PAE: projecte

Miquel Martínez, Joan Laso

Juny 2025

Índex

1 Explicació general de la pràctica

1.1 Objectiu general

L'objectiu general de la pràctica ha sigut, a través de la llibreria que vam fer a la pràctica anterior, crear un algorisme que permeti a temps real les següents funcionalitats:

- Fer que el robot segueixi una paret, tant per l'esquerra com per la dreta, i que pugui canviar de direcció
- Que reajusti el rumb en cas d'allunyar-s'he o apropar-s'he massa a la paret, i eviti obstacles senzills
- Imprimir les informacions necessàries dels sensors al launchpad
- Que soni l'himne del barça just al trobar-s'he la primera paret

1.2 Recursos utilitzats

En aquesta pràctica s'ha utilitzat el robot BOOSTXL-EDUMKII Educational BoosterPack i el microcontrolador MSP432P401R que hi té adherit. Més concretament, s'han utilitzat la UART (concretament la UART 2), les rodes dels dos motors i els tres sensors. A més, també s'ha utilitzat un button del launchpad, la pantalla adherida al launchpad, i un buffer per transmetre so.

2 Configuració dels recursos utilitzats

2.1 Recursos necessaris

2.1.1 UART

Comencem per la UART. La primera linia d'instrucció és per ressetejar la USCI. Tot seguit, configurarem USCI altre cop en el registre UCA2CTLW0; hem posat en els comentaris les eleccions internes que està fent.

Després hem de configurar el baud rate. Recordem que la UART és una comunicació asíncrona que envia i rep bits a una determinada velocitat (el baud rate). En el nostre cas, els motors tenen un baud rate de 500.000bps, i per tant hem de configurar la UART de la mateixa manera. Així doncs, configurem el registre UCA2MCTLW amb un sobremostreix x16. També necessitem configurar el registre UCA2BRW. Com que el SMCLK va a 24Mhz, i volem un baudrate de 500kbps, aleshores fent dos senzills càlculs: $24/0.5 = 48$ i $48/16 = 3$, obtenim que el nostre factor és 3. Com que no tenim cap fracció, no hem de configurar el UCBRSx. Per configurar hem seguit el procediment explicat a la pàgina 915 del datasheet.

Tot seguit hem d'inicialitzar el pin3. El configurem com a sortida, i com a GPIO. A més, inicialitzem el port a 0 (en el P3OUT) que en aquest cas indicara que estem rebent dades. Més tard ja el configurarem, si s'escau, amb el Tx.

Després configurem els pins de la UART, els P3SEL, amb la funció I.O; el p3.2 per la RX i el p3.3 pel TX.

Finalment reactivem la línia de comunicacions serie en el UCA2CTLWO, que habia quedad desactivada, i també configurem les interrupcions locals (netejant primer la flag d'interrupció) i en el NVIC. Notem que això s'ha fet al final per evitar possibles errors mentres configurem. També notem que s'han d'activar les interrupcions globalment, però això o realitzarem al main.

Per la rutina d'atenció a la interrupció, senzillament el que hem de fer és llegir el bit que hem rebut. Notem però que per evitar possibles errors hem de netejar la flag d'interrupció i sobretot desactivar-les mentres estiguem llegint!

```
void init_UART(void){
 // Reg UCA2CTLW0
 // 15       14      13      12      11      10      9       8
 // Parity   Parity  MSB     lenght  stop bit UCMODE(2b)     UCSYNC
 // enable   select  1st sel 7 o 8b  1 o 2b  00= UART        async o sync(1)
 // 7        6       5       4       3       2       1       0
 // clk source(2b)   RX err  RX break Sleep  Tx addr Tx break SW reset Enabled
 // 00 UCLK...       char IE char IE 1=sleep 0=d 1=addr      UCSWRST

UCA2CTLW0 |= UCSWRST;                 // reset de la USCI

//Pag 923 del document
//Configurem la usci amb una variable predefinida
//UCSYINC=0 ->indica mode asincron de configuracio
//UCMODEx=0 ->seleccionem la UART
//UCSPB=0 ->  1 bit de stop
//UC7BIT=0 -> 8bits de dades de transmissio
//UCMSB=0 -> bit de menys pes es el primer
//UCPEN=0 -> bit de paritat NO
//UCPAR=x -> No fem servir bit de paritat
//clock smclk (24MHz amb la funcio que tenim)
UCA2CTLW0 |= UCSSEL__SMCLK;

UCA2MCTLW = UCOS16; // per fer el sobremostreig x16

//Tenim que SMCLK va a 24Mhz. Volem un baud rate de 500kbps=0.5Mbps,
```

```
    //i estem fent sobremostreig de 16.
    //24/0.5=48 ; ara, 48/16=3, on el 16 es el sobremostreig
    UCA2BRW = 3;



    //Incialitzem el pin d'adreçament de dades per a la linia half-duplex dels motors
    P3DIR |= DIR_UART;                    // PORT P3.0 com a sortida (Data direction: Selector Tx/Rx)
    P3SEL0 &= ~DIR_UART;                  // PORT P3.0 com I/O digital (GPIO)
    P3SEL1 &= ~DIR_UART;
    P3OUT &= ~DIR_UART;                   // Inicialitzem port P3.0 a 0 (Rx)

    //Configurem els pins de la UART
    Port_UARTx_SEL0 |= BIT2 | BIT3;       //I/O funcio: P3.2 = UART2RX, P3.3 = UART2TX
    Port_UARTx_SEL1 &= ~ (BIT2 | BIT3);

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

    //Interrupcions
    EUSCI_A2->IFG &= ~EUSCI_A_IFG_RXIFG;    // Clear eUSCI RX interrupt flag
    EUSCI_A2->IE |= EUSCI_A_IE_RXIE;

    NVIC->ICPR[0] |= 1 <<((EUSCIA2_IRQn) & 31);//Comprovem que no hi ha int residual pendent a la USCI
    NVIC->ISER[0] |= 1 <<((EUSCIA2_IRQn) & 31); //Habilitem les int. de la USCI
}

void EUSCIA2_IRQHandler(void)
{
    // we see if the interrupt is related to reception
    EUSCI_A2->IFG &=~ EUSCI_A_IFG_RXIFG; // Clear interrupt
    UCA2IE &= ~UCRXIE; //disabled interrupts in RX

    DatoLeido_UART = UCA2RXBUF;

    Byte_recibido_bool=1;
    UCA2IE |= UCRXIE;
}
```

2.1.2 Timers

Pel que fa als timers, hem modificat lleugerament el funcionament del timer A0, ja que ara, a més de fer parpellejar un LED per portar un control del temps de cicle d'aquest timer, també s'utilitza per temporitzar els temps de delay en els moviments del robot (exemple: petit delay en girar a la dreta per evitar que corregeixi la direcció constantment).

Codis que ens interessen relatius al timer A0, inclosa la rutina d'interrupció:

```
void TA0_0_IRQHandler(void)
{
    const uint8_t LED1_ON_OFF = 0x01;
    TA0CCTL0 &= ~CCIFG;
    P1OUT ^= LED1_ON_OFF;

    //Timer de gir
    timer_gir++;
}
```

```
void init_timer_A0(void){

TIMER_A0->CTL = TIMER_A_CTL_ID__1 | TIMER_A_CTL_SSEL__ACLK | TIMER_A_CTL_CLR | TIMER_A_CTL_MC__UP;
    TIMER_A0->CCR[0] = (1 << 13) - 1;
    TIMER_A0->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0

    NVIC->ICPR[0] |= BIT8;
    //Primero, me aseguro de que no quede ninguna interrupcion residual pendiente para este puerto,
    NVIC->ISER[0] |= BIT8; //y habilito las interrupciones del puerto
}
```

A més, hem hagut de configurar un dels temporitzadors de 32 bits del microcontrolador, per tal de poder ajustar amb una major precisió la durada de cada nota (a la funció de tocar melodia). Es podria haver utilitzat el temporitzador A0 (el A1 no, ja que aquest s'apaga i s'encén per al timeout), però necessitàvem una velocitat molt més alta (més precisió) per a la durada de les notes que per al retard de gir del robot, motiu pel qual hem considerat més apropiat fer ús d'aquest altre temporitzador. Per a això hem creat una funció que genera un delay d'una quantitat de microsegons passada per paràmetre (s'ha utilitzat per gestionar el temps que sonava cada nota en el brunzidor). Aquesta funció fa quatre coses: configura el registre de càrrega (com que el rellotge és de 3 MHz, hi ha 3 cicles per microsegon), configura el registre de control (habilita el temporitzador en mode periòdic i sense prescaler), neteja qualsevol interrupció prèvia pendent, i espera fins que el comptador arribi a 0 (això provoca que mentre el robot toca una melodia, el processador quedi en espera, per la qual cosa cal anar amb compte. En el cas del nostre projecte no és rellevant, ja que simplement volíem que el robot toqués una bella melodia abans de començar la seva marxa).

Codi de la funció:

```
void timer32_delay_us(uint32_t microseconds) {
    // Timer32 usa 3 MHz como default (SMCLK = 3 MHz)
    TIMER32_1->LOAD = microseconds * 3 - 1;  // 3 cycles per microsecond
    //La inicialització de CONTROL es podria haver fet només a l'inici del programa.
    //L'hem posat aquí per provar que tot funcionés inicialment i ens hem despistat de moure-la, tot
    TIMER32_1->CONTROL = TIMER32_CONTROL_ENABLE | TIMER32_CONTROL_MODE | TIMER32_CONTROL_PRESCALE_0;
    TIMER32_1->INTCLR = 0;  // Limpiar interrupción

    while((TIMER32_1->RIS & 1) == 0);  // Esperar que cuente a cero
}
```

Finalment, pel timer A1, en la funció init timers, s'han posat els registres corresponents (és a dir, s'ha posat el mode en UP, i el source en SMCLK. El divider s'ha posat a 1 trivialment). Després, s'ha ajustat la freqüència en el registre CCR[0], amb 0 i s'han activat les interrupcions. Perquè hem posat la freqüència a 0? La raó és que aquest timer l'anirem activant i desactivant, per tal de no gastar recursos, segons anem rebent els packets de la UART. Així doncs, hem creat dues funcions auxiliars, *activatimerA1TimeOut*, *desactivatimerA1TimeOut*, les quals activen i desactiven el timer A1 canviant la freqüència, de tal manera que estem simulant un timeout.

Codis que ens interessen relatius al timer A1. També posem la rutina d'interrupció:

```
void init_timer_A1(void){

TIMER_A1->CTL = TIMER_A_CTL_ID__1 | TIMER_A_CTL_SSEL__SMCLK | TIMER_A_CTL_CLR | TIMER_A_CTL_MC__UP;
    TIMER_A1->CCR[0] = 0;
    TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0

    NVIC->ICPR[0] |= 1 << (TA1_0_IRQn & 31);
    //Primero, me aseguro de que no quede ninguna interrupcion residual pendiente para este puerto,
    NVIC->ISER[0] |= 1 << (TA1_0_IRQn & 31);
```

```
    //y habilito las interrupciones del puerto

}

void Activa_TimerA1_TimeOut(void){
    TIMER_A1->CCR[0] =  240;
    //SI va a 24Mhz, fent 24Mhz / 240 = 0.1Mhz que és 10microsegons (1/0.1 = 10)
    TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0
}

void Desactiva_TimerA1_TimeOut(void){
    TIMER_A1->CCR[0] = 0;
    TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0
}


void Reset_Timeout(void){
    contador_T1_TIMEOUT = 0;
}

byte TimeOut(int umbral){
    //contador_T1_TIMEOUT=contador_T1_TIMEOUT;
    return umbral <= contador_T1_TIMEOUT;
}

void TA1_0_IRQHandler(void)
{
    TA1CCTL0 &= ~CCIFG;
    contador_T1_TIMEOUT++;
}
```

2.2 Recursos opcionals

2.2.1 Button del launchpad

Hem configurat el button del launchpad per tal de poder canviar de referència. Per configurar-lo, hem fet un init_botton seguint l'estil dels anteriors, configurant-lo com a GPIO, posant el pull-up, i activant-hi les interrupcions.

En la interrupció fem un if else que permet veure en quina referència ens trobem, i si cal o no fer el backflip. Per veure que funciones correctament printejavem des de la interrupció, tot i que és mala pràctica deixar-ho en el codi (i pot donar errors) i per això no ho hem posat a la versió final.

```
void init_botton(void){
    //Configuramos boton (5.1)
       //**************************
       P5SEL0 &= ~(BIT1);     //Els polsadors son GPIOs
       P5SEL1 &= ~(BIT1);     //Els polsadors son GPIOs

       P5DIR &= ~(BIT1);     //Un polsador es una entrada
       P5REN |= (BIT1);      //Pull-up/pull-down pel pulsador
       P5OUT |= (BIT1); //Donat que l'altra costat es GND, volem una pull-up

       P5IE |= (BIT1);       //Interrupcions activades
       P5IES &= ~(BIT1);     // amb transicio L->H
       P5IFG = 0;                    // Netegem les interrupcions anteriors
```

```
        NVIC->ICPR[1] |= 1 << (PORT5_IRQn & 31);
        NVIC->ISER[1] |= 1 << (PORT5_IRQn & 31);
}
    /*
 * Interrupcio del boto per canviar la referencia
 */
void PORT5_IRQHandler(void){
    uint8_t flag = P5IV; //guardem el vector d'interrupcions i el netegem
    //char buffer[64];

    P5IE &= ~(BIT1);

    //Posem referencia a 1 i backflip a 1
    //De normal la referencia estar a 0
    if(referencia == 0){
        referencia = 1;
        backflip = 1;
        //sprintf(buffer, "%s", "REF: dreta");
    }else if(referencia == 1){
        referencia = 0;
        backflip = 1;
        //sprintf(buffer, "%s", "REF: esquerra");
    }
    //halLcdClearLine(4);
    //halLcdPrintLine(buffer, 4, NORMAL_TEXT);

    P5IE |= (BIT1);
}
```

2.2.2 Buzzer del launchpad

Hem configurat el brunzidor (buzzer) del LaunchPad per tal que reprodueixi una melodia (en aquest cas hem optat per l'Himne del FC Barcelona) abans de començar la marxa. Per a això ha estat suficient configurar com a GPIO de sortida el port P2.7, que correspon al port J4.40 del BoosterPack:

```
void init_buzzer(void){
    // Configurar P2.7 como salida digital
    P2->SEL0 &= ~BIT7;
    P2->SEL1 &= ~BIT7;
    P2->DIR |= BIT7;
}
```

3 Funcionalitat del programa i testejos

En aquesta secció ens centrarem a explicar el funcionament del programa principal així com d'alguns tests que hem realitzat.

3.1 Algorisme

A continuació explicarem el comportament que descriu el robot al llarg de l'execució del programa. En primer lloc, és pertinent explicar les dues funcions auxiliars perquè el robot executi els girs:

1. **executar_gir(int posicio_gir, int unitat_temps):** aquesta funció simplement executa en el robot un gir en la direcció indicada per `posicio_gir` (esquerra o dreta) i espera la quantitat de temps establerta amb l'objectiu que el gir tingui una durada mínima.
2. **executar_gir_sensors(int posicio_gir, int unitat_temps, int maxim_sensor):** aquesta funció és una variació de l'anterior. En aquest cas també s'executa el gir, amb la diferència de que l'espera posterior també finalitza si el sensor del costat contrari cap on s'està girant detecta una paret més a prop de la distància indicada, o bé si el sensor del costat contrari detecta la paret més lluny de la distància indicada (deixant-hi un marge, és a dir, `maxim_sensor` és el centre d'un interval dins del qual s'ha de moure el robot).

El codi d'aquestes dues funcions és el següent:

```
void executar_gir(int posicio_gir, int unitat_temps){
    //Tenim dos casos, o be esquerra o be dreta
    if(posicio_gir==ESQUERRA){
        girar_esquerra();
        Reset_gir_temps();
        while(!control_gir_temps(unitat_temps)){imprimir_sensors();}

    }else if(posicio_gir==DRETA){
        girar_dreta();
        Reset_gir_temps();
        while(!control_gir_temps(unitat_temps)){imprimir_sensors();}
    }
}

void executar_gir_sensors(int posicio_gir, int unitat_temps, int maxim_sensor){
    int sensor_no_detectat = 1;
    RxReturn packetsensor_dreta;
    RxReturn packetsensor_esquerra;

    if(posicio_gir==ESQUERRA){
        girar_esquerra();
        Reset_gir_temps();
        while(!control_gir_temps(unitat_temps) && sensor_no_detectat){
            imprimir_sensors();
            packetsensor_dreta=llegir_sensor(2);
            //llegirem els sensors; si estem girant cap a l'esquerra aixo vol dir que voldrem tenir
            //la paret de la dreta
            //Si el sensor de la dreta esta entre [-20 + maxim_sensor, maxim_sensor + 20] aixo
            //vol dir que estarem en un rang acceptable

            if(packetsensor_dreta.StatusPacket[5] > -10 + maxim_sensor
            && packetsensor_dreta.StatusPacket[5] < maxim_sensor + 10){
                sensor_no_detectat = 0;
            }
        }
    }else if(posicio_gir==DRETA){
```

```
        girar_dreta();
        Reset_gir_temps();
        while(!control_gir_temps(unitat_temps) && sensor_no_detectat){
            imprimir_sensors();
            packetsensor_esquerra=llegir_sensor(0);
            //llegirem els sensors; si estem girant cap a la dreta aixo vol dir que voldrem tenir
            //la paret de lesquerra
            //Si el sensor de lesq esta entre [-20 + maxim_sensor, maxim_sensor + 20] aixo
            //vol dir que estarem en un rang acceptable
            if(packetsensor_esquerra.StatusPacket[5] > -10 + maxim_sensor
            && packetsensor_esquerra.StatusPacket[5] < maxim_sensor + 10){
                sensor_no_detectat = 0;
            }
        }
    }
}
```

Temporalment, el programa transcorre de la manera següent: després de totes les inicialitzacions, es deté el robot i s'ajusta una velocitat inicial. A partir d'aquí es realitza el següent:

1. **Buscar la primera paret:** aquesta funció inicial llegeix el sensor davanter del robot fins que la distància de lectura és prou gran. Un cop això succeeix, el robot es pararà uns instants per escriure a la pantalla el valor llegit pels sensors. Un cop fet això, el robot comença a girar en la direcció corresponent en funció de la referència inicial (esquerra per defecte en el nostre codi) mentre reprodueix l'himne del FC Barcelona.

   És important notar que en un principi el robot estava quiet, i després girava cap on havia de continuar. Haguéssim pogut deixar-ho així, però era més entretingut que anés girant mentre sonés l'himne. Tanmateix, això fa que potser acabarà en una posició extranya i s'hagi de redirigir a mà.

2. **Bucle principal:** un cop s'ha trobat la primera paret, el robot comença a avançar cap endavant, donant inici al bucle principal. En aquest, primer es llegeixen els valors dels 3 sensors que s'utilitzaran al llarg del bucle. Després, es comprova si s'ha fet un canvi de referència, la cual cosa provocaría que es realitzi un gir de 180^{o} en el sentit de la nova referència.

   A partir d'aquest punt, el codi del bucle és simètric per als casos en què la referència sigui l'esquerra o la dreta, així que ens centrarem en el primer cas, que és el configurat per defecte. En cada cicle del bucle es contemplen els següents quatre casos, en l'ordre indicat:

   (a) S'ha trobat una paret al davant: s'executa un gir cap a la dreta per esquivar la paret, i després es manté avançant uns instants abans de continuar amb el següent cicle del bucle, per tal d'evitar que el robot quedi encallat.

   (b) S'ha deixat de detectar una paret a l'esquerra: s'executa un gir molt curt cap a l'esquerra per tornar a trobar una altra paret. De la mateixa manera que abans, s'avança cap endavant durant una petita fracció de temps.

   (c) S'ha detectat la paret de l'esquerra massa a prop i també una paret al davant: en aquest cas el robot actuarà igual que quan simplement troba una paret al davant, amb la diferència que aquesta vegada s'utilitzarà la funció `executar_gir_sensors` perquè el gir només duri fins que el robot torni a detectar que es troba en la posició adequada segons els sensors. D'aquesta manera, el gir s'executa correctament per a diferents angles. Igual que abans, hi ha un temps de gir per evitar que el robot quedi girant sense trobar paret. Novament es realitza un petit avanç cap endavant.

   (d) S'ha detectat la paret de l'esquerra massa a prop: en aquest cas simplement caldrà corregir la posició executant un gir cap a la dreta fins que el robot es trobi en la posició adequada. També s'executa el petit avanç.

   Notem que l'ordre dels factors és important. S'ha donat prioritat, per exemple, a trobar la paret del davant, ja que és l'acció més urgent per tal que no xoqui. De la mateixa manera, els passos c) d) poden

semblar casi iguals, però en el c) la distancia de dir és major (4ticks) vs el d) (que té 2ticks). Això és perquè si tenim una paret al davant i a lesquerra això voldrà dir que necessite girar bastant més que no que si ens estem apropant.

El codi descrit es el següent:

```
int buscar_primera_paret(){
    RxReturn packetsensor_endavant;
    char buffer[64];

    byte paret_trobada = 0;
    //Assumirem que la primera paret està al davant
    while(!paret_trobada){
        packetsensor_endavant= llegir_sensor(1);
        //Ens hem trobat la primera paret que esta endavant
        if(packetsensor_endavant.StatusPacket[5]>CERCANIA_PARET_MAXIM){
            //ATURAREM EL MOTOR I ENS ESPERAREM FINS A UN TEMPS PETIT QUE LUSUARI PUGUI FER INPUT
            aturar();
            Reset_gir_temps();
            while(!control_gir_temps(6)){imprimir_sensors();}
            paret_trobada = 1;
        }
    }
    //A segons de la referencia girarem a l'esquerra o a la dreta
    if(referencia==0){
        executar_gir(DRETA,5);
        sprintf(buffer, "%s", "REF: esquerra");
    }else{
        executar_gir(ESQUERRA,5);
        sprintf(buffer, "%s", "REF: dreta");
    }

    halLcdClearLine(4);
    halLcdPrintLine(buffer, 4, NORMAL_TEXT);
    tocar_himne_FCB();
    return 0;
}

void buclecodiproj(){

    //Returns de tres sensors
    RxReturn packetsensor_endavant;
    RxReturn packetsensor_esquerra;
    RxReturn packetsensor_dreta;

    //Primer bucle per anar a trobar la pared
    buscar_primera_paret();
    moure_endavant();

    //El primer backflip no el tindrem en compte. El backflip serveix per quan canviem de direcció
    if(backflip){
        backflip = 0;
    }

    //bucle infinit principal del programa
    int i=1;
    while(i){
        //Haurem de llegir ara dos sensors, el del davant i el de la dreta / el de lesquerra
```

```
packetsensor_endavant=llegir_sensor(1);
packetsensor_esquerra=llegir_sensor(0);
packetsensor_dreta=llegir_sensor(2);

//Si hem activat el joystick
if(backflip){
    //Si ref == 0, aleshores volem paret cap a l'esquerra (i la tenim a la dreta)
    if ( referencia == 0){
        executar_gir_sensors(DRETA, 16, 40);
    }else{ //sino hem de girar cap a l'altre costat
        executar_gir_sensors(ESQUERRA, 16, 40);
    }
    //ja haurem fet el gir aixi que resetegem
    backflip = 0;
}

if(caminant){//no hem clicat el boto de parar

    //Anem cap endavant. Aquesta instrucció pot ser una mica redundant ates que a priori
    //sempre que girem posem el moure_endavant
    //Tanmateix la posem aquí per seguretat, ja que sempre que comencem el bucle
    //haurem d'anar cap endavant
    moure_endavant();

    switch(referencia){
    //CASE 0: TENIM UNA PARET A LESQUERRA
    case 0:
        if(packetsensor_endavant.StatusPacket[5]>CERCANIA_PARET_MAXIM){
            //girarem perque haurem de resseguir
            //executar_gir(DRETA,6); //aixo vol dir que s'haura de girar cap a la dreta
            executar_gir(DRETA, 6);
            //continuem movent cap endavant
            moure_endavant();
            Reset_gir_temps();
            while(!control_gir_temps(4)){imprimir_sensors();}
        }
        else if(packetsensor_esquerra.StatusPacket[5]<CERCANIA_PARET_MINIM){
        //Si deixem de tenir la paret a la esquerra
            executar_gir(ESQUERRA,1); //aixo vol dir que s'haura de girar cap a la esquerra

            moure_endavant(); //continuem cap endavant
            Reset_gir_temps();
            while(!control_gir_temps(2)){imprimir_sensors();}
        } //Si tenim una paret al davant i a l'esquerra massa aprop haurem de girar una mica
        else if(packetsensor_esquerra.StatusPacket[5]>CERCANIA_PARET_MITJA
        & packetsensor_endavant.StatusPacket[5]>CERCANIA_PARET_MINIM){
            //girarem perque haurem de resseguir
            //executar_gir(DRETA,4); //aixo vol dir que s'haura de girar cap a la dreta

            executar_gir_sensors(DRETA, 4, 40);
            moure_endavant(); //continuem cap endavant
            Reset_gir_temps();
            while(!control_gir_temps(2)){imprimir_sensors();}
        }
        else if(packetsensor_esquerra.StatusPacket[5]>CERCANIA_PARET_MITJA){
            //girarem perque haurem de resseguir
            //executar_gir(DRETA,2); //aixo vol dir que s'haura de girar cap a la dreta
```

```
                executar_gir_sensors(DRETA, 2, 40);
                //continuem movent cap endavant
                moure_endavant();
                Reset_gir_temps();
                while(!control_gir_temps(3)){imprimir_sensors();}
            }
            break;

        //TENIM UNA PARET A LA DRETA
        case 1:
            if(packetsensor_endavant.StatusPacket[5]>CERCANIA_PARET_MAXIM){
                //girarem perque haurem de resseguir
                //executar_gir(ESQUERRA,6); //aixo vol dir que s'haura de girar cap a l'esquerra
                executar_gir(ESQUERRA, 6);

                //continuem movent cap endavant
                moure_endavant();
                Reset_gir_temps();
                while(!control_gir_temps(4)){imprimir_sensors();}
            }
            else if(packetsensor_dreta.StatusPacket[5]<CERCANIA_PARET_MINIM){
            //Si deixem de tenir la paret a la dreta
                executar_gir(DRETA,1); //aixo vol dir que s'haura de girar cap a la dreta

                moure_endavant(); //continuem cap endavant
                Reset_gir_temps();
                while(!control_gir_temps(2)){imprimir_sensors();}
            } //Si tenim una paret al davant i a la dreta massa aprop haurem de girar una mica
            else if(packetsensor_dreta.StatusPacket[5]>CERCANIA_PARET_MITJA
            & packetsensor_endavant.StatusPacket[5]>CERCANIA_PARET_MINIM){
                //girarem perque haurem de resseguir
                executar_gir_sensors(ESQUERRA, 4, 40);
                //aixo vol dir que s'haura de girar cap a l'esquerra

                moure_endavant(); //continuem cap endavant
                Reset_gir_temps();
                while(!control_gir_temps(2)){imprimir_sensors();}
            } //Aixo vol dir que hem pillat una paret a la dreta molt, aixi q toca girar
            //cap a l'altra banda
            else if(packetsensor_dreta.StatusPacket[5]>CERCANIA_PARET_MITJA){
                //girarem perque haurem de resseguir
                executar_gir_sensors(ESQUERRA, 2, 40);
                //aixo vol dir que s'haura de girar cap a l'esquerra

                //continuem movent cap endavant
                moure_endavant();
                Reset_gir_temps();
                while(!control_gir_temps(3)){imprimir_sensors();}
            }
            break;
        case 2:
            break;
        }
    }else{ //caminant==0;
        aturar();
    }
};
```

```
}
```

3.2 Tocar melodia

Per a la implementació de la melodia que hem estat comentant, hem hagut de (a més d'inicialitzar el brunzidor) crear 4 mètodes:

1. D'una banda, hem creat mètodes de busy delay: un fent ús d'un dels temporitzadors que té el microcontrolador (vist a la secció de inicialització de temporitzadors), que ens ofereix una gran precisió a l'hora de calcular la durada de cada so, i un altre fent ús d'un comptador simple, per als silencis corresponents.

2. D'altra banda, hem creat la funció `playTone`, que rep la freqüència (nota que sonarà) i la durada d'aquesta. Amb això, la funció calcula el període, el temps d'encesa i d'apagada (que representen el volum que tindrà el so) i el nombre de cicles. Amb això, es fa ús del delay del temporitzador de 32 bits perquè el brunzidor s'encengui i s'apagui en cada cicle el temps corresponent, de manera que soni la nota adequada amb el volum adequat.

3. Finalment, la funció per tocar la melodia corresponent (en el nostre cas, l'himne del FC Barcelona campió de lliga). Aquesta funció crea dos arrays que transcriuen la partitura. En un es guarden les freqüències de les notes en l'ordre en què s'han de tocar, i en l'altre les durades de cadascuna d'elles. Amb això, només cal, per a cada nota de la "partitura", reproduir el so al brunzidor amb les funcions que hem creat anteriorment. En cas que la nota sigui un silenci, simplement s'espera la durada corresponent en lloc de cridar la funció `playTone`.

Codi descrit:

```
void simple_delay_ms(uint32_t ms) {
    volatile uint32_t count;
    while (ms--) {
        count = 3000;
        while (count--) ;
    }
}

void playTone(uint32_t freq, uint32_t duration_ms) {
    uint32_t period_us = 1000000 / freq;
    uint32_t on_time = period_us / 4;        // 25% encendido
    uint32_t off_time = period_us - on_time; // 75% apagado
    uint32_t cycles = (duration_ms * 1000) / period_us;

    uint32_t i;
    for(i = 0; i < cycles; i++) {
        P2->OUT |= BIT7;
        timer32_delay_us(on_time);
        P2->OUT &= ~BIT7;
        timer32_delay_us(off_time);
    }
}

void tocar_himne_FCB(void) {
    uint16_t notes[] = {
            329, 392, 523, 0,  // MI, SOL, DO'
            329, 392, 523, 0,  // MI, SOL, DO'
            329, 392, 523, 587, 523, 493, // MI, SOL, DO', RE', DO', SI
            440, 493, 523, 493, 440, 392, // LA, SI, DO', SI, LA, SOL
            349, 392, 440, 392, 349, 329, // FA, SOL, LA, SOL, FA, MI
            293, 329, 370, 392, 370, 392, 440, 493, // RE, MI, FA#, SOL, FA#, SOL, LA, SI
            523, 493, 440, 392, 293, 392, 493, 440,493,440,493,440,493,440,493,440,493,440,
```

```
            493,440,493,440,493, 392 // DO', SI, LA, SOL, RE, SO  L, SI, LA (trino), SOL

        };

    uint16_t durations[] = {
            1125, 375, 375, 1687,
            1125, 375, 375, 1687,
            750, 188, 375, 375, 188, 1125,
            562, 188, 375, 188, 188, 1125,
            562, 188, 375, 188, 188, 1125,
            750, 375, 375, 188, 188, 188, 188, 750,
            375, 282, 94, 188, 188, 188, 188, 47,47,47,47,47,47,47,47,47,47,47,47,47,47,47, 375
        };

    int length = sizeof(notes) / sizeof(notes[0]);
    int i;
    for(i = 0; i < length; i++) {
        if (notes[i] == 0) {
            // Silencio: solo espera el tiempo de la duración
            simple_delay_ms(durations[i]);
        } else {
            playTone(notes[i] / 4, durations[i] * 10);
        }
    }
}
```

3.3 Testejos

Per comprovar el correcte funcionament del codi hem realitzat una sèrie de proves. Per una banda, provar el funcionament de la melodia ha estat simplement cridar el mètode `tocar_himne_FCB` a l'inici del programa. Per altra banda, hem realitzat les proves en els següents casos per testar el moviment i la reacció del robot en diferents situacions:

- Deixar que el robot busqui una primera paret, executant el programa des d'un punt proper a una paret.
- Recórrer una paret recta amb la referència per l'esquerra, comprovant que el robot corregia la seva posició si s'apropava massa a la paret o bé si se n'allunyava i desviava el rumb.
- Gir de 180 graus. Vam posar una paret fina amb un final, de manera que quan el robot hi arribés, deixés de detectar la paret de referència i girés fins a tornar-la a trobar, havent de fer un gir brusc de 180º.
- Gir de 90 graus. Vam col·locar dues parets formant un angle de 90 graus, de manera que quan el robot arribés a la intersecció hagués de realitzar un gir sobre el seu eix per poder continuar per la paret.
- Girs amb diferents angles. També hem sotmès el robot a situacions en què havia de fer girs més tancats o més oberts, per comprovar que no quedava encallat i feia un gir amb la durada adequada.
- Finalment hem repetit totes aquestes proves amb la referència per la dreta. Amb això hem aprofitat per provar la funcionalitat del botó, clicant-lo durant la marxa per comprovar que aquest feia un gir de 180 graus sobre el seu eix canviant la referència.

4 Problemes que han surgit i conclusions

4.1 Problemes que han surgit

A mesura que anàvem implementant, han sorgit els següents problemes que hem solucionat:

- En una primera instància vam probar de fer el canvi de direcció amb el joystick. Tanmateix, el joystick tenia problemes seriosos, ja que quan el clickavem es tornava a clickar immediatament, fet que feia que s'anules la instrucció. Per aquest motiu al final ho hem hagut de fer amb el button, tot i que també té problemes similars, però apretan-t'ho fort va bé (no sempre). De fet, en un principi haviem configurat el joystick per aturar el robot, però donava masses problemes pel mateix motiu, i al final no ho hem implementat.
- Per fer l'algorisme, ens passava que llegia un sensor, i començava a girar a l'esquerra per exemple. Tanmateix, potser llegia un altre sensor que enviava informació contradictòria, i ara havia de girar a la dreta, fent que es quedes en bucle parat. També passava que no teniem manera de controlar quant de temps girava. Per fer això, hem introduït els delays amb els timers ja mencionats.
- El robot 9 va deixar de funcionar, i per tant vam haver de canviar de robot. A vegades els sensors dels robots són lleugerament diferents, i també tenen una velocitat diferent als motors. Això dificulta una mica les proves, tot i que amb la versió final s'ha comprovat que funcionés correctament.

A més, el projecte té els següents problemes, els quals poden afectar el rendiment del robot i s'han de millorar:

- Actualment printegem cada vegada que llegim sensors. Tot i que això ens dona una òptima informació, també pot donar problemes, ja que ho estem fent tot en l'execució principal del programa. Pot passar (i algun cop passa de tant en tant) que es quedi aturat perquè hi ha hagut algun problema de connexió amb la pantalla. No està a dins del nostre control, i la solució seria o bé fer-ho amb el FreeRTOS (com es proposa més abaix al document) o bé eliminar-ho completament o bé imprimir moltes menys vegades.
- El fet que hi hagi delays implica que hi ha un temps en el programa que no estem actualitzant posicions o possibilitats de gir. En cas que es canvii la velocitat, això vol dir que s'han de tornar a programar els delays, per tal d'ajustar el moviment. De la mateixa manera, quan llegim dels sensors, hem d'anar en compte amb la velocitat que tenim, ja que si llegim un sensor i hem de girar però estem massa aprop, potser no donarà temps sense xocar amb la paret. Idealment, el que hauriem de fer és llegir la velocitat exacta del motor, i després fer una funció, en funció d'aquesta velocitat, que ens retorni una variable per poder ajustar millor els delays i/o els sensors.

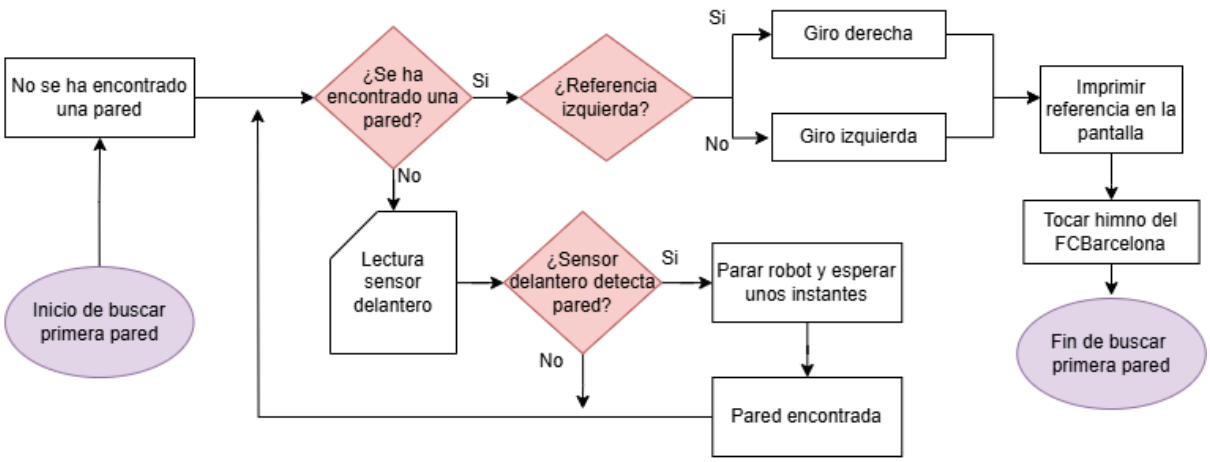
4.2 Conclusions

Al llarg d'aquesta pràctica hem anat aprenent les dificultats a l'hora de programar en un sistema a 'temps real', el qual pateix de problemàtiques molt diferents al que estem acostumats. Hem hagut d'afinar els algorismes i fer testejos exhaustius per tal de veure la diferència entre el paper i la realitat, que no és una correspondència 1 a 1. També s'han configurat nous elements segons han estat necessaris, denotant un domini sobre el hardware i habilitat per adaptar-nos a les especificacions necessàries.

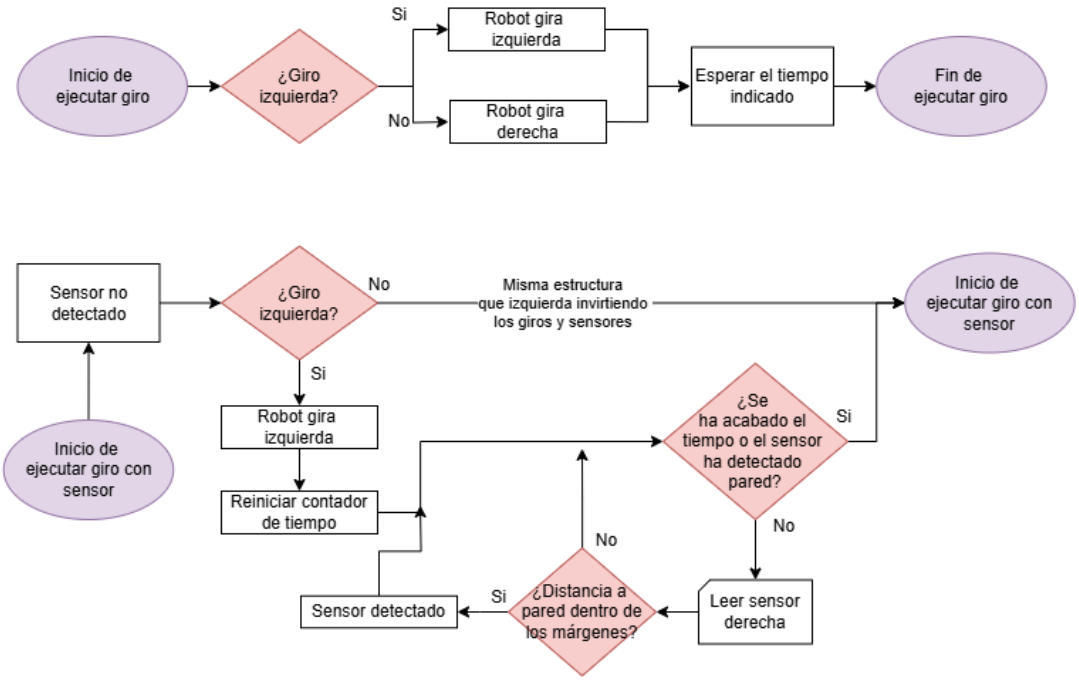
Tot i les problemàtiques trobades, s'ha aconseguit fer un algorisme prou robust. Tanmateix, per solucionar alguns dels problemes trobats, s'ha intentat trobar una altra solució, no bare metal, explicada a la secció següent.

5 Diagrames de flux

En esta sección adjuntaremos los diagramas de flujo. En primer lugar veremos los diagramas de las funciones auxiliares. Por un lado tenemos la función `buscar_primera_paret`:



Por otra parte podemos observar los diagramas de las funciones para ejecutar giros, tanto `executar_gir` como `executar_gir_sensors`:



Finalmente, mostramos el diagrama de flujo correspondiente a la ejecución principal del programa, en el que se utilizan estos diagramas que hemos adjuntado a parte. Se entiende que al escribir "girar izquierda"o "girar izquierda con sensores"se estan utilizando las funciones `executar_gir` y `executar_gir_sensors` respectivamente:

6 Projecte RTOS

6.1 Explicació general

De manera opcional i seguint el que s'ha proposat a classe de teoria, hem implementat tot el projecte en FreeRTOS, un SO de temps real de codi obert.

Els objectius d'implementar el projecte en RTOS són dos: d'una banda, obtenir la paralel·lització de tasques de tal manera que no bloquegem el thread del programa per algun error en alguna part de l'algorisme; i d'altra banda, tenir tot el codi implementat a través de les llibreries que proporciona FreeRTOS, de tal manera que si decidissim canviar de microcontrolador, no hauriem de tornar a configurar tots els pins manualment seguint el datasheet, sinó que això ja se n'encarrega la llibreria interna.

Les proves s'han fet amb el robot 12. S'ha transportat tot el programa integrament, menys l'himne que no ha donat temps a fer. Els sensors també s'haurien d'ajustar una mica, com la velocitat, però el 'core' de l'algorisme funciona bé.

A continuació explicarem les diferències amb el projecte bare-metal. L'objectiu no és reexplicar el projecte, sinó comentar els canvis respecte l'altra implementació.

Explicarem només els detalls i les funcionalitats essencials del programa. El codi està força comentat de totes maneres, així que si apareix alguna instrucció o funcionalitat no comentada s'hauria de poder entendre.

6.2 Estructura del projecte (carpetes)

A dins del ZIP hi ha diverses carpetes. Per no tenir problemes amb el FreeRTOS en termes de configuració hem partit de la Demo (https://www.freertos.org/Documentation/02-Kernel/01-About-the-FreeRTOS-kernel/03-Download-freeRTOS/01-DownloadFreeRTOS) que estava inclosa en la descàrrega del kernel del FreeRTOS, i hem modificat les funcions per tal de redigir els fitxers que volem executar.

El fitxer on està essencialment tot el projecte és el fitxer .c que es troba a dins de "SimplyBlinkyDemo", que es diu "tasques.c". Els altres fitxers a dins de la carpeta de "SimplyBlinkyDemo", com 'main-proj.c' són versions antigues que hem anat modificant.

Quan s'executa el projecte s'inicia un main.c que es troba a fora de la carpeta "SimplyBlinkyDemo", i aquest main redirigeix al 'main' de 'tasques.c'. A partir d'allà s'inicien tasques i l'scheduler i ja executa el programa.

A dins del projecte també hi ha altres carpetes. 'Driverlib' és una biblioteca de drivers pel Microcontrolador que estem utilitzant. 'Full-Demo' és una carpeta on hi ha altres demos que venien incloses al projecte que feien diferentes funcionalitats (força útils per veure com funciona).

Apart d'això també hi ha fitxers de configuracio. El FreeRTOSConfig.h configura unes quantes coses del SO, com ara prioritats base per les tasques, activament de funcionalitats, etc. L'altre fitxer que també hem modificat és el fitxer msp432-startup-css.c, que es troba a dins de la carpeta system. En aquest fitxer hi ha la taula del vector d'interrupcions, que hem modificat per redirigir algunes interrupcions a handlers específics.

6.3 Scheduler i taskes

L'avantatge principal d'utilitzar un sistema operatiu en temps real és que s'ocupa de diferents qüestions de manera no visible pel programador. En particular, FreeRTOS proporciona un Scheduler configurable. A través d'aquest Scheduler podem crear diferentes tasques, les quals s'executaran seqüencialment a segons de les prioritats que el programador les otorgui. Per iniciar un scheduler, cridem

```
vTaskStartScheduler();
```

Abans, però, hem de crear com a mínim una tasca. Per crear una tasca, utilitzarem el següent template:

```
xTaskCreate(vImprimirSensors, "ImprimirSensors", TEST_TASK_STACK_SIZE,
(void *)1, TEST_TASK_PRIORITY, NULL);
```

vImprimirSensors és el nom de la funció de l'entrada de la tasca. És el que s'executara quan la tasca s'inicii el primer cop. 'ImprimirSensors' és el nom que assignarem a la tasca. El StackSize són els bytes que donem màxim de memòria a la tasca, que en aquest cas hem posat 512 per totes menys el bucle, que són 2048. S'ha de mantenir un bon equilibri entre tenir-n'he suficient i no passar per optimitzar memòria.
L'altra paràmetre important és la prioritat de la tasca. En aquest cas, li hem donat la prioritat de la idle (que té la prioritat més baixa) + 1. En el projecte, l'única tasca que té una prioritat diferent és el bucle, que té prioritat de idle + 2. S'ha fet d'aquesta manera perquè així ens assegurem que en fer el bucle sempre tinguem l'informació dels sensors més actualitzada possible.

En el projecte hem fet les tasques de vLlegirSensors, vImprimirSensors, vInitTask, vBucleCodiprojecteTask, vHimneBarcaTask (que no s'ha implementat al final del tot), i vSpawnerTask.

En el main s'inicialitza la vInitTask i la vSpawnerTask. La vInitTask s'encarrega d'inicialitzar les configuracions que faltin, i a més s'encarrega de buscar la primera paret. D'altra banda, la vSpawnerTask s'encarregarà de crear les altres tasques del projecte; així doncs, la vSpawnerTask només s'executarà quan la vInitTask s'acabi. Un cop la vSpawnerTask hagi acabat, les altres tasques, llegir, imprimir, i fer el bucle, s'executaran seqüencialment. El scheduler assignarà un temps determinat a cada tasca (temps configurable), i quan el temps s'hagi acabat canviarà de tasca (excepte si ha entrat en alguna regió crítica).

Hem de tenir present algunes coses també:

- Si una tasca s'acaba, hem d'eliminar la tasca manualment amb vTaskDelete, ja que una tasca mai pot arribar al final de la funció handler.
- Si estem en una tasca i volem fer un delay (per exemple, perquè no volem estar llegint constantment els sensors) enlloc d'utilitzar un timer de 'hardware' de la placa, hem utilitzat un vTaskDelay(pdMS_TO_TICKS(n)). La funció pdMS_TO_TICKS transforma els milisegons a ticks del scheduler (a segons de com tinguem el clock configurat). La vTaskDelay fa un delay de la tasca - si és molt gran, aleshores l'scheduler ens canvia de tasca, i així l'execució del programa no es para!

6.4 Inicialització de la UART i dispositius I/O

Com s'ha explicat, un dels avantatges de fer el programa en FreeRTOS és la possibilitat d'inicialitzar tot des d'una llibreria ja implementada.

Per exemple, per inicialitzar la UART hem fet:

```
    const eUSCI_UART_Config xUARTConfig =
{
    EUSCI_A_UART_CLOCKSOURCE_SMCLK, /* SMCLK Clock Source. */
    3,                             /* BRDIV (es el mateix que amb la configuracio manual)*/
    0,                               /* UCxBRF */
    0,                               /* UCxBRS */
    EUSCI_A_UART_NO_PARITY,          /* No Parity. */
    EUSCI_A_UART_LSB_FIRST,          /* MSB First. */
    EUSCI_A_UART_ONE_STOP_BIT,       /* One stop bit. */
    EUSCI_A_UART_MODE,               /* UART mode. */
    EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION /* Low Frequency Mode. */
};
void init_uart0(void){
    uartTxQueue = xQueueCreate(UART_TX_QUEUE_LENGTH, sizeof(uint8_t));
    RxQueue = xQueueCreate(UART_RX_QUEUE_LENGTH, sizeof(uint8_t));

    MAP_GPIO_setAsPeripheralModuleFunctionInputPin( UART_GPIO_PORT_DADES, GPIO_PIN2 | GPIO_PIN3, GPIO_

    /* Use the library functions to initialise and enable the UART. */
    MAP_UART_initModule( EUSCI_AN_BASE, &xUARTConfig );
    MAP_UART_enableModule( EUSCI_AN_BASE );
```

```
    MAP_GPIO_setAsOutputPin(UART_GPIO_PORT_DADES, GPIO_PIN0);
    // PORT P1.0 com a sortida (Data direction: Selector Tx/Rx), GPIO
    MAP_GPIO_setOutputLowOnPin(UART_GPIO_PORT_DADES, GPIO_PIN0);
    // Inicialitzem port P1.0 a 0 (Rx)

    MAP_UART_clearInterruptFlag( EUSCI_AN_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT | EUSCI_A_UART_TRANSMIT
    MAP_UART_enableInterrupt( EUSCI_AN_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT );


    /* The interrupt handler uses the FreeRTOS API function so its priority must
    be at or below the configured maximum system call interrupt priority.
    configKERNEL_INTERRUPT_PRIORITY is the priority used by the RTOS tick and
    (should) always be set to the minimum priority. */
    MAP_Interrupt_setPriority( INT_EUSCIAN, configKERNEL_INTERRUPT_PRIORITY );
    MAP_Interrupt_enableInterrupt( INT_EUSCIAN );
}
```

Com es pot veure, s'utilitzen les funcions de initModule i enableModule, i els GPIO's dels ports es configuren amb funcions. A més, també podem netejar les interrupts flags, i activar les interrupcions (les funcions són força autexplicatives). Això redueix i simplifica el codi, ja que per exemple per inicialitzar el buffer podem fer senzillament

```
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN7);
```

Per inicialitzar algun GPIO que tingui interrupcions i sigui d'entrada, utilitzem un esquema com el següent (està comentat les equivalències amb el bareMetal)

```
void init_botton(void){
   // Aixo d'aqui es equivale a posar els P5SEL0, P5SEL1, P5DIR, P5REN i P5OUT
   MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P5, GPIO_PIN1);

   // Equivalent a P5IE |=
   MAP_GPIO_enableInterrupt(GPIO_PORT_P5, GPIO_PIN1);

   // Equivalent a posar el P5IES &=
   MAP_GPIO_interruptEdgeSelect(GPIO_PORT_P5, GPIO_PIN1, GPIO_LOW_TO_HIGH_TRANSITION);

   // Equivalent a P5IFG = 0
   MAP_GPIO_clearInterruptFlag(GPIO_PORT_P5, GPIO_PIN1);

   // Equivalent als les NVIC -> ICPR[1] i ISER[1].
   MAP_Interrupt_enableInterrupt(INT_PORT5);
}
```

Al projecte també hem configurat uns timers, tot i que al final no s'han utilitzat. Aquests timers són de 'software' interns del propi sistema operatiu, i es creen seguint el template:

```
one_shot_timer = xTimerCreate(
                  "One-shot timer",     // Name of timer
                  delay,            // Period of timer (in ticks)
                  pdFALSE,              // Auto-reload
                  (void *)0,            // Timer ID
                  raiseFlagRxPacket);  // Callback function
```

Finalment, també hem configurat els clocks utilitzant la llibreria poporcionada:

```
    void prvConfigureClocks( void )
{
    /* Set Flash wait state for high clock frequency.  Refer to datasheet for
```

```
    more details. */
    FlashCtl_setWaitState( FLASH_BANK0, 2 );
    FlashCtl_setWaitState( FLASH_BANK1, 2 );

    /* The full demo configures the clocks for maximum frequency, whereas the
    blinky demo uses a slower clock as it also uses low power features.  Maximum
    freqency also needs more voltage.

    From the datashee:  For AM_LDO_VCORE1 and AM_DCDC_VCORE1 modes, the maximum
    CPU operating frequency is 48 MHz and maximum input clock frequency for
    peripherals is 24 MHz.

    S'ha canviat a 24Mhz per ajustar-ho al projecte*/
    PCM_setCoreVoltageLevel( PCM_VCORE1 );
    CS_setDCOCenteredFrequency( CS_DCO_FREQUENCY_24 );
    CS_initClockSignal( CS_HSMCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1 );
    CS_initClockSignal( CS_SMCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1 );
    CS_initClockSignal( CS_MCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1 );
    CS_initClockSignal( CS_ACLK, CS_REFOCLK_SELECT, CS_CLOCK_DIVIDER_1 );
}
```

6.5 Comunicació en la UART

Com ja s'ha comentat, per activar les interrupcions dels ports s'ha hagut de redirigir al handler adequat des del vector d'interrupcions. Comentem la IRS de la UART perquè és força diferent. Apart d'utilitzar funcions per netejar flags, deshabilitar interrupcions i etc, també hem fet de tal manera que enviar i rebre des de la UART no es llegeixin els registres adequats, sinó que s'utilitzin les funcions de la llibreria. A més, fixem-nos que en la comunicació de les queues s'ha d'utilitzar la funció _FromISR, ja que algunes funcions quan es criden des de una ISR necessiten aquest sufixe (per tal que s'executin correctament i no hi hagin problemes amb l'scheduler).

A més, a diferència del projecte, aquí també transmeten les dades des de la ISR. Per fer-ho, primer transmetem la primera dada en el TxPackage, i activem les interrupcions de transmissió. Un cop haguem transmés la primera dada i tinguem les interrupcions activades, s'activa la flag d'interrupció de transmetre i salta la ISR, des d'on podrem enviar correctament la dada (veient quina flag s'ha aixecat).

A continuació el codi del handler de la ISR de la UART:

```
    void EUSCIA2_IRQHandler(void)
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    // we see if the interrupt is related to reception or the tx
    uint_fast8_t xInterruptStatus = MAP_UART_getEnabledInterruptStatus(EUSCI_AN_BASE);

    uint8_t txByte;
    uint8_t data;

    if ( (xInterruptStatus & EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG ) != 0x00 ) {
        // FromISR es perque son funcions que es fan desde una ISR i tenen atencio especial
        MAP_UART_clearInterruptFlag(EUSCI_AN_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG);
        data = MAP_UART_receiveData(EUSCI_AN_BASE); //rebem la data
        xQueueSendFromISR( RxQueue, &data, &xHigherPriorityTaskWoken );
    }

    //Una idea naif podria ser aqui fer un while amb la cua plena
    //Aixo no funcionara xk podriem estar transmetent massa rapid
    if ((xInterruptStatus & EUSCI_A_UART_TRANSMIT_INTERRUPT_FLAG) != 0x00) {
        if (xQueueReceiveFromISR(uartTxQueue, &txByte, &xHigherPriorityTaskWoken) == pdPASS) {
```

```
            MAP_UART_transmitData(EUSCI_AN_BASE, txByte);
            // We don't need to re-enable the interrupt here, as the act of
            // writing to the transmit buffer will likely set the flag again
            // if the UART is ready for more data.
        }else{
            MAP_UART_clearInterruptFlag(EUSCI_AN_BASE, EUSCI_A_UART_TRANSMIT_INTERRUPT_FLAG);
            // No more data to transmit - disable the transmit interrupt
            MAP_UART_disableInterrupt(EUSCI_AN_BASE, EUSCI_A_UART_TRANSMIT_INTERRUPT);
            Sentit_Dades_Rx(); //Aixo ho posem aqui dsp de que acabi la transmissio
            //Sentit_Dades_Rx();
            // The direction will be switched to RX after the transmission is complete,
        }

    }

    /* portYIELD_FROM_ISR() will request a context switch if executing this
    interrupt handler caused a task to leave the blocked state, and the task
    that left the blocked state has a higher priority than the currently running
    task (the task this interrupt interrupted).  See the comment above the calls
    to xSemaphoreGiveFromISR() and xQueueSendFromISR() within this function. */
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}
```

6.6 Comunicació entre tasques: queues, semàfors i timeouts

És natural que entre tasques ens volguem comunicar. Podriem estar temptats d'utilitzar variables globals, però no sol ser recomanat per temes de corrupció de dades quan canviem de tasca, i problemes de sincronització.

Per la comunicació entre tasques hem utilitzat dos instruments: les queues i els semàfors.

6.6.1 Queues

La queue és una queue de tota la vida, que es guarda en el nostre programa. La declarem amb

```
QueueHandle_t uartTxQueue = NULL;
    uartTxQueue = xQueueCreate(UART_TX_QUEUE_LENGTH, sizeof(uint8_t));
```

on la lenght serà quants objectes hi poden cabre, i el sizeof sera el size de la cua. En aquest cas, aquesta cua és la comunicació entre TxPackage i la ISR, i per tant hem definit una lenght àmplia. Hem fet una queue semblant pel RxPackage.

Per la comunicació entre tasques del bucle, on principalment volen transmetre la informació dels sensors, creem queues de lenght 1, i que tinguin un size d'un array de les dades dels sensors. Això s'ha fet així perquè si per un casual posem una dada de sensors, i després n'escrivim una altra, des del bucle de l'aplicació sempre voldrem la última dada més actualitzada, i per tant descartem la resta de dades. Una queue d'un sol item s'ha d'emplenar amb xQueueOverwrite; les altres amb xQueueSend. Per rebre, utilitzem xQueueReceive, i per fer un reset de la cua xQueueReset.

6.6.2 Semàfors

Per la vSpawnerTask hem d'esperar que s'acabi la tasca d'inicialització. Però com que les creem al mateix moment, les dues estan executant-se. Una manera de impedir que la Spawner no avanci és amb un semàfor. Un semàfor és basicament l'anàleg a una variable global booleana, que podem 'encendre' i 'apagar' per tal de bloquejar seccions de codi. La sintaxis és:

```
    SemaphoreHandle_t finishInitTaskSemaphore;
    finishInitTaskSemaphore = xSemaphoreCreateBinary();
```

```
//....
 //indiquem amb un semafor que ja hem acabat, i poden començar les altres tasques
xSemaphoreGive(finishInitTaskSemaphore);

//....
 if (xSemaphoreTake(finishInitTaskSemaphore, portMAX_DELAY) == pdTRUE) { ... }
```

Els noms de les funcions són autoexplicatius.

6.6.3 Timeouts

Si ens hi fixem en la instrucció anterior, apareix el portMAX_DELAY; aquest és un paràmetre que indica quan de temps hem d'esperar abans de continuar el programa. El portMAX_DELAY és una manera de tenir aquest timeout casi infinit. En el cas de les queues, per exemple, quan rebem també utilitzem un delay:

```
xQueueReceive(controlQueue, &packets, pdMS_TO_TICKS(100)) == pdTRUE
```

En aquest cas, a les queues, a diferència dels semàfors, ens interessa tenir un timeout finit per si hi ha algun problema amb la comunicació, de tal manera que no tinguem la tasca infinitament bloquejada.

D'altra banda, també ens pot interessar fer algun timeout en alguna altra situació. Per exemple, quan executem el gir a través dels sensors, utilitzem una altra funcionalitat. El següent codi és un codi simplificat del que tenim al projecte:

```
TickType_t start_time;
start_time = xTaskGetTickCount(); // Agafem els "ticks" de la tasca
girar_esquerra();
while (sensor_no_detectat && (xTaskGetTickCount() - start_time < pdMS_TO_TICKS(unitat_temps))) {
    // Si el sensor de la dreta esta entre [maxim_sensor - 10, maxim_sensor + 10]

    if (xQueueReceive(controlQueue, &packets, pdMS_TO_TICKS(50)) == pdTRUE){
        packetsensor_dreta = packets[2];
        if (packetsensor_dreta.StatusPacket[4] > maxim_sensor - 30 && packetsensor_dreta.StatusPacket[
            sensor_no_detectat = 0;
        }
    }
    vTaskDelay(pdMS_TO_TICKS(1)); // Per evitar busy-waiting
}
```

Aquest codi llegeix els sensors a través de la cua, un màxim de 50MS, fins que o bé haguem detectat un sensor (i estiguem en el rang correcte), o bé haguem arribat a un timeout. Per fer això, agafem els ticks de la tasca (quants ticks portem executats), i després els anem contant fins que la resta entre final - inicial sigui major a una certa quantitat de temps (utilitzant pdMS_TO_TICKS).

6.7 Problemàtiques trobades

Finalment, expliquem una problemàtica trobada la qual no ha donat temps a solucionar.

Com bé sabem necessitem un baudrate de 500kbs per comunicar-n'hos amb el motor. A més, a la pràctica s'ha utilitzat el clock a 24Mhz. A efectes pràctics, això vol dir que tenim molt poc temps (molts pocs ticks de clock) des de que enviem fins a que rebem. En el bare metal això no era un problema, perquè com que ho feiem amb pins directament hi havia molt poques instruccions a executar. Ara, tanmateix, com que ho fem desde el FreeRTOS tenim instruccions amb molts ticks extres. Així doncs, des de que enviem l'últim byte passa massa poc temps, i el perdem. Per sort, en aquest cas és un 0xFF sense informació adicional, però això podria ser un problema seriós. Hi ha diverses solucions al problema, com ara apujar el clock o bé llegir a un buffer directament.

De fet vam preguntar al foro de FreeRTOS, vegeu el post: https://forums.freertos.org/t/configuration-of-uart-for-msp432/23150/8 .

6.8 Conclusions

Amb aquesta implementació s'ha intentat solucionar el problema de la concurrència, de manera que problemes a l'hora de llegir o imprimir no donin problemes a l'execució del programa. Tanmateix, s'han introduït nous problemes, com ara la sincronització entre tasques i enviar i rebre a través de les interrupcions, els quals s'haurien de solucionar (amb més temps). El codi també s'ha complicat una mica, tot i que és més senzill de llegir degut als noms de les funcions.

Amb tot, ha sigut una bona experiència per aprendre les diferències entre bare metal i implementacions amb un sistema operatiu, i ha solucionat algun dels problemes trobats anteriorment.

7 Annex: codi

```
#include "robot_lib.h"
#include "stdio.h"
#include "lib_PAE.h"
#include "msp.h"

#define TXD2_READY (UCA2IFG & UCTXIFG) // interrupt flag reg and flag de tx

//Defines pels ports de la UART
#define Port_UARTx_SEL0 P3SEL0
#define Port_UARTx_SEL1 P3SEL1
#define DIR_UART BIT0

//Identificadors dels ids dels motors esquerra dreta i sensor
#define MOTOR_ESQUERRA_ID 2
#define MOTOR_DRETA_ID 4
#define SENSOR_ID 100

//Numero d'instruccio que enviem a la UART
#define WR_INST_BIT 3

typedef uint8_t byte;

uint8_t DatoLeido_UART;
//Time Out per rebre informacio de la UART
uint16_t contador_T1_TIMEOUT;

//Velocitat actual del robot
uint16_t velocitat_actual;

//Boolea per saber si hem rebut de la UART
byte Byte_recibido_bool;

//Timer que va contant quant de temps portem girant
//(o executant alguna accio, ho fem servir com a delay)
byte timer_gir;

byte referencia = 0; //Esquerra = 0; dreta = 1; deafault = 2
byte caminant = 1; //Parat = 0; caminant = 1
byte backflip = 0; //0 no gir; 1 gir



//----------------------------------------TXCPACKET-------------------------------

/*TxPacket() 3 parmetres: ID del Dynamixel, Mida dels parmetres, Instruction byte. torna la mida del '
 * Funcio:
 *
 */
byte TxPacket(byte bID, byte bParameterLength, byte bInstruction, byte Parametros[16]) {

    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;
    }

    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 mdul al que volem enviar el missatge
    TxBuffer[3] = bParameterLength+2;
    //Length(Parameter,Instruction,Checksum)
    TxBuffer[4] = bInstruction;
    //Instrucci que enviem al Mdul

    for(bCount = 0; bCount < bParameterLength; bCount++) {
    //Comencem a generar la trama que hem denviar
        TxBuffer[bCount+5] = Parametros[bCount];
    }

    bCheckSum = 0;
    bPacketLength = bParameterLength+4+2;
    //Aixo hauria de ser +3. EL +3 fa per ID; LENGHT, INSTRUCTION
    //(pero posem +4 per com fem els bucles)

    //El +2 es pels 2 byts dinici de trama


    //el bcount comena per 2 perque els dos primers bytes son d'inici de trama.
    for(bCount = 2; bCount < bPacketLength-1; bCount++) {   //Clcul del checksum
        bCheckSum += TxBuffer[bCount];
    }

    TxBuffer[bCount] = ~bCheckSum;                          //Escriu el Checksum (complement a 1)
    for(bCount = 0; bCount < bPacketLength; bCount++) {
    //Aquest bucle s el que envia la trama al Mdul Robot
        TxUACx(TxBuffer[bCount]);
    }

    //UCBUSY es BIT0, por tanto estamos cogiendo el BIT0 del registro UCA2STATW que indica
    //si hay una operacin en marcha
    while((UCA2STATW & UCBUSY));
    //Espera fins que sha transms el ltim byte

    Sentit_Dades_Rx();
    //Posem la lnia de dades en Rx perqu el mdul Dynamixel envia resposta

    return(bPacketLength);
}

//------------------------------------------------------------------------------

//-----------------------------------CANVI SENTIT COMUNICACIONS---------------------------------
```

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

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

/* funci TxUACx(byte): envia un byte de dades per la UART 2 */

void TxUACx(uint8_t bTxdData) {
    while(!TXD2_READY); // Espera a que estigui preparat el buffer de transmissi
    UCA2TXBUF = bTxdData;
}

//-------------------------------------------------------------------------------

//-------------------------------------RXPACKET---------------------------------
/*
    Veure documentacio al projecte
*/

RxReturn RxPacket(void) {
    RxReturn respuesta; //La resposta que rebem del robot
    byte bCount, bLenght, bChecksum; //
    byte Rx_time_out=0;
    byte check_sum=0;

    respuesta.error=0;


    //volem timeout quan hi hagi 1 milisegon; la rutina dinterrupcio fa cada 10 microsegons

    Sentit_Dades_Rx();                          //Ponemos la linea half duplex en Rx;
    Activa_TimerA1_TimeOut();

    //Primer llegim 3 bytes
    for(bCount = 0; bCount < 4; bCount++) {     //bRxPacketLength; bCount++)
        Reset_Timeout(); //posa el cont a 0

        Byte_recibido_bool=0;                       //No_se_ha_recibido_Byte();

        while (!Byte_recibido_bool) {               //Se_ha_recibido_Byte())
            Rx_time_out = TimeOut(1000);          // tiempo en decenas de microsegundos
            if (Rx_time_out){
                respuesta.byteTimeOut = 1;
                break;              //sale del while
            }
        }

        if (Rx_time_out) break;                 //sale del for si ha habido Timeout

        //Si no, es que todo ha ido bien, y leemos un dato:
        respuesta.StatusPacket[bCount] = DatoLeido_UART; //Get_Byte_Leido_UART();
```

```
    }

    //Desprs llegim tot
    if (!Rx_time_out){
        bLenght = DatoLeido_UART; //Aquest daqui es [3], lultim que hem llegit.

        for (bCount = 0; bCount < bLenght; bCount++){
            Reset_Timeout(); //posa el cont a 0

            Byte_recibido_bool=0;                       //No_se_ha_recibido_Byte();

            while (!Byte_recibido_bool) {               //Se_ha_recibido_Byte())
                Rx_time_out = TimeOut(1000);          // tiempo en decenas de microsegundos
                if (Rx_time_out){
                    respuesta.byteTimeOut = 1;
                    break;              //sale del while
                }
            }

            if (Rx_time_out) break;                  //sale del for si ha habido Timeout

            //Si no, es que todo ha ido bien, y leemos un dato:
            //Posem un +4 perque ens estem saltant els primers 4 parametres que ya hem llegit.
            respuesta.StatusPacket[bCount+4] = DatoLeido_UART; //Get_Byte_Leido_UART();
        }

        if (!Rx_time_out){
            check_sum+= respuesta.StatusPacket[2]; //afegim el ID
            check_sum+= respuesta.StatusPacket[3]; //afegim la lenght

            for (bCount = 0; bCount < bLenght-1; bCount++){ //menys el checksum (que es lultim)
                check_sum+= respuesta.StatusPacket[bCount+4]; //afegim tots els parametres
            }
            check_sum=~check_sum;
        }

        respuesta.byteTimeOut = 0;
    }


    //Comprovem el checksum, i tamb si l'error no s 0 (llavors hi ha algun error).
    if(check_sum != respuesta.StatusPacket[bLenght + 4 -1] || respuesta.StatusPacket[4] != 0){
        respuesta.error = 1;
        //TODO: En un futur podriem gestionar diferents errors.
        //Si per exemple lerror es dinvalid parameters farem una cosa
        //Si per exemple tenim el checksum malament podriem tornar a provar denviarho.
    }else{
        respuesta.error=0;
    }

    Desactiva_TimerA1_TimeOut();

    return respuesta;
}

//---------------------------------------------------------------------------------
```

```
//------------------------------COMUNICATION PACKET-------------------------------------

//Funcio que fa un txpacket i un rxpacket. Tamb te en compte si hi ha hagut algun error, i torna a fer
RxReturn ComunicationPacket(byte bID, byte bParameterLength, byte bInstruction, byte Parametros[16]) {
    TxPacket(bID,bParameterLength,bInstruction,Parametros);
    RxReturn retornpacket= RxPacket();

    //Fem bucle fins que no hi hagi error
    while(retornpacket.error==1 || retornpacket.byteTimeOut==1){
        TxPacket(bID,bParameterLength,bInstruction,Parametros);
        retornpacket= RxPacket();
    }
    return retornpacket;
}

//-----------------------------------------------------------------------------------------



//-----------------------------------------MOTOR-------------------------------------------

RxReturn EncendreMotor(byte motor_id) {

    //Torque enable = 1
    byte torque_enable[2] = {24, 1};
    return ComunicationPacket(motor_id, 2, WR_INST_BIT, torque_enable);

    //Configurar modo roda continua (CW y CCW angle limits a 0)
    /*
    byte cw_limit_low[2]  = {6, 0};
    byte cw_limit_high[2] = {7, 0};
    byte ccw_limit_low[2] = {8, 0};
    byte ccw_limit_high[2] = {9, 0};

    ComunicationPacket(motor_id, 2, WR_INST_BIT, cw_limit_low);
    ComunicationPacket(motor_id, 2, WR_INST_BIT, cw_limit_high);
    ComunicationPacket(motor_id, 2, WR_INST_BIT, ccw_limit_low);
    ComunicationPacket(motor_id, 2, WR_INST_BIT, ccw_limit_high);
     */
}

RxReturn AturaMotor(byte motor_id) {
    // Torque Enable = 0
    byte torque_disable[2] = {24, 0};

    return  ComunicationPacket(motor_id, 2, WR_INST_BIT, torque_disable);
}



RxReturn SetVelocitatMotor(byte motor_id, uint16_t velocitat) {
    byte speed[3] = {32, (byte) (velocitat & 0xFF), (byte) ((velocitat >> 8) & 0xFF)};
    // Less significant bits

    //byte speed_high[2] = {33, (velocitat >> 8) & 0xFF};   // Most significant bits
    //ComunicationPacket(motor_id, 2, WR_INST_BIT, speed_high);
```

```
    return ComunicationPacket(motor_id, 3, WR_INST_BIT, speed);
}

//------------------------------BUZZER CONFIG---------------------------------

void init_buzzer(void){
    // Configurar P2.7 como salida digital
    P2->SEL0 &= ~BIT7;
    P2->SEL1 &= ~BIT7;
    P2->DIR |= BIT7;
}

//----------------------------------------------------------------------------



//------------------------------BOTO CONFIG-----------------------------------

void init_botton(void){
    //Configuramos boton (5.1)
       //*************************
       P5SEL0 &= ~(BIT1);     //Els polsadors son GPIOs
       P5SEL1 &= ~(BIT1);     //Els polsadors son GPIOs

       P5DIR &= ~(BIT1);     //Un polsador es una entrada
       P5REN |= (BIT1);      //Pull-up/pull-down pel pulsador
       P5OUT |= (BIT1); //Donat que l'altra costat es GND, volem una pull-up

       P5IE |= (BIT1);       //Interrupcions activades
       P5IES &= ~(BIT1);     // amb transicio L->H
       P5IFG = 0;                    // Netegem les interrupcions anteriors

       NVIC->ICPR[1] |= 1 << (PORT5_IRQn & 31);
       NVIC->ISER[1] |= 1 << (PORT5_IRQn & 31);
}

//----------------------------------------------------------------------------



//------------------------------TIMER CONFIG----------------------------------
/**
 * Funcio copiada del campus; inicialitza el timer.
 */
void init_timer_A0(void){

    TIMER_A0->CTL = TIMER_A_CTL_ID__1 | TIMER_A_CTL_SSEL__ACLK | TIMER_A_CTL_CLR | TIMER_A_CTL_MC__UP;
    TIMER_A0->CCR[0] = (1 << 13) - 1;
    TIMER_A0->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0

    NVIC->ICPR[0] |= BIT8; //Primero, me aseguro de que no quede ninguna interrupcion residual pendier
    NVIC->ISER[0] |= BIT8; //y habilito las interrupciones del puerto
}

//Timer pel timeOUT
void init_timer_A1(void){
```

```
    TIMER_A1->CTL = TIMER_A_CTL_ID__1 | TIMER_A_CTL_SSEL__SMCLK | TIMER_A_CTL_CLR | TIMER_A_CTL_MC__UP
    TIMER_A1->CCR[0] = 0;
    TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0

    NVIC->ICPR[0] |= 1 << (TA1_0_IRQn & 31); //Primero, me aseguro de que no quede ninguna interrupcio
    NVIC->ISER[0] |= 1 << (TA1_0_IRQn & 31); //y habilito las interrupciones del puerto

}

void Activa_TimerA1_TimeOut(void){
    TIMER_A1->CCR[0] =  240; //SI va a 24Mhz,
    // fent 24Mhz / 240 = 0.1Mhz que s 10microsegons (1/0.1 = 10)
    TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0
}

void Desactiva_TimerA1_TimeOut(void){
    TIMER_A1->CCR[0] = 0;
    TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0
}


void Reset_Timeout(void){
    contador_T1_TIMEOUT = 0;
}

byte TimeOut(int umbral){
    //contador_T1_TIMEOUT=contador_T1_TIMEOUT;
    return umbral <= contador_T1_TIMEOUT;
}

void Reset_gir_temps(void){
    timer_gir=0;
}

byte control_gir_temps(int unitat_temps){
    return unitat_temps <= timer_gir;
}

//-------------------------------------------------------------------------


//----------------------------UART CONFIG--------------------------------

void init_UART(void){
    // Reg UCA2CTLW0
    // 15       14      13      12      11      10      9       8
    // Parity   Parity  MSB     lenght  stop bit UCMODE(2b)     UCSYNC
    // enable   select  1st sel 7 o 8b  1 o 2b  00= UART        async o sync(1)
    // 7        6       5       4       3       2       1       0
    // clk source(2b)   RX err  RX break Sleep  Tx addr Tx break SW reset Enabled
    // 00 UCLK...       char IE char IE 1=sleep 0=d 1=addr      UCSWRST

   UCA2CTLW0 |= UCSWRST;                // reset de la USCI

   //Pag 923 del document
   //Configurem la usci amb una variable predefinida
```

```
    //UCSYINC=0 ->indica mode asincron de configuracio
    //UCMODEx=0 ->seleccionem la UART
    //UCSPB=0 ->  1 bit de stop
    //UC7BIT=0 -> 8bits de dades de transmissio
    //UCMSB=0 -> bit de menys pes es el primer
    //UCPEN=0 -> bit de paritat NO
    //UCPAR=x -> No fem servir bit de paritat
    //clock smclk (24MHz amb la funcio que tenim)
    UCA2CTLW0 |= UCSSEL__SMCLK;

    UCA2MCTLW = UCOS16; // per fer el sobremostreig x16

    //UCA2BRW = 13;     // CONFIGURATION UART TO 115200bps (antiga)
    //UCA2MCTLW &= ~(0x25<<8);

    //Tenim que SMCLK va a 24Mhz. Volem un baud rate de 500kbps=0.5Mbps, i estem fent sobremostreig 16.
    //24/0.5=48 ; ara, 48/16=3, on el 16 es el sobremostreig
     UCA2BRW = 3;

    //Incialitzem el pin d'adreament de dades per a la linia half-duplex dels motors
    P3DIR |= DIR_UART;                  // PORT P3.0 com a sortida (Data direction: Selector Tx/Rx)
    P3SEL0 &= ~DIR_UART;                // PORT P3.0 com I/O digital (GPIO)
    P3SEL1 &= ~DIR_UART;
    P3OUT &= ~DIR_UART;                 // Inicialitzem port P3.0 a 0 (Rx)

    //Configurem els pins de la UART
    Port_UARTx_SEL0 |= BIT2 | BIT3;      //I/O funcio: P3.2 = UART2RX, P3.3 = UART2TX
    Port_UARTx_SEL1 &= ~ (BIT2 | BIT3);

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

    //Interrupcions
    EUSCI_A2->IFG &= ~EUSCI_A_IFG_RXIFG;    // Clear eUSCI RX interrupt flag
    EUSCI_A2->IE |= EUSCI_A_IE_RXIE;

    NVIC->ICPR[0] |= 1 <<((EUSCIA2_IRQn) & 31);
    // Comprovem que no hi ha int residual pendent a la USCI
    NVIC->ISER[0] |= 1 <<((EUSCIA2_IRQn) & 31);
    // Habilitem les int. de la USCI
}

//------------------------------------------------------------------------


//-----------------------------BUZZER INSTRUCCIONS-------------------------

void timer32_delay_us(uint32_t microseconds) {
    // Timer32 usa 3 MHz como default (SMCLK = 3 MHz)
    TIMER32_1->LOAD = microseconds * 3 - 1;  // 3 cycles per microsecond
    TIMER32_1->CONTROL = TIMER32_CONTROL_ENABLE | TIMER32_CONTROL_MODE | TIMER32_CONTROL_PRESCALE_0;
    TIMER32_1->INTCLR = 0;  // Limpiar interrupcin

    while((TIMER32_1->RIS & 1) == 0);  // Esperar que cuente a cero
}

void simple_delay_ms(uint32_t ms) {
```

```
    volatile uint32_t count;
    while (ms--) {
        count = 3000; // Ajusta este valor para que 3000 ciclos sean ~1 ms
        while (count--) ;
    }
}

void playTone(uint32_t freq, uint32_t duration_ms) {
    uint32_t period_us = 1000000 / freq;
    uint32_t on_time = period_us / 4;        // 25% encendido
    uint32_t off_time = period_us - on_time; // 75% apagado
    uint32_t cycles = (duration_ms * 1000) / period_us;

    uint32_t i;
    for(i = 0; i < cycles; i++) {
        P2->OUT |= BIT7;
        timer32_delay_us(on_time);
        P2->OUT &= ~BIT7;
        timer32_delay_us(off_time);
    }
}

void tocar_himne_FCB(void) {
    uint16_t notes[] = {
            329, 392, 523, 0,  // MI, SOL, DO'
            329, 392, 523, 0,  // MI, SOL, DO'
            329, 392, 523, 587, 523, 493, // MI, SOL, DO', RE', DO', SI
            440, 493, 523, 493, 440, 392, // LA, SI, DO', SI, LA, SOL
            349, 392, 440, 392, 349, 329, // FA, SOL, LA, SOL, FA, MI
            293, 329, 370, 392, 370, 392, 440, 493, // RE, MI, FA#, SOL, FA#, SOL, LA, SI
            523, 493, 440, 392, 293, 392, 493, 440,493,440,493,440,493,440,493,440,493,440,493,440,493

        };

    uint16_t durations[] = {
            1125, 375, 375, 1687,
            1125, 375, 375, 1687,
            750, 188, 375, 375, 188, 1125,
            562, 188, 375, 188, 188, 1125,
            562, 188, 375, 188, 188, 1125,
            750, 375, 375, 188, 188, 188, 188, 750,
            375, 282, 94, 188, 188, 188, 188, 47,47,47,47,47,47,47,47,47,47,47,47,47,47,47,47, 375
        };

    int length = sizeof(notes) / sizeof(notes[0]);
    int i;
    for(i = 0; i < length; i++) {
        if (notes[i] == 0) {
            // Silencio: solo espera el tiempo de la duracin
            simple_delay_ms(durations[i]);
        } else {
            playTone(notes[i] / 4, durations[i] * 10);
        }
    }
}
```

```
void tocar_himne_CDT(void) {
    uint16_t notes[] = {
            329, 392, 523, 0,  // MI, SOL, DO'
            329, 392, 523, 0,  // MI, SOL, DO'
            329, 392, 523, 587, 523, 493, // MI, SOL, DO', RE', DO', SI
            440, 493, 523, 493, 440, 392, // LA, SI, DO', SI, LA, SOL
            349, 392, 440, 392, 349, 329, // FA, SOL, LA, SOL, FA, MI
            293, 329, 370, 392, 370, 392, 440, 493, // RE, MI, FA#, SOL, FA#, SOL, LA, SI
            523, 493, 440, 392, 293, 392, 493, 440,493,440,493,440,493,440,493,440,493,440,493,440,493

        };

    uint16_t durations[] = {
            1125, 375, 375, 1687,
            1125, 375, 375, 1687,
            750, 188, 375, 375, 188, 1125,
            562, 188, 375, 188, 188, 1125,
            562, 188, 375, 188, 188, 1125,
            750, 375, 375, 188, 188, 188, 188, 750,
            375, 282, 94, 188, 188, 188, 188, 47,47,47,47,47,47,47,47,47,47,47,47,47,47,47,47, 375
        };

    int length = sizeof(notes) / sizeof(notes[0]);
    int i;
    for(i = 0; i < length; i++) {
        if (notes[i] == 0) {
            // Silencio: solo espera el tiempo de la duracin
            simple_delay_ms(durations[i]);
        } else {
            playTone(notes[i] / 4, durations[i] * 10);
        }
    }
}
//-------------------------------------------------------------------------------



//-------------------------RUTINAS DE INTERRUPCIN-------------------------
/**
 * Rutina d'interrupcio del timer de gir
 */
void TA0_0_IRQHandler(void)
{
    const uint8_t LED1_ON_OFF = 0x01;
    TA0CCTL0 &= ~CCIFG;
    P1OUT ^= LED1_ON_OFF;

    //Timer de gir
    timer_gir++;
}

/*
 * Rutina d'interrupcio del timer per fer el timeout
 */
void TA1_0_IRQHandler(void)
{
```

```
    TA1CCTL0 &= ~CCIFG;
    contador_T1_TIMEOUT++;
}

/*
 * Interrupcio de la UART2 quan rep informacio
 */
void EUSCIA2_IRQHandler(void)
{
    // we see if the interrupt is related to reception
    EUSCI_A2->IFG &=~ EUSCI_A_IFG_RXIFG; // Clear interrupt
    UCA2IE &= ~UCRXIE; //disabled interrupts in RX

    //UCA2RXBUF estara el dato que volem llegir
    DatoLeido_UART = UCA2RXBUF;

    Byte_recibido_bool=1;
    UCA2IE |= UCRXIE;

}

/*
 * Interrupcio del boto per canviar la referencia
 */
void PORT5_IRQHandler(void){
    uint8_t flag = P5IV; //guardem el vector d'interrupcions i el netegem
    char buffer[64];

    P5IE &= ~(BIT1);

    //Posem referencia a 1 i backflip a 1
    //De normal la referencia estar a 0
    if(referencia == 0){
        referencia = 1;
        backflip = 1;
        sprintf(buffer, "%s", "REF: dreta");
    }else if(referencia == 1){
        referencia = 0;
        backflip = 1;
        sprintf(buffer, "%s", "REF: esquerra");
    }

    halLcdClearLine(4);
    halLcdPrintLine(buffer, 4, NORMAL_TEXT);

    P5IE |= (BIT1);
}

//-------------------------------------------------------------------------


//-----------------------------IMPLEMENTACIONS LLIBRERIA--------------------------------

void set_velocitat(uint16_t velocitat){
    SetVelocitatMotor(MOTOR_ESQUERRA_ID, velocitat ^ 1024);
    SetVelocitatMotor(MOTOR_DRETA_ID, velocitat);
    velocitat_actual=velocitat;
```

```
}

// Moviments
void moure_endavant(void) {
    set_velocitat(velocitat_actual & (~0x400));
    EncendreMotor(MOTOR_ESQUERRA_ID);
    EncendreMotor(MOTOR_DRETA_ID);
    caminant=1;
}

void moure_enrere(void) {
    set_velocitat(velocitat_actual | 0x400);
    EncendreMotor(MOTOR_ESQUERRA_ID);
    EncendreMotor(MOTOR_DRETA_ID);
    caminant=1;
}

void girar_dreta(void){
    SetVelocitatMotor(MOTOR_ESQUERRA_ID, velocitat_actual & (~0x400));
    SetVelocitatMotor(MOTOR_DRETA_ID, velocitat_actual & (~0x400));


    EncendreMotor(MOTOR_ESQUERRA_ID);
    EncendreMotor(MOTOR_DRETA_ID);
}

void girar_esquerra(void){
    SetVelocitatMotor(MOTOR_ESQUERRA_ID, velocitat_actual | 0x400);
    SetVelocitatMotor(MOTOR_DRETA_ID, velocitat_actual | 0x400);


    EncendreMotor(MOTOR_ESQUERRA_ID);
    EncendreMotor(MOTOR_DRETA_ID);
}

void aturar(void){
    SetVelocitatMotor(MOTOR_ESQUERRA_ID, 0 ^ 1024);
    SetVelocitatMotor(MOTOR_DRETA_ID, 0);

    //AturaMotor(MOTOR_ESQUERRA_ID);
    //AturaMotor(MOTOR_DRETA_ID);
    caminant=0;
}

// Sensors
RxReturn llegir_sensor(uint8_t id_sensor) {
    char buffer[64];  // Espacio suficiente para la cadena a imprimir
    byte param[2];
    const char* direccion = "";  // Para guardar el texto correspondiente al sensor

    switch(id_sensor) {
    case 0:
        param[0] = 0x1A;
        param[1] = 0x01;
        direccion = "izq";
        break;
    case 1:
```

```
        param[0] = 0x1B;
        param[1] = 0x01;
        direccion = "frente";
        break;
    case 2:
        param[0] = 0x1C;
        param[1] = 0x01;
        direccion = "der";
        break;
    default:
        direccion = "desconocido";
        break;
    }

    RxReturn rxReturn = ComunicationPacket(0x64, 0x02, 0x02, param);
    sprintf(buffer, "S. %s: %03u", direccion, rxReturn.StatusPacket[5]);
    //Ponemos como numero de lnea el id del sensor para que as cada uno se imprima en una lnea diferen
    halLcdPrintLine(buffer, id_sensor, NORMAL_TEXT);
    return rxReturn;
}


//-------------------------------------------------------------------------------------

#include "msp.h"
#include "lib_PAE.h"
#include "robot_lib.h"


#define CONF_OUTPUT_LED1 0x01  //0000 0001

//Aquests define serveixen per poder canviar els motors rapidament
#define MOTOR_ESQUERRA_ID 2
#define MOTOR_DRETA_ID 4
#define SENSOR_ID 100

//Indiquen ubicacio del robot
#define ESQUERRA 0
#define DRETA 1

//Indiquen distancia del sensor fins a l'obstacle (rang: 0<x<255)
#define CERCANIA_PARET_MAXIM_LIMIT 254
#define CERCANIA_PARET_MAXIM 180
#define CERCANIA_PARET_MITJA 70
#define CERCANIA_PARET_MINIM 20
#define WR_INST_BIT 3

typedef uint8_t byte;

uint8_t inputUSUARI; //Input usuari per saber si ha d'anar esquerra o dreta

/**
 * Funcio copiada del campus; inicialitza el led1.
 * Deixem el led1 perqu est per debugging!
 */
void init_LED1(void){
    const uint8_t CONF_GPIO_LED1 = 0xFE;
```

```
    const uint8_t INIT_STATE_OFF = 0xFE;
    // Configuration of GPIO connection for LED1
    P1SEL0 &= CONF_GPIO_LED1;
    P1SEL1 &= CONF_GPIO_LED1;
    // Configuration of pin0's port 1 as output
    P1DIR |= CONF_OUTPUT_LED1;
    P1OUT &= INIT_STATE_OFF;
}

//-------------------------------------------------------------------------


//-----------------------------------MAIN----------------------------------

void main(void)
  {
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer


    //Inicialtizacions
    init_ucs_24MHz();

    //Init dispositius GPIO
    init_LED1();
    init_botton();
    init_buzzer();

    //Init timers
    init_timer_A0();
    init_timer_A1();

    //Init UART
    init_UART();

    __enable_interrupt();

    //Configura els angles
    byte cw_limit_low[16]  = {6, 0,0,0,0};
    ComunicationPacket(MOTOR_ESQUERRA_ID, 5, WR_INST_BIT, cw_limit_low);
    ComunicationPacket(MOTOR_DRETA_ID, 5, WR_INST_BIT, cw_limit_low);


    //Posa els leds (serveix per debuging)
    byte array_led[2]={25,0};//25 es direccio, 0 o 1 si volem encendre o apagar
    ComunicationPacket(MOTOR_ESQUERRA_ID,2,WR_INST_BIT,array_led); //aquesta encen o apaga el led
    ComunicationPacket(MOTOR_DRETA_ID,2,WR_INST_BIT,array_led); //aquesta encen o apaga el led

    halLcdInit(); //Inicializar y configurar la pantallita
    halLcdClearScreenBkg(); //Borrar la pantalla, rellenando con el color de fondo


    //Aturem el robot per si estava en marxa prviament
         aturar();

    //Inicialment setegem una velocitat dels dos motors, i fem que avanci
    set_velocitat(400);
    moure_endavant();
```

```
    //Cridem el bucle principal
    buclecodiproj();

    //Llegim el sensor en bucle infinit; mai hauriem d'arribar aqui!
    while(1){
        llegir_sensor(0);
        llegir_sensor(1);
        llegir_sensor(2);
        aturar();
    }


}
//---------------------------------------------------------------------------

/**
 * Aquesta funci llegeix els sensors i els imprimeix
 */
void imprimir_sensors(){
    llegir_sensor(0);
    llegir_sensor(1);
    llegir_sensor(2);
}

/*
 * Aquesta funci busca la primera paret (en linia recta). Un cop buscada la primera paret,
 * s'espera un input de l'usuari, i a segons d'aquest input
 * anir cap a l'esquerra o cap a la dreta
 */
int buscar_primera_paret(){
    RxReturn packetsensor_endavant;
    char buffer[64];

    byte paret_trobada = 0;
    //Assumirem que la primera paret est al davant
    while(!paret_trobada){
        packetsensor_endavant= llegir_sensor(1);
        //Ens hem trobat la primera paret que esta endavant
        if(packetsensor_endavant.StatusPacket[5]>CERCANIA_PARET_MAXIM){
            //ATURAREM EL MOTOR I ENS ESPERAREM FINS A UN TEMPS PETIT QUE LUSUARI PUGUI FER INPUT
            aturar();
            Reset_gir_temps();
            while(!control_gir_temps(6)){imprimir_sensors();}
            paret_trobada = 1;
        }
    }
    //A segons de la referencia girarem a l'esquerra o a la dreta
    if(referencia==0){
        executar_gir(DRETA,5);
        sprintf(buffer, "%s", "REF: esquerra");
    }else{
        executar_gir(ESQUERRA,5);
        sprintf(buffer, "%s", "REF: dreta");
    }
```

```
    halLcdClearLine(4);
    halLcdPrintLine(buffer, 4, NORMAL_TEXT);
    tocar_himne_FCB();
    return 0;
}

/*
 * Funcio: girar en la direccio indicada fins que passi la unitat de temps dita
 * POSICIO_GIR: esquerra o dreta (0 o 1)
 * UNITAT TEMPS: unitat de temps que estar executant el gir
 */

void executar_gir(int posicio_gir, int unitat_temps){
    //Tenim dos casos, o be esquerra o be dreta
    if(posicio_gir==ESQUERRA){
        girar_esquerra();
        Reset_gir_temps();
        while(!control_gir_temps(unitat_temps)){imprimir_sensors();}

    }else if(posicio_gir==DRETA){
        girar_dreta();
        Reset_gir_temps();
        while(!control_gir_temps(unitat_temps)){imprimir_sensors();}
    }
}

/*
 * Funcio: girar en la direccio indicada fins que detecti una pared indicada per maxim_sensor
 * o fins que s'esgoti el temps
 *
 * Params: posicio_gir = cap a on volem girar
 *         unitat_temps = quantes unitats de temps ser el MAXIM que girarem
 *         maxim_sensor = fins a on haurem de girar
 *
 */
void executar_gir_sensors(int posicio_gir, int unitat_temps, int maxim_sensor){
    int sensor_no_detectat = 1;
    RxReturn packetsensor_dreta;
    RxReturn packetsensor_esquerra;

    if(posicio_gir==ESQUERRA){
        girar_esquerra();
        Reset_gir_temps();
        while(!control_gir_temps(unitat_temps) && sensor_no_detectat){
            imprimir_sensors();
            packetsensor_dreta=llegir_sensor(2);
            //llegirem els sensors; si estem girant cap a l'esquerra aixo vol
            //dir que voldrem tenir la paret de la dreta
            //Si el sensor de la dreta esta entre [-10 + maxim_sensor, maxim_sensor + 10]
            //aixo vol dir que estarem en un rang acceptable
            if(packetsensor_dreta.StatusPacket[5] > -10 + maxim_sensor && packetsensor_dreta.StatusPac
                sensor_no_detectat = 0;
            }
        }
    }else if(posicio_gir==DRETA){
        girar_dreta();
        Reset_gir_temps();
```

```
        while(!control_gir_temps(unitat_temps) && sensor_no_detectat){
            imprimir_sensors();
            packetsensor_esquerra=llegir_sensor(0);
            //llegirem els sensors; si estem girant cap a la dreta aixo vol dir
            //que voldrem tenir la paret de lesquerra
            //Si el sensor de lesq esta entre [-20 + maxim_sensor, maxim_sensor + 20]
            //aixo vol dir que estarem en un rang acceptable
            if(packetsensor_esquerra.StatusPacket[5] > -10 + maxim_sensor && packetsensor_esquerra.Sta
                sensor_no_detectat = 0;
            }
        }
    }
}


/*
 * Bucle principal del projecte.
 * S'encarrege de rebre els sensors i a segons del seu estat i de la referncia que tinguem
 * cridar la funci de gir adequada
 * Implementaci de pooling bsica, amb if-else.
 */
void buclecodiproj(){

    //Returns de tres sensors
    RxReturn packetsensor_endavant;
    RxReturn packetsensor_esquerra;
    RxReturn packetsensor_dreta;

    //Primer bucle per anar a trobar la pared
    buscar_primera_paret();
    moure_endavant();

    //El primer backflip no el tindrem en compte. El backflip serveix per quan canviem de direcci
    if(backflip){
        backflip = 0;
    }

    //bucle infinit principal del programa
    int i=1;
    while(i){
        //Haurem de llegir ara dos sensors, el del davant i el de la dreta / el de lesquerra
        packetsensor_endavant=llegir_sensor(1);
        packetsensor_esquerra=llegir_sensor(0);
        packetsensor_dreta=llegir_sensor(2);

        //Si hem activat el joystick
        if(backflip){
            //Si ref == 0, aleshores volem paret cap a l'esquerra (i la tenim a la dreta)
            if ( referencia == 0){
                executar_gir_sensors(DRETA, 16, 40);
            }else{ //sino hem de girar cap a l'altre costat
                executar_gir_sensors(ESQUERRA, 16, 40);
            }
            //ja haurem fet el gir aixi que resetegem
            backflip = 0;
        }
```

```
if(caminant){//no hem clicat el boto de parar

    //Anem cap endavant. Aquesta instrucci pot ser una mica redundant ates que a priori
    //sempre que girem posem el moure_endavant
    //Tanmateix la posem aqu per seguretat, ja que sempre que
    //comencem el bucle haurem d'anar cap endavant
    moure_endavant();

    switch(referencia){
    //CASE 0: TENIM UNA PARET A LESQUERRA
    case 0:
        if(packetsensor_endavant.StatusPacket[5]>CERCANIA_PARET_MAXIM){
            //girarem perque haurem de resseguir
            //executar_gir(DRETA,6); //aixo vol dir que s'haura de girar cap a la dreta
            executar_gir(DRETA, 6);
            //continuem movent cap endavant
            moure_endavant();
            Reset_gir_temps();
            while(!control_gir_temps(4)){imprimir_sensors();}
        }
        else if(packetsensor_esquerra.StatusPacket[5]<CERCANIA_PARET_MINIM){
        //Si deixem de tenir la paret a la esquerra
            executar_gir(ESQUERRA,1); //aixo vol dir que s'haura de girar cap a la dreta

            moure_endavant(); //continuem cap endavant
            Reset_gir_temps();
            while(!control_gir_temps(2)){imprimir_sensors();}
        } //Si tenim una paret al davant i a la dreta massa aprop haurem de girar una mica
        else if(packetsensor_esquerra.StatusPacket[5]>CERCANIA_PARET_MITJA & packetsensor_enda
            //girarem perque haurem de resseguir
            //executar_gir(DRETA,4); //aixo vol dir que s'haura de girar cap a l'esquerra

            executar_gir_sensors(DRETA, 4, 40);
            moure_endavant(); //continuem cap endavant
            Reset_gir_temps();
            while(!control_gir_temps(2)){imprimir_sensors();}
        }
        else if(packetsensor_esquerra.StatusPacket[5]>CERCANIA_PARET_MITJA){
            //girarem perque haurem de resseguir
            //executar_gir(DRETA,2); //aixo vol dir que s'haura de girar cap a l'esquerra
            executar_gir_sensors(DRETA, 2, 40);
            //continuem movent cap endavant
            moure_endavant();
            Reset_gir_temps();
            while(!control_gir_temps(3)){imprimir_sensors();}
        }
        break;

    //TENIM UNA PARET A LA DRETA
    case 1:
        if(packetsensor_endavant.StatusPacket[5]>CERCANIA_PARET_MAXIM){
            //girarem perque haurem de resseguir
            //executar_gir(ESQUERRA,6); //aixo vol dir que s'haura de girar cap a l'esquerra
            executar_gir(ESQUERRA, 6);

            //continuem movent cap endavant
            moure_endavant();
```

```
                    Reset_gir_temps();
                    while(!control_gir_temps(4)){imprimir_sensors();}
                }
                else if(packetsensor_dreta.StatusPacket[5]<CERCANIA_PARET_MINIM){
                //Si deixem de tenir la paret a la dreta
                    executar_gir(DRETA,1); //aixo vol dir que s'haura de girar cap a la dreta

                    moure_endavant(); //continuem cap endavant
                    Reset_gir_temps();
                    while(!control_gir_temps(2)){imprimir_sensors();}
                } //Si tenim una paret al davant i a la dreta massa aprop haurem de girar una mica
                else if(packetsensor_dreta.StatusPacket[5]>CERCANIA_PARET_MITJA & packetsensor_endavar
                    //girarem perque haurem de resseguir
                    executar_gir_sensors(ESQUERRA, 4, 40);
                    //aixo vol dir que s'haura de girar cap a l'esquerra

                    moure_endavant(); //continuem cap endavant
                    Reset_gir_temps();
                    while(!control_gir_temps(2)){imprimir_sensors();}
                }
                //Aixo vol dir que hem pillat una paret a la dreta molt, aixi q toca girar cap
                //a l'altra banda
                else if(packetsensor_dreta.StatusPacket[5]>CERCANIA_PARET_MITJA){
                    //girarem perque haurem de resseguir
                    executar_gir_sensors(ESQUERRA, 2, 40);
                    //aixo vol dir que s'haura de girar cap a l'esquerra

                    //continuem movent cap endavant
                    moure_endavant();
                    Reset_gir_temps();
                    while(!control_gir_temps(3)){imprimir_sensors();}
                }
                break;
            case 2:
                break;
            }
        }else{ //caminant==0;
            aturar();
        }
    };
}
```

8 Annex: codi RTOS

```
#include "msp.h"
#include "lib_PAE.h"
#include "robot_lib.h"
#include "stdio.h"
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"
#include "semphr.h"
#include "queue.h"


////////////////////////////////////////////////////////////////////////
/*
 * A continuacio es definiran totes les constants globals
 *
 */

//UART i queue de comunicacio
#define UART_TX_QUEUE_LENGTH 64
#define UART_RX_QUEUE_LENGTH 64
QueueHandle_t uartTxQueue = NULL;
QueueHandle_t RxQueue = NULL;

//Indiquen distancia del sensor fins a l'obstacle (rang: 0<x<255)
#define CERCANIA_PARET_MAXIM_LIMIT 254
#define CERCANIA_PARET_MAXIM 200
#define CERCANIA_PARET_MITJA_MAXIM 100
#define CERCANIA_PARET_MITJA 60
#define CERCANIA_PARET_MINIM 30

//Indiquen ubicacio del robot
#define ESQUERRA 0
#define DRETA 1

//MOTORS
#define MOTOR_ESQUERRA_ID 2
#define MOTOR_DRETA_ID 3
#define SENSOR_ID 100

//Numero d'instruccio que enviem a la UART PER ESCRIURE
#define WR_INST_BIT 3

//Comunicacio bare metal (en desus)
#define TXDx_READY (UCA0IFG & UCTXIFG) // interrupt flag reg and flag de tx
#define UCAxTXBUF_SELECTOR UCA0TXBUF
#define Port_UARTx_SEL0 P1SEL0
#define Port_UARTx_SEL1 P1SEL1
#define DIR_UART BIT0
#define UCAxSTATW_SELECTOR UCA0STATW //UCA2 en el real
#define Sentit_Dades_PORT_SELECTOR P1OUT //P3OUT en el real

//Comunicacio UART: EMULADOR VS VIDA REAL
#define EUSCI_AN_BASE EUSCI_A2_BASE
#define INT_EUSCIAN INT_EUSCIA2
#define UART_GPIO_PORT_DADES GPIO_PORT_P3
```

```
#define CONF_OUTPUT_LED1 0x01  //0000 0001

typedef uint8_t byte;

//PEL TIMER (no utilitzat al final)
uint8_t xByteTimeoutFlag = 0; // Global flag for timeout
static const TickType_t delay = 200; //Delay pel timer
static TimerHandle_t one_shot_timer = NULL; //El handler

//Algunes constants utils de debugging
uint8_t flag;

//Velocitat actual del robot
uint16_t velocitat_actual;

byte referencia = 0; //Esquerra = 0; dreta = 1; deafault = 2
byte caminant = 1; //Parat = 0; caminant = 1
byte backflip = 0; //0 no gir; 1 gir

//Constants per les tasques

// Semafor per comprova si sha executat la taska de init
SemaphoreHandle_t finishInitTaskSemaphore; //aquest de moment no sutilitza

// Els sizes (en bytes) de cada tasca
#define TEST_TASK_STACK_SIZE 512
#define BUCLE_TASK_STACK_SIZE 2048

// Define the priority of the task (nota: ms proper a 0 ms baix. tskIdle te la mateixa)
#define TEST_TASK_PRIORITY tskIDLE_PRIORITY + 1
#define BUCLE_TASK_PRIORITY tskIDLE_PRIORITY + 2

// Queue per comunicacio entre tasques (hi hauran sensors)
#define IMP_QUEUE_LENGTH 1 //posem una per no llegir dades erronees
#define CTR_QUEUE_LENGTH 1
#define GIR_QUEUE_LENGHT 1
QueueHandle_t impQueue = NULL;
QueueHandle_t girQueue = NULL;
QueueHandle_t controlQueue = NULL;




////////////////////////////////////////////////////////////////////////
/*
 *  A continuacio hi ha el codi pel main + les tasques
 *  Es a dir, en el projecte aixo equivaldria a tot el main
 */


//------------------------------TASQUES I FUNCIONS RELATIVES---------------------------------
/*
 * Tasca inicial, que inicia el robot i crida la funcio de buscar parets.
 */
void vInitTask(void *pvParameters)
{
```

```
    //Recollim els parmetres
    int taskParameter;
    taskParameter = ( int ) pvParameters;

    //init_buzzer();

    // Small delay to let everything initialize (sincerament nose si aixo fa alguna cosa util)
    vTaskDelay(pdMS_TO_TICKS(100));

    aturar();

    moure_endavant();

    // Inicialitzem les ques que utilitzarem per les tasques
    impQueue = xQueueCreate(IMP_QUEUE_LENGTH, sizeof(RxReturn) * 3);
    controlQueue = xQueueCreate(CTR_QUEUE_LENGTH, sizeof(RxReturn) * 3);
    girQueue = xQueueCreate(GIR_QUEUE_LENGHT, sizeof(RxReturn) * 3);

    // Configura els angles
    byte cw_limit_low[16]  = {6, 0,0,0,0};
    ComunicationPacket(MOTOR_ESQUERRA_ID, 5, WR_INST_BIT, cw_limit_low);
    ComunicationPacket(MOTOR_DRETA_ID, 5, WR_INST_BIT, cw_limit_low);

    //Inicialment setegem una velocitat dels dos motors, i fem que avanci
    set_velocitat(350);
    moure_endavant();

    // Enceme led per debugging
    byte array_led[2]={25,1};//25 es direccio, 0 o 1 si volem encendre o apagar
    ComunicationPacket(MOTOR_ESQUERRA_ID,2,3,array_led); //aquesta encen el led

    // Ara cridem la funcio de primera paret
    buscar_primera_paret();

    moure_endavant(); //tirem cap endavant un altre cop

    //indiquem amb un semafor que ja hem acabat, i poden comenar les altres tasques
    xSemaphoreGive(finishInitTaskSemaphore);

    vTaskDelete(NULL); // Ens carreguem la tasca
}

/*
 * Tasca per llegir sensors. Posa a les ques corresponents
 per altres tasques la informacio dels sensors
 */
void vLlegirSensors(void *pvParameters){
    //Llegirem els sensors (de moment tambe els imprimim);
    RxReturn izq;
    RxReturn endavant;
    RxReturn dret;
    RxReturn sensors[3];

    while(1){
        izq = llegir_sensor(0);
        endavant = llegir_sensor(1);
        dret = llegir_sensor(2);
```

```
        //Fem un array de RxReturn
        sensors[0] = izq;
        sensors[1] = endavant;
        sensors[2] = dret;

        //Ho posem a la cua overwriting (si no ho fem, i la cua ja te un item, estariem bloquejant!)
        xQueueOverwrite( impQueue, &sensors);
        xQueueOverwrite( controlQueue, &sensors);
        xQueueOverwrite( girQueue, &sensors);

    }
}

/*
 * Taska per imprimir els sensors. Rep a traves de la cua la informacio de sensors
 */
void vImprimirSensors(void *pvParameters){
    char buffer[64];
    RxReturn packets[3];
    const char *direccion[3] = {"Esq", "End", "Dreta"}; // Direccions
    int i;

    // Bucle infinit on mai sortirem
    while(1){
        // Quan rebem de la cua imprimim
        if (xQueueReceive(impQueue, &packets, pdMS_TO_TICKS(50)) == pdTRUE) {
            for (i = 0; i < 3; i++) {
                // Access sensor ID from packets[i].StatusPacket[1]
                // Access sensor value from packets[i].StatusPacket[4]
                sprintf(buffer, "S. %s: %03u", direccion[i], packets[i].StatusPacket[4]);

                halLcdPrintLine(buffer, i, NORMAL_TEXT); // La i actua com a number de linia
            }
        }
    }
}

// Tasca per imprimir l'himne del barca. Crida la funcio apropiada senzillament
void vHimneBarcaTask(void *pvParameters){
    while(1){
        tocar_himne_FCB();
        vTaskDelay(pdMS_TO_TICKS(10000)); //10s de delay aprox
    }
}

/*
 * Bucle principal del projecte.
 * S'encarrege de rebre els sensors i a segons del seu estat i
 * de la referncia que tinguem cridar la funci de gir adequada
 * Implementaci de pooling bsica, amb if-else.
 */
void vBucleCodiProjecteTask(void *pvParameters){

    //Returns de tres sensors
    RxReturn packetsensor_endavant;
    RxReturn packetsensor_esquerra;
```

```
RxReturn packetsensor_dreta;
RxReturn packets[3];

//El primer backflip no el tindrem en compte. El backflip serveix per quan canviem de direcci
if(backflip){
    backflip = 0;
}

//bucle infinit principal del programa

while(1){
    // Rebem de la cua.
    if (xQueueReceive(controlQueue, &packets, pdMS_TO_TICKS(50)) == pdTRUE) {

        //Copiem la dada dels sensors
        packetsensor_esquerra = packets[0];
        packetsensor_endavant = packets[1];
        packetsensor_dreta = packets[2];

        //Si hem activat el joystick
        if(backflip){
            //Si ref == 0, aleshores volem paret cap a l'esquerra (i la tenim a la dreta)
            if ( referencia == 0){
                //executar_gir_sensors(DRETA, 14000, 40);
                executar_gir_sensors(DRETA,4000,40);
            }else{ //sino hem de girar cap a l'altre costat
                //executar_gir_sensors(ESQUERRA, 14000, 40);
                executar_gir_sensors(ESQUERRA,4000,40);
            }
            //ja haurem fet el gir aixi que resetegem
            backflip = 0;
        }

        if(caminant){//no hem clicat el boto de parar

            //Anem cap endavant. Aquesta instrucci pot ser una mica redundant
            //ates que a priori sempre que girem posem el moure_endavant
            //Tanmateix la posem aqu per seguretat, ja que sempre
            //que comencem el bucle haurem d'anar cap endavant
            moure_endavant();

            switch(referencia){
            //CASE 0: TENIM UNA PARET A LESQUERRA
            case 0:
                if(packetsensor_endavant.StatusPacket[4]>CERCANIA_PARET_MAXIM){
                    //girarem perque haurem de resseguir
                    //executar_gir(DRETA,6); //aixo vol dir que s'haura de girar cap a la dreta
                    executar_gir(DRETA, 1500);
                    //continuem movent cap endavant
                    moure_endavant();
                    vTaskDelay(pdMS_TO_TICKS(200));
                }
                else if(packetsensor_esquerra.StatusPacket[4]<CERCANIA_PARET_MINIM){
                //Si deixem de tenir la paret a la esquerra
                    executar_gir(ESQUERRA,200);
                    //aixo vol dir que s'haura de girar cap a la dreta
```

```
                moure_endavant(); //continuem cap endavant
                vTaskDelay(pdMS_TO_TICKS(200));
            }
            //Si tenim una paret al davant i a la dreta massa aprop haurem de girar una mica
            else if(packetsensor_esquerra.StatusPacket[4]>CERCANIA_PARET_MITJA & packetsensor_
                //girarem perque haurem de resseguir

                //aixo vol dir que s'haura de girar cap a l'esquerra
                executar_gir_sensors(DRETA, 2000, 50);
                moure_endavant(); //continuem cap endavant
                vTaskDelay(pdMS_TO_TICKS(250));
            }
            else if(packetsensor_esquerra.StatusPacket[4]>CERCANIA_PARET_MITJA_MAXIM){
                //girarem perque haurem de resseguir
                //aixo vol dir que s'haura de girar cap a l'esquerra
                executar_gir_sensors(DRETA, 1000, 50);
                //continuem movent cap endavant
                moure_endavant();
                vTaskDelay(pdMS_TO_TICKS(250));
            }
            break;

        //TENIM UNA PARET A LA DRETA
        case 1:
            if(packetsensor_endavant.StatusPacket[4]>CERCANIA_PARET_MAXIM_LIMIT){
                //girarem perque haurem de resseguir
                //aixo vol dir que s'haura de girar cap a l'esquerra
                executar_gir(ESQUERRA, 1500);

                //continuem movent cap endavant
                moure_endavant();
                vTaskDelay(pdMS_TO_TICKS(200));
            }
            else if(packetsensor_dreta.StatusPacket[4]<CERCANIA_PARET_MINIM){
            //Si deixem de tenir la paret a la dreta
                executar_gir(DRETA,200);
                //aixo vol dir que s'haura de girar cap a la dreta

                moure_endavant(); //continuem cap endavant
                vTaskDelay(pdMS_TO_TICKS(200));
            } //Si tenim una paret al davant i a la dreta massa aprop
            //haurem de girar una mica
            else if(packetsensor_dreta.StatusPacket[4]>CERCANIA_PARET_MITJA & packetsensor_end
                //girarem perque haurem de resseguir
                //aixo vol dir que s'haura de girar cap a l'esquerra
                executar_gir_sensors(ESQUERRA, 2000, 50);

                moure_endavant(); //continuem cap endavant
                vTaskDelay(pdMS_TO_TICKS(250));
            } //Aixo vol dir que hem pillat una paret a la dreta molt,
            //aixi q toca girar cap a l'altra banda
            else if(packetsensor_dreta.StatusPacket[4]>CERCANIA_PARET_MITJA_MAXIM){
                //girarem perque haurem de resseguir
                //aixo vol dir que s'haura de girar cap a l'esquerra
                executar_gir_sensors(ESQUERRA, 1000, 50);

                //continuem movent cap endavant
```

```
                        moure_endavant();
                        vTaskDelay(pdMS_TO_TICKS(250));
                    }
                    break;
                case 2:
                    break;
                }
            }else{ //caminant==0;
                aturar();
            }
        }
        xQueueReset(controlQueue);
        //vTaskDelay(pdMS_TO_TICKS(1));
    };
}

/*
 * Taska de spawner. Aquesta tasca s'inicia just despres
 * de que acabi la primera tasca, i creara totes les altres
 */
void vSpawnerTask(void *pvParameters)
{
    if (xSemaphoreTake(finishInitTaskSemaphore, portMAX_DELAY) == pdTRUE) {
        // Creem la resta de taskes

        // pel bucle principal
        xTaskCreate(vBucleCodiProjecteTask, "BucleCodi", BUCLE_TASK_STACK_SIZE,
        (void *)1, BUCLE_TASK_PRIORITY, NULL);

        // per llegir sensors
        xTaskCreate(vLlegirSensors, "LlegirSensors", TEST_TASK_STACK_SIZE,
        (void *)1, TEST_TASK_PRIORITY, NULL);

        //Imprimir sensors
        xTaskCreate(vImprimirSensors, "ImprimirSensors", TEST_TASK_STACK_SIZE,
        (void *)1, TEST_TASK_PRIORITY, NULL);

        // Best feature (himne del barca)
        //xTaskCreate(vHimneBarcaTask, "HimneBarca", TEST_TASK_STACK_SIZE,
        (void *)1, TEST_TASK_PRIORITY, NULL);

        // Eliminem la tasca que no necessitem
        vTaskDelete(NULL);
    } else {
        // NO HAURIA DARRIBAR AQUI
        while(1);
    }
}

//------------------------------MAIN-----------------------------------

/*
 * MAIN: inicia el programa, iniciant els perfierics necessaris
 */
void tasques(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;      // Stop watchdog timer (tot i que ja s'ha aturat p
```

```
    //Inicialtizacions: millor fer-les aqui i no en un una tasca pel tema de clocks i timers
    prvConfigureClocks();
    init_uart0();
    //Aquest de fet no fa falta pero lhavia utilitzat previament per provar una cosa
    initRxTimeoutTimer();
    init_botton();

    __enable_interrupt(); //activem les interrupcions globalment

    //Creem el semafor ABANS de les tasques
    finishInitTaskSemaphore = xSemaphoreCreateBinary();

    halLcdInit(); //Inicializar y configurar la pantallita
    halLcdClearScreenBkg(); //Borrar la pantalla, rellenando con el color de fondo

    // Ara crearem les tasques necessaries per comencar

    xTaskCreate(vInitTask,       /* Function that implements the task. */
                        "InitTask",      /* Text name for the task. */
                        TEST_TASK_STACK_SIZE, /* Stack size in words, not bytes. */
                        ( void * ) 1,    /* Parameter passed into the task. */
                        TEST_TASK_PRIORITY, /* Priority at which the task is created. */
                        NULL );

    // Spawnejara les altres tasques! Quan hagi acabat la init
    xTaskCreate(vSpawnerTask,
                "SpawnerTask",
                TEST_TASK_STACK_SIZE, /* Stack size in words, not bytes. */
                ( void * ) 1,    /* Parameter passed into the task. */
                TEST_TASK_PRIORITY, /* Priority at which the task is created. */
                NULL );

    vTaskStartScheduler();

    while(1); //mai shauria darribar aqui

}

//-------------------------------------Funcions que implementen logica del robot---------------
/*
 * Aquesta funci busca la primera paret (en linia recta). Un cop buscada la primera paret,
 * s'espera un input de l'usuari, i a segons d'aquest input
 * anir cap a l'esquerra o cap a la dreta
 */
void buscar_primera_paret(){
    RxReturn packetsensor_endavant;

    byte paret_trobada = 0;
    //Assumirem que la primera paret est al davant
    while(!paret_trobada){
        //No llegirem encara de la taska de sensors perque es abans de tot el programa
        packetsensor_endavant = llegir_sensor(1);
        //Ens hem trobat la primera paret que esta endavant
        if(packetsensor_endavant.StatusPacket[4]>CERCANIA_PARET_MAXIM){
            //ATURAREM EL MOTOR I ENS ESPERAREM FINS A UN TEMPS PETIT QUE LUSUARI PUGUI FER INPUT
            aturar();
```

```
            vTaskDelay(pdMS_TO_TICKS(4000)); //5 segons
            paret_trobada = 1;
        }
        vTaskDelay(pdMS_TO_TICKS(10));
    }
    //A segons de la referencia girarem a l'esquerra o a la dreta
    if(referencia==0){
        executar_gir(DRETA,2000);
    }else{
        executar_gir(ESQUERRA,2000);
    }
}
/*
 * Funcio: girar en la direccio indicada fins que passi la unitat de temps dita
 * POSICIO_GIR: esquerra o dreta (0 o 1)
 * UNITAT TEMPS: unitat de temps que estar executant el gir (en milisegons!
 */

void executar_gir(int posicio_gir, int unitat_temps){
    //Tenim dos casos, o be esquerra o be dreta
    if(posicio_gir==ESQUERRA){
        girar_esquerra();
        vTaskDelay(pdMS_TO_TICKS(unitat_temps));

    }else if(posicio_gir==DRETA){
        girar_dreta();
        vTaskDelay(pdMS_TO_TICKS(unitat_temps));
    }
}

/*
 * Funcio: girar en la direccio indicada fins que detecti una pared indicada
 * per maxim_sensor o fins que s'esgoti el temps
 * Siguent estrictes crec que lo mes eficient aqui no seria rebre altre cop
 * els RxReturn packets dels sensors, sino passarho per parametre
 * Faig aixi per practicar una mica de dles cues i per veure el tema de la concurrencia
 * i com de rapid va
 *
 * Params: posicio_gir = cap a on volem girar
 *         unitat_temps = quantes unitats de temps ser el MAXIM que girarem
 *         maxim_sensor = fins a on haurem de girar
*/

void executar_gir_sensors(int posicio_gir, int unitat_temps, int maxim_sensor){
    int sensor_no_detectat = 1;
    RxReturn packetsensor_dreta;
    RxReturn packetsensor_esquerra;
    RxReturn packets[3];
    TickType_t start_time;

    if (xQueueReceive(controlQueue, &packets, pdMS_TO_TICKS(100)) == pdTRUE){
        packetsensor_dreta = packets[2];
        packetsensor_esquerra = packets[0];
        start_time = xTaskGetTickCount(); // Agafem els "ticks" de la tasca

        if(posicio_gir==ESQUERRA){
            girar_esquerra();
```

```
            // Aqui estem restant els ticks que tenim de la tasca - els ticks inicials
            while (sensor_no_detectat && (xTaskGetTickCount() - start_time < pdMS_TO_TICKS(unitat_temp
                // Si el sensor de la dreta esta entre [maxim_sensor - 10, maxim_sensor + 10]

                if (xQueueReceive(controlQueue, &packets, pdMS_TO_TICKS(50)) == pdTRUE){
                    packetsensor_dreta = packets[2];
                    if (packetsensor_dreta.StatusPacket[4] > maxim_sensor - 30 && packetsensor_dreta.S
                        sensor_no_detectat = 0;
                    }
                }
                vTaskDelay(pdMS_TO_TICKS(1)); // Per evitar busy-waiting
            }

        }else if(posicio_gir==DRETA){
            girar_dreta();

            while (sensor_no_detectat && (xTaskGetTickCount() - start_time < pdMS_TO_TICKS(unitat_temp
                // Si el sensor de l'esquerra esta entre [maxim_sensor - 10, maxim_sensor + 10]
                if (xQueueReceive(controlQueue, &packets, pdMS_TO_TICKS(50)) == pdTRUE){
                    packetsensor_esquerra = packets[0];
                    if (packetsensor_esquerra.StatusPacket[4] > maxim_sensor - 30 && packetsensor_esqu
                        sensor_no_detectat = 0;
                    }
                }
                vTaskDelay(pdMS_TO_TICKS(1)); // evitar Busy-waiting
            }
        }
    }
}

/*
 *
 */

//-------------------------------------------Funcions del himne del barca--------------------------

/*
 * Funcio que fa que el freeRTOS tingui un delay dels microsegons seleccionats
 * S'ha mantingut per mantenir la coherencia amb la prctica per a
 * efectes prctics tb es podria haver fet amb vTaskDelay.
 */
static void freertos_delay_us(uint32_t microseconds) {
    TickType_t ticks_to_delay;
    // Approximate: microseconds / (1000000 / ticks per second)
    ticks_to_delay = (microseconds * configTICK_RATE_HZ) / 1000000;
    if (ticks_to_delay > 0) {
        vTaskDelay(ticks_to_delay);
    }
}

// Les segents funcions toquen l'himne; estan explicades a l'altre docu o al pdf d'entrega

void playTone(uint32_t freq, uint32_t duration_ms) {
    uint32_t period_us = 1000000 / freq;
    uint32_t on_time = period_us / 4;        // 25% encendido
    uint32_t off_time = period_us - on_time; // 75% apagado
```

```
    uint32_t cycles = (duration_ms * 1000) / period_us;

    uint32_t i;
    for(i = 0; i < cycles; i++) {
        P2->OUT |= BIT7;
        freertos_delay_us(on_time);
        P2->OUT &= ~BIT7;
        freertos_delay_us(off_time);
    }
}

void tocar_himne_FCB(void) {
    uint16_t notes[] = {
            329, 392, 523, 0,  // MI, SOL, DO'
            329, 392, 523, 0,  // MI, SOL, DO'
            329, 392, 523, 587, 523, 493, // MI, SOL, DO', RE', DO', SI
            440, 493, 523, 493, 440, 392, // LA, SI, DO', SI, LA, SOL
            349, 392, 440, 392, 349, 329, // FA, SOL, LA, SOL, FA, MI
            293, 329, 370, 392, 370, 392, 440, 493, // RE, MI, FA#, SOL, FA#, SOL, LA, SI
            523, 493, 440, 392, 293, 392, 493, 440,493,440,493,440,493,440,493,440,493,440,493,440,493

        };

    uint16_t durations[] = {
            1125, 375, 375, 1687,
            1125, 375, 375, 1687,
            750, 188, 375, 375, 188, 1125,
            562, 188, 375, 188, 188, 1125,
            562, 188, 375, 188, 188, 1125,
            750, 375, 375, 188, 188, 188, 188, 750,
            375, 282, 94, 188, 188, 188, 188, 47,47,47,47,47,47,47,47,47,47,47,47,47,47,47, 375
        };

    int length = sizeof(notes) / sizeof(notes[0]);
    int i;
    for(i = 0; i < length; i++) {
        if (notes[i] == 0) {
            // Silencio: solo espera el tiempo de la duracion
            vTaskDelay(durations[i]); //task delay ja treballa amb milisegons ossigui que no fa falta
        } else {
            playTone(notes[i] / 4, durations[i] * 10);
        }
    }
}


void tocar_himne_CDT(void) {
    uint16_t notes[] = {
            329, 392, 523, 0,  // MI, SOL, DO'
            329, 392, 523, 0,  // MI, SOL, DO'
            329, 392, 523, 587, 523, 493, // MI, SOL, DO', RE', DO', SI
            440, 493, 523, 493, 440, 392, // LA, SI, DO', SI, LA, SOL
            349, 392, 440, 392, 349, 329, // FA, SOL, LA, SOL, FA, MI
            293, 329, 370, 392, 370, 392, 440, 493, // RE, MI, FA#, SOL, FA#, SOL, LA, SI
            523, 493, 440, 392, 293, 392, 493, 440,493,440,493,440,493,440,493,440,493,440,493,440,493

        };
```

```
    uint16_t durations[] = {
            1125, 375, 375, 1687,
            1125, 375, 375, 1687,
            750, 188, 375, 375, 188, 1125,
            562, 188, 375, 188, 188, 1125,
            562, 188, 375, 188, 188, 1125,
            750, 375, 375, 188, 188, 188, 188, 750,
            375, 282, 94, 188, 188, 188, 188, 47,47,47,47,47,47,47,47,47,47,47,47,47,47,47,47, 375
        };

    int length = sizeof(notes) / sizeof(notes[0]);
    int i;
    for(i = 0; i < length; i++) {
        if (notes[i] == 0) {
            // Silencio: solo espera el tiempo de la duracio
            simple_delay_ms(durations[i]);
        } else {
            playTone(notes[i] / 4, durations[i] * 10);
        }
    }
}
//------------------------------------------------------------------------------


////////////////////////////////////////////////////////////////////////////
/*
 * A continuacio hi ha les funcions de comunicacio: TX, RX i semblants
 *
 */

//-----------------------------------TXCOMUNICACIO--------------------------------

//ELS MOTORS TENEN ID 1,2 I EL SENSOR TE ID 100, i 3 EN EL SIMULADOR

//TxPacket() 3 parmetres: ID del Dynamixel, Mida dels parmetres, Instruction byte. torna la mida del '
byte TxPacket(byte bID, byte bParameterLength, byte bInstruction, byte Parametros[16]) {

    char error[] = "adr. no permitida";
    if ((Parametros[0] < 6) && (bInstruction == 3)){
        //emitir mensaje de error de direccion prohibida:
          //halLcdPrintLine(error, 8, INVERT_TEXT);
          //y salir de la funcion sin mas:
          return 0;
    }

    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 mdul al que volem enviar el missatge
```

```
    TxBuffer[3] = bParameterLength+2;
    //Length(Parameter,Instruction,Checksum)
    TxBuffer[4] = bInstruction;
    //Instrucci que enviem al Mdul

    for(bCount = 0; bCount < bParameterLength; bCount++) {
    //Comencem a generar la trama que hem denviar
        TxBuffer[bCount+5] = Parametros[bCount];
    }

    bCheckSum = 0;
    bPacketLength = bParameterLength+4+2;
    //Aixo hauria de ser +3. EL +3 fa per ID; LENGHT, INSTRUCTION.
    //El +2 es pels 2 byts dinici de trama

    //el bcount comena per 2 xk els dos primers bytes son d'inici de trama.
    for(bCount = 2; bCount < bPacketLength-1; bCount++) {   //Clcul del checksum
        bCheckSum += TxBuffer[bCount];
    }

    // Posem tot el paquet en una cua!
    TxBuffer[bCount] = ~bCheckSum;
    for (bCount = 1; bCount < bPacketLength; bCount++) {
         if (xQueueSend(uartTxQueue, &TxBuffer[bCount], pdMS_TO_TICKS(40)) != pdPASS) {
             return 0; //error que no hauriem de trobar
         }
     }

    // Enable the transmit interrupt - aixo ens portara a la ISR automaticament,
    // un cop hem transmes el primer byte
    MAP_UART_enableInterrupt(EUSCI_AN_BASE, EUSCI_A_UART_TRANSMIT_INTERRUPT);
    MAP_UART_transmitData(EUSCI_AN_BASE, TxBuffer[0]);


    vTaskDelay(pdMS_TO_TICKS(1)); // Small delay, util sovint per que no es sobreescriguin coses

    //Posem la lnia de dades en Rx perqu el mdul Dynamixel envia resposta
    //MAP_UART_enableInterrupt(EUSCI_AN_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT);
    //UCA2IFG |= EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG;

    return(bPacketLength);
}

//-------------------------------------CANVI DE SENTIT COMUNICACIO-------------------------

//Configuraci del Half Duplex dels motors: Recepci
void Sentit_Dades_Rx(void) {
    MAP_GPIO_setOutputLowOnPin(UART_GPIO_PORT_DADES, GPIO_PIN0); // El posem a 0
    // Equivalent a P3OUT &=~BIT0;
    //P3OUT &=~BIT0;
}

//Configuraci del Half Duplex dels motors: Transmissi
void Sentit_Dades_Tx(void) {
    MAP_GPIO_setOutputHighOnPin(UART_GPIO_PORT_DADES, GPIO_PIN0); // Inicialitzem port P1.0 a 0 (Rx)
    //Equivlaent a P3OUT |= BIT0;
    //P3OUT |= BIT0;
```

```
}

/* funci TxUACx(byte): envia un byte de dades per la UART
Esta en desus! Ara ho fem des de la rutina dinterrupcio

void TxUACx(uint8_t bTxdData) {
    //Equivalent a while(!TXDx_READY); UCAxTXBUF_SELECTOR = bTxdData;
    MAP_UART_transmitData(EUSCI_A0_BASE,bTxdData);
}
*/

//-------------------------------------------RXCOMUNICACIO-----------------------------

/*
 * RxPacket. Compte, que per algun motiu no rebem el primer byte (un dels 0xff)
 * i per tant hem hagut d'ajustar alguns numeros.
 */
RxReturn RxPacket() {
    RxReturn respuesta;
    byte bCount, bLenght, bChecksum;
    byte check_sum = 0;
    respuesta.error = 0;
    respuesta.byteTimeOut = 0;
    // Inicialitzem a 0; aixo es per debugging i per controlar quines coses estem sobreescrivint
    respuesta.StatusPacket[0]=0;
    respuesta.StatusPacket[1]=0;
    respuesta.StatusPacket[2]=0;
    respuesta.StatusPacket[3]=0;
    respuesta.StatusPacket[4]=0;
    respuesta.StatusPacket[5]=0;
    respuesta.StatusPacket[6]=0;

    uint8_t receivedByte;
    BaseType_t xResult;

        // Read the first 4 bytes (header, ID, length, instruction/error)
        for (bCount = 0; bCount < 4; bCount++) {
            //xTimerStart(one_shot_timer, portMAX_DELAY); Per ferho amb timers

            // Agafem de la cua
            xResult = xQueueReceive( RxQueue, &receivedByte, pdMS_TO_TICKS(50) );

            if( xResult == pdPASS ) {
                respuesta.StatusPacket[bCount] = receivedByte; //Posem el byte
            } else {
                respuesta.byteTimeOut = 1;
                break;
            }
        }

        // If the header was received without timeout, read the rest of the packet
        if (!respuesta.byteTimeOut) {
            bLenght = respuesta.StatusPacket[2]; // Length byte; el [3] s per l'error

            // Read the parameters and checksum
            for (bCount = 0; bCount < bLenght-1; bCount++) {
                //xTimerStart(one_shot_timer, portMAX_DELAY); (per ferho amb timers)
```

```
                // Ojo el pdMS!! Si posem infinit es queda encallat si hi ha algun error
                xResult = xQueueReceive( RxQueue, &receivedByte, pdMS_TO_TICKS(50) );

                if( xResult == pdPASS ) {
                    respuesta.StatusPacket[bCount+4] = receivedByte;
                } else {
                    respuesta.byteTimeOut = 1;
                    break;
                }
                if (respuesta.byteTimeOut) break;
                // If timeout occurred for a data byte, exit the data loop
            }

            // Checksum
            if (!respuesta.byteTimeOut) {
                check_sum = 0;
                check_sum += respuesta.StatusPacket[1]; // ID
                check_sum += respuesta.StatusPacket[2]; // Length

                for (bCount = 0; bCount < bLenght-1 ; bCount++) { // Exclude checksum byte
                    check_sum += respuesta.StatusPacket[bCount + 3]; // Parameters
                }
                check_sum = ~check_sum;

                if (check_sum != respuesta.StatusPacket[bLenght + 3 - 1] || respuesta.StatusPacket[3]
                    respuesta.error = 1;
                } else {
                    respuesta.error = 0;
                }
            }
        }
        // Reset queu; aixo va be per netejar el que tinguem si hi ha hagut algun error
        xQueueReset(RxQueue);
        //xByteTimeoutFlag = 0; (per ferho amb timers)
        return respuesta;

}

/*
 * Comunicatin packet que serveix basicament per cridar el TxPacket i
 * retornar el packet de retorn fins que no i hagi cap error.
 */
RxReturn ComunicationPacket(byte bID, byte bParameterLength, byte bInstruction, byte Parametros[16]) {
    TxPacket(bID,bParameterLength,bInstruction,Parametros);
    RxReturn retornpacket= RxPacket();

    //Fem bucle fins que no hi hagi error
    while(retornpacket.error==1 || retornpacket.byteTimeOut==1){
        TxPacket(bID,bParameterLength,bInstruction,Parametros);
        retornpacket= RxPacket();
    }
    return retornpacket;
}
////////////////////////////////////////////////////////////////////////////
/*
 * A continuacio hi han les inicialitazions dalguns periferics o sistemes
```

```
 * Hi ha la inicialitzacio del timer, la dels clocks, la de la UART, la de un polsador.
 */

//------------------------------BOTO CONFIG-------------------------------------

void init_botton(void){
   // Aixo d'aqui es equivale a posar els P5SEL0, P5SEL1, P5DIR, P5REN i P5OUT
   MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P5, GPIO_PIN1);

   // Equivalent a P5IE |=
   MAP_GPIO_enableInterrupt(GPIO_PORT_P5, GPIO_PIN1);

   // Equivalent a posar el P5IES &=
   MAP_GPIO_interruptEdgeSelect(GPIO_PORT_P5, GPIO_PIN1, GPIO_LOW_TO_HIGH_TRANSITION);

   // Equivalent a P5IFG = 0
   MAP_GPIO_clearInterruptFlag(GPIO_PORT_P5, GPIO_PIN1);

   // Equivalent als les NVIC -> ICPR[1] i ISER[1].
   MAP_Interrupt_enableInterrupt(INT_PORT5);
}

//------------------------------TIMERS-------------------------------------
//Configuracio de timers per fer proves. Son timers de SOFTWARE del RTOS.
// No al final no els he utilitzat, pero funcionen :)

// Funcio que criadara quan el timer estigui
void raiseFlagRxPacket( TimerHandle_t pxTimer)
{
    /* The timer expired before a byte was received. Set the timeout flag. */
    xByteTimeoutFlag = 1;
}

// Init del timer
void initRxTimeoutTimer()
{
    // Create a one-shot timer
      one_shot_timer = xTimerCreate(
                          "One-shot timer",      // Name of timer
                          delay,             // Period of timer (in ticks)
                          pdFALSE,               // Auto-reload
                          (void *)0,             // Timer ID
                          raiseFlagRxPacket);  // Callback function
}
//------------------------------CLOCKS-------------------------------------

/*
 * Funcio que configura clocks. Esta agafada d'algun lloc, i he twekeat par de parametres.
 * De totes maneres crec que la de la lib de PAE ja funcionaria prou be
 */
void prvConfigureClocks( void )
{
    /* Set Flash wait state for high clock frequency.  Refer to datasheet for
    more details. */
    FlashCtl_setWaitState( FLASH_BANK0, 2 );
    FlashCtl_setWaitState( FLASH_BANK1, 2 );
```

```
    /* The full demo configures the clocks for maximum frequency, whereas the
    blinky demo uses a slower clock as it also uses low power features.  Maximum
    freqency also needs more voltage.

    From the datashee:  For AM_LDO_VCORE1 and AM_DCDC_VCORE1 modes, the maximum
    CPU operating frequency is 48 MHz and maximum input clock frequency for
    peripherals is 24 MHz.

    S'ha canviat a 24Mhz per ajustar-ho al projecte*/
    PCM_setCoreVoltageLevel( PCM_VCORE1 );
    CS_setDCOCenteredFrequency( CS_DCO_FREQUENCY_24 );
    CS_initClockSignal( CS_HSMCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1 );
    CS_initClockSignal( CS_SMCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1 );
    CS_initClockSignal( CS_MCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1 );
    CS_initClockSignal( CS_ACLK, CS_REFOCLK_SELECT, CS_CLOCK_DIVIDER_1 );
}


//------------------------------UART-----------------------------------------
const eUSCI_UART_Config xUARTConfig =
{
    EUSCI_A_UART_CLOCKSOURCE_SMCLK, /* SMCLK Clock Source. */
    3,                             /* BRDIV (es el mateix que amb la configuracio manual)*/
    0,                               /* UCxBRF */
    0,                               /* UCxBRS */
    EUSCI_A_UART_NO_PARITY,          /* No Parity. */
    EUSCI_A_UART_LSB_FIRST,          /* MSB First. */
    EUSCI_A_UART_ONE_STOP_BIT,       /* One stop bit. */
    EUSCI_A_UART_MODE,               /* UART mode. */
    EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION /* Low Frequency Mode. */
};
void init_uart0(void){
    uartTxQueue = xQueueCreate(UART_TX_QUEUE_LENGTH, sizeof(uint8_t));
    RxQueue = xQueueCreate(UART_RX_QUEUE_LENGTH, sizeof(uint8_t));

    MAP_GPIO_setAsPeripheralModuleFunctionInputPin( UART_GPIO_PORT_DADES, GPIO_PIN2 | GPIO_PIN3, GPIO_

    /* Use the library functions to initialise and enable the UART. */
    MAP_UART_initModule( EUSCI_AN_BASE, &xUARTConfig );
    MAP_UART_enableModule( EUSCI_AN_BASE );

    MAP_GPIO_setAsOutputPin(UART_GPIO_PORT_DADES, GPIO_PIN0);
    // PORT P1.0 com a sortida (Data direction: Selector Tx/Rx), GPIO
    MAP_GPIO_setOutputLowOnPin(UART_GPIO_PORT_DADES, GPIO_PIN0);
    // Inicialitzem port P1.0 a 0 (Rx)

    MAP_UART_clearInterruptFlag( EUSCI_AN_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT | EUSCI_A_UART_TRANSMIT
    MAP_UART_enableInterrupt( EUSCI_AN_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT );


    /* The interrupt handler uses the FreeRTOS API function so its priority must
    be at or below the configured maximum system call interrupt priority.
    configKERNEL_INTERRUPT_PRIORITY is the priority used by the RTOS tick and
    (should) always be set to the minimum priority. */
    MAP_Interrupt_setPriority( INT_EUSCIAN, configKERNEL_INTERRUPT_PRIORITY );
    MAP_Interrupt_enableInterrupt( INT_EUSCIAN );
}
```

```
//------------------------------BUZZER----------------------------------------
void init_buzzer(void){
    // Configurar P2.7 como salida digital
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN7);
}
//------------------------------LED 1-----------------------------------------
/*
 * Utilitzat per debugging, de moment el desactivem
void init_LED1(void){
    const uint8_t CONF_GPIO_LED1 = 0xFE;
    const uint8_t INIT_STATE_OFF = 0xFE;
    // Configuration of GPIO connection for LED1
    P1SEL0 &= CONF_GPIO_LED1;
    P1SEL1 &= CONF_GPIO_LED1;
    // Configuration of pin0's port 1 as output
    P1DIR |= CONF_OUTPUT_LED1;
    P1OUT &= INIT_STATE_OFF;
}*/

////////////////////////////////////////////////////////////////////////
/*
 * Implementacio de la llibreria del projecte
 * Aqui estan les funcions que implementen la logica interna
 */

//----------------------------------------MOTOR-----------------------------------------

RxReturn EncendreMotor(byte motor_id) {
    //Torque enable = 1
    byte torque_enable[2] = {24, 1};
    return ComunicationPacket(motor_id, 2, WR_INST_BIT, torque_enable);
}

RxReturn AturaMotor(byte motor_id) {
    // Torque Enable = 0
    byte torque_disable[2] = {24, 0};

    return  ComunicationPacket(motor_id, 2, WR_INST_BIT, torque_disable);
}

RxReturn SetVelocitatMotor(byte motor_id, uint16_t velocitat) {
    byte speed[3] = {32, (byte) (velocitat & 0xFF), (byte) ((velocitat >> 8) & 0xFF)};
    // Less significant bits
    return ComunicationPacket(motor_id, 3, WR_INST_BIT, speed);
}


//------------------------------IMPLEMENTACIONS LLIBRERIA---------------------------------

void set_velocitat(uint16_t velocitat){
    SetVelocitatMotor(MOTOR_ESQUERRA_ID, velocitat ^ 1024);
    SetVelocitatMotor(MOTOR_DRETA_ID, velocitat);
    velocitat_actual=velocitat;
}

// Moviments
void moure_endavant(void) {
```

```
    set_velocitat(velocitat_actual & (~0x400));
    EncendreMotor(MOTOR_ESQUERRA_ID);
    EncendreMotor(MOTOR_DRETA_ID);
    caminant=1;
}

void moure_enrere(void) {
    set_velocitat(velocitat_actual | 0x400);
    EncendreMotor(MOTOR_ESQUERRA_ID);
    EncendreMotor(MOTOR_DRETA_ID);
    caminant=1;
}

void girar_dreta(void){
    SetVelocitatMotor(MOTOR_ESQUERRA_ID, velocitat_actual & (~0x400));
    SetVelocitatMotor(MOTOR_DRETA_ID, velocitat_actual & (~0x400));

    EncendreMotor(MOTOR_ESQUERRA_ID);
    EncendreMotor(MOTOR_DRETA_ID);
}

void girar_esquerra(void){
    SetVelocitatMotor(MOTOR_ESQUERRA_ID, velocitat_actual | 0x400);
    SetVelocitatMotor(MOTOR_DRETA_ID, velocitat_actual | 0x400);

    EncendreMotor(MOTOR_ESQUERRA_ID);
    EncendreMotor(MOTOR_DRETA_ID);
}

void aturar(void){
    SetVelocitatMotor(MOTOR_ESQUERRA_ID, 0 ^ 1024);
    SetVelocitatMotor(MOTOR_DRETA_ID, 0);

    caminant=0;
}

// Sensors
RxReturn llegir_sensor(uint8_t id_sensor) {
    //char buffer[64];  // Espacio suficiente para la cadena a imprimir
    byte param[2];
    //const char* direccion = "";  // Para guardar el texto correspondiente al sensor

    switch(id_sensor) {
    case 0:
        param[0] = 0x1A;
        param[1] = 0x01;
        //direccion = "izq";
        break;
    case 1:
        param[0] = 0x1B;
        param[1] = 0x01;
        //direccion = "frente";
        break;
    case 2:
        param[0] = 0x1C;
        param[1] = 0x01;
        //direccion = "der";
```

```
        break;
    default:
        //direccion = "desconocido";
        break;
    }

    RxReturn rxReturn = ComunicationPacket(SENSOR_ID, 0x02, 0x02, param);

    return rxReturn;
}


//-------------------------------------------------------------------------------------

////////////////////////////////////////////////////////////////////////
/*
 * A continuacio hi ha les rutines d'interrupcio: per la UART i el P5
 *
 *
 */

//-------------------------RUTINAS DE INTERRUPCIN-------------------------

void EUSCIA2_IRQHandler(void)
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    // we see if the interrupt is related to reception or the tx
    uint_fast8_t xInterruptStatus = MAP_UART_getEnabledInterruptStatus(EUSCI_AN_BASE);

    uint8_t txByte;
    uint8_t data;

    if ( (xInterruptStatus & EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG ) != 0x00 ) {
        // FromISR es perque son funcions que es fan desde una ISR i tenen atencio especial
        MAP_UART_clearInterruptFlag(EUSCI_AN_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG);
        data = MAP_UART_receiveData(EUSCI_AN_BASE); //rebem la data
        xQueueSendFromISR( RxQueue, &data, &xHigherPriorityTaskWoken );
    }

    //Una idea naif podria ser aqui fer un while amb la cua plena
    //Aixo no funcionara xk podriem estar transmetent massa rapid
    if ((xInterruptStatus & EUSCI_A_UART_TRANSMIT_INTERRUPT_FLAG) != 0x00) {
        if (xQueueReceiveFromISR(uartTxQueue, &txByte, &xHigherPriorityTaskWoken) == pdPASS) {
            MAP_UART_transmitData(EUSCI_AN_BASE, txByte);
            // We don't need to re-enable the interrupt here, as the act of
            // writing to the transmit buffer will likely set the flag again
            // if the UART is ready for more data.
        }else{
            MAP_UART_clearInterruptFlag(EUSCI_AN_BASE, EUSCI_A_UART_TRANSMIT_INTERRUPT_FLAG);
            // No more data to transmit - disable the transmit interrupt
            MAP_UART_disableInterrupt(EUSCI_AN_BASE, EUSCI_A_UART_TRANSMIT_INTERRUPT);
            Sentit_Dades_Rx(); //Aixo ho posem aqui dsp de que acabi la transmissio
            //Sentit_Dades_Rx();
            // The direction will be switched to RX after the transmission is complete,
        }

    }
```

```
    /* portYIELD_FROM_ISR() will request a context switch if executing this
    interrupt handler caused a task to leave the blocked state, and the task
    that left the blocked state has a higher priority than the currently running
    task (the task this interrupt interrupted).  See the comment above the calls
    to xSemaphoreGiveFromISR() and xQueueSendFromISR() within this function. */
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);



}

void PORT5_IRQHandler(void){
    //Equivalent a agafar flag = P5IV
    uint16_t interruptFlag = MAP_GPIO_getInterruptStatus(GPIO_PORT_P5, GPIO_PIN1);

    //Equivalent a netejar la flag? crec que aixo es fa automaticament quan l'agafem del registre
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P5, GPIO_PIN1);

    //Equivalent a P5IE &= ~(BIT1);
    MAP_GPIO_disableInterrupt(GPIO_PORT_P5, GPIO_PIN1);

    //Posem referencia a 1 i backflip a 1
    //De normal la referencia estar a 0
    //Nota: el & es per operacions binaries.
    //Per tant aqui estem agafant nomes la part de la interrupt flag que ens interessa!
    if(referencia == 0 && (interruptFlag & GPIO_PIN1)){
        referencia = 1;
        backflip = 1;
    }else if(referencia == 1 && (interruptFlag & GPIO_PIN1)){
        referencia = 0;
        backflip = 1;
    }

    //equivalent a P5IE |= (BIT1);
    MAP_GPIO_enableInterrupt(GPIO_PORT_P5, GPIO_PIN1);

}
//Aquestes daqui son pel timer perque estan configurades globalment al projecte
//Les posem perque el compilador no es queixi
void vT32_0_Handler( void )
{
//
}
/*-----------------------------------------------------------*/

void vT32_1_Handler( void )
{
//
}
```