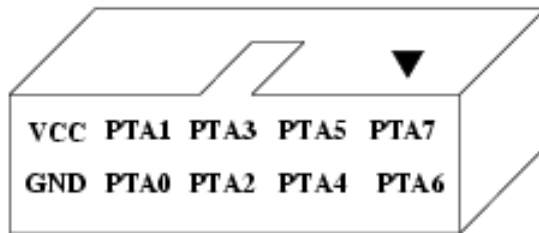
<http://www.iearobotics.com/personal/juan/proyectos/gpbot/gpbot.html>

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)

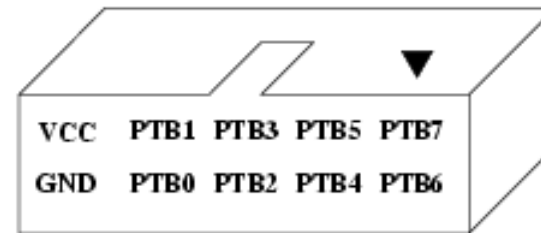


$$R = \frac{V_{out} - V_{diodo}}{I_{diodo}} = 330 \ \Omega$$

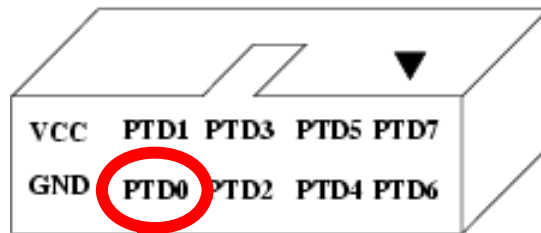
**PUERTO A**



**PUERTO B**

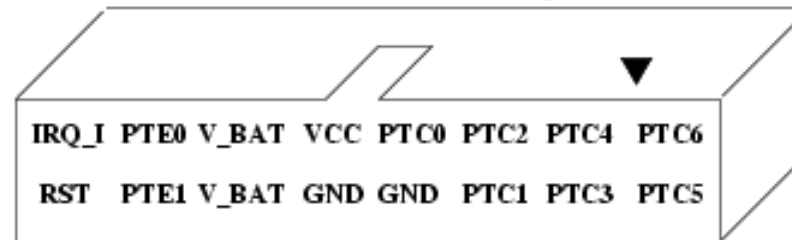


**PUERTO D**



LED

**PUERTO C y E**





```
/* **** */
/* Ejemplo para el PUERTO D. */
/* Se configura para salida y se envia un dato */

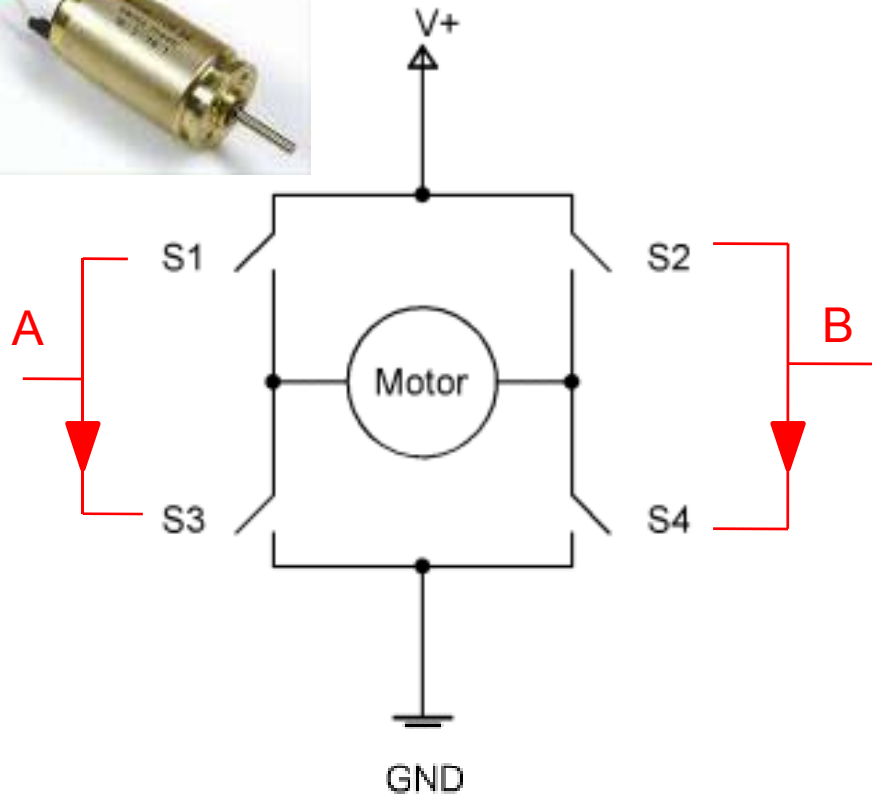
/* **** */

#include <mc68hc908gp32.h>

void main(void) {
    DDRD = 0xFF; //-- Configurar Puerto B para salida
    PORTD = 0x01; //-- Enviar un dato
}
```

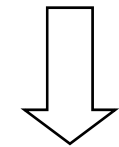
SDCC: Software de prueba [portd\\_sal.c](#)

CodeWarrior: [LED ON.zip](#)



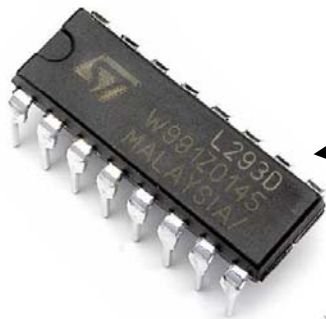
Puente en H: Controlador 1 motor

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

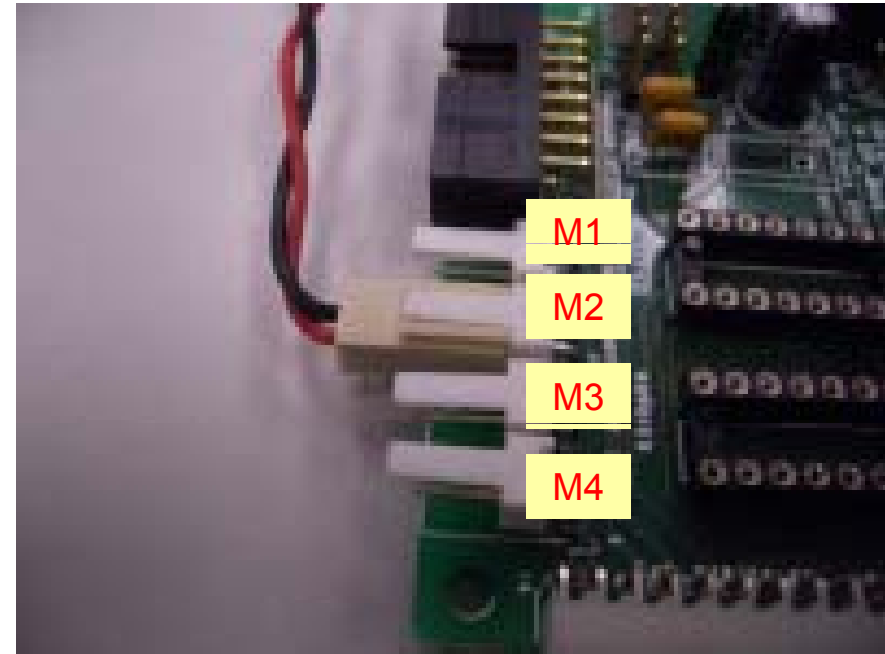
A	B	
on	Off	Izq
off	on	Der
Off	Off	Stop



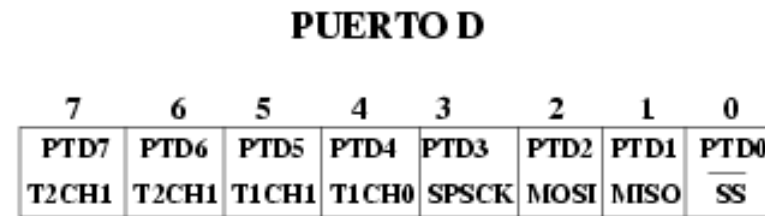
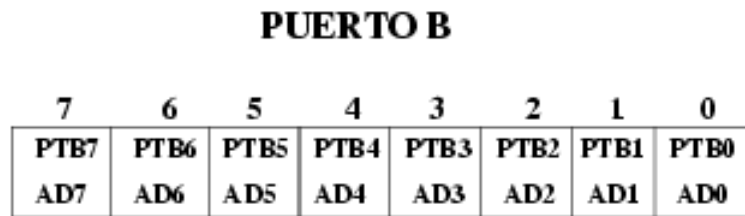
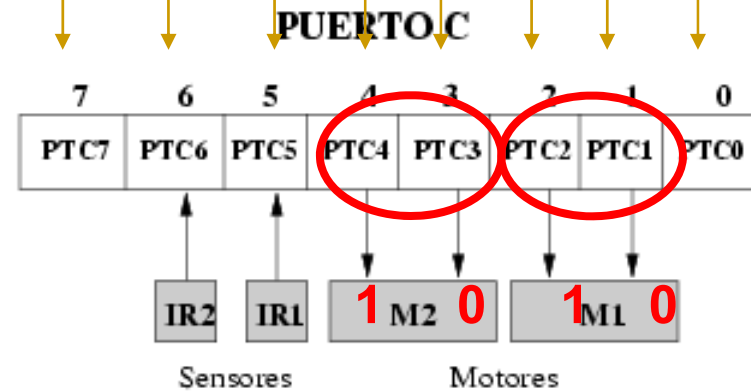
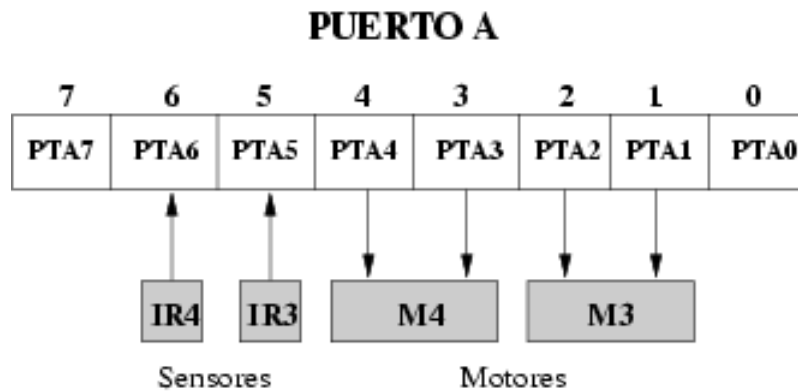
x2

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){
```

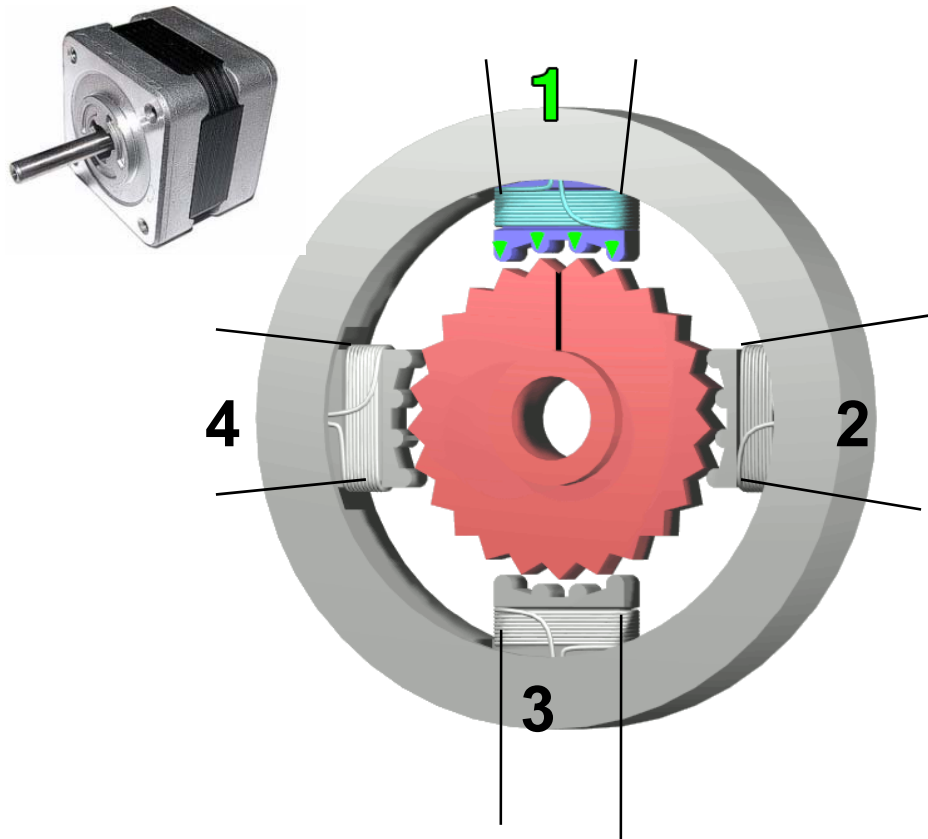
```
    //-- Configurar pines PTC1, PTC2, PTC3 y PTC4 para salida  
    DDRC=0x1E;
```

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

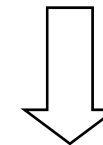
```
    //-- Bucle infinito  
    for (;;);
```

```
}
```

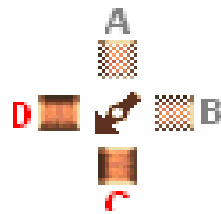
Software de prueba [motor\\_on.c](#)



B1	B2	B3	B4	
1	1	0	0	Step 1
0	1	1	0	Step 2
0	0	1	1	Step 3
1	0	0	1	Step 4
0	0	0	0	Stop

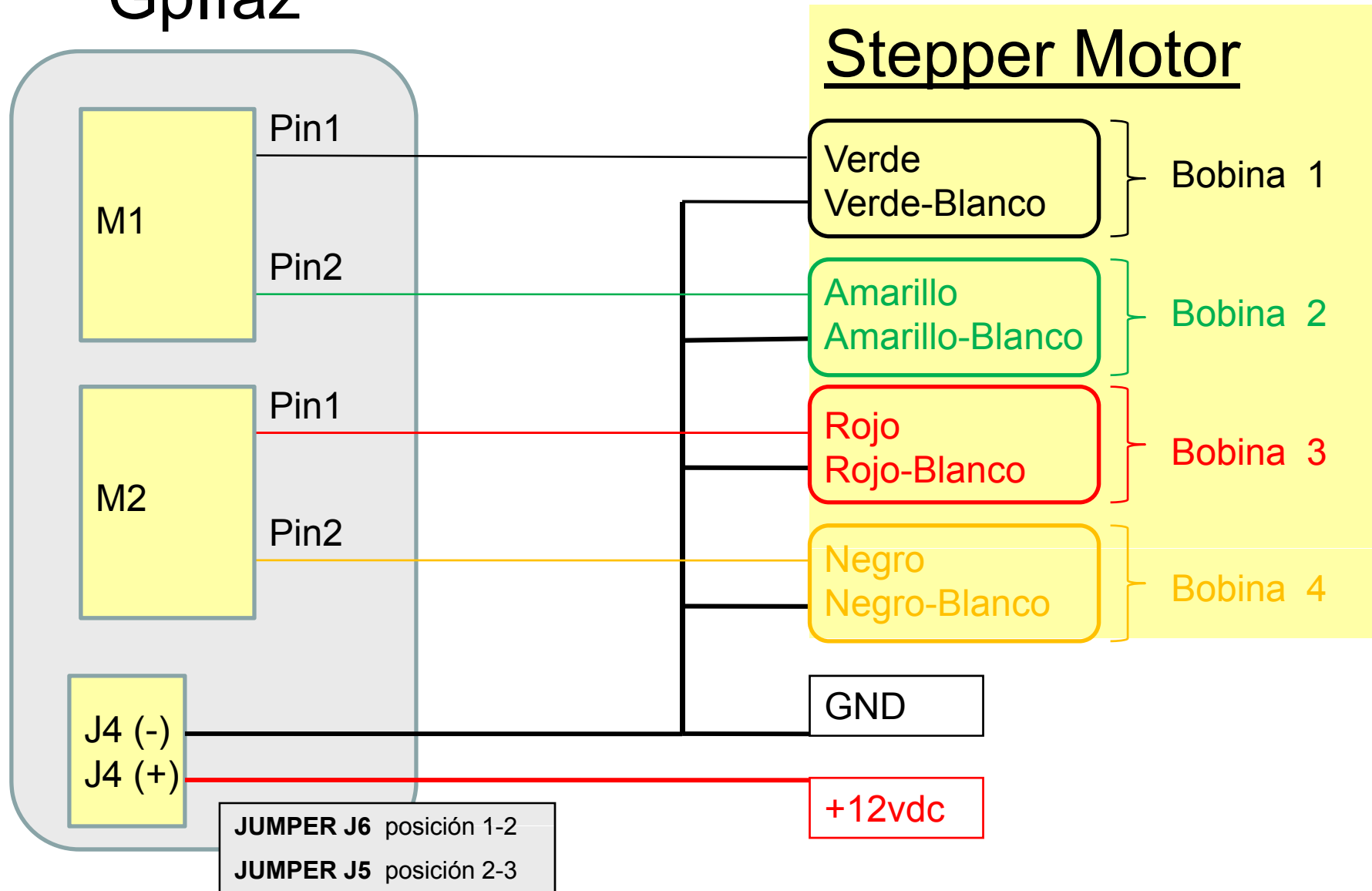


Tipo Unipolar  
Control dos bobinas a la vez (+ fuerza)



L293D ofrece 2 P-H para 1 motor Paso a Paso

# Gplfaz



```
#include <mc68hc908gp32.h>

#define STEP1 0x18; // 0001-1000
#define STEP2 0x0C; // 0000-1100
#define STEP3 0x06; // 0000-0110
#define STEP4 0x12; // 0001-0010

void main(void){

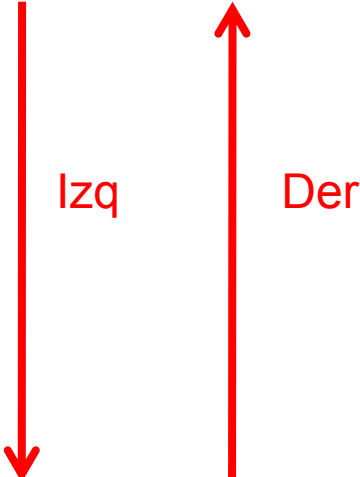
    DDRC=0x1E;    //-- Configura los pines de los motores de salida

    for (;;) {
        PTC=STEP1;
        Pausa();

        PTC=STEP2;
        Pausa();

        PTC=STEP3;
        Pausa();

        PTC=STEP4;
        Pausa();
    }
}
```

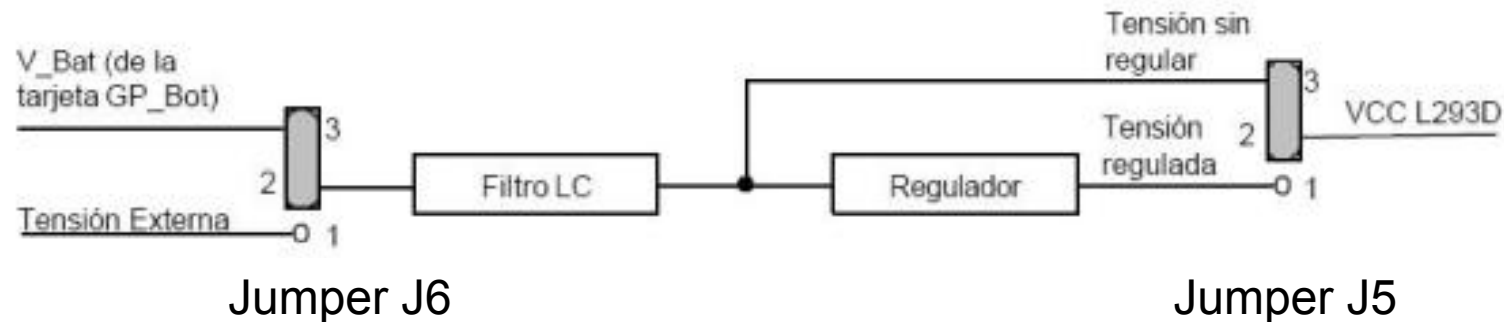


Software de prueba [stepper.c](#)



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 J5



**JUMPER J6** tiene que colocarse en la posición 1-2

**JUMPER J5** tiene que colocarse en la posición 2-3

## 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.

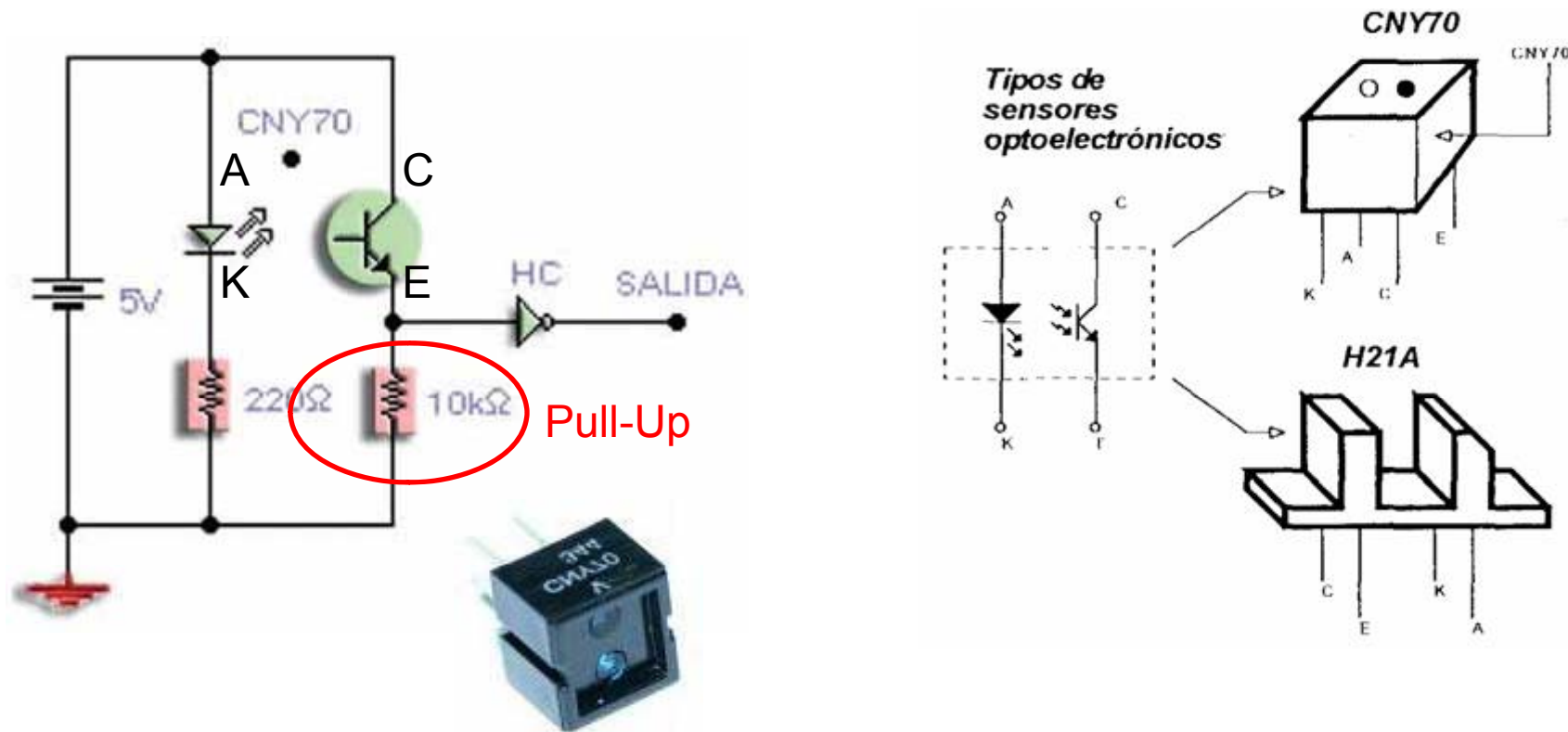
Al invertir el sentido de giro de un motor se produce un pico de corriente que puede provocar una caída de tensión por debajo de 3v. !!!!

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

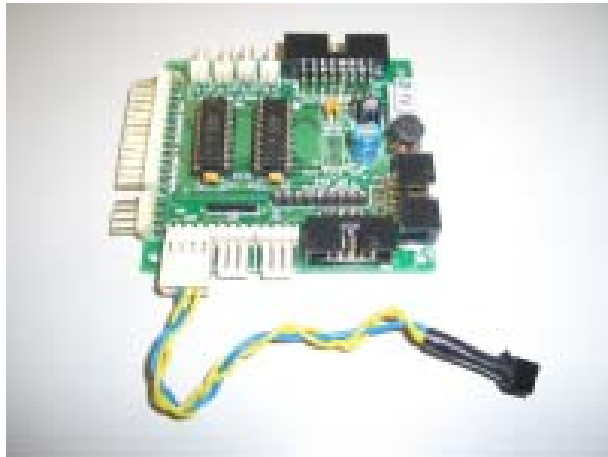
**CONFIG1|=0x10;**

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

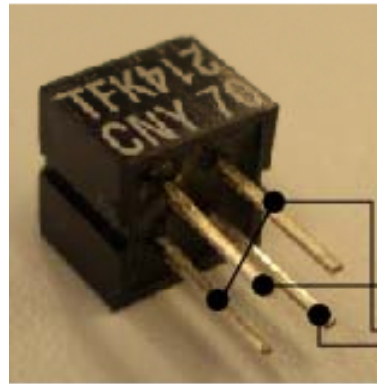
Ejemplo de lectura de "Pin de entrada" usando un CNY70



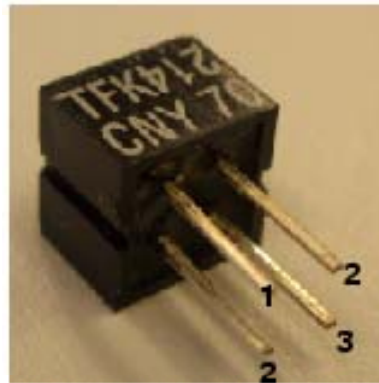
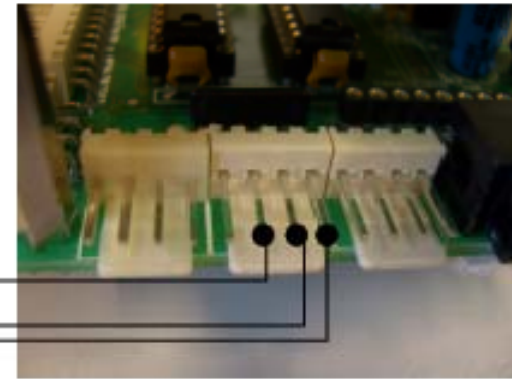
<http://arantxa.ii.uam.es/~gdrivera/robotica/curso0607/p1/p1-doc.html>

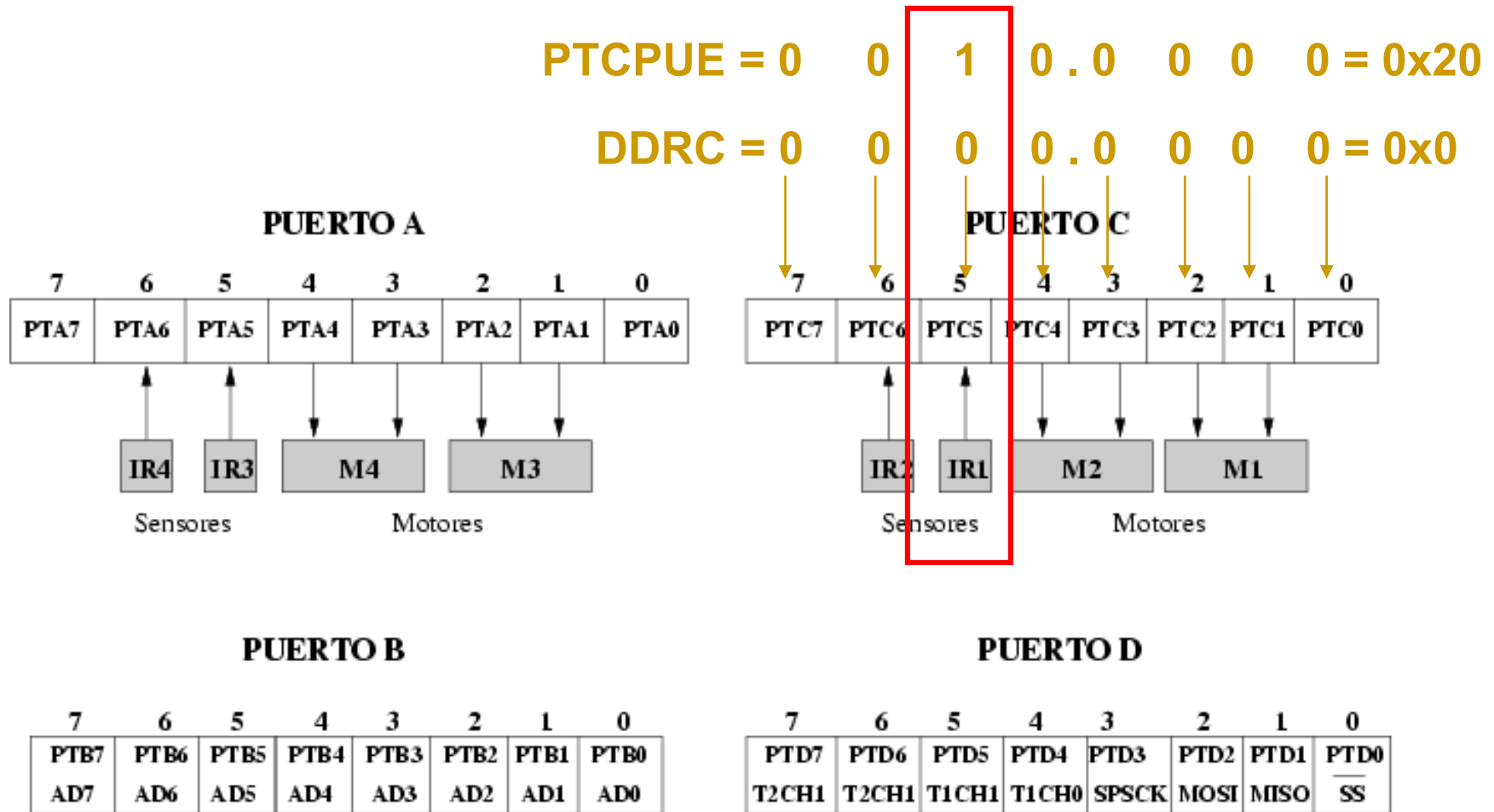


Sensor CNY70



GP\_IFAZ





**PUERTO B**

7	6	5	4	3	2	1	0
PTB7	PTB6	PTB5	PTB4	PTB3	PTB2	PTB1	PTB0
AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0

**PUERTO D**

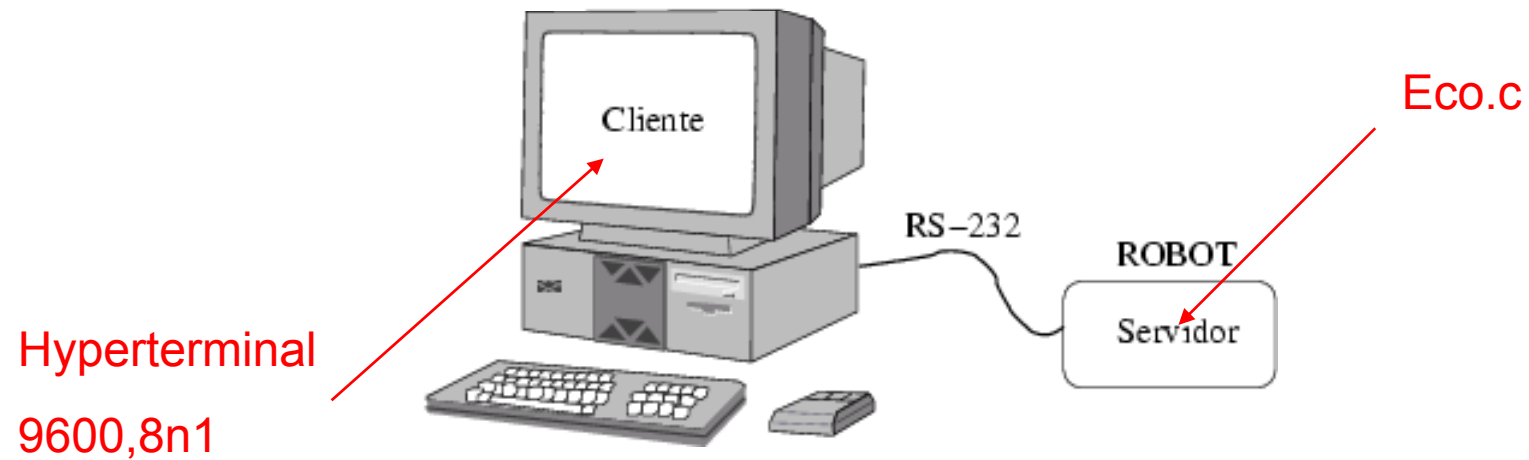
7	6	5	4	3	2	1	0
PTD7	PTD6	PTD5	PTD4	PTD3	PTD2	PTD1	PTD0
T2CH1	T2CH1	T1CH1	T1CH0	SPSCK	MOSI	MISO	SS

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

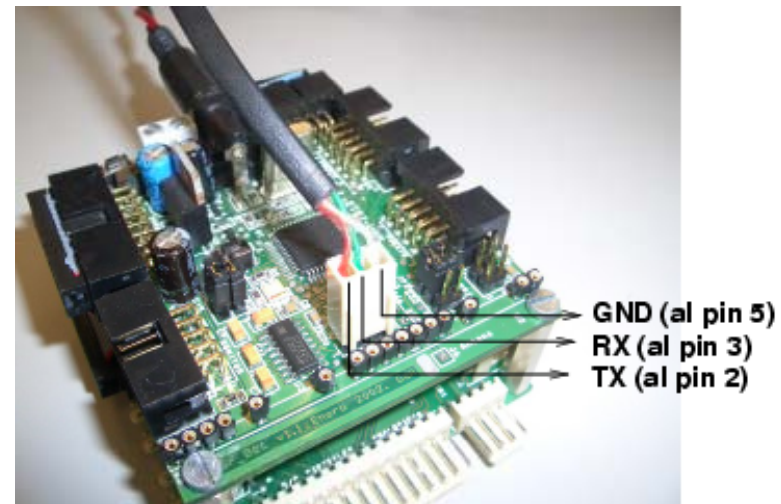
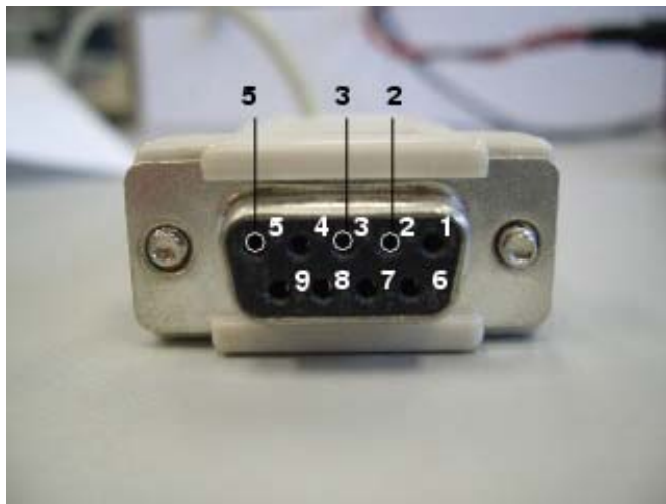
Software de prueba **sensor1.c**

```
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;
    }
}
```



<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:	LOOP	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
- SCRIF** 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  
}
```

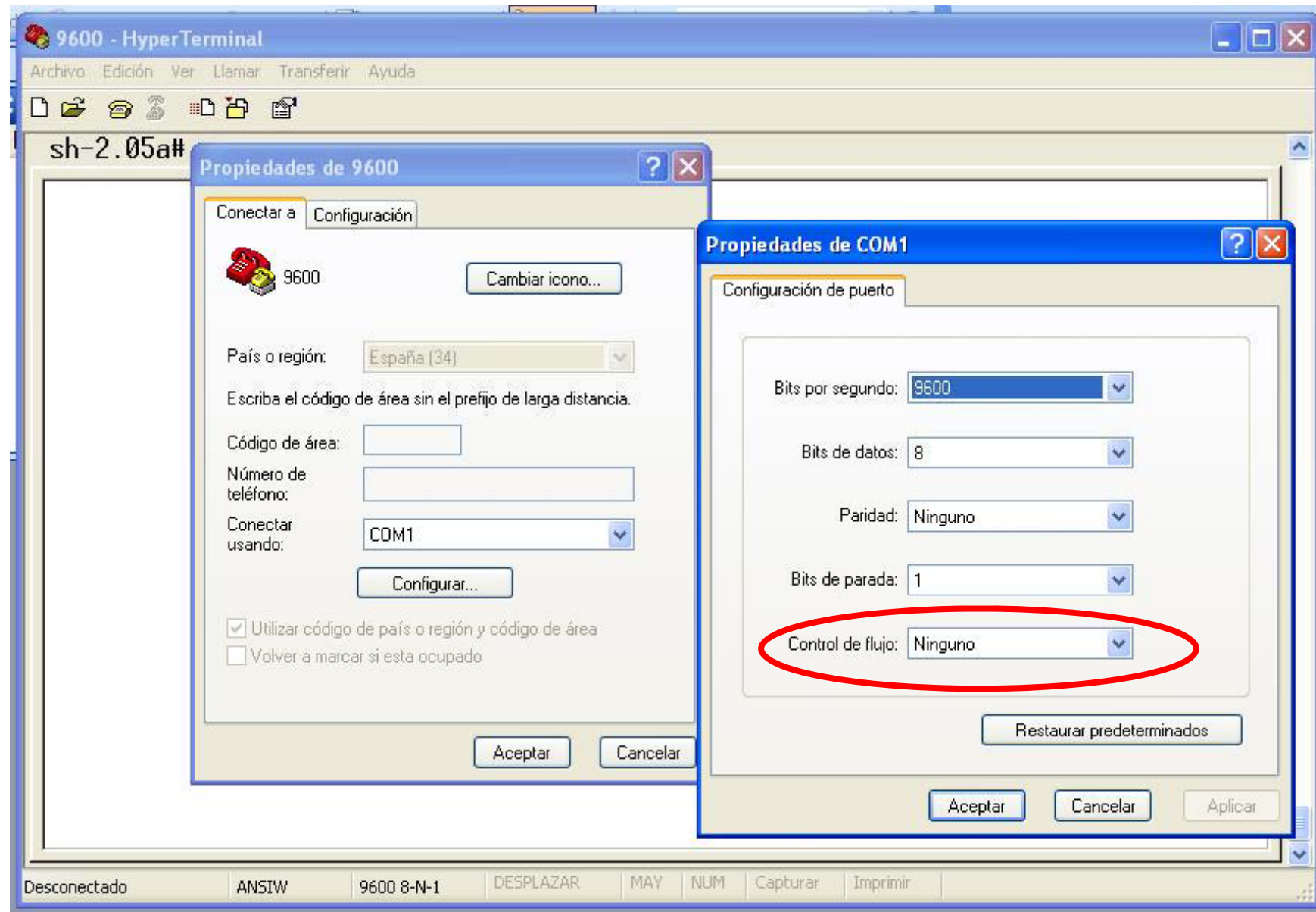
```
unsigned char sci_leer_car(void){  
    //-- Esperar a que se ponga a '1' el Flag (Bit 5)  
    while(!(SCS1 & 0x20));  
    return SCDR;  
}
```

```
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.

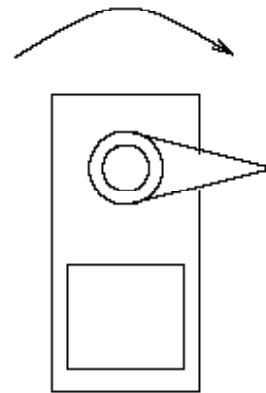
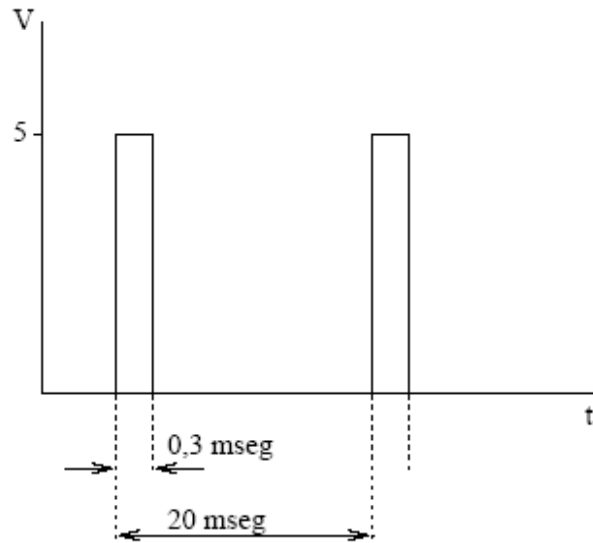
CMGMX\_CLK : Señal equivalente al reloj externo = 9.8304 Mhz

CGMV\_CLK: Señal generada en el PLL ( **No se usa** el PLL )

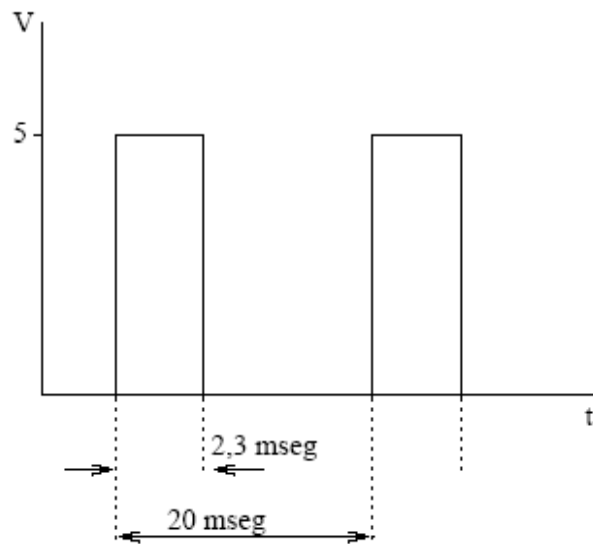
CGMOUT: Señal de reloj Base = CMGMX\_CLK / 2

Reloj interno (Fbus) = CGMOUT / 2 = CMGMX\_CLK / 4

**Reloj Externo 9.8304 Mhz -> Fbus= 2.45 MHz**



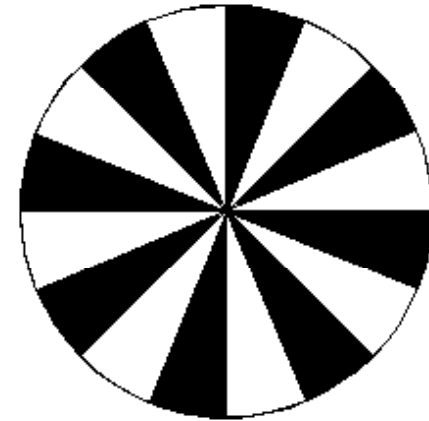
Futaba S3003 Analógico



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

Software por interrupciones



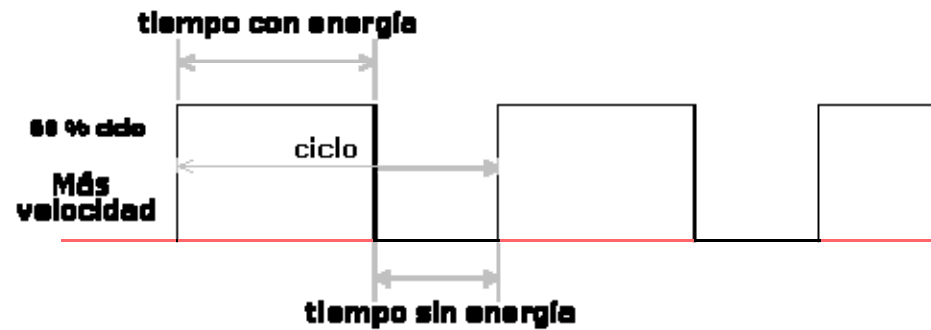


**PWM** = Modulación por ancho de pulso

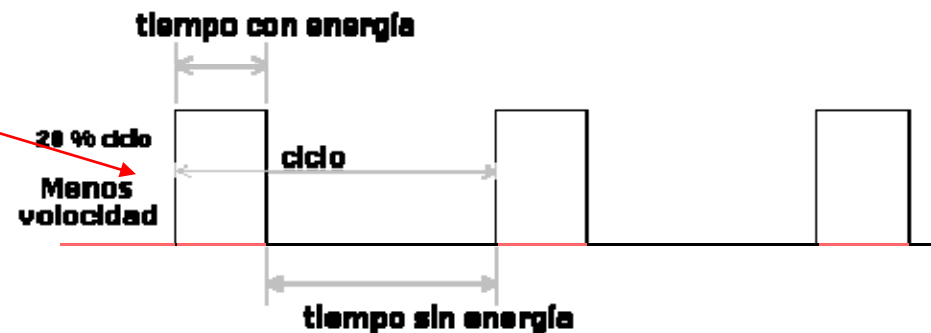
**Nos va a permitir controlar la velocidad de un motor**

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

Ejemplo de programación