Sensorless DC Motor Control

Andreu Playà Caamaño i Jordi Hernández Albà

18 de desembre de 2012

Índex

Capítol 1

Especificació

1.1 Objectius generals

L'objectiu general d'aquest projecte consisteix en dissenyar un controlador d'un motor de corrent continua amb un senyal d'entrada PWM (Pulse Width Modulated), utilitzant com a realimentació únicament la back-EMF (Electro Motrice Force). Aquest voltatge es generat pel propi motor en els períodes de temps en els que no esta alimentat, ja que actua com una dinamo. De la mesura de voltatge que aporta en podem obtenir una aproximació prou bona de la velocitat de gir del motor.

1.2 Objectius concrets

Els objectius concrets del projecte estan dividits en dos grups:

Hardware

- Alimentació correcta del motor DC.
- Muntatge de la placa PIC.

1.2.1 Software

- Lectura correcta del convertidor analògic digital (ADC).
- Sincronització lectura – actuació.
- Obtenció d'un model de la planta.
- Disseny d'un controlador.
- Implementació del controlador en el PIC.

1.3 Metodologia de desenvolupament

La metodologia ha consistit en assolir els objectius de un en un i per l'ordre establert en l'apartat anterior. En alguns casos s'han modificant les solucions trobades en primer lloc per solucions millors ja que s'ha observat que els resultats no eren exactament el que es pretenia.

En cada cas, després de assolir un objectiu s'ha procedit a comprovar que aquest era correcte mitjançant mètodes experimentals. Aquest mètodes han consistit segons cada cas en mostrejar dades per pantalla, observar senyals en el oscil·loscopi o observar el comportament del motor entre d'altres.

1.4 Punts del projecte

1.5 Estudi previ

- Recopilació Informació.
- Estudi de la placa (Hello World).
- Estudi de projectes similars.

1.5.1 Estudi de la planta

- Estudi dels components.
- Disseny de la circuiteria de la adaptació de variables.

1.5.2 Disseny del software

- Sincronitzar la lectura del ADC amb el PWM.
- Trobar un model de la planta mitjançant una entrada graó.
- Trobar funcions de canvi de variable.
- Disseny d'un controlador per la planta.
- Implementació del controlador.

Capítol 2

Projecte

2.1 Hardware

Per a aquest projecte s'ha utilitzat una PIC model "MicroStick dsPic33FJ64MC" per que es un PIC de baix cost, poques prestacions, però alhora dóna gran llibertat de disseny necessària per la sincronització del pols PWM amb la lectura del ADC, més concretament:

```
Architecture  16-bit
CPU Speed (MIPS)  40
Memory Type  Flash
Program Memory (KB)  64
RAM Bytes  16,384
Temperature Range C  -40 to 150
Operating Voltage Range (V)  3 to 3.6
I/O Pins  21
Pin Count  28
System Management Features  PBOR
POR  Yes
WDT  Yes
Internal Oscillator  7.37 MHz, 512 kHz
nanoWatt Features  Fast Wake/Fast Control
Digital Communication Peripherals  2-UART, 2-SPI, 1-I2C
Codec Interface  NO
Analog Peripherals  1-A/D 6x10-bit @ 1100(ksps)
Op Amp  NO
Comparators  2
CAN (#, type)  1 ECAN
Capture/Compare/PWM Peripherals  4/4
PWM Resolution bits  16
Motor Control PWM Channels  8
Quadrature Encoder Interface (QEI)  2
Timers  5 x 16-bit 2 x 32-bit
Parallel Port  PMP
Hardware RTCC  Yes
DMA  8
```

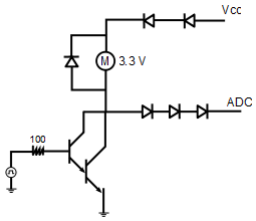
La planta utilitzada es un motor de corrent continua estandard de voltatge de 1.5V a 3V amb reductor, de carecterístiques:

```
Tensión de alimentación  1.5 -> 3 V dc
Par de salida máximo  20 gcm
Velocidad de salida  7800 rpm
Gearhead Type  Cajas reductoras
Diámetro del eje  2 mm
Potencia nominal  1,6W
Core Construction  Iron Core
Corriente nominal  0.85 A
Long.  50 mm
Capacidad  27 mm; 1,1/16pulgadas
```

2.2 Adaptació de variables

En primer lloc s'ha agut de adaptar el voltatge que en proporcionava la pròpia placa del PIC ja que aquesta es de 5V mentre que el voltatge nominal del motor es de 3V. Per a fer-ho s'ha fet ús de una circuit de adaptació enter la font i el motor.

Figura 2.1: Adaptador de voltatge



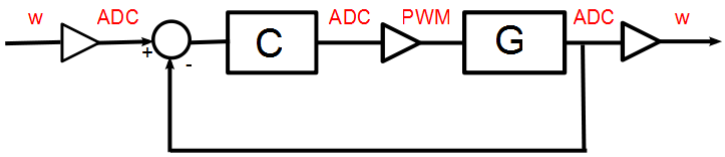
2.3 Conceptual

S'ha utilitzat MATLAB per realitzar els càlculs del controlador, les rectes de regressió i per buscar el model de la planta. Les simulacions per contrastar els resultats han estat obtingudes mitjançant SimuLink.

2.3.1 Canvis de variables

Una vegada tot el sistema ha estat connectat i les mesures del ADC son correctes s'ha procedit a fer un estudi de la planta, per trobar les relacions entre PWM, velocitat angular a la sortida del motor mesurada en el seu eix. El següent esquema mostra la estructura general de la utilització d'aquestes transformacions. Com que el control es fa amb dades del ADC i les convertirem internament a PWM, es necessari ajustar amb màxima precisió aquesta recta. Per aconseguir aquest objectiu s'ha excitat la planta amb una senyal triangular de pendent

Figura 2.2: esquema del recorregut de les variables



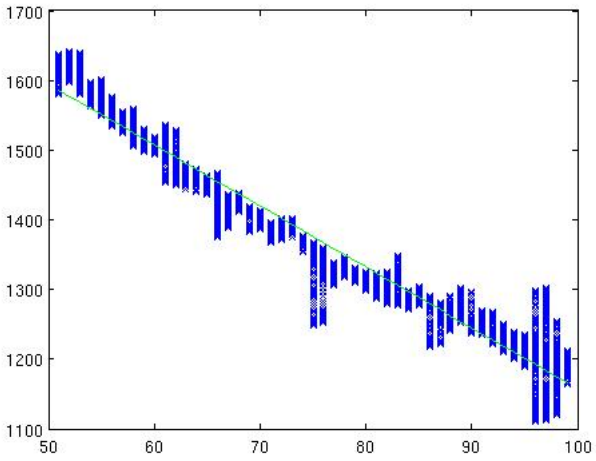
molt baixa, i s'han anat prenent mostres consecutivament. Així doncs per a cada valor de PWM,$[50, 100] \epsilon \mathbb{N}$, es pren aproximadament 100 mostres.

Si es mostren els valors de PWM en front dels de la lectura del ADC obtenim la següent recta:

$$y(x) = -8,75 \times x + 2032$$

Seguint el mateix procediment que en els casos anterior s'ha trobat que la recta

Figura 2.3: recta de regressió PWM - ADC



de regressió entre el valor del ADC i la velocitat angular. Aquesta última és molt més senzilla, ja que només servirà per mostrar les dades per pantalla.

$$y(x) = -0.0021 \times x + 3.8885$$

2.3.2 Filtre

S'ha utilitzat un filtre de mitjana mòbil a l'entrada del ADC. En la següent figura es mostra la senyal ADC i la mateixa senyal filtrada.

Figura 2.4: recta de regressió ADC - rads/s

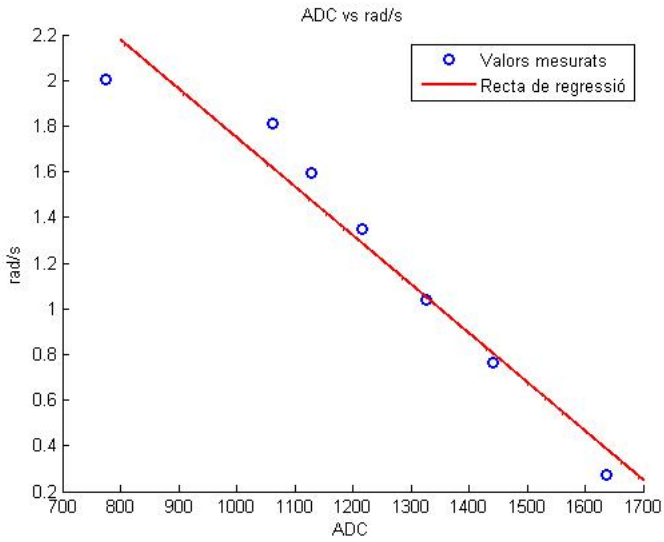
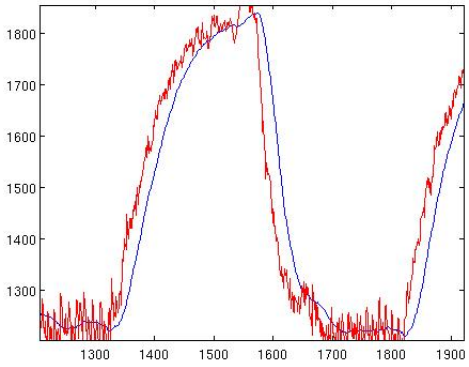


Figura 2.5: Senyal filtrada



El filtre de mitjana mòbil tot i ser un dels mes simples, introdueix molt retard. Per atenuar aquest efecte es sobremostreja la senyal, a efectes pràctics s'utilitza una mostra de cada 10 que equival a un retard de una mostra.

2.3.3 Model

Per tal de crear el model s'ha excitat la planta amb una senyal quadrada i s'ha agafat un número suficient de mostres, aproximadament 10.000, durant 10 períodes aproximadament. Amb aquestes dades s'ha creat un model utilitzant models de estimació paramètrics. El model triat ha estat un ARX 111 que un

cop estimat té variables $A(q) \times y(t) = B(q) \times u(t) + e(t)$

$$A(q) = 1 - 0.883 \times q^{-1}$$

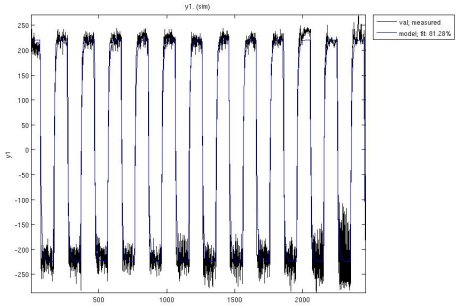
$$B(q) = 0.3317 \times q^{-1}$$

La funció de transferència obtinguda és de primer ordre:

$$\frac{0.1317}{z - 0.8813}$$

Si aquest model li posem la mateixa senyal d'entrada que la planta s'obté una estimació de la correspondencia del model amb la realitat:

Figura 2.6: comparativa output model - planta

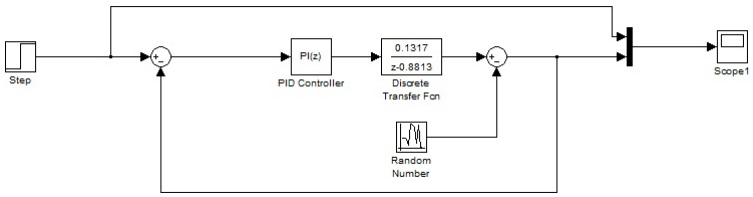


S'obté un fit del 80% que és prou bó tenint en compte la quantitat de soroll que té el sensor.

2.3.4 Diseny del controlador

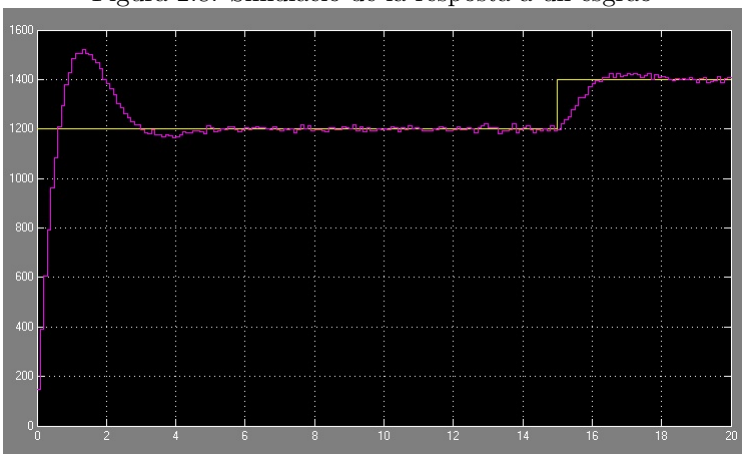
Un cop obtingut el model, es dissenya un controlador PI utilitzant els algoritmes propis del Simulink per a una planta com la de la figura 2.7

Figura 2.7: Model de la planta en Simulink



La solució optima obtinguda per simulació resulta en els valors de Kp=0.7 i Ki=1.9. Amb aquest valors s'obté una resposta al esgraó que amortitza el soroll numèric. En la següent figura es pot apreciar la senyal de referència i la senyal obtinguda.

Figura 2.8: Simulació de la resposta a un esgraó



2.4 Implementació Software

La implementació d'aquest projecte al PIC s'ha realitzat en llenguatge C utilitzant MplabX per ser un software de distribució lliure.

2.4.1 Estructura del Software

El programa dissenyat per a aquest projecte consta principalment de dues tasques. La de lectura del ADC, i la de control-actuació. Ambdues tasques estan sincronitzades de manera que la lectura es fa instants abans de fer la actuació de manera que el càlcul de control sigui el més pròxim possible a l'actuació.

La jerarquia d'aquestes tasques son de productor consumidor, el lector ADC proporciona dades contínuament mentre que una tasca secundaria s'encarrega de fer la mitja de 10 valors consecutius per tal de filtrar la senyal, i entrega el resultat a la tasca de control-actuació.

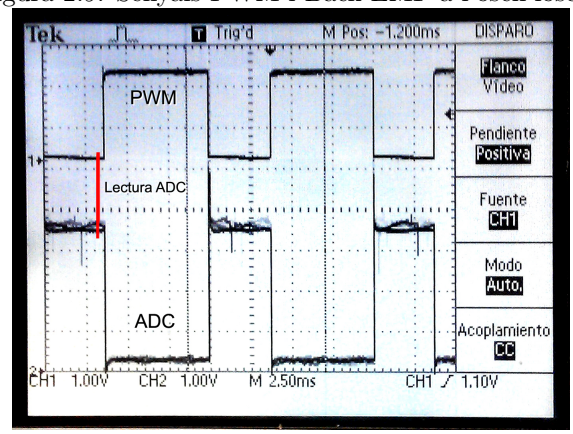
2.4.2 Sincronisme – Lectura ADC

S'ha estudiat com fer la lectura de un convertidor AD amb un PIC, i transformar aquestes dades a valors de velocitat angular. Donat que els valors de la back-EMF eren majors als que el ADC es capaç de llegir, s'han posat dos díodes connectats en sèrie entre el lector ADC i el motor per adaptar el voltatge al desitjat. Per tal de que el control sigui el mes eficient possible es fa la lectura del ADC instants abans de que s'iniciï el PWM tal i com es mostra en la següent figura 2.9.

2.4.3 Controlador

En primera instancia s'implementa el controlador PI dissenyat pel model fet en Simulink. Però com que no s'han tingut en compte els efectes de la saturació, el sistema (controlador+planta) real es inestable i oscil·la. Per tant s'han modificat

Figura 2.9: Senyals PWM i Back-EMF a l'oscil·loscopi

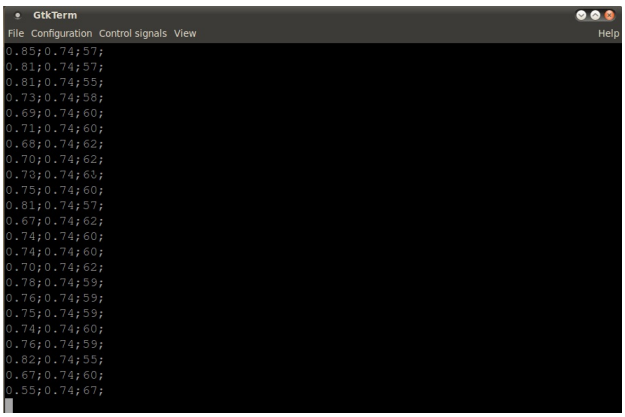


els valors de Kp i Ki manualment fins obtenir la resposta adequada. Els valor finals son: Kp=0.5 i Ki=0.3.

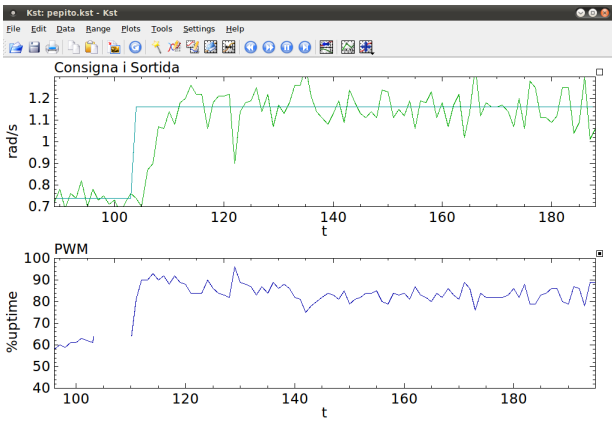
2.4.4 Connexió amb el PC

Per connectar amb el PC s'utilitza el protocol UART (Universal Asynchronous Reciever-Transmitter) que generalment es RS232 però en aquest cas s'utilitza amb connexió USB. En aquest cas s'envien 16 caràcters en total, separats per ';' per poder ser tractats posteriorment.

Per tal de mostrar les dades al PC s'utilitza GtkTerm el qual ens permet veure les dades directament en pantalla. Això s'ha utilitzat per tasques de depuració de codi.



Finalment per mostrar les dades de velocitat angular i senyals de control s'utilitza el KST, programa de distribució lliure que permet mostrar dades de arxius .csv (comma separated values).


Kst: pepito.kst - Kst
File Edit Data Range Plots Tools Settings Help
Consigna i Sortida
rad/s
1.2
1.1
1
0.9
0.8
0.7
100 120 140 160 180
t
PWM
%uptime
100
90
80
70
60
50
40
100 120 140 160 180
t

Capítol 3

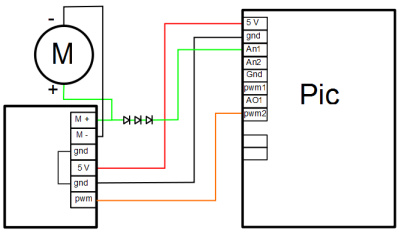
Manual d'usuari

3.1 Instruccions d'instal·lació, preparació i muntatge

3.1.1 Hardware

El hardware d'aquest projecte consta de 3 parts. El Pic, el motor de corrent continua i el circuit d'adaptació. S'han de connectar entre ells tal com s'indica en el següent esquema. NOTA: La connexió entre el motor i el ADC del Pic esta composta per 3 diodes connectats en sèrie.

Figura 3.1: Connexions entre la placa i el Pic



3.1.2 Software

Per poder mostrar les dades per pantalla cal un PC amb linux que tingui instalada la aplicació lliure KST.

Primerament cal comfigurar el canal uart; per fer-ho escriurem les seguents línies de codi a la finestra de comandes:

```
>> screen /dev/ttyACM0 115200
```

El procedimen és correcte si es poden veure les dades del motor per pantalla. A continuacio es prem 'CTRL-A' i seguidament 'K' i es confirma la operació de tancar la finestra amb 'y'.

A continuació enviem aquestes dades a un arxiu utilitzant la següent comana:

```
>> cat /dev/ttyACM0 > data.csv
```

I per acabar s'executa el KST i es carrega l'arxiu de configuració adjunt al cd de recursos.

Bibliografia

[1] Kerhuel, Lubin, *PIC32 Sensorless Speed Controller for DC Motor Applied on Picooz Main Rotor wiki*

http://www.kerhuel.eu/wiki/PIC32_Sensorless_Speed_Controller_for_DC_Motor_Applied_on_Picooz_Main_Rotor

[2] Burroughs, Jon, *Controlling 3-Phase AC Induction Motors Using the PIC18F4431,(29/6/04)*

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en020394

[3] Microchip, *Datasheet for dsPIC33FJ64MC802*

http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en532314

Appendices

```
#include <p33FJ64MC802.h>
#include <stdio.h> //Funciones de entrada salida (fprintf, scanf)
#include <stdbool.h> //Funciones booleanas
_FOSCSEL(FNOSC_FRCPLL) // PLL permite que el reloj FRC corra mas rapido, no es
//_FWDT(FWDTEN_OFF);// Desactiva el watchdog
char data_buf[16];
int valor_ADC;
int valor_mig;
float v_anterior=0;
float mean=1400;
float kp=0.5;//constant proporcional
float ki=0.3;//constant integrador
float e=0;//error instantani
float ea=0;//error acumulat
float up=0;//senyal de control del proporcional
float ui=0;//senyal de control de l'integrador
float u=1400;//senyal de control
int r=0;//contador filtre
int kcons=1000;//contador canvi consigna
int M=10;//longitud filtre mitjana mobil
int stack_ADC[10];//stack del shift register dels valors de l'adc
float wactual;//valor convertit de lADC->rad/s
float k1=-0.0021;//pendent recta de regresio ADC->rad/s
float o1=3.8885;//offset recta de regresio ADC->rad/s
float consigna_rads=1.5;//rad/s
int consigna_ADC=0;
float a1=-8.34;
float a0= 2032.0;
bool debug=false;



/*UART Data parity mask selection */
#define BIT9       0x06
#define BIT8_ODD   0x04
#define BIT8_EVEN  0x02
#define BIT8_NO    0x00

/*UART Stop bits mask selection */
#define BIT_STOP_1 0x00
#define BIT_STOP_2 0x01

/*UART Error codes */
#define UART_ERR_OVERFLOW -1
#define UART_ERR_NO_DATA  -2

static volatile unsigned fcy_ = 39.61375;
#define MEGA 1000000ul

volatile unsigned int DutyCicle=1;
```

```
volatile unsigned int T_alt,T_baix;
volatile unsigned long Taux;

void __attribute__((interrupt)) _MPWM1Interrupt (void) //Manera de declarar int
{
    IFS3bits.PWM1IF=0;
    ////LATBbits.LATB4 = ~LATBbits.LATB4;
    //LATAbits.LATA3 = 1;
    filtreMM();
    canvi_consigna();
}

void __attribute__((interrupt)) _T1Interrupt (void) //Manera de declarar interr
{
       // Entra 5 cops per segon
       IFS0bits.T1IF=0;
}
void __attribute__((interrupt)) _T3Interrupt (void) //Manera de declarar interr
{
    IFS0bits.T3IF=0;
}

void __attribute__((interrupt)) _ADC1Interrupt (void) //Manera de declarar inte
{
    IFS0bits.AD1IF=0; // Aquesta RSI S'executa en 2,96us
    valor_ADC=ADC1BUF0;

   // LATBbits.LATB4 = ~LATBbits.LATB4;
}

void Init_AD(void)
{

//Pasos para configurar el ADC 16.7
    //1. Select 10-bit or 12-bit mode
    AD1CON1bits.ADON=0;      // Inicializamos el ADC
          //AD1PCFGL Port Configuration Register
    AD1PCFGL=0xFFFE;         // Solo la primera ANO es una entrada analogica (bi
    //2. Select the voltage reference source to match the expected range on ana
        //VCFG<2:0> Converter Voltage Reference Configuration
    AD1CON2bits.VCFG2=0;
    AD1CON2bits.VCFG1=0;
    AD1CON2bits.VCFG0=0;     //  000 -> Referencia del AD: Vss-Vdd
   //3. Select the analog conversion clock to match the desired data rate with

    //AD1CON3=AD1CON3&0x7F00; // Usamos el system clocK (bit15=0)
    AD1CON3bits.ADRC=0;
    //AD1CON3=AD1CON3|0x0003; // divisor 4 (bit7-0=1) TAD=4*TCY

    AD1CON3bits.ADCS = 2; //TAD = (ADCS+1)*TCY, TCY = 33.9nSec TAD = 2.22 uSec
```

```
    AD1CON2bits.CHPS0=0;
    AD1CON2bits.CHPS1=0;      // Usem un unic S&H: canal0
    AD1CSSL=0x0001;           // Canal 0 entrada
    AD1CHS0=0x0000;           // Canal 0 triat
    AD1CON1bits.AD12B=1;      // 12 bits !
//AD1CON1bits.SSRC    = 3; // Sample Clock Source: Motor Control PWM
    AD1CON1bits.SSRC0=0;// LO he cambiado porque para sincronizar el ADC con el
    AD1CON1bits.SSRC1=1;
    AD1CON1bits.SSRC2=1;// Automatic Conversion
    AD1CON1bits.SSRC=3;
    AD1CON1bits.ASAM=1;       // Automatic Sampling:Esta Activo en el ejemplo de

    AD1CON3bits.SAMC0=0; //SAMC Auto Sample Time bits
    AD1CON3bits.SAMC1=0;
    AD1CON3bits.SAMC2=0;
    AD1CON3bits.SAMC3=0;
    AD1CON3bits.SAMC4=0;    // Sample= 0 TAD


//Data Output Format
    AD1CON1bits.FORM0=0;
    AD1CON1bits.FORM1=0;    // Salida entera de 12 bits

    IEC0bits.AD1IE=1;         // Enable interrupt AD
    IFS0bits.AD1IF=0;         // AD1IF:Interrupt Flag Status, Instruccion: Borra
    IPC3bits.AD1IP=7;         // ***Ojo amb la prioritat
    AD1CON1bits.ADON=1;       // Engeguem AD

}
void Init_PWM(void)
{
    //FPOR
    P1TCONbits.PTCKPS=0b10;   //Preescaler Input clock period is 16 TCY because
    P1TCONbits.PTOPS =0b00;   // Postscaler  1:1
    P1TCONbits.PTMOD =0b00;   // Free running Count Mode
                              // FOSC=79227500 Hz -> Fcyc=39613750 Hz
                              // FCY device operating frequency.
                              // FPWM is the PWM switching frequency
                              // PRE=16->2475859Hz -> CNT=25758=100 Hz
    P1TPER= 24908;            // Equation 14.1 P1TER= (FCY/(FPWM*PxTMR Prescaler
                              // Periode     = 10ms (corregit respecte al teoric)
    P1DC1 = 12306;            // Duty Cicle = 25%
    P1TMR=0;                  // Iniciem timer dedicat a 0


    PWM1CON1bits.PMOD1=0;
    PWM1CON1bits.PEN1H=1;
    PWM1CON1bits.PEN1L=0;
```

```
    PWMCON2bits.IUE=1;
    PWMCON2bits.UDIS=0;
    P1OVDCONbits.POVD1H=1;
    P1OVDCONbits.POVD1L=0;

    IFS3bits.PWM1IF=0;          // Borrem pendents...
    IEC3bits.PWM1IE=1;
    IPC14bits.PWM1IP=6;         // Interrupcio PWM

    P1SECMPbits.SEVTDIR = 1;     // trigger ADC when PWM counter is in upwards c
                                                          //....Tad= 2 uSec, Tpwm
    P1SECMPbits.SEVTCMP = 24700;

    TRISBbits.TRISB14=0;        //Sortida
    P1TCONbits.PTEN = 1;        // Enable PWM
}
void Set_PWM (unsigned int duty)  // duty pot valdre fins 88% tal com esta
programat
{
    P1DC1 = 2*249*duty;                 // Duty Cicle = 1% equival a 2x249,08
}
void Init_CLK()
{
    CLKDIVbits.PLLPRE=0;
    PLLFBD=41;
    CLKDIVbits.PLLPOST=0;
    while(!OSCCONbits.LOCK); //CLK=79.2275MHz
}
void Init_T1(void)
{
    T1CON=0;                    //Reset Timer
    IFS0bits.T1IF=0;            //Borrem pendents...
    IPC0bits.T1IP=7;            //Ojo amb la prioritat
    IEC0bits.T1IE=1;            //Enable timer
    TMR1=0x0000;                //Posem a 0
                                //FOSC=79227500 Hz -> Fcyc=39613750 Hz
    T1CONbits.TCKPS0=1;
    T1CONbits.TCKPS1=1;         //Prescaler a 256 sobre Fcy=39.61375 MHz->154751Hz
    PR1=30946;                  //30946 -> 5 IRQ per segon
    T1CONbits.TON=1;            //Engeguem
}
void Init_T32(void)
{
    T2CONbits.T32=1;        // Timer de 32 bits composat pel 2 i 3
    T2CONbits.TCKPS=0;
    T2CONbits.TCS=0;
    T2CONbits.TGATE=0;
    PR3=0x262;              // Part alta
    PR2=0x5A00;             // Part , un segon
    IFS0bits.T3IF=0;
```

```
    IPC2bits.T3IP=6;
    IEC0bits.T3IE=1;     // Preparem interrupcio
    T2CONbits.TON=1;     // Posem en marxa
}

void serial_port_init(unsigned long baud)
{
    RPOR5bits.RP11R = 0b00011;/* Set on pin RP11 the output signal U1TX */

    RPINR18bits.U1CTSR = 0b11111; /* U1CTS is routed through no pin */
    RPINR18bits.U1RXR  = 10; /* Route U1RX through pin RP10 */

    U1MODE           = 0;                  /* reset UART 1 mode registry */
    IEC0bits.U1RXIE = 0;                    /* Disable Interrupts */
    IEC0bits.U1TXIE = 0;
    IFS0bits.U1RXIF = 0;                  /* Clear Interrupt flag bits */
    IFS0bits.U1TXIF = 0;

    U1MODEbits.BRGH    = 0;               /* UART bit lenght is 16x of
clock (istruction not needed, judspst to make it explicit) */
    U1MODE |= BIT_STOP_1 | BIT8_NO & 0x07;        /* Number of bit,
Parity and Stop bits. Only the 3 lsb mans for this */

    /*Per far esguire l'arrotondamemto si utilizza il trucco (x + (y/2))y */
    U1BRG = (fcy_ *MEGA + (16 * baud)/2)/(16 * baud) - 1; /* UART Baud
Rate Generator (BRG): vedi dspic family reference manual section 19-3
*/

    U1MODEbits.UARTEN = 1;                /* enable UART */

    /* TX & RX interrupt modes */
    U1STA = 0;
    U1STAbits.UTXEN = 1;

}

void uart1_write_byte(unsigned char data)
{
    /* Polling mode */
    while (U1STAbits.UTXBF) ;
    U1TXREG = data;
    while (!U1STAbits.TRMT) ;
}

void uart1_write_string(const char* str)
{
    int i;
    for(i=0; str[i]!='\0'; i++)
        uart1_write_byte(str[i]);
}
```

```
void control(void)
{
    float tmp;
    float ref=(float)consigna_ADC-mean;
    float y=(float)valor_mig-mean;
    e=ref-y;
    ea=ea+e;
    up=kp*e;
    ui=ki*ea;
    u=ui+up;
    u=u+mean;
    tmp=(u-a0)/a1;
    DutyCicle=(int)tmp;
    if(debug==true)
    {
        if(DutyCicle>100)
        {
            uart1_write_string("\r\nERROR: saturacio PWM>100%\r\n");
            sprintf(data_buf,"DutyCicle=%d\r\n",DutyCicle);
            uart1_write_string(data_buf);
        }
        if(DutyCicle<40)
        {
            uart1_write_string("\r\nERROR: saturacio PWM<40%\r\n");
            sprintf(data_buf,"DutyCicle=%d\r\n",DutyCicle);
            uart1_write_string(data_buf);
        }
    }
    v_anterior=(float)valor_mig;
}
void filtreMM (void)
{
        int s=0;
        long int sum=0;
        stack_ADC[r]=valor_ADC;
        r++;
        if (r==M)
        {
            r=0;
        }
        for(s=0;s<M;s++)
        {
             sum=sum+stack_ADC[s];
        }
        valor_mig=sum/M;

}

void canvi_consigna(void)
```

```
{
    kcons++;
    int max=1300;
    int min=1500;
    if(kcons>1000){


        if (consigna_ADC==max)
        {
            consigna_ADC=min;
        }
            else
            {
                consigna_ADC=max;
            }
              kcons=0;
    }
}

int main(void)
{

    Init_CLK();
    serial_port_init(115200);
    Init_T1();
    Init_T32();
    Init_PWM();
    Init_AD();
    TRISBbits.TRISB4 = 0;
    LATBbits.LATB4 = 1;
    TRISBbits.TRISB15=0;

    DutyCicle=60;
    Set_PWM(DutyCicle);

    int i;
    int j;
    uart1_write_string("\r\nHELLO_WORLD\r\n");

    if(debug==true)
        {
            uart1_write_string("\r\nvalor_mig;consigna_ADC;DutyCicle\r\n");

        }
    if(debug==false)
        {
            uart1_write_string("\r\nwactual,consigna_rads,DutyCicle\r\n");
        }
    while(1)
    {
```

```
            wactual=((float)valor_mig*k1+o1);
            if (wactual<0)
            {
                wactual=0;
            }
            consigna_rads=(float)consigna_ADC*k1+o1;

            if(debug==true)
            {
                sprintf(data_buf,"%d;%d;%d\r\n",valor_mig,consigna_ADC,DutyCicle);/
            }
            if(debug==false)
            {
                sprintf(data_buf,"%.2f;%.2f;%d;\r\n",wactual,consigna_rads,DutyCicl
            }
            uart1_write_string(data_buf);
            for(i=1;i<1000;i++)
            {
                for(j=1;j<770;j++){}//introdueix una latencia de 100ms
            }
            control();
            Set_PWM(DutyCicle);
        }
}
```