UNIVERSITAT DE BARCELONA

Pràctica 4. Creació de les funcions bàsiques de comunicació i d'una llibreria de funcions per controlar el robot

Noah Márquez Vara
Alejandro Guzman Requena

10 de Maig 2022

ÍNDEX

1 INTRODUCCIÓ

1 Què es vol fer a la pràctica

En aquesta pràctica l'objectiu principal és conèixer com es realitzen les comunicacions bàsiques entre la placa del microprocessador i el robot (més concretament, els seus mòduls) i crear els recursos bàsics per fer possible aquesta comunicació. És a dir, haurem de definir la UART, la qual és un protocol de comunicació asíncron.

El segon objectiu principal de la pràctica es crear una llibreria d'usuari que contingui les funcions bàsiques de moviment a realitzar pel robot, per tant per poder fer aquesta llibreria primer hem d'haver configurat correctament la UART i saber quins paquets cal enviar per definir concretament els moviments bàsics del robot.

Per realitzar la comunicació placa-robot ho farem amb dos versions; la emulada i la real. La versió emulada no utilitzarà motors Dynamixel reals ni sensors reals, sinó una versió emulada en el PC, utilitzant la UART del microcontrolador que va cap al PC. En la versió real farem ús dels motors/sensors *Dynamixel* reals que es troben en el robot, utilitzant una altra UART que és la que realment està connectada a aquests dispositius.

Tal i com podem comprovar en el codi del programa 2, no es pretén tenir dos codis diferents, sinó que fem ús de *defines* i variables per tal de canviar còmodament entre la versió emulada i la real de la comunicació entre microcontrolador i motors/sensors *Dynamixel*, que com ja hem comentat, en la emulada són ficticis.

A continuació detallem els recursos utilitzats, així com els ports definits per a la programació d'aquests recursos, distingint en tot moment si es tracta de la versió emulada o la real.

2 Recursos utilitzats

La comunicació entre el microcontrolador i els mòduls del robot (AX-12: Motors; AX-S1: Sensor) és fa mitjançant un protocol sèrie asíncron. El recurs del processador que s'encarrega d'això es la USCI (*Universal Serial Communication Interface*) programada com a UART (*Universal Asincronous Receiver Transmiter*). El microcontrolador que utilizarem per a la realització d'aquesta pràctica disposa de 4 USCIs que poden funcionar com a UART.

Segons la versió tractarem amb diferents ports per a la USCI, això és degut a que també en la emulada utilizarem el microcontrolador de la placa amb uns ports i en la real utilitzarem el microcontrolador del robot de laboratori.

2.1 Versió emulada

En la versió emulada els motors Dynamixel seran ficticis, amb IDs 1 i 2, el sensor (també fictici) tindrà l'ID 3. L'emulació del funcionament del robot s'executarà en una aplicació **python** que emularà el seu funcionament. Aquesta aplicació s'executarà en l'ordinador i es comunicarà mitjançant l'entrada USB d'aquest dispositiu amb el microcontrolador MSP432. Per part del microcontrolador, la comunicació és realitzarà amb el debugger integrat en la placa d'avaluació, i aquest farà la translació entre UART i USB. El protocol de comunicació emprat en la comunicació entre els motors i el sensor del programa i el microntrolador de la placa serà UART.

En aquesta versió farem servir *Backchannel UART*, el qual és un dels UART connectats a través del XDS110 A la placa com un port sèrie en el PC i és la UCA0. Està al port 1 (pins P1.2 = RX, P1.3 = TX).

2.2 Versió real

En la versió real els motors Dynamixel seran part del robot del laboratori, amb IDs 1 i 2, el sensor del robot tindrà l'ID 100. Hi ha la possibilitat que s'hagin pogut reprogramar els IDs i, com a conseqüència, que fossin valors diferents, però no ha sigut el nostre cas.

Per a poder activar la eUSCI en mode UART (protocol asíncron de comunicació) hem d'utilitzar la UCA2, que es troba en el port 3 (pins P3.2 = RX, P3.3 = TX) del robot de laboratori.

En el cas de la versió real, al ser *half-duplex* cal utilitzar el pin *Direction port*, el qual serveix per alternar el sentit de les dades en els busos de comunicació entre els mòduls i el microcontrolador. Aquest pin és el 3.0 A continuació indiquem una taula resum dels recursos utilitzats que estan connectats a certs pins del microprocessador:

Recurs	**Versió**	**Port.Pin**
UCA0RXD	Emulada	P1.2
UCA0TXD	Emulada	P1.3
UCA2RXD	Real	P3.2
UCA2TXD	Real	P3.3
Direction port	Real	P3.0

Taula 1.1: Connexió dels recursos al Microcontrolador

3 Configuració de recursos

Tal i com hem vist a teoria, el primer que hem de dur a terme quan fem un programa per a un microcontrolador és configurar els ports de tal manera que treballin de la forma esperada.

Abans de procedir a la configuració dels recursos hem de tenir clar quins dispositius són d'entrada i quins de sortida, per tal de programar els seus respectius pins de manera correcta. A continuació mostrem una taula amb la distinció de quins dispositius són d'entrada i quins de sortida dels utilitzats en la pràctica:

ENTRADA	**SORTIDA**
UCA0RXD	UCA0TXD
UCA2RXD	UCA2TXD
Direction port	Direction port

Taula 1.2: Dispositius d'entrada/sortida

Nota: El pin *direction port* el configurem a l'inici del programa com a entrada, però en qualsevol moment de la seva execució es pot configurar com a entrada o com a sortida, depenent de quina part del procés de comunicació es trobi en aquell moment el flux del programa.

Un cop feta la distinció de quins són els dispositius d'entrada i de sortida hem de procedir a la seva inicialització/configuració a nivell dels seus pins. Per tal de fer això hem d'escriure el contingut d'uns registres específics del microcontrolador. És molt important estudiar aquests registres abans de configurar res, ja que una mala configuració pot provocar un curt-circuit en algun pin.

Hem de tenir en compte també que si configurem un pin específic com a entrada/sortida digital hem d'especificar si és *entrada* o *sortida*. Un pin també pot treballar amb interrupcions, i això també ho hem hagut de configurar.

Totes aquestes configuracions inicials que fem les podrem anar canviant durant l'execució del programa segons ens interessi. És a dir, la configuració inicial no és restrictiva, la podem canviar durant el transcurs de l'execució del nostre programa.

*3.1 Funcio **init_UART(void)***

És la funció encarregada d'inicialitzar els pins per a poder fer possible la transmissió i recepció dels paquets d'instruccions els quals tenen un format específic que veurem més endavant quan analitzem les rutines *TxPacket* i *RxPacket*.

Com per a la recepció de bytes mitjançant la UART es fa per interrupcions, necessitem habilitarles, això es realitza mitjançant els ports ICPR i ISPR.

- ICPR: Manipulant aquest port ens assegurem que no quedi cap interrupció residual pendent per a aquest.
- ISPR: Amb aquest port habilitem la interrupció a nivell de dispositiu. En aquest cas la rutina de la interrupció, l'anomenada *void EUSCIA2_IRQHandler(void)*, desactiva la interr-upció en el pin de lectura, es llegeixen les dades en qüestió i a continuació torna a habilitar la interrupció.

*3.2 Funcions **void Sentit_Dades_Tx/Rx(void)***

En la UART en mode real, la transmissió de dades es troba en *Half Duplex*, això implica que hem d'indicar en tot moment si volem transmetre o rebre dades per evitar un conflicte de bus. Això ho aconseguirem controlant el senyal mitjançançant el *direction port* on ho posem a 1 si volem transmetre dades als mòduls i viceversa.

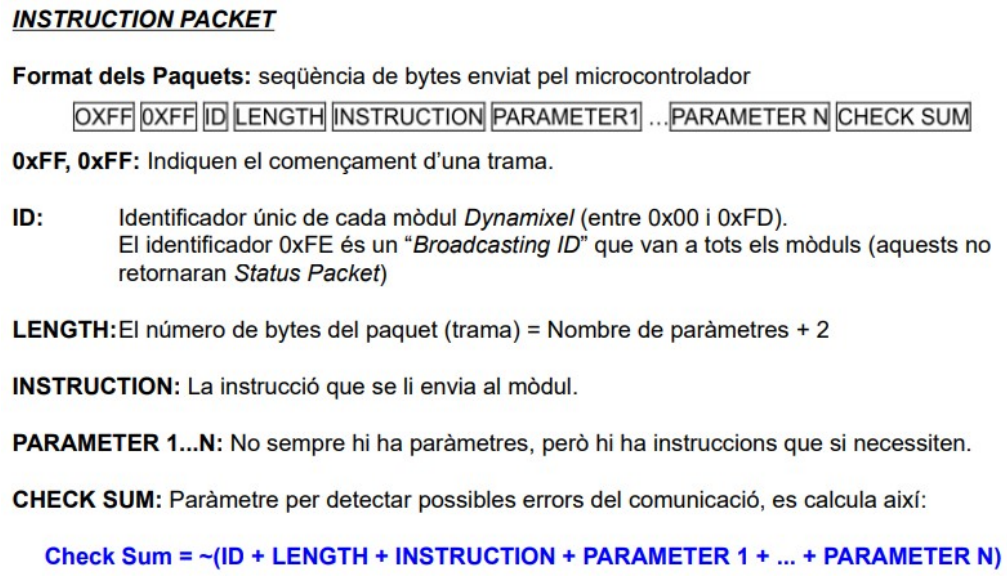
*3.3 Funció **void TxUAC2(byte bTxdData***

Aquesta funció fa que el processador esperi mentre el buffer de dades està ocupat.

*3.4 Format Instruction packet i funció **TxPacket***

L'instruction packet és el paquet que enviarem per transmetre les dades contingudes en aquest, el format de les dades és molt específic, tal i com es mostra a la figura 1.1

La funció ***byte TxPacket(byte bID, byte bParameterLenght, byte bInstruction, byte Parametres[16])***, s'encarregarà de comunicar a un motor/sensor en específic (mitjançant l'ID) quina instrucció volem que executi i la direcció de memòria on volem escriure els diferents paràmetres que coneixem a través de 'Parametros[16]'. La forma de crear el paquet d'instruccions és, a part d'assegurar-nos que no es pot escriure fora de les direccions permeses de memòria, creant un *buffer* per inserir les dades segons el format d'instruccions. Un cop el paquet ha estat enviat



INSTRUCTION PACKET

Format dels Paquets: seqüència de bytes enviat pel microcontrolador

OXFF	0XFF	ID	LENGTH	INSTRUCTION	PARAMETER1	...	PARAMETER N	CHECK SUM

0xFF, 0xFF: Indiquen el començament d'una trama.

ID: Identificador únic de cada mòdul *Dynamixel* (entre 0x00 i 0xFD).
El identificador 0xFE és un "*Broadcasting ID*" que van a tots els mòduls (aquests no retornaran *Status Packet*)

LENGTH: El número de bytes del paquet (trama) = Nombre de paràmetres + 2

INSTRUCTION: La instrucció que se li envia al mòdul.

PARAMETER 1...N: No sempre hi ha paràmetres, però hi ha instruccions que si necessiten.

CHECK SUM: Paràmetre per detectar possibles errors del comunicació, es calcula així:

Check Sum = ~(ID + LENGTH + INSTRUCTION + PARAMETER 1 + ... + PARAMETER N)

Figura 1.1: Format de l'*instruction packet*

correctament, habilitem el sentit de les dades mitjançant la funció ***void Sentit_Dades_Rx(void)*** per tal de poder rebre dades.

3.5 Format ***Status packet*** *i funció* ***RxPacket***

La funció *RxPacket* s'encarrega de llegir els bytes rebuts i detectar si hi ha algun error mitjançant funcions auxiliars:

- *checkSum*: Paràmetre que comprova si hi ha hagut errors en la comunicació i es calcula segons la fórmula blava de la figura 1.2.
- *time_out*: Comprova si el temps que passem per paràmetre es major o menor que el guardat en una variable que augmenta mitjançant la subrutina del *timer*. Retorna *true* si hi ha un excés de temps en llegir l'*Status Packet*. Com a conseqüència d'això es retornarà un error.

STATUS PACKET

Format dels Paquets: seqüència de bytes amb que respon el mòdul

OXFF	0XFF	ID	LENGTH	ERROR	PARAMETER1	PARAMETER2	...	PARAMETER N	CHECK SUM

0xFF, 0xFF: Indiquen el començament d'una trama.

ID: Identificador del mòdul.

LENGTH: El número de bytes del paquet.

ERROR:

Bit	Name	Details
Bit 7	0	-
Bit 6	Instruction Error	Set to 1 if an undefined instruction is sent or an action instruction is sent without a Reg_Write instruction.
Bit 5	Overload Error	Set to 1 if the specified maximum torque can't control the applied load.
Bit 4	Checksum Error	Set to 1 if the checksum of the instruction packet is incorrect.
Bit 3	Range Error	Set to 1 if the instruction sent is out of the defined range.
Bit 2	Overheating Error	Set to 1 if the internal temperature of the Dynamixel unit is above the operating temperature range as defined in the control table.
Bit 1	Angle Limit Error	Set as 1 if the Goal Position is set outside of the range between CW Angle Limit and CCW Angle Limit.
Bit 0	Input Voltage Error	Set to 1 if the voltage is out of the operating voltage range as defined in the control table.

PARAMETER 1...N: Si es necessiten.

CHECK SUM: Paràmetre per detectar possibles errors del comunicació, es calcula així:

Check Sum = ~(ID + LENGTH + ERROR + PARAMETER 1 + ... + PARAMETER N)

Figura 1.2: Format de l'*status packet*

Com haurem de fer servir les interrupcions en els *timer* **A0** i **A1**, les hem de configurar a nivell del controlador d'interrupcions del processador, anomenat **NVIC** (*Nested Vectored Interrupt Controller*). Això ho fem de la següent forma:

```
1
2 // Timer A0
3 NVIC->ICPR[0] |= BIT8;
4 NVIC->ISER[0] |= BIT8;
```

Com hem d'habilitar les corresponents interrupcions, hem d'anar a la taula 6-39 del *Datasheet* (pàg. 116 i 117) on comprovem en primer lloc que el port 4 es igual a 38 i correspon al *BIT6* del segon registre *ISER1* i *ICPR1*, i com que els registres que fem servir són de 32 bits, realitzem la operació lògica PORT4_IRQn & 31 = 6, i desplacem 1 sis cops a l'esquerra per obtenir el *BIT6*. Anàlogament ho realitzem per al port 5, que resulta ser el *BIT7* del segon registre *ICPR1* i *ISER1*.

Per a la configuració del NVIC (*Nested Vectored Interrupt Controller*) dels timers ho fem realitzant només la operació lògica *OR* per tal d'indicar que volem posar un 1 en el *BIT8* del primer registre *ISER0* i *ICPR0* (per al cas del *Timer* A0). De manera anàloga per a les següents configuracions dels dos *timers*, tal i com es pot veure en el codi mostrat anteriorment.

Nota 1: Amb la primera instrucció ens assegurem que no quedin interrupcions pendents als ports on estem configurant les interrupcions i amb la segona instrucció les habilitem.

Nota 2: La primera forma de configuració de l'*NVIC* pot semblar més enrevessada a simple vista però és més sofisticada ja que et neteja tot el registre i només activa les interrupcions que estàs configurant en aquell moment.

A continuació analitzarem els registres que hem hagut de configurar a l'inici del nostre programa per tal de configurar correctament els nostres recursos:

- **PxSEL0, PxSEL1**: Com volem que els pins d'aquests ports treballin com a GPIO (entrada/sortida digital), hem de configurar els seus respectius bits a 0 (fent ús d'operacions lògiques). Això ho fem amb les instruccions següents:

```
    // I/O function: P1.2 UART0RX, P1.3 UART0TX
    P1SEL0 |= BIT2 | BIT3;
    P1SEL1 &= ~(BIT2 | BIT3);

    // Port 3.0 com I/O digital
    P3SEL0 &= ~BIT0;
    P3SEL1 &= ~BIT0;

```

- **PxDIR**: Aquest registre indica quin dels pins d'un port (dels que hem configurat com a entrada/sortida digital) seran d'entrada (bit = 0) i quins seran de sortida (bit = 1).

  En el cas de mode real, indiquem que el pin 3.0 (corresponent al *direction port*) s'inicialitza com a sortida. Durant l'execució del programa aquest valor canvia segons en quin punt ens trobem de la comunicació.

```
    // Port 3.0 com a sortida (Data Direction: selector Tx/Rx)
    P3DIR |= BIT0;

```

- **PxOUT**: És un registre destinat als pins que haguem configurat com a sortida digital (en el nostre cas a l'inici quan configurem el pin *direction port* com a sortida). Indicarà si escrivim l'estat del pin a la sortida o no.

```
    //Inicialitzem Sentit Dades a 0 (Rx)
    P3OUT &= ~BIT0;

```

- **UCAxIE**: Com per a la recepció de bytes mitjançant la UART es fa per interrupcions, necessitem habilitar-les, per això acudim als ports que ja coneixem de les pràctiques anteriors; els ports **ICPR** i **ISPR**. Amb aquests dos registres habilitarem les interrupcions a nivell de dispositiu (perifèric). Les instruccions utilitzades són les següents:

```
    UCA0IE |= UCRXIE;
    UCA2IE |= UCRXIE;

```

  Amb la primera instrucció de cada port habilitem a nivell de dispositiu les interrupcions en els pins indicats.
  Amb la segona instrucció de cada port volem que les interrupcions saltin al flanc de pujada L->H en els pins indicats.

- **TAxCCTLn**: Per tal d'activar les fonts d'interrupcions dels dos *timers* ho hem de fer configurant el seu registre *TaxCCTLn* (*BIT CCIE*; on **x** és el numero del *timer*, i **n** és la interrupció que volem configurar. Les instruccions utilitzades són les següents:

```
    // Deshabilitem el mode "capture" (treballem en mode "compare") y treiem el flag
     d'interrupcio
    TA0CCTL0 &= ~(CAP+CCIFG);

    // Deshabilitem el mode "capture" (treballem en mode "compare") y treiem el flag
     d'interrupcio
    TA1CCTL0 &= ~(CAP+CCIFG);

```

- **TAxCTL**: Amb aquest registre configurem la forma de treballar del *timer* actual. Els bits MC s'utilitzen per a configurar els modes de funcionament (**Stop**: MC=00, **Up**: MC=01, **Continous**: MC=10, **Up/Down**: MC=11).

```
    // Usem ACLK (2^15 Hz), reiniciem el comptador
```

```
    TA0CTL |= TASSEL_1 + TACLR;

    // Usem ACLK (2^15 Hz), reiniciem el comptador
    TA1CTL |= TASSEL_1 + MC_1 + TACLR;

```

- **TAxCCR**: Aquest registre s'utilitza per aconseguir una freqüència determinada en el *timer*.

```
    // Registre TA0CCR0
    // 32768hz / 32 = 1024hz ~= 1ms
    TA0CCR0 = 33;

    // Registre TA1CCR0
    // 32768hz / 32768 ~= 1hz ~= 1000ms ~= 1sec
    TA1CCR0 = 33;

```

4 Funcions dels recursos: Llibreria d'usuari

Els recursos en aquesta pràctica s'utilitzen única i exclusivament per a configurar la comunicació entre diversos dispositius. Tot i així, el segon gran objectiu de la pràctica és dissenyar una llibreria d'usuari la qual estigui composada per funcions que permetin realitzar els moviments més bàsics amb el robot. D'aquesta manera comprovarem el correcte funcionament de la comunicació prèviament configurada.

La llibreria d'usuari creada té 13 funcions que utilitzen els motors *Dynamixel* i el sensor del robot per a realitzar moviments específics i fer lectures de l'entorn.

Les funcions creades són les següents:

- **wheelMode**: Les rodes giren de forma contínua.
- **moveWheel**: Es mou una roda amb o sense rotació i una certa velocitat.
- **stop**: Les dues rodes s'aturen.
- **turnLeft**: Gira a la esquerra; s'atura la roda esquerra i només gira la dreta.
- **turnLeftD**: Gira a la esquerra; es mouen les dues rodes a la dreta amb un cert angle.
- **turnOnItselfLeft**: Gira sobre si mateix a l'esquerra; les dues rodes es mouen a la dreta a la vegada.
- **turnRight**: Gira a la dreta; es para la roda dreta i només es mou l'esquerra.
- **turnRightD**: Gira a la dreta; les dues rodes es mouen a la dreta a la vegada.
- **turnOnItselfRight**: Gira a la dreta sobre si mateix, les dues rodes es mouen a la esquerra a la vegada.
- **forward**: Es mou cap a endavant.
- **backward**: Es mou cap a endarrere.
- **motorLed**: El led del motor s'encén.
- **readSensor**: Llegeix la informació del sensor.

5 Problemes

5.1 1ª sessió

La primera sessió la vàrem invertir sobretot en aprendre com configurar correctament la UART abans de començar a programar, tot i així, ens van sorgir bastants problemes.

- **Alternar entre modes de la UART**: Tal i com està estructurada la pràctica hem d'alternar entre dos modes de funcionament, el mode real i el mode simulació. En un principi no

sabíem com ho podíem programar de forma simple en el codi, però al final vàrem optar per definir (amb un *define*) una variable global i segons si aquesta està definida o no, el codi actuarà per la UART simulada o per la real.

5.2 2ª sessió

- **Problemes amb el mode simulat**:
  En aquesta sessió vam poder configurar el mode simulat, però alhora de provar-ho ho vam realitzar amb un portàtil *Macbook* i no funcionava, llavors no sabíem si era problema del dispositiu o del programa. Vam invertir una gran part de la sessió en buscar els problemes ajudant-nos del *debugger* que incorpora el *Code Composer Studio*, mirant els valors dels registre a cada pas de l'execució del problema, fins que vam trobar que el problema estava relacionat amb la tramesa del paquet d'instruccions i el càlcul del *checkSum*. Tot i així no ens va donar temps a solucionar-ho del tot.

5.3 3ª sessió

- **Problemes amb el return** En aquesta sessió vam continuar resolent el problema de la sessió anterior, i també ens vam donar compte que l'*status packet* no es retornava de forma correcta, llavors vam haver d'analitzar un altre cop amb ajuda del *debugger* on es trobava el problema. Després d'una gran part de la sessió vam trobar que el problema era ocasionat principalment per les condicions i (de nou) el càlcul del *checkSum*. Això feia que el *return* donés error a cada execució del codi.

5.4 4ª sessió

- **Solució de tots els problemes** En l'última sessió (que ens va ser molt profitosa ja que considerem que és una de les pràctiques més difícils fins el moment), vam provar el codi que portàvem revisat de casa i vam acabar de fer els últims ajustaments fins que vam aconseguir transmetre i rebre correctament els paquets mitjançant la UART.

  Tot i així, no ens va donar temps a provar la versió emulada al laboratori i ho vam deixar per provar a casa.

6 Conclusions

Aquesta última pràctica abans del projecte ens ha servit com a introducció i tecnificació en la comunicació entre dispositius mitjançant la UART. Hem continuat familiaritzant-nos amb el programari *Code Composer*, amb l'emulador ofert pel professorat i amb el Robot de pràctiques.

En la primera sessió vam repassar tots els coneixements donats a teoria sobre la comunicació entre dispositius, la tramesa i rebuda de paquets, ... I vam dedicar la sessió a intentar començar a programar el màxim possible, però va prevaldre el voler entendre correctament el que volíem i havíem de fer.

Durant les següents sessions ens vam haver d'enfrontar a diversos problemes que feien que aprenguéssim molt sobre la comunicació mitjançant la UART i sobre el Robot de pràctiques. Això és perquè fins ara aquesta ha estat la pràctica més complicada, tant a nivell de temps com de complexitat, ja que els conceptes teòrics semblaven ser molt clars però a l'hora de voler implementar-ho no ho eren tant.

En general ha resultat ser una pràctica molt útil on hem aprés a relacionar hardware i software d'una manera interessant i intuïtiva. A part, el temps atorgat en la realització tant de la pràctica com de l'informe ha estat bastant encertat (sobretot la sessió extra de la que vam poder gaudir).

2 PROGRAMA COMENTAT

En aquesta secció s'inclourà el programa realitzat al *Code Composer* per la realització de la present pràctica. El codi inclourà comentaris detallats del seu funcionament per tal de demostrar la compressió del que s'està fent en tot moment.

A continuació adjuntem el codi de la pràctica degudament comentat i dividit en diferents classes i *headers* per tal de poder mantenir i reutilitzar el codi més fàcilment en un futur:

1 main.c

```
#include "msp432p401r.h"
#include <stdio.h>
#include <stdint.h>
#include "lib_PAE2.h"

/* LLibreries ajuda */
#include <helper.h>
#include <interrupt.h>
#include <uart.h>
#include <control_robot.h>

/* Aquest define s'haurï¿½ de descomentar en cas de voler fer servir el simulador*/
//#define SIMULACIO "simulacio"

/* Inici timers */
void init_timers(void){
    // De moment deshabilitem el timer de 1 ms, ja que nomes s'activara al utilizar la funcio delay_t
    // Timer a 1ms
    TA0CTL |= TASSEL_1 + TACLR;                 // Usem ACLK (2^15 Hz), reiniciem el comptador
    TA0CCTL0 &= ~(CAP+CCIFG);                   // Deshabilitem el mode "capture" (treballem en mode "compare") y quitem el flag d'interrupcio
    TA0CCTL0 |= CCIE;                           // Habilitem les interrupcions (Per CCR0)
    TA0CCR0 = 33;                               // 32768hz / 32 = 1024hz ~= 1ms

    // Timer a 1ms (Para el timeout)
    TA1CTL |= TASSEL_1 + MC_1 + TACLR;          // Usem ACLK (2^15 Hz), reiniciem el comptador
    TA1CCTL0 &= ~(CAP+CCIFG);                   // Deshabilitem el mode "capture" (treballem en mode "compare") y quitem el flag d'interrupcio
    TA1CCTL0 |= CCIE;                           // Habilitem les interrupcions (Per CCR0)
    TA1CCR0 = 33;                               // 32768hz / 32768 ~= 1hz ~= 1000ms ~= 1 sec

}

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;       // stop watchdog timer

    // Cridem a les funcions d'inicialitzacio
    init_ucs_24MHz();
    init_timers();

    // Segons en quin mode de la UART haguem seleccionat, es crida a una funcio de configuracio de la UART o a una altra
    #ifdef SIMULACIO
        init_UART_SIMULACIO();
```

```
    #else
        init_UART();
    #endif

    // Cridem a la funcio d'inicialitzacio de les interrupcions
    init_interrupciones();

    // En aquest punt del programa ja esta la UART del mode seleccionat degudament configurada, i les interrupcions i timers tambe
    // aixi que podem cridar a diverses funcions de la llibreria d'usuari per tal de realitzar moviments especifics amb el robot

    // Com a primera comprovacio cridem a la seguent funcio que estableix un moviment continu de les rodes del robot
    wheelMode();

    // Una possible funcio de moviment de rodes a cridar es moure cap a la dreta
    turnRight(400);

    /* Altres crides possibles de la nostra llibreria */
    //forward(1000);
    //turnOnItselfLeft(400);
    //turnLeft(400);

    // Bucle infinit, per evitar que finalitzi l'execucio del programa
    while(true){
        ;
    }
}

// Interrupcions per al TIMER DE 1 MS
void TA0_0_IRQHandler(void) {

    TA0CCTL0 &= ~CCIE;        // Deshabilitem interrupcions mentres tractem aquesta

    timer_ms_count++;         // Augmentem el temps actual

    TA0CCTL0 &= ~CCIFG;       // Quitem el flag d'interrupcio
    TA0CCTL0 |= CCIE;         // Habilitem les interrupcions
}
```

2 uart.h

```
#ifndef UART_H_
#define UART_H_

//extern uint32_t current_time;
//extern byte received_data;
//extern byte read_data_UART;

void init_UART(void);
void wheelMode(void);
struct RxReturn RxPacket(void);
byte TxPacket(byte bID, byte bParameterLength, byte bInstruction, byte Parametros[16]);
void sentit_dades_Rx(void);
void sentit_dades_Tx(void);
bool time_out(int time);
void TxUAC2(byte bTxdData);

void EUSCIA2_IRQHandler(void);
void TA1_0_IRQHandler(void);
```

```
19
20 #endif /* UART_H_ */
```

3 uart.c

```
1  #include <msp432p401r.h>
2  #include <stdint.h>
3  #include <stdio.h>
4  #include <helper.h>
5
6  uint32_t current_time = 0; // Variable que se usa en el Timer0
7  byte received_data;
8  byte read_data_UART;
9  /* Per tal de fer servir el codi en l'emulador cal descomentar el define */
10 //#define SIMULACIO "simulating"
11
12 // Funcio time_out es True si el temps del comptador (current_time) supera el temps que
       hem passat com a parametre (time)
13 bool time_out(int time){
14     return (current_time>=time);
15 }
16
17
18 // Sentit: Rebre dades
19 void sentit_dades_Rx(void)
20 { //Configuracio del Half Duplex dels motors: Recepcio
21     P3OUT &= ~BIT0; //El pin P3.0 (DIRECTION_PORT) el posem a 0 (Rx)
22 }
23
24 // Sentit: Enviar les dades
25 void sentit_dades_Tx(void)
26 { //Configuracio del Half Duplex dels motors: Transmissio
27     P3OUT |= BIT0; //El pin P3.0 (DIRECTION_PORT) el posem a 1 (Tx)
28 }
29
30 // Configuracio de la UART en mode simulacio (aplicacio programada amb Phyton)
31 // La UART funciona en mode simulacio configurant la UCA0 situada en el port 1 (pin 1.0
       de direction port, 1.2 UCA0RXD, 1.3 UCA2TXD).
32 void init_UART_SIMULACIO(void){
33     // A continuacio configurem tres registres importants de la UCA0 amb les
       caracterï¿½stiques que volem que
34     // tingui la UART configurada
35     UCA0CTLW0 |= UCSWRST;          // Fem un reset de la USCI i es desactiva
36     UCA0CTLW0 |= UCSSEL__SMCLK;   // UCSYNC=0 mode asíncron
37                                   // UCMODEx=0 seleccionem mode UART
38                                   // UCSPB=0 nomes 1 stop bit
39                                   // UC7BIT=0 8 bits de dades
40                                   // UCMSB=0 bit de menys pes primer
41                                   // UCPAR=x no es fa servir bit de paritat
42                                   // UCPEN=0 sense bit de paritat
43
44     UCA0MCTLW = UCOS16;   // Triem SMCLK (24MHz) com a font del clock BRCLK
45     UCA0BRW = 3;          // Prescaler de BRCLK fixat a 13 (UCA0BR0 = (24MHz / 115200 kb/s
       ) / 16. Com SMCLK va a 24MHz
46                           // volem un baud rate de 115,200kb/s i fem sobre-mostreig
47                           // de 16
48
49                          // Necessitem sobre-mostreig -> bit 0 = UCOS16 = 1
50
51
52
```

```
53      // Configurem els pins de la UART
54      // Bacnkchannel UART -> UCA0 (versio emulada)
55      // P1.2 = RX, P1.3 = TX
56
57      P1SEL0 |= BIT2 | BIT3;        // I/O function: P1.2 UART0RX, P1.3 UART0TX
58      P1SEL1 &= ~(BIT2 | BIT3);
59
60      P3SEL0 &= ~BIT0;              // Port 3.0 com I/O digital
61      P3SEL1 &= ~BIT0;
62      P3DIR |= BIT0;                // Port 3.0 com a sortida (Data Direction: selector Tx/Rx
        )
63      P3OUT &= ~BIT0;               // Inicialitzem el port P3.0 a 0 (Rx)
64      UCA0CTLW0 &= ~UCSWRST;        // Reactivem la linia de comunicacions serie
65      UCA0IE |= UCRXIE;
66  }
67
68  // Configuracio de la UART en mode real, utilitzant els moduls Dynamixel i el sensor del
        robot de laboratori
69  // La UART funciona en mode real configurant la UCA2 situada en el port 3 (pin 3.0 de
        direction port, 3.2 UCA0RXD, 3.3 UCA2TXD).
70  void init_UART(void)
71  {
72      // A continuaciï¿½ configurem tres registres importants de la UCA2 amb les
        caracterï¿½stiques que volem que
73     // tingui la UART configurada
74      UCA2CTLW0 |= UCSWRST;             //Fem un reset de la USCI, desactiva la USCI
75      UCA2CTLW0 |= UCSSEL__SMCLK;       //UCSYNC=0 mode asincron
76                                        //UCMODEx=0 seleccionem mode UART
77                                        //UCSPB=0 nomes 1 stop bit
78                                        //UC7BIT=0 8 bits de dades
79                                        //UCMSB=0 bit de menys pes primer
80                                        //UCPAR=x ja que no es fa servir bit de paritat
81                                        //UCPEN=0 sense bit de paritat
82                                        //Triem SMCLK (16MHz) com a font del clock BRCLK
83
84      UCA2MCTLW = UCOS16;               // Necessitem sobre-mostreig => bit 0 = UCOS16 = 1
85      UCA2BRW = 3;                      //Prescaler de BRCLK fixat a 3. Com SMCLK va a24MHz,
86                                        //volem un baud rate de 500kb/s i fem sobre-mostreig
        de 16
87                                        //el rellotge de la UART ha de ser de 8MHz (24MHz/3).
88
89      //Configurem els pins de la UART
90      // P3.2 = RX, P3.3 = TX, P3.0 = Direction port
91      P3SEL0 |= BIT2 | BIT3; //I/O funcio: P3.3 = UART2TX, P3.2 = UART2RX
92      P3SEL1 &= ~ (BIT2 | BIT3);
93
94      //Configurem pin de seleccio del sentit de les dades Transmissio/Recepecio
95      P3SEL0 &= ~BIT0; //Port P3.0 com GPIO
96      P3SEL1 &= ~BIT0;
97      P3DIR |= BIT0; //Port P3.0 com sortida (Data Direction selector Tx/Rx)
98      P3OUT &= ~BIT0; //Inicialitzem Sentit Dades a 0 (Rx)
99      UCA2CTLW0 &= ~UCSWRST; //Reactivem la linia de comunicacions serie
100     UCA2IE |= UCRXIE; //Aixo nomes s'ha d'activar quan tinguem la rutina de recepcio
101 }
102
103 // Enviar un byte per la UART en mode simulacio
104 void TxUAC0(byte bTxdData)
105 {
106     while(!TXD0_READY); // Espera a que estigui preparat el buffer de transmissio
107     UCA0TXBUF = bTxdData;
108 }
109
```

```
110 // Enviar un byte por la UART en mode real
111 void TxUAC2(byte bTxdData)
112 {
113     while(!TXD2_READY); // Espera a que estigui preparat el buffer de transmissio
114     UCA2TXBUF = bTxdData;
115 }
116
117 //TxPacket() 3 parametres: ID del Dynamixel, Mida dels parametres, Instruction byte.
        torna la mida del "Return packet"
118 byte TxPacket(byte bID, byte bParameterLength, byte bInstruction, byte bParameters[16])
119 {
120     byte bCount,bCheckSum,bPacketLength;
121     byte TxBuffer[32];
122
123     if(bParameters[0] <= 5){ // Comprovacio per seguretat
124         return 0;
125     }
126
127     sentit_dades_Tx();  //El pin P3.0 (DIRECTION_PORT) el posem a 1 (Transmetre)
128
129     TxBuffer[0] = 0xff; //Primers 2 bytes que indiquen inici de trama FF, FF.
130     TxBuffer[1] = 0xff;
131
132     TxBuffer[2] = bID; //ID del modul al que volem enviar el missatge
133     TxBuffer[3] = bParameterLength+2; //Length(Parameter,Instruction,Checksum)
134     TxBuffer[4] = bInstruction; //Instruccio que enviem al Modul
135
136     for(bCount = 0; bCount < bParameterLength; bCount++) //Comencem a generar la trama
        que hem d'enviar
137     {
138         TxBuffer[bCount+5] = bParameters[bCount];
139     }
140
141     // Inicialitzem a 0 la variable del checksum que utilizarem per comprovar si s'ha
        enviat correctament les dades
142     bCheckSum = 0;
143     // Inicialitzem i assignem la longitud del paquet d'enviament com la longitud de
        parametres + 4 + 2
144     bPacketLength = bParameterLength+4+2;
145
146     for(bCount = 2; bCount < bPacketLength-1; bCount++) //Calcul del checksum
147     {
148         bCheckSum += TxBuffer[bCount];
149     }
150
151     TxBuffer[bCount] = ~bCheckSum; //Escrivim el Checksum (complement a 1)
152
153     for(bCount = 0; bCount < bPacketLength; bCount++) //Aquest bucle es el que envia la
        trama al Modul Robot
154     {
155         // Segons si hem seleccionat un mode o un altre, enviem el packet emprant una UAC
         o una altra
156         #ifdef SIMULACIO
157             TxUAC2(TxBuffer[bCount]);
158         #else
159             TxUAC0(TxBuffer[bCount]);
160         #endif
161     }
162
163     /* Segons estiguem en el mode simulador o no */
164     #ifdef SIMULACIO
165         while((UCA0STATW & UCBUSY));
```

```
166     #else
167         while( (UCA2STATW&UCBUSY)); //Espera fins que s'ha transmes l'ultim byte
168     #endif
169 
170     sentit_dades_Rx(); //Possem la linia de dades en Rx perque el modul Dynamixel envii
        una resposta
171 
172     return bPacketLength;
173 }
174 
175 
176 // Configuracio del paquet que el modul Dynamixel enviara al microcontrolador com a
        resposta de la transmissio de
177 // dades realitzada
178 struct RxReturn RxPacket(void)
179 {
180     // Inicialitzem una variable de tipus struct RxReturn definida com una estructura de
        dades
181     struct RxReturn respuesta;
182     // Inicialitzem
183     byte bCount, bLenght, bCheckSum = 0;
184     bool Rx_time_out = false;
185 
186     sentit_dades_Rx(); // Possem la linia de dades en Rx perque el modul Dynamixel envii
        una resposta
187 
188     //Activa_Interrupcion_TimerA1();
189 
190     // Llegim els quatre primers parametres
191     for(bCount = 0; bCount < 4; bCount++)
192     {
193         // Inicialitzem una variable que guardi el temps actual, aixo servira per
        gestionar els errors de tipus time_out
194         current_time=0;
195 
196         // Inicialitzem una variable que indiqui si s'ha rebut la dada o no
197         received_data=0; // No
198 
199         while (!received_data) // No s'ha rebut la dada?
200         {
201             Rx_time_out = time_out(20);  // Temps d'espera en milisegons
202             if (Rx_time_out)break;       // Si s'ha esgotat el temps d'espera per a rebre
        una dada sortim del bucle
203         }
204 
205         if (Rx_time_out)break; // Sortim del bucle si s'ha produit un time_out
206 
207         // Si tot ha anat correctament, llegim una dada del paquet rebut
208         respuesta.StatusPacket[bCount] = read_data_UART;
209 
210     }
211 
212 
213     if (!Rx_time_out){  // Si no n'hi ha time_out
214 
215         // Inicialitzem una variable que guarda la longitud del total que anem a llegir
216         bLenght = respuesta.StatusPacket[3]+4;
217 
218         // Llegim la resta de dades
219         for(bCount = 4; bCount < bLenght; bCount++)
220         {
221             // Inicialitzem una variable que guardi el temps actual, aixo servira per
```

```
    gestionar els errors de tipus time_out
222             current_time=0;
223
224             // Inicialitzem una variable que indiqui si s'ha rebut la dada o no
225             received_data=0; // No
226
227             while (!received_data) // No s'ha rebut la dada?
228             {
229                 Rx_time_out = time_out(20);  // Temps d'espera en milisegons
230                 if (Rx_time_out)break;       // Si s'ha esgotat el temps d'espera per a
    rebre una dada sortim del bucle
231             }
232
233             if (Rx_time_out)break; // Sortim del bucle si n'hi ha hagut un time_out
234
235             // Si tot ha anat correctament, llegim una dada del paquet rebut
236             respuesta.StatusPacket[bCount] = read_data_UART;
237         }
238     }
239
240     if(!Rx_time_out){    // Si no n'hi ha time_out
241
242         for(bCount = 2; bCount < bLenght-1; bCount++) // Realitzem el calcul del check
    sum
243         {
244             bCheckSum += respuesta.StatusPacket[bCount];
245         }
246
247         bCheckSum = ~bCheckSum; // Checksum a complement a 1
248         // Comparem el checksum rebut amb el calculat, si dona diferent, n'hi ha un error
249         if(respuesta.StatusPacket[bLenght-1] != bCheckSum){
250             // Utilitzem una variable inicialitzada dintre de la struct RxReturn per
    saber si s'ha produit un error
251             respuesta.error = true;
252         }
253     }
254
255     return respuesta;
256 }
257
258 // Interrupcio al rebre una dada
259 void EUSCIA0_IRQHandler(void)
260 {
261     // Interrupcio de recepcio en la UART en mode simulacio
262     UCA0IE &= ~UCRXIE; // Interrupcions desactivades en Rx
263
264     read_data_UART = UCA0RXBUF; // Assignem a una variable global el valor de la seguent
    dada a llegir
265     received_data = 1;
266     UCA0IE | UCRXIE; // Interrupcions reactivades en Rx
267 }
268
269 // Interrupcio al rebre una dada
270 void EUSCIA2_IRQHandler(void)
271 {
272     // Interrupcio de recepcio en la UART en mode real
273     UCA2IE &= ~UCRXIE; //Interrupcions desactivades en RX
274
275     read_data_UART = UCA2RXBUF; // Assignem a una variable global el valor de la seguent
    dada a llegir
276     received_data = 1; // La dada ha arribat
277
```

```
278     UCA2IE |= UCRXIE; //Interrupcions reactivades en RX
279 }
280
281 // Interrupcio pel TIMER DE 1 MS
282 void TA1_0_IRQHandler(void) {
283
284     TA1CCTL0 &= ~CCIE;      // Deshabilitem les interrupcions mentre tractem aquesta
285
286     current_time++;         // Augmentem el temps actual
287
288     TA1CCTL0 &= ~CCIFG;     // Quitem el flag d'interrupcio
289     TA1CCTL0 |= CCIE;       // Habilitem les interrupcions
290 }
```

4 interrupt.h

```
1 #ifndef INTERRUPT_H_
2 #define INTERRUPT_H_
3
4 void init_interrupciones(void);
5
6 #endif /* INTERRUPT_H_ */
```

5 interrupt.c

```
1  #include <msp432p401r.h>
2  #include <stdio.h>
3  #include <stdint.h>
4
5
6  /************************************************************************
7   * INICIALIZACIO DEL CONTROLADOR D'INTERRUPCIONS (NVIC).
8   *
9   * Sense dades d'entrada
10  *
11  * Sense dades de sortida
12  *
13  ************************************************************************/
14 void init_interrupciones(void)
15 {
16     // Configuracio al estil MSP430 "clasico":
17     // Enable Port 4 interrupt on the NVIC
18     // segun datasheet (Tabla "6-12. NVIC Interrupts", capitulo "6.6.2 Device-Level User
       Interrupts", p80-81 del documento SLAS826A-Datasheet),
19     // la interrupcio del port 4 es la User ISR numero 38.
20     // Segons el document SLAU356A-Technical Reference Manual, capitol "2.4.3 NVIC
       Registers"
21     // n'hi han 2 registres de'habilitacio ISER0 y ISER1, cadascun per a 32 interrupcions
        (0..31, y 32..63, resp.),
22     // accessibles mitjanÃ§ant l'estructura NVIC->ISER[x], amb x = 0 o x = 1.
23     // Aixi mateix, n'hi han 2 registres per deshabilitar-les: ICERx, y dos registres per
        a limpiar-les: ICPRx.
24
25     // Int. port 3 = 37 correspon al bit 5 del segon registre ISER1:
26     NVIC->ICPR[1] |= BIT5; // Primer, ens asegurem de que no quedi cap interrupcio
       residual en aquest port,
27     NVIC->ISER[1] |= BIT5; // i habilitem les interrupcions al port
28
29     //Int. port 4 = 38 correspon al bit 6 del segon registre ISERx:
```

```
30      NVIC->ICPR[1] |= BIT6; // Primer, ens asegurem de que no quedi cap interrupció
        residual en aquest port,
31      NVIC->ISER[1] |= BIT6; // i habilitem les interrupcions al port
32
33      //Int. port 5 = 39 correspon al bit 7 del segon registre ISERx:
34      NVIC->ICPR[1] |= BIT7; // Primer, ens asegurem de que no quedi cap interrupcio
        residual en aquest port,
35      NVIC->ISER[1] |= BIT7; // i habilitem les interrupcions al port
36
37      // Timer A0
38      NVIC->ICPR[0] |= BIT8;  // Primer, ens asegurem de que no quedi cap interrupcio
        residual en aquest port,
39      NVIC->ISER[0] |= BIT8;  // i habilitem les interrupcions al port
40
41      // Timer A1
42      NVIC->ICPR[0] |= BITA;  // Primer, ens asegurem de que no quedi cap interrupcio
        residual en aquest port,
43      NVIC->ISER[0] |= BITA;  // i habilitem les interrupcions al port
44
45      NVIC->ICPR[0] |= 0x40000;
46      NVIC->ISER[0] |= 0x40000;
47
48      __enable_interrupt(); // Habilitem les interrupcions a nivell global del
        microcontrolador.
49  }
```

6 helper.h

```
1   #ifndef HELPER_H_
2   #define HELPER_H_
3
4   typedef uint8_t byte;
5   typedef int bool;
6
7   extern uint32_t timer_ms_count;
8
9   #define false 0
10  #define true 1
11
12  #define LEFT 0
13  #define RIGHT 1
14
15  #define LEFT_WHEEL 3
16  #define RIGHT_WHEEL 2
17
18  // --- UART  ---
19
20  typedef struct RxReturn{
21      byte StatusPacket[16];
22      byte time_out;
23      bool error;
24  } RxReturn;
25
26  #define TXD0_READY (UCA0IFG & UCTXIFG)
27  #define TXD2_READY (UCA2IFG & UCTXIFG)
28
29  // --- Instruccions ---
30
31  #define WRITE_DATA 0x03
32  #define READ_DATA 0x02
33  #define BROADCASTING 0xFE
```

```

// --- Control Table ---

// EEPROM AREA - Direccions d'inici
#define CW_ANGLE_LIMIT_L 0x06
#define CW_ANGLE_LIMIT_H 0x07
#define CCW_ANGLE_LIMIT_L 0x08
#define CCW_ANGLE_LIMIT_H 0x09


// RAM AREA - Direccions d'inici
#define MOV_SPEED_L 0x20
#define MOV_SPEED_H 0x21
#define M_LED 0x19
#define LEFT_SENSOR 0x1A
#define CENTER_SENSOR 0x1B
#define RIGHT_SENSOR 0x1C

// Funcions per al temps (Opcional)
void delay_t(uint32_t temps_ms);

#endif /* HELPER_H_ */
```

7 helper.c

```
#include <msp432p401r.h>
#include <stdio.h>
#include <stdint.h>

uint32_t timer_ms_count;

/* Funcio que ens servira per girar amb un cert angle */
void delay_t(uint32_t temps_ms)
{
    timer_ms_count = 0;                     // Reiniciar Timer ms
    TA0CTL |= MC_1;                         // S'habilita el Timer
    do {

    } while (timer_ms_count <= temps_ms);   // Timer anira sumant +1 a la varibale
    timer_ms_count fins arribar al temps establert
    TA0CTL &= ~MC_1;                        // Es deshabilita el timer
}
```

8 control_robot.h

```
/*
 * control_robot.h
 */

#ifndef CONTROL_ROBOT_H_
#define CONTROL_ROBOT_H_


void wheelMode(void);
void moveWheel(byte ID, bool rotation, unsigned int speed);
void turnLeft(unsigned int speed);
void turnLeftD(unsigned int degree);
void turnOnItselfLeft(unsigned int speed);
void turnRight(unsigned int speed);
void turnRightD(unsigned int degree);
```

```
void turnOnItselfRight(unsigned int speed);
void motorLed(byte ID, bool status);
void stop(void);
void forward(unsigned int speed);
void backward(unsigned int speed);
void motorLed(byte ID, bool status);
int readSensor(byte ID, byte sensor);


#endif /* CONTROL_ROBOT_H_ */
```

9 control_robot.c

```
#include <msp432p401r.h>
#include <stdio.h>
#include <stdint.h>
#include <helper.h>
#include <uart.h>
#include <lib_PAE.h>

// Funcio per a que les rodes girin de forma continua
void wheelMode(void){

    byte bID = BROADCASTING; // No obtenim resposta en aquest mode
    byte bInstruction = WRITE_DATA;
    byte bParameterLength = 5;
    byte bParameters[16];

    // Comencem per la direccio 0x06 (6)
    bParameters[0] = CW_ANGLE_LIMIT_L;

    // Dades a escriure
    bParameters[1] = 0;  // CW_ANGLE_LIMIT_L = 0
    bParameters[2] = 0;  // CW_ANGLE_LIMIT_H = 0
    bParameters[3] = 0;  // CCW_ANGLE_LIMIT_L = 0
    bParameters[4] = 0;  // CCW_ANGLE_LIMIT_H = 0

    // Enviem les dades
    TxPacket(bID, bParameterLength, bInstruction, bParameters);
}

// Funcio per moure una roda amb o sense rotacio i una certa velocitat
void moveWheel(byte ID, bool rotation, unsigned int speed)
{
    // Variable de tipus struct RxReturn que guarda el paquet que retorna els moduls
    struct RxReturn returnPacket;
    // Variables que indiquen la velocitat
    byte speed_H,speed_L;
    speed_L = speed;

    if(speed<1024){ // Velocitat max. 1023

        if(rotation){ // Rotation == 1
            speed_H = (speed >> 8)+4;   // Moure a la dreta (CW)
        }else{
            speed_H = speed >> 8;       // Moure a la esquerra (CCW)
        }

        byte bInstruction = WRITE_DATA;
        byte bParameterLength = 3;
        byte bParameters[16];
```

```
49
50          // Comencem per la direccio 0x20 (32)
51          bParameters[0] = MOV_SPEED_L;
52
53          // Escribim la velocitat y la direccio
54          bParameters[1] = speed_L;
55          bParameters[2] = speed_H;
56
57          // Cridem a la funcio TxPacket per tal d'enviar el paquet d'instruccions
58          TxPacket(ID, bParameterLength, bInstruction, bParameters);
59          // Assignem a la variable creada anteriorment returnPacket el paquet de dades
        retornat
60          returnPacket = RxPacket();
61
62      }
63  }
64
65  // Funcio per aturar les dues rodes
66  void stop(void)
67  {
68      //Cridem a la funcio moveWheel per possar la rotacio i la velocitat a 0 d'ambdues
        rodes
69      moveWheel(RIGHT_WHEEL, 0, 0);
70      moveWheel(LEFT_WHEEL, 0, 0);
71  }
72
73  // Funcio per moure nomes la roda dreta i aixi girar a l'esquerra
74  void turnLeft(unsigned int speed){
75      if(speed < 1024){
76          moveWheel(RIGHT_WHEEL, RIGHT, speed);
77          moveWheel(LEFT_WHEEL, RIGHT, 0);
78      }
79  }
80
81  // Funcio per moure nomes la roda dreta i aixi girar a l'esquerra (amb un cert angle)
82  void turnLeftD(unsigned int degree){
83      moveWheel(RIGHT_WHEEL, RIGHT, 300);
84      moveWheel(LEFT_WHEEL, RIGHT, 0);
85      delay_t(degree*28.5);
86      stop();
87  }
88
89  // Funcio per girar les dues rodes a la dreta a la vegada i aixi que el robot doni una
        volta a si mateix per l'esquerra
90  void turnOnItselfLeft(unsigned int speed)
91  {
92      if(speed < 1024){
93          moveWheel(RIGHT_WHEEL, RIGHT, speed);
94          moveWheel(LEFT_WHEEL, RIGHT, speed);
95      }
96  }
97
98  // Funcio per moure nomes la roda esquerra i aixi girar a la dreta
99  void turnRight(unsigned int speed)
100 {
101     if(speed < 1024){
102         moveWheel(RIGHT_WHEEL, LEFT, 0);
103         moveWheel(LEFT_WHEEL, LEFT, speed);
104     }
105 }
106
107 // Funcio per moure nomes la roda esquerra i aixi girar a la dreta (amb un cert angle)
```

```
108 void turnRightD(unsigned int degree)
109 {
110     moveWheel(RIGHT_WHEEL, LEFT, 0);
111     moveWheel(LEFT_WHEEL, LEFT, 300);
112     delay_t(degree*28.5);
113     stop();
114 }
115
116 // Funcio per girar les dues rodes a l'esquerra a la vegada i aixï¿½ que el robot doni
    una volta a si mateix per la dreta
117 void turnOnItselfRight(unsigned int speed)
118 {
119     if(speed < 1024){
120         moveWheel(RIGHT_WHEEL, LEFT, speed);
121         moveWheel(LEFT_WHEEL, LEFT, speed);
122     }
123 }
124
125 // Funcio que mou el robot cap a endavant
126 void forward(unsigned int speed)
127 {
128     if(speed < 1024){
129         moveWheel(RIGHT_WHEEL, RIGHT, speed);
130         moveWheel(LEFT_WHEEL, LEFT, speed);
131     }
132 }
133
134 // Funcio que mou el robot cap a endarrere
135 void backward(unsigned int speed)
136 {
137     if(speed < 1024){
138         moveWheel(RIGHT_WHEEL, LEFT, speed);
139         moveWheel(LEFT_WHEEL, RIGHT, speed);
140     }
141 }
142
143 // Funcio que encen el led del motor
144 void motorLed(byte ID, bool status)
145 {
146     // Variable de tipus struct RxReturn que guarda el paquet que retorna els moduls
147     struct RxReturn returnPacket;
148     byte bInstruction = WRITE_DATA;
149     byte bParameterLength = 2;
150     byte bParameters[16];
151     bParameters[0] = M_LED;
152     bParameters[1] = status;
153
154     // Cridem a la funcio TxPacket per tal d'enviar el paquet d'instruccions
155     TxPacket(ID, bParameterLength, bInstruction, bParameters);
156     // Assignem a la variable creada anteriorment returnPacket el paquet de dades
    retornat
157     returnPacket = RxPacket();
158 }
159
160 // Funcio que llegeix informacio del sensor el ID del qual s'ha passat per parametre
161 int readSensor(byte ID, byte sensor)
162 {
163     // Variable de tipus struct RxReturn que guarda el paquet que retorna els moduls
164     struct RxReturn returnPacket;
165
166     byte bInstruction = READ_DATA;
167     byte bParameterLength = 2;
```

```
168     byte bParameters[16];
169     bParameters[0] = sensor;
170     bParameters[1] = 1;
171 
172     // Cridem a la funcio TxPacket per tal d'enviar el paquet d'instruccions
173     TxPacket(ID, bParameterLength, bInstruction, bParameters);
174     // Assignem a la variable creada anteriorment returnPacket el paquet de dades
        retornat
175     returnPacket = RxPacket();
176 
177     return returnPacket.StatusPacket[5];
178 }
```

3 DIAGRAMA DE FLUX

Els diagrames de flux representen un conjunt d'instruccions que es relacionen entre si formant una seqüència de passos que representen una certa operació que realitza, en aquest cas, un programa informàtic.

A continuació adjuntem el diagrama de flux d'aquesta pràctica. Cal remarcar que al ser una pràctica d'introducció a la comunicació mitjançant la UART i la conexió amb el Robot, el diagrama mostra el flux general que donaria una comunicació amb el Robot:

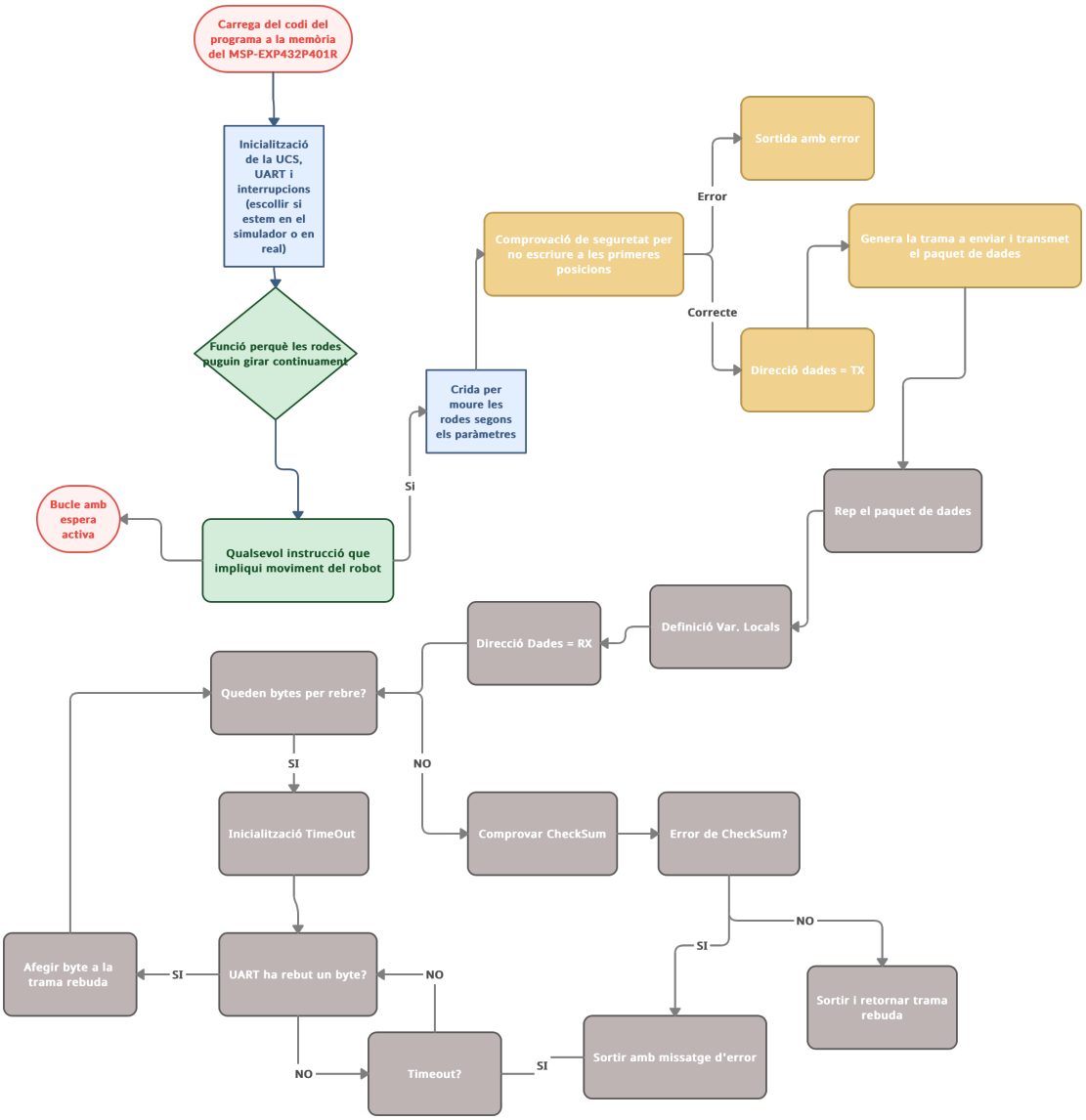


Figura 3.1: Diagrama de flux del programa