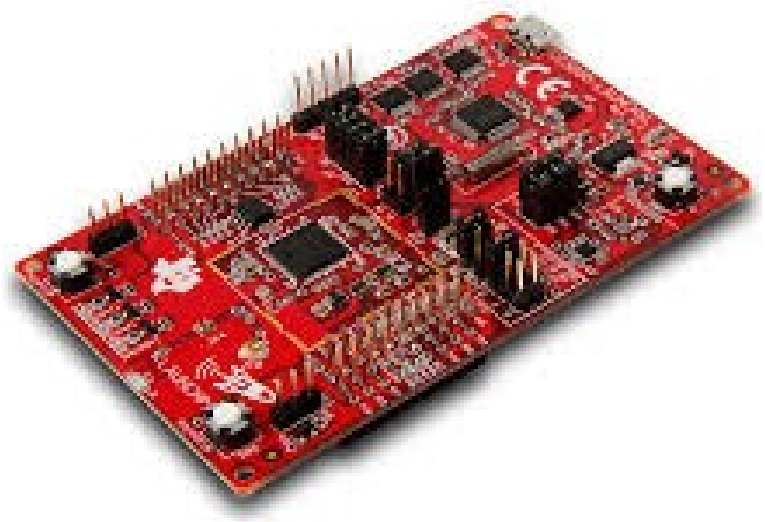
Programació d'Arquitectures Encastades

Pràctica 4 – UART i llibreria del robot

Semestre de Primavera 2023



David Fernàndez
Víctor Sort

David Fernàndez
Víctor Sort

ÍNDEX

David Fernàndez
Víctor Sort

1. OBJECTIUS

A diferència de les anteriors pràctiques, on l'objectiu consistia en crear un codi que implementés una funcionalitat específica a la placa del Boosterpack o Launchpad usant el microcontrolador MSP432P401R, en aquesta l'objectiu és crear les funcions bàsiques de comunicació amb el robot que usarem en el projecte final i preparar una llibreria de funcions per ell.

Per fer-ho, inicialment s'han hagut de generar els recursos bàsics per comunicar la placa del microcontrolador amb el robot, fent una funció per configurar les comunicacions amb els mòduls del robot, una altre per enviar "comandaments" als mòduls del robot i una útlima per rebre informació dels mòduls del Robot. Corresponentment, són els mètodes init_UART(), TxPacket() i RxPacket().

Inicialment es va fer una versió emulada on ens comunicàvem amb una versió emulada en el PC. Per aconseguir-ho s'ha d'usar el mòdul eUSCI amb el protocol asíncron UART del microcontrolador que va cap el PC (Backchannel UART). Es va començar per aquí, doncs és molt més senzill detectar errors i corregir-los.

A posterior, ja sí que es va fer una versió real on ens comunicàvem amb els motors/sensors Dynamixel. Va caldre canviar a un altre mòdul eUSCI, també amb el protocol asíncron UART que és el que realment està connectat a aquests dispositius. Per provar aquesta versió, es va prendre com a objectiu encendre els LEDs dels motors del robot.

Finalment s'ha fet una llibreria de funcions que permetin controlar el moviment del robot. Inclouen totes les següents funcions:

- Encendre i apagar els LEDs dels motors.
- Clnfigurar el mode continu dels motos (endless turn).
- Aturar-se.
- Avançar recte.
- Retrocedir recte.
- Avançar amb gir cap a la dreta i cap a l'esquerra.
- Retrocedir amb gir cap a la dreta i cap a l'esquerra.
- Girar en sentit horari i antihorari.
- Pivotar sobre ell mateix en sentit horari i antihorari.
- Llegir els tres sensors de distància.

Així doncs, com a resum, es pot dir que l'objectiu d'aquesta pràctica ha sigut deixar-ho tot preparat per començar amb el projecte principal d'aquesta assignatura.

David Fernàndez
Víctor Sort

2. RECURSOS UTILITZATS DEL ROBOT

En aquesta pràctica, no s'han fet servir ni recursos de la placa Launchpad ni de la placa d'experimentació (Boosterpack) del robot.

S'han usat els 2 mòduls Dynamixel AX-12 del robot, on hem usat els seus LEDs (per comprovar la correcta configuració) i els seus motors (per moure el robot). A continuació es dona informació sobre aquests mòduls. Més informació detallada de com ha estat configurada la UART i la seva connexió amb els mòduls es troba al punt 4.1. Realment, es pot dir que de la Launchpad sí que s'ha usat l'eUSCI en mode UART per enviar comunicacions a aquests mòduls.

La connexió entre la placa i els mòduls, que es connecten en daisy chain, es fa mitjançant una UART A2 del port 3. En el punt 4. s'especifica com s'han utilitzat els pins 3.2 i 3.3, configurant-los correctament per permetre la comunicació. El baud rate programat dels mòduls Dynamixel és de 500 kbps, i per tant la UART s'ha de configurar per treballar a aquesta velocitat.

Els mòduls Dynamixel fan servir una comunicació asíncrona però Half-duplex (només té una línia "Data" per transmetre i rebre) mentre que la UART en principi és Full-duplex (té una línia per transmetre UCAxTXD i una per rebre UCAxRXD). Per solucionar-ho s'ha de fer un circuit que passi de dues línies a una i viceversa.

S'ha hagut de tenir en compte que des del microcontrolador, per programa, s'ha de controlar el senyal "DIRECTION_PORT". A la nostra placa l'hem connectat al port 3, al pin 3.0. S'ha d'inicialitzar com GPIO de sortida, i gestionar el senyal en funció de si volem enviar o rebre missatges.

Codi per configurar el pin 3.0 com a GPIO de sortida:

```
void config_Direction_Port(void){
      P3SEL0 &= ~BIT0; //P3.0 configurat com a GPIO
      P3SEL1 &= ~BIT0; //P3.0 configurat com a GPIO
      P3DIR |= BIT0;}   //És de sortida}
```

Funcions per canviar el sentit de les comunicacions:

```
void Sentit_Dades_Rx(void){ //Configuració del Half Duplex dels motors: Recepció
      P3OUT &= ~BIT0;} //El pin P3.0 (DIRECTION_PORT) el posem a 0 (Rx)


void Sentit_Dades_Tx(void){ //Configuració del Half Duplex dels motors
Transmissió
      P3OUT |= BIT0;} //El pin P3.0 (DIRECTION_PORT) el posem a 1 (Tx)
```

David Fernàndez
Víctor Sort

Mòduls Dynamixel AX-12

A l'hora de fer la llibreria, s'ha inclòs una funció que retorni la distància a la que es troba el robot d'altres objectes o obstacles. Per això s'ha fet servir també el mòdul de sensor. No cal una configuració d'aquest sensor, ja que ja existeix una funció de lectura. Simplement cal saber l'ID del sensor.

Un altre recurs que s'ha usat és el timer A0, per controlar el temps del codi del main (que es troba explicat al punt 4.3) i el timer A1 per controlar els possibles timeouts de la comunicació amb el robot. A continuació es troba el codi comentat de configuració del timer A1, fet perquè generi interrupcions cada desena d'un microsegon. El del timer A0 és completament idèntic.

```
void init_timers(void){

    //Timer A1, used for calculating possible timeout

    //Divider = 1; CLK source is SMCLK; clear the counter; MODE is stop

    TIMER_A1->CTL = TIMER_A_CTL_ID_1 | TIMER_A_CTL_SSEL_SMCLK | TIMER_A_CTL_CLR
| TIMER_A_CTL_MC_STOP;

    TIMER_A1->CCR[0] = 240 - 1;     //100kHz 10^-5 s (decenas de microsegundos)

   TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Activem les interrupcions del
timer A1

}
```

David Fernàndez
Víctor Sort

3. PROBLEMES

Realitzant aquesta pràctica, a part de petits problemes de compilació, hem tingut com a major dificultat la configuració de la UART2, la corresponent a la versió real. El motiu principal era que ens calia configurar el port 3 (DIRECTION_PORT) com a GPIO de sortida, tal i com s'ha vist en el punt anterior. La configuració final es mostra i s'explica en el punt 4.1.

A més a més, en haver d'usar el mateix codi per l'emulador i el robot real, va suposar que bastantes parts del codi estiguessin quasi repetides i s'haguessin d'utilitzar diversos defines per reaprofitar el codi.

Un exemple d'això és que hi ha dos mètodes per inicialitzar les UARTs: init_UART2 per la UART del robot i init_UART0 per la del emulador. Es defineix a l'inici:

```
#define initUART init_UART0
#define initUART init_UART2
```

i es comenta la que no estigui sent utilitzada. Així amb moltes altres variables. Això va suposar igualment diversos errors que vàrem trigar bastanta estona en corregir. També vam invertir bastant temps intentant optimitzar el codi.

Un altre gran problema que vam trobar va ser la instal·lació de l'emulador, ja que no funcionava correctament a causa de la versió més recent del Matplotlib que venia per defecte en la instal·lació de l'Anaconda. A més a més, va ser impossible que en la versió emulada el robot fes els moviments desitjats i durant el temps desitjat. Això és així perquè si enviem la comanda de girar i la mantenim durant 0.1s no gira res, amb 0.11s gira 100º i si demanem que mantingui la comanda durant 10s amb un timer ben configurat, la simulació la manté durant 16s. El codi presentat en aquest informe presenta els intervals de temps entre comandes perquè el robot en l'emulador pugui fer un recorregut sense xocar-se amb les parets, però, el projecte entregat, modifica els intervals a 1s perquè funcioni correctament amb el robot real.

David Fernàndez
Víctor Sort

4. EXPLICACIÓ DE MÈTODES

4.1 Inicialització de la UART

Els codis per inicialitzar la UART es troben comentats a l'annex 6.2 (tant la UART2 usada pel robot com la UART0 usada per l'emulador). Com no tenen cap diferència més enllà del baud rate usat, explicarem només la configuració de la UART2, i la diferència en el baud rate de la UART0.

```
void init_UART2(void){
    UCA2CTLW0 |= UCSWRST;
    UCA2CTLW0 |= UCSSEL__SMCLK;
    UCA2MCTLW = UCOS16;
    UCA2BRW = 3;
    P3SEL0 |= (BIT2 | BIT3);
    P3SEL1 &= ~(BIT2 | BIT3 );
    UCA2CTLW0 &= ~UCSWRST;
    EUSCI_A2->IFG &= ~EUSCI_A_IFG_RXIFG;
    EUSCI_A2->IE |= EUSCI_A_IE_RXIE;
    config_Direction_Port();
}
```

La primera línia de codi té moltes funcionalitats diferents. Serveix per fer un reset de la eUSCI, ficar-la en mode asíncron i seleccionar el mode UART. Serveix també per indicar que només té 1 stop bit, 8 bits de dades, que el bit de menys pes va primer i que no es fa servir bit de paritat.

Les següents 3 línies serveixen per seleccionar el clock SMCLK que funciona a 24MHz, indicar que es fa sobre-mostreig de 16 i seleccionar un prescaler de 3, ja que com el baud rate que volem és de 500kb/s, el rellotge ha d'anar a 8MHz (24MHz/3).

En el cas de la UART0, com el baud rate desitjat és de 115200 bp/s, se selecciona un prescaler de 13 ja que el rellotge ha d'anar a ~1.85Hz (24MHz/13). Com no és exacte, cal afegir la següent línia per fixar la part fraccionària:

```
UCA0MCTLW |= (0x25 << 8);
```

Seguint amb l'UART2, les següents dues línies serveixen per configurar els pins 3.2 i 3.3 com a GPIO (P3.2 = UART2RX i P3.3 = UART2TX). Les tres següents línies són necessàries per reactivar la línia de comunicacions sèrie, netejar el flag d'interrupcions de l'eUSCI RX i habilitar la interrupció de la eUSCI_A2 quan es tingui recepció.

L'última línia crida el mètode que es troba explicat en el punt 2 i configura el pin 3.0 com a GPIO de sortida.

David Fernàndez
Víctor Sort

4.2 Funcions de transmissió de dades

No posem aquí el codi del mètode TxPacket() ja que és bastant llarg. Es troba sencer en l'annex 6.2.

Aquest mètode rep la ID del mòdul desitjat, la mida dels paràmetres, la instrucció a enviar i els paràmetres adients. Inicialment es declaren totes les variables necessàries, es comprova que no s'estigui intentant escriure a una direcció prohibida (i en cas que així sigui, el mètode surt amb un missatge d'error) i es posa el pin 3.0 a 1 (és a dir, el Direction Port es posa en mode transmetre). Entre les declaracions, la més important és la de TxBuffer, un array de 32 posicions on a les dues primeres s'escriu 0xFF per indicar l'inici de la trama i, a continuació, les tres primeres dades rebudes per paràmetre (no afegim els paràmetres ja que és un array i s'ha de fer amb cura a continuació).

A continuació es troben 3 bucles. El primer segueix omplint la trama que es vol enviar, omplint el TxBuffer amb els paràmetres passats per paràmetre. La segona fa el càlcul del checksum, un valor per controlar que la trama enviada no hagi sigut modificada en el procés de l'enviament i, un cop acaba, l'escriu també a l'array TxBuffer. El tercer serveix per enviar la trama al mòdul del robot, usant la funció TxUACx (on BUFFER és un define diferent segons si treballem amb l'emulador (UCA0TXBUF) o el robot real (UCA2TXBUF)):

```
void TxUACx(byte bTxdData){
      while (!TXD_READY); // Espera a que estigui preparat el buffer de
transmissió
      BUFFER = bTxdData; //escrivim el byte a ser enviat
}
```

Un cop s'ha transmès l'últim byte, es torna a canviar el Direction Port en mode recepció, perquè el mòdul a qui se li ha enviat la trama enviï la resposta. El mètode acaba retornant la mida del Return Packet.

La resposta és gestionada al mètode RxPacket, que és molt extens i està extremadament comentat a l'annex 6.2.

David Fernàndez
Víctor Sort

4.3 Llibreria de funcions de moviment

No posem aquí el codi del main ja que és bastant llarg. Es troba sencer en l'annex 6.3.

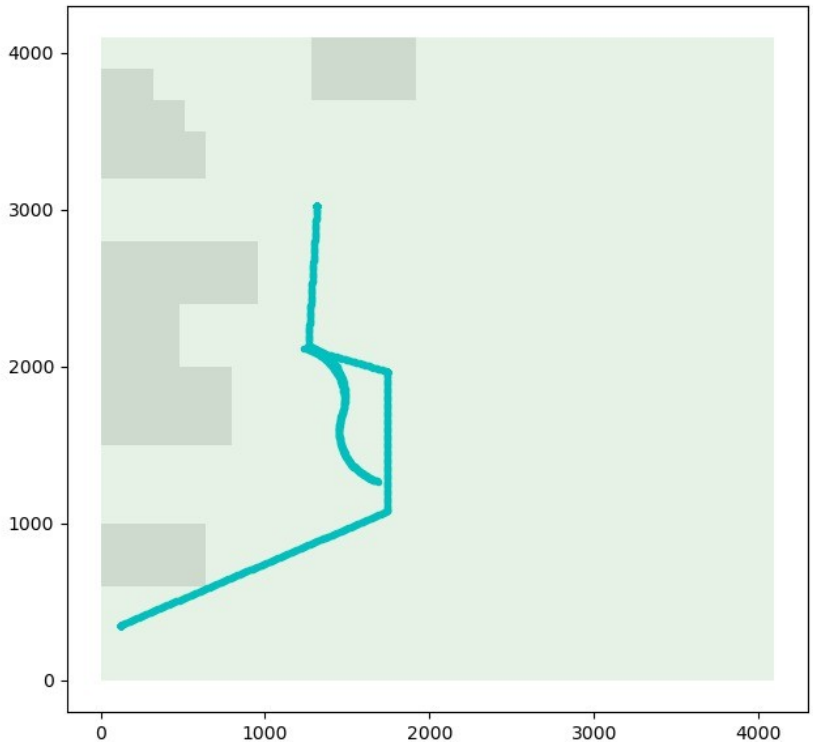
Primerament, inicialitzem els timers i fem totes les configuracions necessàries. Un cop això, el que farem serà repetir el següent codi tantes vegades com funcions de la llibreria per controlar el robot haguem programat, variant el temps depenent de la necessitat:

```
while(!TimeOutA0(1000000)); //10s
pivotar_antihorario(velocitat)
Reset_TimeoutA0();
```

S'ha provat el codi al laboratori modificant els temps entre ordres a 1s (com s'entrega el projecte) i funciona perfectament. Es mou tal i com s'espera, encén i apaga els LEDs correctament i llegix bé els sensors de distància.

A continuació es mostra una imatge del moviment del robot fet a l'emulador, amb els intervals de temps presents al codi de l'annex. Si es compara amb el codi de prova del main, s'observa que és corresponent.

Podem concloure doncs amb la satisfactòria validació de la llibreria de funcions.



David Fernàndez
Víctor Sort

5. CONCLUSIONS

Per començar, volem dir que estem molt satisfets amb els nostres codis finals ja que creiem que satisfan l'objectiu de la pràctica en tots els casos possibles. Hem aconseguit configurar bé les UARTs del robot i de l'emulador i hem creat una prou completa llibreria per gestionar tots els moviments del robot i la lectura dels seus sensors.

Creiem que les classes de teoria ens han ajudat molt en aquesta pràctica, al repassar tan detalladament com fer la correcta configuració de la UART.

La realització d'aquesta pràctica ens ha ajudat a entendre conceptes explicats a teoria, per exemple, referents als mòduls Dynamixel AX-12.

Respecte el temps invertit en la realització de la pràctica, creiem que ha sigut similar a les dues anteriors i hem notat satisfets que el treball que ja hem fet anteriorment, com la configuració de ports com a GPIOs, timers o interrupcions l'hem realitzat de manera molt més ràpida en aquesta pràctica.

Finalment, creiem que hem treballat molt bé en equip, que ens hem repartit les tasques equitativament, en hem ajudat mútuament i esforçat per aprendre i optimitzar al màxim, dins les nostres capacitats, el codi d'aquesta pràctica. En resum, i en la nostra opinió, hem fet una molt bona pràctica.

David Fernàndez
Víctor Sort

6. ANNEXOS

6.1 Programa Comentat

6.1.1 Llibreria_robot.h

```
#ifndef LLIBRERIA_ROBOT_H_
#define LLIBRERIA_ROBOT_H_
#include <msp432p401r.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "lib_PAE.h"
typedef uint8_t byte;
#define INDEX_LENGTH_STATUS_PACKET  3
#define INDEX_ERROR_STATUS_PACKET  4
#define WRITE_INSTRUCTION 3
#define READ_INSTRUCTION 2

//Defines per funcionar a l'emulador
/* #define initUART init_UART0
#define ID_MOTOR_ESQUERRE 1
#define ID_MOTOR_DRET 2
#define ID_SENSOR 3
#define TXD_READY (UCA0IFG & UCTXIFG)
#define BUSY UCA0STATW & UCBUSY
#define BUFFER UCA0TXBUF */

//Defines per funcionar amb el robot
#define initUART init_UART2
#define ID_MOTOR_ESQUERRE 3
#define ID_MOTOR_DRET 2
#define ID_SENSOR 100
#define TXD_READY (UCA2IFG & UCTXIFG)
#define BUSY UCA2STATW & UCBUSY
#define BUFFER UCA2TXBUF
```

```
typedef struct{
    byte StatusPacket[16];  //el status packet rebut
    byte TimeOut;           //byte que indica si hi ha hagut timeout rebent el
status packet
    byte StatusCheckSum;    //byte que indica si el checksum rebut coincideix
amb el checksum del status packet rebut
} RxReturn;


typedef struct{
    byte Left;              //la lectura del sensor esquerre
    byte Center;            //la lectura del sensor central
    byte Right;             //la lectura del sensor dret
} LecturaSensores;


void init_interrupciones(void);
void init_timers(void);
void config_Direction_Port(void);
void Sentit_Dades_Rx(void);
void Sentit_Dades_Tx(void);
void TxUACx(byte bTxdData);
void init_UART2(void);
void init_UART0(void);
byte TxPacket(byte bID, byte bParameterLength, byte bInstruction, byte
Parametros[16]);
void Activa_TimerA1_TimeOut(void);
void Desactiva_TimerA1_TimeOut(void);
void Reset_TimeoutA1(void);
byte TimeOutA1(int TimeOut);
RxReturn RxPacket(void);
void TA1_0_IRQHandler(void);
void EUSCIA2_IRQHandler(void);
void EUSCIA0_IRQHandler(void);
RxReturn enviarInstruccion(byte bID, byte bParameterLength, byte bInstruction,
byte Parametros[16]);
void encenderLeds();
```

```
void encenderLed(byte bID);

void apagarLeds();

void apagarLed(byte bID);

void enableTorque(void);

void configurar_CONT_MODE(void);

void mover_rueda(byte bID, uint16_t speed, byte direction);

void avanzar(uint16_t speed);

void retroceder(uint16_t speed);

void parar();

void avanzar_con_giro(uint16_t speed_left, uint16_t speed_right);

void retroceder_con_giro(uint16_t speed_left, uint16_t speed_right);

void pivotar_sobre_si_mismo_horario(uint16_t speed);

void pivotar_sobre_si_mismo_antihorario(uint16_t speed);

void pivotar_horario(uint16_t speed);

void pivotar_antihorario(uint16_t speed);

LecturaSensores leer_sensores(void);

#endif /* LLIBRERIA_ROBOT_H_ */
```

6.1.2 Llibreria_robot.c

```
#include "llibreria_robot.h"

volatile int cont_time_out; //variable que ens servira per comptar el nombre de
cicles temps

volatile byte Byte_Recibido = 0;

volatile byte DatoLeido_UART = 0;


void init_interrupciones(void){

    // Configuracion al estilo MSP430 "clasico":

    // --> Enable Port 4 interrupt on the NVIC.

   // Segun el Datasheet (Tabla "6-39. NVIC Interrupts", apartado "6.7.2 Device-
Level User Interrupts"),

    // la interrupcion del puerto 1 es la User ISR numero 35.

    // Segun el Technical Reference Manual, apartado "2.4.3 NVIC Registers",

  // hay 2 registros de habilitacion ISER0 y ISER1, cada uno para 32
interrupciones (0..31, y 32..63, resp.),

   // accesibles mediante la estructura NVIC->ISER[x], con x = 0 o x = 1.
```

```
    // Asimismo, hay 2 registros para deshabilitarlas: ICERx, y dos registros
para limpiarlas: ICPRx.

    //Int. TA1 de CCTL0, corresponde al bit 10 del primer registro ISER0

    NVIC->ICPR[0] |= 1 << TA1_0_IRQn; //Primero, me aseguro de que no quede
ninguna interrupcion residual pendiente para este puerto,

    NVIC->ISER[0] |= 1 << TA1_0_IRQn; //y habilito las interrupciones del puerto

    //Int. EUSCIA0, corresponde al bit 16 del primer registro ISER0

    NVIC->ICPR[0] |= 1 << EUSCIA0_IRQn; //Primero, me aseguro de que no quede
ninguna interrupcion residual pendiente para este puerto,

    NVIC->ISER[0] |= 1 << EUSCIA0_IRQn; //y habilito las interrupciones del
puerto

    //Int. EUSCIA0, corresponde al bit 18 del primer registro ISER0

    NVIC->ICPR[0] |= 1 << EUSCIA2_IRQn; //Primero, me aseguro de que no quede
ninguna interrupcion residual pendiente para este puerto,

    NVIC->ISER[0] |= 1 << EUSCIA2_IRQn; //y habilito las interrupciones del
puerto

    //Les interrupcions al port 3, no calen, a més a més, no estan
inicialitzades a nivell de dispositiu

    //Int. PORT3, corresponde al bit 5 del segundo registro ISER1

    //NVIC->ICPR[1] |= 1 << (PORT3_IRQn & 31); //Primero, me aseguro de que no
quede ninguna interrupcion residual pendiente para este puerto,

    //NVIC->ISER[1] |= 1 << (PORT3_IRQn & 31); //y habilito las interrupciones
del puerto

}


void init_timers(void){

    //Timer A1, used for calculating possible timeout

    //Divider = 1; CLK source is SMCLK; clear the counter; MODE is stop

    TIMER_A1->CTL = TIMER_A_CTL_ID_1 | TIMER_A_CTL_SSEL_SMCLK | TIMER_A_CTL_CLR
| TIMER_A_CTL_MC_STOP;

    TIMER_A1->CCR[0] = 240 - 1;     //100kHz 10^-5 s (decenas de microsegundos)

   TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Activem les interrupcions del
timer A1

}
```

```
void config_Direction_Port(void){
     //Configurem el pin que es connectara a la senyal DIRECTION_PORT (P3.0)
     P3SEL0 &= ~BIT0; //P3.0 configurat com a GPIO
     P3SEL1 &= ~BIT0; //P3.0 configurat com a GPIO
     P3DIR |= BIT0;   //Es de sortida
 }

// funcions per canviar el sentit de les comunicacions
void Sentit_Dades_Rx(void){ //Configuracio del Half Duplex dels motors: Recepcio
    P3OUT &= ~BIT0; //El pin P3.0 (DIRECTION_PORT) el posem a 0 (Rx)
}

void Sentit_Dades_Tx(void){ //Configuracio del Half Duplex dels motors:
Transmissio
    P3OUT |= BIT0; //El pin P3.0 (DIRECTION_PORT) el posem a 1 (Tx)
}

// funcio TxUACx(byte): envia un byte de dades per la UART0 o UART2
void TxUACx(byte bTxdData){
    while (!TXD_READY); // Espera a que estigui preparat el buffer de transmissio
    BUFFER = bTxdData; //escrivim el byte a ser enviat
}

void init_UART2(void){
    UCA2CTLW0 |= UCSWRST;       //Fem un reset de la USCI, desactiva la USCI
    UCA2CTLW0 |= UCSSEL__SMCLK; //UCSYNC=0 mode asincron
                                //UCMODEx=0 seleccionem mode UART
                                //UCSPB=0 nomes 1 stop bit
                                //UC7BIT=0 8 bits de dades
                                //UCMSB=0 bit de menys pes primer
                                //UCPAR=x ja que no es fa servir bit de paritat
                                //UCPEN=0 sense bit de paritat
                                //Triem SMCLK (24MHz) com a font del clock BRCLK
    UCA2MCTLW = UCOS16;        //Necessitem sobre-mostreig => bit 0 = UCOS16 = 1
```

```
    UCA2BRW = 3;                //Prescaler de BRCLK fixat a 3. Com SMCLK va a
            24MHz, volem un baud rate de 500 kb/s i fem sobre-mostreig de 16
            //el rellotge de la UART ha de ser de 8MHz (24MHz/3).
            //Per tant no hi ha part fraccionaria del baud rate

    //Configurem els pins de la UART

    P3SEL0 |= (BIT2 | BIT3);        //I/O funcio: P3.3 = UART2TX, P3.2 = UART2RX

    P3SEL1 &= ~(BIT2 | BIT3 );

    UCA2CTLW0 &= ~UCSWRST;            //Reactivem la linia de comunicacions serie

    EUSCI_A2->IFG &= ~EUSCI_A_IFG_RXIFG;    //Clear eUSCI RX interrupt flag

    EUSCI_A2->IE |= EUSCI_A_IE_RXIE;        //Enable USCI_A2 RX interrupt, nomes
quan tinguem la recepcio

   config_Direction_Port();                //Configurem el pin que es connectarara
la senyal DIRECTION_PORT (P3.0)

}


void init_UART0(void){

    UCA0CTLW0 |= UCSWRST;       //Fem un reset de la USCI, desactiva la USCI

    UCA0CTLW0 |= UCSSEL__SMCLK; //UCSYNC=0 mode asincron

                                //UCMODEx=0 seleccionem mode UART

                                //UCSPB=0 nomes 1 stop bit

                                //UC7BIT=0 8 bits de dades

                                //UCMSB=0 bit de menys pes primer

                                //UCPAR=x ja que no es fa servir bit de paritat

                                //UCPEN=0 sense bit de paritat

                                //Triem SMCLK (24MHz) com a font del clock BRCLK

    UCA0MCTLW = UCOS16;        //Necessitem sobre-mostreig => bit 0 = UCOS16 = 1

    UCA0BRW = 13;          //Prescaler de BRCLK fixat a 13. Com SMCLK va a 24MHz,
                volem un baud rate de 115200 bps i fem sobre-mostreig de 16

                //el rellotge de la UART ha de ser de ~1.85Hz (24MHz/13).

    UCA0MCTLW |= (0x25 << 8);   //UCBRSx, part fractional del baud rate

    //Configurem els pins de la UART

    P1SEL0 |= (BIT2 | BIT3);        //I/O funcio: P3.3 = UART2TX, P3.2 = UART2RX

    P1SEL1 &= ~(BIT2 | BIT3 );

    UCA0CTLW0 &= ~UCSWRST;         //Reactivem la linia de comunicacions serie

    EUSCI_A0->IFG &= ~EUSCI_A_IFG_RXIFG;    //Clear eUSCI RX interrupt flag
```

```
    EUSCI_A0->IE |= EUSCI_A_IE_RXIE;        //Enable USCI_A2 RX interrupt, nomes
quan tinguem la recepcio
    config_Direction_Port();                //Configurem el pin que es connectara
a la senyal DIRECTION_PORT (P3.0)
}


//TxPacket() 3 parametres: ID del Dynamixel, Mida dels parametres, Instruction
byte. torna la mida del "Return packet"
byte TxPacket(byte bID, byte bParameterLength, byte bInstruction, byte
Parametros[16]){
    byte bCount, bCheckSum, bPacketLength;
    byte TxBuffer[32];
    Sentit_Dades_Tx();  //El pin P3.0 (DIRECTION_PORT) el posem a 1 (Transmetre)
    TxBuffer[0] = 0xff;     //Primers 2 bytes que indiquen inici de trama FF, FF.
    TxBuffer[1] = 0xff;
    TxBuffer[2] = bID;            //ID del modul al que volem enviar el missatge
    TxBuffer[3] = bParameterLength + 2;  //Length(Parameter,Instruction,Checksum)
    TxBuffer[4] = bInstruction;             //Instruccio que enviem al Modul
    char error[] = "adr. no permitida";
    if ((Parametros[0] < 6) && (bInstruction == 3)){//si se intenta escribir en
una direccion <= 0x05,
        //emitir mensaje de error de direccion prohibida:
        halLcdPrintLine(error, 8, INVERT_TEXT);
        //y salir de la funcion sin mas:
        return 0;
    }
    for (bCount = 0; bCount < bParameterLength; bCount++){ //Comencem a generar
la trama que hem d'enviar
        TxBuffer[bCount + 5] = Parametros[bCount];
    }
    bCheckSum = 0;
    bPacketLength = bParameterLength + 4 + 2;
    for (bCount = 2; bCount < bPacketLength - 1; bCount++){//Calcul del checksum
        bCheckSum += TxBuffer[bCount];
    }
    TxBuffer[bCount] = ~bCheckSum; //Escriu el Checksum (complement a 1)
```

```
    for (bCount = 0; bCount < bPacketLength; bCount++){ //Aquest bucle es el que
envia la trama al Modul Robot
        TxUACx(TxBuffer[bCount]);
    }
    while (BUSY); //Espera fins que s'ha transmes l'ultim byte
    Sentit_Dades_Rx(); //Posem la linia de dades en Rx perque el modul Dynamixel
envii resposta
    return (bPacketLength); //retornem la llargada de la trama
}

void Activa_TimerA1_TimeOut(void){
    TIMER_A1->CTL |= TIMER_A_CTL_MC__UP; //posem el timer A1 en mode UP,
activant-lo
}

void Desactiva_TimerA1_TimeOut(void){
    TIMER_A1->CTL &= ~TIMER_A_CTL_MC__UP; //posem el timer A1 en mode STOP,
desactivant-lo
}

void Reset_TimeoutA1(void){
    cont_time_out = 0; //fem reset al comptador que controla el timeout
}

byte TimeOutA1(int TimeOut){
    //retornem 1 si el nostre comptador de timeout supera el valor passat per
parametre, si no, retornem 0
    if (cont_time_out > TimeOut){
        return 1;
    }
    else{
        return 0;
    }
}
```

```
RxReturn RxPacket(void){
    //see if activate aqui el return packet
    RxReturn respuesta;
    byte bCount, bLength;
    byte bCheckSum = 0;
    byte Rx_time_out = 0;
    respuesta.TimeOut = Rx_time_out;
    respuesta.StatusCheckSum = bCheckSum;
    Activa_TimerA1_TimeOut(); //posem el timer A1 en mode UP
    //Llegim els 4 primers bytes del status packet
    for (bCount = 0; bCount < 4; bCount++){
        Reset_TimeoutA1(); //fem reset al timer
        Byte_Recibido = 0; //Posem que no s'ha rebut el byte
        while (!Byte_Recibido){ //Iterem si no s'ha rebut cap byte
            Rx_time_out = TimeOutA1(1000); // comprovem si tenim timeout
            if (Rx_time_out) //si tenim timeout, sortim del bucle
                break; //sale del while
        }
        if (Rx_time_out){ //si hem sortit del bucle per timeout
            respuesta.TimeOut = 1; //posem timeOut=1 a la resposta
            break; //sale del for si ha habido Timeout
        }
        //Si no, es que hem llegit un byte
        respuesta.StatusPacket[bCount] = DatoLeido_UART; //Posem a la repsosta
el byte rebut per la UART
    } //fin del for
    // Continua llegint la resta de bytes del Status Packet si no tenim timeout
    if (!Rx_time_out){
        //guardem el nombre de  bytes totals a partir de la posicio 3 del
StatusPacket,
        //que dona el nombre de bytes que queden despres dels 4 primers
        bLength = respuesta.StatusPacket[INDEX_LENGTH_STATUS_PACKET] + 4;
        for (bCount = 4; bCount < bLength; bCount++){
            Reset_TimeoutA1(); //fem reset al timer
            Byte_Recibido = 0; //Posem que no s'ha rebut el byte
```

```
            while (!Byte_Recibido){ //Iterem si no s'ha rebut cap byte
                Rx_time_out = TimeOutA1(1000); // comprovem si tenim timeout
                if (Rx_time_out) //si tenim timeout, sortim del bucle
                    break; //sale del while
            }
            if (Rx_time_out){ //si hem sortit del bucle per timeout
                respuesta.TimeOut = 1; //posem timeOut=1 a la resposta
                break; //sale del for si ha habido Timeout
            }
            respuesta.StatusPacket[bCount] = DatoLeido_UART; //Posem a la
repsosta el byte rebut per la UART
        } //fin del for
    }
    Desactiva_TimerA1_TimeOut(); //posem el timer A1 en mode STOP, ja que ja no
volem comprovar si tenim timeout
    //Comprovem el checkSum si no tenim timeout
    if (!Rx_time_out){   //sin no hem tingut timeout, comprovem el checksum
        for (bCount = 2; bCount < bLength - 1; bCount++){
            bCheckSum += respuesta.StatusPacket[bCount];
        }
        bCheckSum = ~bCheckSum;
        if (bCheckSum != respuesta.StatusPacket[bLength - 1]){
            respuesta.StatusCheckSum = 1;
        }
    }
    return respuesta;
}


void TA1_0_IRQHandler(void){
    TA1CCTL0 &= ~TIMER_A_CCTLN_CCIFG; //Clear interrupt flag
    cont_time_out++; //Sumem 1 al comptador per controlar el timeout
}
```

```
void EUSCIA2_IRQHandler(void){ //interrupcion de recepcion en la UART A2
    EUSCI_A2->IFG &= ~ EUSCI_A_IFG_RXIFG; // Clear interrupt
    UCA2IE &= ~UCRXIE; //Interrupciones desactivadas en RX
    DatoLeido_UART = UCA2RXBUF; //Llegim el byte enviat pel modul corresponent
    Byte_Recibido = 1; //Imformem que s'ha rebut un byte per que RXPacket ho
pugui llegir
    UCA2IE |= UCRXIE; //Interrupciones reactivadas en RX
}

void EUSCIA0_IRQHandler(void){ //interrupcion de recepcion en la UART A0
    EUSCI_A0->IFG &= ~ EUSCI_A_IFG_RXIFG; // Clear interrupt
    UCA0IE &= ~UCRXIE; //Interrupciones desactivadas en RX
    DatoLeido_UART = UCA0RXBUF; //Llegim el byte enviat pel modul corresponent
    Byte_Recibido = 1; //
    UCA0IE |= UCRXIE; //Interrupciones reactivadas en RX
}

RxReturn enviarInstruccion(byte bID, byte bParameterLength, byte bInstruction,
byte Parametros[16]){
    RxReturn ret;
    byte error;
    byte time_out;
    byte check_sum_error;
    do{
            TxPacket(bID, bParameterLength, bInstruction, Parametros);  //enviem
la instruccio
            ret = RxPacket();                                           //rebem
la resposta
            error = ret.StatusPacket[INDEX_ERROR_STATUS_PACKET];
//guardem el byte dels errors rebent resposta
            time_out = ret.TimeOut;
//guardem si hi ha hagut timeout rebent la resposta
            check_sum_error = ret.StatusCheckSum;
//guardem si hi ha hagut un error en la comprovacio del checksum
        }
```

```
        while (time_out != 0 || error!= 0 || check_sum_error != 0);     //mentre
hi hagi timeout, checksum error o error en la comunicacio, repetim el proces
    return ret; //retornem la resposta
}

void encenderLeds(){
    byte bID = ID_MOTOR_ESQUERRE;
    encenderLed(bID);
    bID = ID_MOTOR_DRET;
    encenderLed(bID);
}

void encenderLed(byte bID){
    byte bParameterLength = 2;
    byte bInstruction = WRITE_INSTRUCTION;  //WRITE-DATA (0x03)
    byte Parametros[16];
    Parametros[0] = 0x19;      //0x19: LED
    Parametros[1] = 0x01;
    //encenem el LED
    enviarInstruccion(bID, bParameterLength, bInstruction, Parametros);
//enviem la instruccio pel motor de ID indicat
}

void apagarLeds(){
    byte bID = ID_MOTOR_ESQUERRE;
    apagarLed(bID);
    bID = ID_MOTOR_DRET;
    apagarLed(bID);
}

void apagarLed(byte bID){
    byte bParameterLength = 2;
    byte bInstruction = WRITE_INSTRUCTION;  //WRITE-DATA (0x03)
    byte Parametros[16];
    Parametros[0] = 0x19;      //0x19: LED
```

```
    Parametros[1] = 0x00;
    //apaguem el LED
    enviarInstruccion(bID, bParameterLength, bInstruction, Parametros);
//enviem la instruccio pel motor de ID indicat
}

void enableTorque(void){
    byte bID = ID_MOTOR_ESQUERRE;
    byte bParameterLength = 3;
    byte bInstruction = WRITE_INSTRUCTION;  //WRITE-DATA (0x03)
    byte Parametros[16];
    Parametros[0] = 0x18;      //0x18: moment i 0x19: LED
    Parametros[1] = 0x01;
    Parametros[2] = 0x01;
    //encenem els LEDs i habilitem el moment dels dos motors
    enviarInstruccion(bID, bParameterLength, bInstruction, Parametros);
//enviem la instruccio pel motor esquerre
    bID = ID_MOTOR_DRET;
    enviarInstruccion(bID, bParameterLength, bInstruction, Parametros);  //envie
la instruccio pel motor dret
}

void configurar_CONT_MODE(void){
    byte bID = ID_MOTOR_ESQUERRE;
    byte bParameterLength = 5;
    byte bInstruction = WRITE_INSTRUCTION;  //WRITE-DATA (0x03)
    byte Parametros[16];
    Parametros[0] = 0x06;       //0x06, 0x07, 0x08 i 0x09 a 0, per configurar els
motors a mode continu
    Parametros[1] = 0x00;
    Parametros[2] = 0x00;
    Parametros[3] = 0x00;
    Parametros[4] = 0x00;
    enviarInstruccion(bID, bParameterLength, bInstruction, Parametros);
//enviem la instruccio pel motor esquerre
```

```
    bID = ID_MOTOR_DRET;

    enviarInstruccion(bID, bParameterLength, bInstruction, Parametros);
//enviem la instruccio pel motor dret

}


void mover_rueda(byte bID, uint16_t speed, byte direction){

    byte bParameterLength = 3;

    byte bInstruction = WRITE_INSTRUCTION;       //WRITE-DATA (0x03)

    byte Parametros[16];

    Parametros[0] = 0x20;                        //0x20 i 0x021 son les posicions
de la velocitat i la direccio

    Parametros[1] = (0xFF & speed);              //a la posicio 0x20 posem els 8
bits menys significactius de la velocitat

    Parametros[2] = (((direction << 2) & 0x04) | ((speed >> 8) & 0x03)); //a la
posicio 0x21 posem els 2 bits mes significactius de la velocitat i despres el bit
de la direccio

    enviarInstruccion(bID, bParameterLength, bInstruction, Parametros); //enviem
la instruccio

}


void avanzar(uint16_t speed){

    byte direction;

    byte bID = ID_MOTOR_ESQUERRE;

    direction = 0x00;   //direccio endavant

    mover_rueda(bID, speed, direction); //movem la roda a la velocitat passada
per parametre a la direccio indicada

    bID = ID_MOTOR_DRET;

    direction = 0x01;   //direccio endarrere

    mover_rueda(bID, speed, direction); //movem la roda a la velocitat passada
per parametre a la direccio indicada

}


void retroceder(uint16_t speed){

    byte direction;

    byte bID = ID_MOTOR_ESQUERRE;

    direction = 0x01;   //direccio endarrere
```

```
    mover_rueda(bID, speed, direction); //movem la roda a la velocitat passada
per parametre a la direccio indicada

    bID = ID_MOTOR_DRET;

    direction = 0x00;   //direccio endavant

    mover_rueda(bID, speed, direction); //movem la roda a la velocitat passada
per parametre a la direccio indicada

}


void parar(){

    avanzar(0);

}


void avanzar_con_giro(uint16_t speed_left, uint16_t speed_right){

    byte direction;

    byte bID = ID_MOTOR_ESQUERRE;

    direction = 0x00;   //direccio endavant

    mover_rueda(bID, speed_left, direction); //movem la roda a la velocitat
passada per parametre a la direccio indicada

    bID = ID_MOTOR_DRET;

    direction = 0x01;   //direccio endarrere

    mover_rueda(bID, speed_right, direction); //movem la roda a la velocitat
passada per parametre a la direccio indicada

}


void retroceder_con_giro(uint16_t speed_left, uint16_t speed_right){

    byte direction;

    byte bID = ID_MOTOR_ESQUERRE;

    direction = 0x01;   //direccio endarrere

    mover_rueda(bID, speed_left, direction); //movem la roda a la velocitat
passada per parametre a la direccio indicada

    bID = ID_MOTOR_DRET;

    direction = 0x00;   //direccio endavant

    mover_rueda(bID, speed_right, direction); //movem la roda a la velocitat
passada per parametre a la direccio indicada

}
```

```
void pivotar_sobre_si_mismo_horario(uint16_t speed){
    byte direction;
    byte bID = ID_MOTOR_ESQUERRE;
    direction = 0x00;   //direccio endavant
    mover_rueda(bID, speed, direction); //movem la roda a la velocitat passada
per parametre a la direccio indicada
    bID = ID_MOTOR_DRET;
    direction = 0x00;   //direccio endavant
    mover_rueda(bID, speed, direction); //movem la roda a la velocitat passada
per parametre a la direccio indicada
}

void pivotar_sobre_si_mismo_antihorario(uint16_t speed){
    byte direction;
    byte bID = ID_MOTOR_ESQUERRE;
    direction = 0x01;   //direccio endarrere
    mover_rueda(bID, speed, direction); //movem la roda a la velocitat passada
per parametre a la direccio indicada
    bID = ID_MOTOR_DRET;
    direction = 0x01;   //direccio endarrere
    mover_rueda(bID, speed, direction); //movem la roda a la velocitat passada
per parametre a la direccio indicada
}

void pivotar_horario(uint16_t speed){
    avanzar_con_giro(speed, 0); //avancem endavant, bloquejant la roda dreta
}

void pivotar_antihorario(uint16_t speed){
    avanzar_con_giro(0, speed); //avancem endavant, bloquejant la roda esquerra
}

LecturaSensores leer_sensores(void){
    RxReturn ret;
    LecturaSensores let;
```

```
    byte bID = ID_SENSOR;
    byte bParameterLength = 2;
    byte bInstruction = READ_INSTRUCTION;  //READ-DATA (0x02)
    byte Parametros[16];
    Parametros[0] = 0x1A;   //Llegim les posicions 0x01A, 0x01B i 0x01C
    Parametros[1] = 0x03;
    ret = enviarInstruccion(bID, bParameterLength, bInstruction,
Parametros); //enviem la instruccio i recuperem la resposta
    let.Left = ret.StatusPacket[5];     //guardem la lectura del sensor esquerre
    let.Center = ret.StatusPacket[6];   //guardem la lectura del sensor centrel
    let.Right = ret.StatusPacket[7];    //guardem la lectura del sensor dret
    return let;                         //retornem la estructura que conte les 3
lectures
}
```

6.1.3 Main.c

```
#include "llibreria_robot.h"
volatile int cont_time_out;
volatile uint16_t velocitat = 1023; //Maxima velocitat
volatile LecturaSensores let;

void init_timer_main(void){
    //Timer A1, used for calculating possible timeout
    //Divider = 1; CLK source is SMCLK; clear the counter; MODE is stop
    TIMER_A0->CTL = TIMER_A_CTL_ID_1 | TIMER_A_CTL_SSEL_SMCLK | TIMER_A_CTL_CLR
| TIMER_A_CTL_MC_STOP;
    TIMER_A0->CCR[0] = 240 - 1;     //100kHz 10^-5 s (decenas de microsegundos)
    TIMER_A0->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Activem les interrupcions del
timer A1
}

void init_interrupcion_main(){
    //Int. TA0 de CCTL0, corresponde al bit 10 del primer registro ISER0
    NVIC->ICPR[0] |= 1 << TA0_0_IRQn; //Primero, me aseguro de que no quede
ninguna interrupcion residual pendiente para este puerto,
    NVIC->ISER[0] |= 1 << TA0_0_IRQn; //y habilito las interrupciones del puerto
}

void Activa_TimerA0_TimeOut(void){
    TIMER_A0->CTL |= TIMER_A_CTL_MC__UP; //posem el timer A1 en mode UP,
activant-lo
}
```

```
void Reset_TimeoutA0(void){
    cont_time_out = 0; //fem reset al comptador que controla el timeout
}

byte TimeOutA0(int TimeOut){
    //retornem 1 si el nostre comptador de timeout supera el valor passat per
parametre, si no, retornem 0
    if (cont_time_out > TimeOut){
        return 1;
    }
    else{
        return 0;
    }
}


void main(void){

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
    init_ucs_24MHz();
    init_timers();          //configurem els Timers
    init_timer_main();
    initUART();             //Inicialitzem la eUSCI Ax com a UART
    init_interrupciones(); //Configurar i activar les interrupcions (TA1 i
eUSCIAx)
    init_interrupcion_main();
    __enable_interrupts(); //Activem les interrupcions a nivell global
    enableTorque();
    configurar_CONT_MODE();
    let = leer_sensores();
    Activa_TimerA0_TimeOut();
    pivotar_horario(velocitat);
    Reset_TimeoutA0();
    while(!TimeOutA0(11000)); //0.11s
    avanzar(velocitat);
    Reset_TimeoutA0();
    while(!TimeOutA0(1000000)); //10s
    pivotar_antihorario(velocitat);
    Reset_TimeoutA0();
    while(!TimeOutA0(11000)); //0.4s
    avanzar(velocitat);
    Reset_TimeoutA0();
    while(!TimeOutA0(500000)); //5s
    Reset_TimeoutA0();
    pivotar_sobre_si_mismo_horario(velocitat);
    Reset_TimeoutA0();
    while(!TimeOutA0(100000)); //1s
```

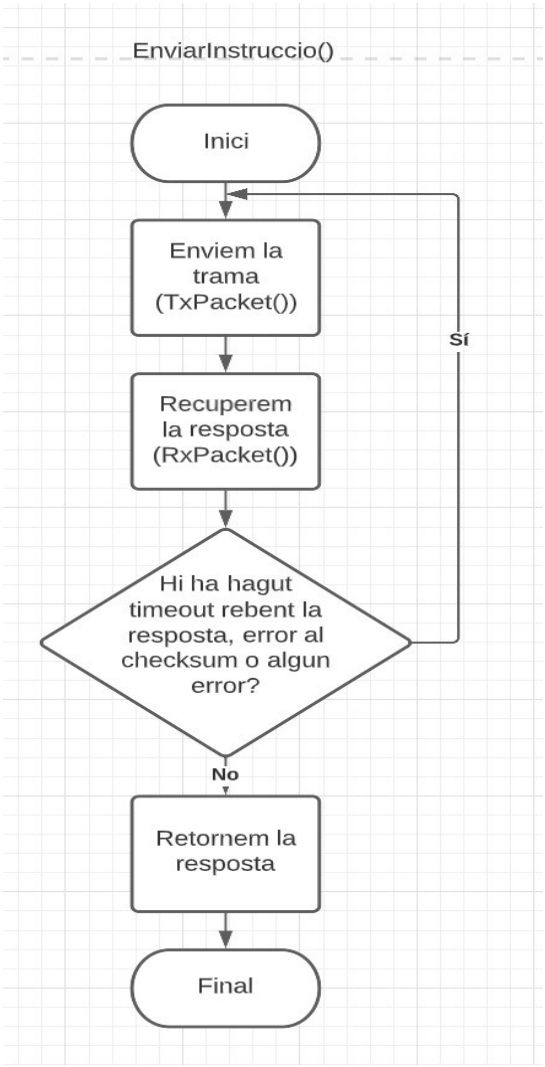
```
    Reset_TimeoutA0();
    retroceder(velocitat);
    Reset_TimeoutA0();
    while(!TimeOutA0(300000)); //3s
    Reset_TimeoutA0();
    avanzar_con_giro(velocitat, 1020);
    Reset_TimeoutA0();
    while(!TimeOutA0(300000)); //3s
    avanzar_con_giro(1020, velocitat);
    Reset_TimeoutA0();
    while(!TimeOutA0(300000)); //3s
    retroceder_con_giro(1020, velocitat);
    Reset_TimeoutA0();
    while(!TimeOutA0(300000)); //3s
    retroceder_con_giro(velocitat, 1020);
    Reset_TimeoutA0();
    while(!TimeOutA0(300000)); //3s
    pivotar_sobre_si_mismo_antihorario(velocitat);
    Reset_TimeoutA0();
    while(!TimeOutA0(100000)); //1s
    avanzar(velocitat);
    Reset_TimeoutA0();
    while(!TimeOutA0(500000)); //5s
    parar();
    apagarLeds();
    while (1);
}

void TA0_0_IRQHandler(void){
    TA0CCTL0 &= ~TIMER_A_CCTLN_CCIFG; //Clear interrupt flag
    cont_time_out++; //Sumem 1 al comptador per controlar el timeout
}
```

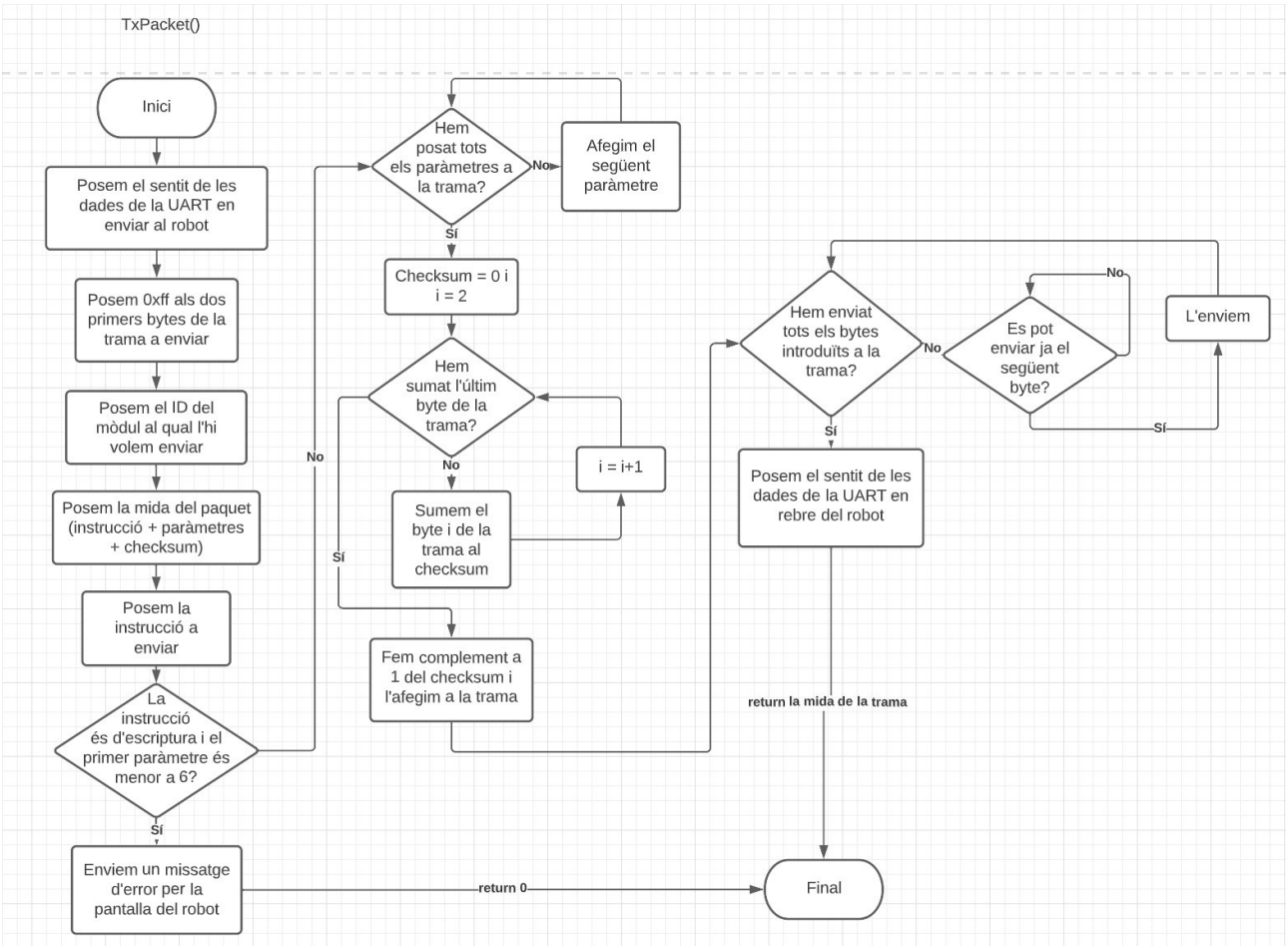
David Fernàndez
Víctor Sort

6.2 Diagrames de Flux

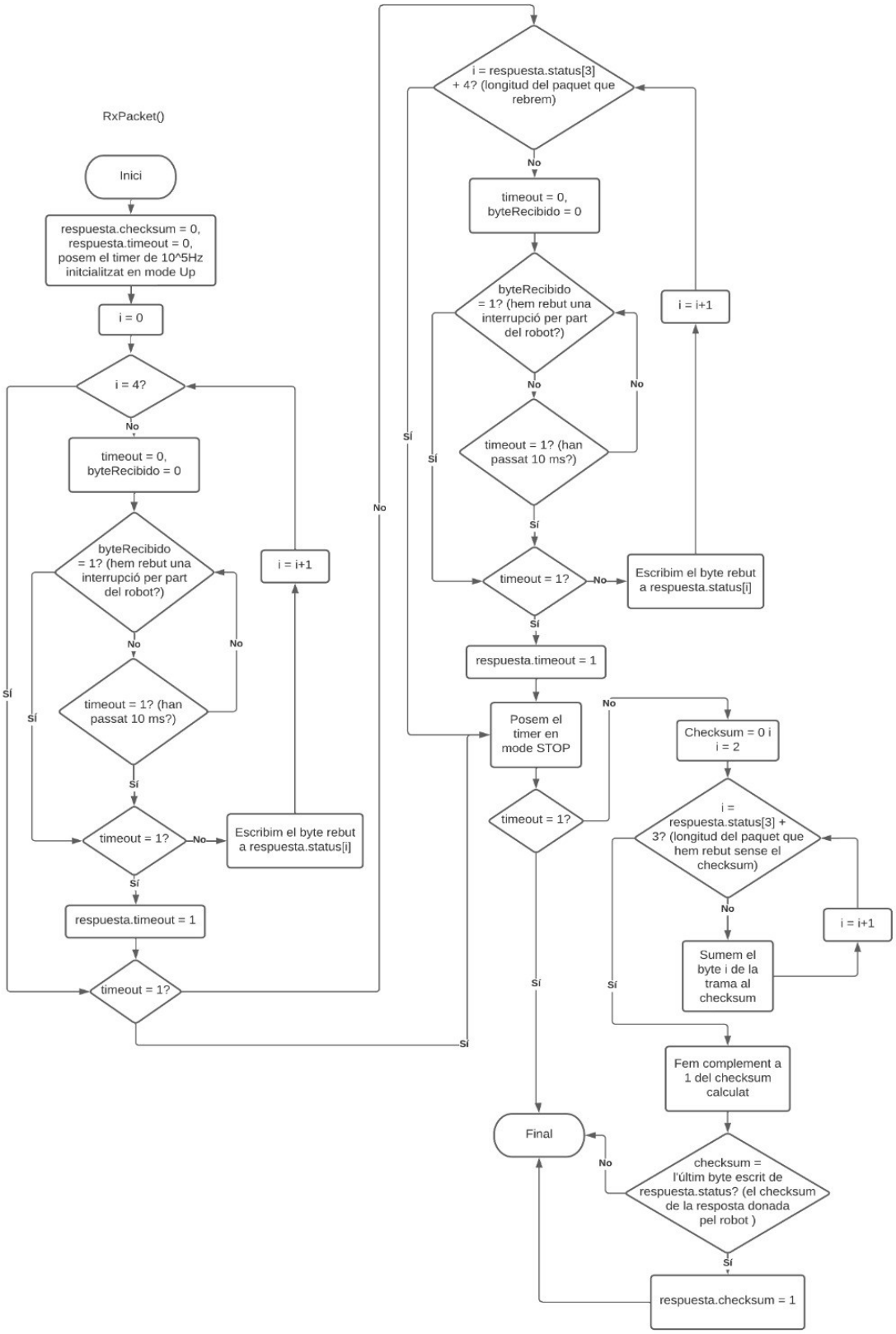
6.2.1 EnviarInstruccio()



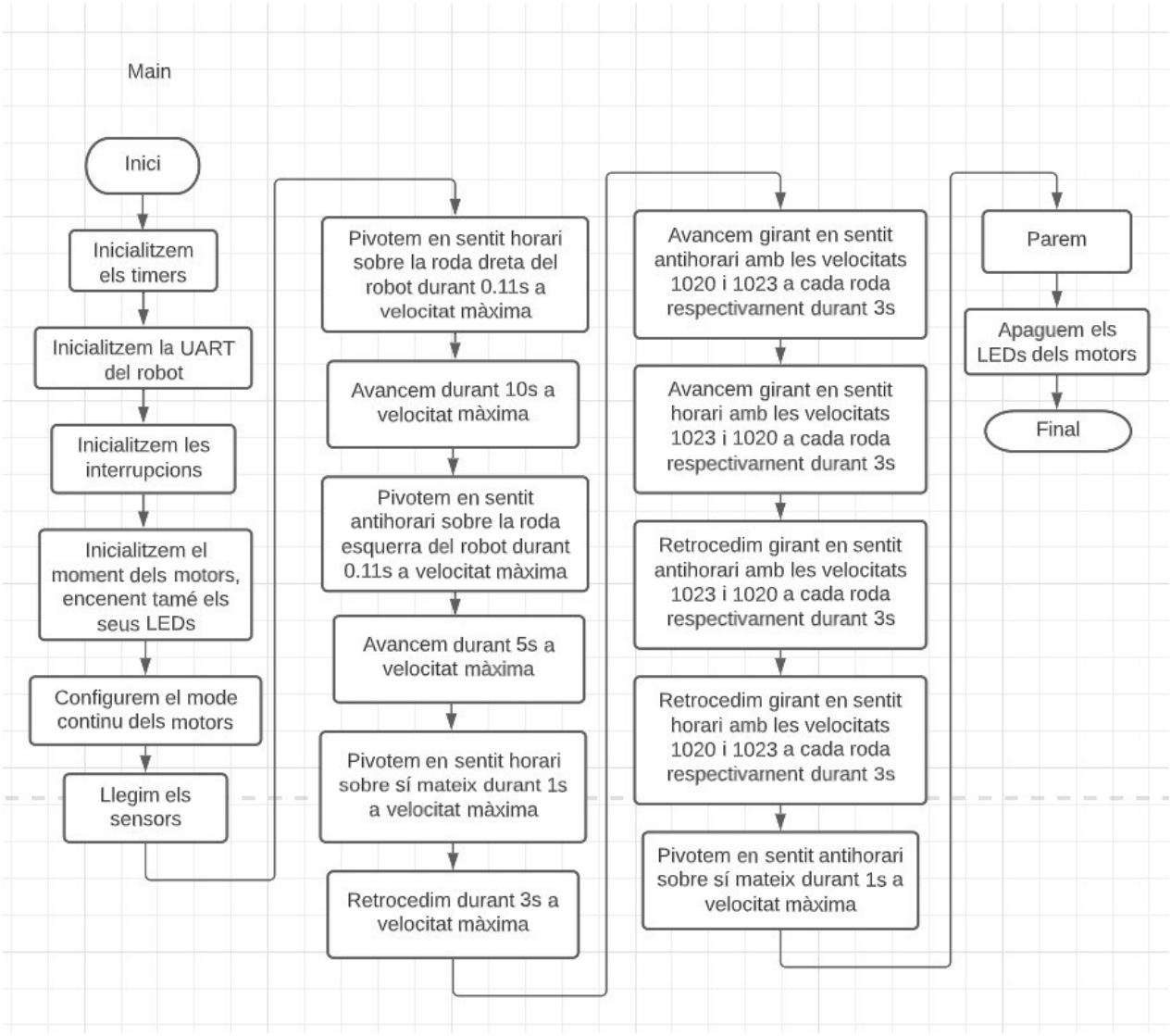
6.2.2 TxPacket()



6.2.3 RxPacket()



6.2.4 Main()


Main
Inici
Inicialitzem els timers
Inicialitzem la UART del robot
Inicialitzem les interrupcions
Inicialitzem el moment dels motors, encenent també els seus LEDs
Configurem el mode continu dels motors
Llegim els sensors
Pivotem en sentit horari sobre la roda dreta del robot durant 0.11s a velocitat màxima
Avancem durant 10s a velocitat màxima
Pivotem en sentit antihorari sobre la roda esquerra del robot durant 0.11s a velocitat màxima
Avancem durant 5s a velocitat màxima
Pivotem en sentit horari sobre sí mateix durant 1s a velocitat màxima
Retrocedim durant 3s a velocitat màxima
Avancem girant en sentit antihorari amb les velocitats 1020 i 1023 a cada roda respectivament durant 3s
Avancem girant en sentit horari amb les velocitats 1023 i 1020 a cada roda respectivament durant 3s
Retrocedim girant en sentit antihorari amb les velocitats 1023 i 1020 a cada roda respectivament durant 3s
Retrocedim girant en sentit horari amb les velocitats 1020 i 1023 a cada roda respectivament durant 3s
Pivotem en sentit antihorari sobre sí mateix durant 1s a velocitat màxima
Parem
Apaguem els LEDs dels motors
Final