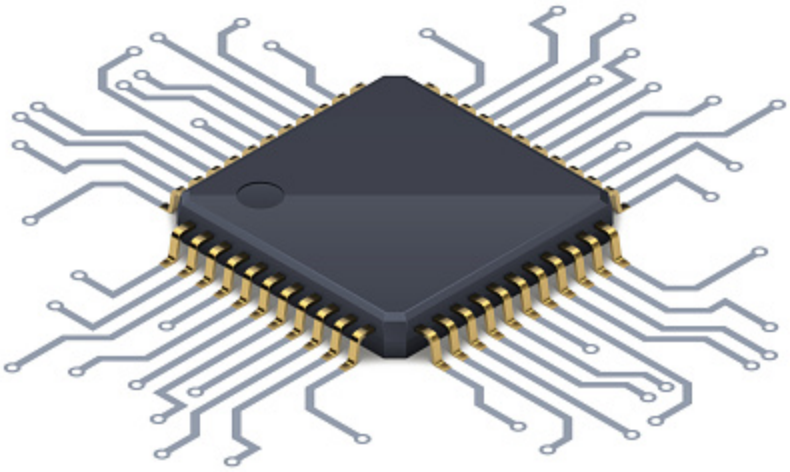
Projecte PAE

Andrés Rio & Adrià Vico

Juny de 2024



Contents

1 Explicació del projecte

1.1 Què es vol fer al projecte

Com a objectiu final de l'assignatura de programació d'arquitectures encastades, implementarem un moviment completament autònom en un robot. Aquest moviment permetrà que el robot es mogui en qualsevol circuit sense tenir cap tipus de col·lisió. Les seves decisions hauran de ser preses sense cap tipus d'ajuda o coneixement previ de l'entorn.

1.2 Funcions bàsiques del moviment

El projecte té com a objectiu desenvolupar un robot autònom capaç d'evitar col·lisions amb objectes dins d'un espai determinat. Per aconseguir-ho, el robot ha de realitzar diverses funcions. Inicialment, ha de buscar i trobar una paret. Un cop el robot detecta una paret, ha de seguir recorrent-la en un sentit determinat (en el nostre cas, seguirà el camí per l'esquerra). Si durant el seu recorregut el robot es troba amb un obstacle diferent, una cantonada de la paret o qualsevol altra interrupció, ha de ser capaç de rodejar aquest obstacle i continuar movent-se en el mateix sentit, sempre que sigui possible. Aquestes capacitats permetran que el robot es desplaci de manera autònoma i eficient, evitant col·lisions i mantenint una trajectòria contínua al llarg del seu recorregut. Es considera un moviment òptim aquell que aprofita al màxim la velocitat del robot, no té cap tipus de col·lisió i no s'allunya en cap moment de l'obstacle trobat inicialment.

1.3 Recursos utilitzats per al moviment

Per aconseguir el moviment, el robot disposa de dues rodes paral·leles, un suport posterior amb una bola omnidireccional per mantenir un equilibri estable, tres sensors a la part davantera i una pantalla. Aquests sensors poden detectar objectes que es trobin just davant i als costats del robot i la pantalla ens permet imprimir a temps real els valors dels sensors i text personalitzat, una manera molt útil de debugar el codi.

Disposem de les eines suficients per detectar qualsevol col·lisiós frontal i gairebé lateral del robot, a més de dictar el seu moviment.

1.4 Limitacions del robot

Amb dues rodes, el moviment del robot és limitat a moviments rectilinis, rotacions i pivotaments. Aquest moviment és poc flexible i requereix atenció especial en certs escenaris de gir.

A més, cal tenir molta cura a l'hora de fer marxa enrere, ja que el robot no disposa de sensors a la part posterior i és bastant important estar atent durant qualsevol gir, ja que, tot i que el robot disposa de dos sensors laterals, aquests només cobreixen el cap del robot. Durant un gir, pot ser que els sensors no detectin una paret dreta existent que simplement està a una altura menor a la del sensor, creant així un risc de col·lisió.

1.5 Funcions Utilitzades

A l'hora d'implementar el moviment del robot, hem simplificat tots els possibles moviments en diferents mètodes. Així podem reutilitzar el codi de manera reiterada, evitant possibles errors. Les funcions que hem utilitzat són:

1.5.1 Leer Sensor

Aquesta funció crida als sensors mitjançant el `TxPacket()`, on li passem com a paràmetre els 3 sensors a la vegada i, un cop rep aquesta informació, cridem al `RxPacket()` per obtenir els valors llegits. Aquesta funció ens permet tenir els valors dels 3 sensors de forma simultània. És important destacar que aquesta informació ve inclosa en un struct `RxReturn`, llavors, per obtenir aquests valors, cal agafar-los de l'array intern. La posició 5 seria el sensor esquerre, la posició 6 seria el sensor del mig i la posició 7 seria el sensor dret. Per exemple, si volem comparar el valor del sensor dret amb un valor arbitrari, caldria fer: `resulSensor.StatusPacket[7] < 30`.

1.5.2 Move Motor

Aquesta funció mai la cridem directament des del `main`, sinó que la usem com a funció auxiliar per als altres mètodes implementats. El que fa és rebre el valor ID de la roda que s'ha de moure, el sentit cap a on es mou i la velocitat a aplicar. Mitjançant el `TxPacket()` enviem la informació al robot

1.5.3 Move UP

Aquesta funció només rep un valor de velocitat i fa que el robot es mogui cap endavant de manera uniforme. El que fa aquest mètode interiorment és trucar al `Move Motor` per les dues rodes, la dreta amb sentit 1 i l'esquerra amb sentit 0 per assegurar que ambdues vagin en la mateixa direcció.

1.5.4 Move UP custom

Aquesta funció té una utilitat similar a l'anterior, només que en comptes de rebre un valor de velocitat, en rep dos: el primer valor per la roda dreta i l'altre per l'esquerra. Aquest mètode s'utilitza principalment per fer girs, ja que ambdues rodes aniran cap endavant, però podem controlar la velocitat de les dues. Així, si una roda va més ràpida que l'altra, el robot farà un gir.

1.5.5 Move rotation

Aquesta funció rep un valor de velocitat i el que fa interiorment és trucar al `Move Motor` però amb el mateix sentit de les rodes. Així, el que fa el robot és pivotar a la velocitat indicada.

1.5.6 Move Rotation Left

Aquesta funció rep un valor de velocitat i simplement truca al `Move Motor` per la roda esquerra, de manera que es mou cap endavant amb la roda dreta completament parada. Seria com trucar al `Move UP custom` amb valor 0 a la roda esquerra (seria menys òptim, ja que trucar la roda esquerra per res seria inútil). Utilitzem aquest mètode per fer girs més exagerats, sobretot en el cas de tenir la paret molt a prop (on no volem avançar amb l'altra roda).

1.5.7 Stop

Aquesta funció s'utilitza principalment per testejar i no per l'algoritme, ens ajuda a parar el robot. Per implementar aquesta funció, indiquem a ambdues rodes que la velocitat sigui 0.

1.5.8 Altres funcions de la llibreria

A la nostra llibreria vam implementar més mètodes que finalment no hem utilitzat. La majoria d'aquestes funcions són versions utilitzades per testeig, com per exemple el `Move Down`, implementat per moure el robot marxa enrere (idea que vam rebutjar), el `Move Rotation Inverse`, implementat per girar en direcció contrària a l'altre `Move Rotation`, igual que les seves versions `custom` que mai vam arribar a utilitzar. Aquestes funcions van ser útils per poder provar les nostres idees de manera eficaç, sense haver de repetir codi i de manera simple i ràpida

1.6 Algoritme implementat

Per fer funcional el nostre robot, hem fet un algorisme que està estructurat en dues parts, principalment. El nostre robot constarà de dos estats, **ESTADO_RECTO** i **ESTADO_PAREDDERECHA**. El nostre primer estat tracta simplement de anar en linia recta fins trobar un obstacle. Un cop l'hagi trobat, maniobrarà fins estar situat paral·lelament per evitar la col·lisió i passarà al següent estat **ESTADO_PAREDDERECHA**. Aquest estat té l'objectiu de seguir la pared sense estar massa a prop ni massa lluny. Per aconseguir aquest objectiu hem dividit el nostre estat en 3 tipus de moviment. Un per quan el nostre robot tingui la pared dreta a molta distància, altre per quan aquesta distància sigui molt petita i un altre per quan trobi un obstacle just davant seu.

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

    //iniciamos todos los metodos antes de iniciar el main
    init_ucs_24MHz();
    Init_UART();

    init_interrupciones();
    __enable_interrupts();
    init_timers();

    volatile uint32_t i;
    uint32_t temps;
    uint8_t linea = 0;
    struct RxReturn resulSensor;
    //uint8_t resulSensorInt;

    halLcdInit();

    char numeroDerecha[3];          // variable para tener valor del sensor como string
    char numeroMedio[3];            //
    char numeroIzquierda[3];        //
    char Derecha[8] = "DERECHA";    // fila 0
    char Medio[6] = "MEDIO";        // fila 3
    char Izquierda[10] = "IZQUIERDA";  // fila 6

    // CODIGO PARA IMPRIMIR NUESTROS NOMBRES

    char nombre[19] = "PAE Andres y Adria";

    halLcdClearScreen(0);

    halLcdPrintLine(Derecha, 0 ,0);  // Imprimir texto derecha
    halLcdPrintLine(Medio, 3 ,0);    // Imprimir texto medio
    halLcdPrintLine(Izquierda, 6 ,0);  // Imprimir texto izquierda

    halLcdPrintLine(nombre, 8 ,0);      // nombre

    move_up(VELOCIDADESTANDAR); //empezamos a movernos rectos
    //esta variable se encargarà de guardar el estado actual de movimiento de nuestro robot
    uint8_t estado = ESTADO_RECTO; //En el caso del comienzo, el estado inicial será el movimiento recto hacia delante

    //prueba de mover hacia delante, parar, mover hacia atras, parar, rotar a la derecha, parar, mover a la izquierda y parar
    while(1){
```

Figure 1: Main

La estructura del nostre main consisteix bàsicament d'un bucle infinit, on dins d'aquest, el nostre robot anirà canviant d'estat depenent de que detectem als sensors. Abans de fer qualsevol cosa, com ja hem vist a la pràctica passada, inicialitzarem la UART (per rebre i enviar dades al robot), les interrupcions i els timers. També farem ús de la llibrería **halLcdInit**, que ens va ser porpocionada pel professorat, la cual utilitzarem per imprimir les distàncies que detecta en tot moment el robot i l'estat actual del robot.

1.6.1 Estat Recte

```
/* *****
 *
 * ESTADO INICIAL
 *
 * ******
 */
if ( resulSensor.StatusPacket[6] > 30 && estado == ESTADO_RECTO ) {   //si el sensor frontal(en el estado recto) detecta una pared cerca

    stop();
    char estado1[15] = "Estado Inicial";
    halLcdPrintLine(estado1, 8 ,0);
    tiempoRotacion = 0; //variable con el que regularemos el tiempo de giro del robot

    Activa_TimerA1_TimeOut(); //activamos timer
    //para hacer el giro a la izquierda antes de que se choque con la pared, lo regularemos mediante dos condicionales dentro de un while
    do {

        move_up_custom(1000,200);   //empezamos a girar a la izquierda
        resulSensor = leer_sensor();

        sprintf(numeroIzquierda, "%03d", resulSensor.StatusPacket[5]); // Convertir a cadena
        sprintf(numeroDerecha, "%03d", resulSensor.StatusPacket[7]); // Convertir a cadena
        sprintf(numeroMedio, "%03d", resulSensor.StatusPacket[6]); // Convertir a cadena

        halLcdPrintLine(numeroDerecha, 1 ,0);
        halLcdPrintLine(numeroMedio,4,0);
        halLcdPrintLine(numeroIzquierda,7,0);

        if (resulSensor.StatusPacket[6] > 110) {  //si al estar girando vemos que se acerca demasiado a la pared, paramos
                        break;
                    }
        //el tiempo de duración del giro estará regulado por un timer, el cual, si el tiempo de rotación es mayor que 1, paramos de girar
        if (tiempoRotacion > 1) {
            break;
        }

    } while (resulSensor.StatusPacket[7] < 80); // esto se repetirá siempre que la distancia del sensor derecho a la pared sea menos de 80

    //Desactiva_TimerA1_TimeOut();

    //una vez hemos acabado el giro, seguiremos el camino yendo recto
    move_up(VELOCIDADESTANDAR);
    estado = ESTADO_PAREDDERECHA; //Ahora nuestro estado cambia a pared derecha
}
```

Figure 2: Estat Recte

A l'iniciar el programa, el robot comença movent-se en línia recta i, en aquest estat, si el sensor frontal detecta una paret massa a prop (valor del sensor frontal més gran que 30), el robot girarà a la esquerra per evitar xocar-se amb la paret. Aquest gir està controlat per la distància a la paret frontal i un timer, de manera que si el sensor frontal detecta una altre paret, al girar, massa propera o el temps de gir supera al d'un segon, el robot deixa de girar i continua movent-se recte, canviant a l'estat "ESTADO_PAREDDERECHA". Volem recalcar que a partir d'aquí, a no ser que reiniciem el programa, el robot no tornarà a passar mai més per aquest estat degut a que ja tindrem una paret dreta com a referència.

1.6.2 Estat Paret Dreta

En aquest estat, ja tindrem una paret dreta com a referència per reseguir. Hem dividit aquest estat en 3 cassos anidats:

```
        /* **************
         *
         * ESTADO PARED DERECHA
         *
         * **************
         */
        //SE VA A CHOCAR CONTRA LA PARED FRONTAL, ROTAMOS SU EJE HASTA QUE SE ALINIE EN PARALELO CON LA PARED Y SEGUIMOS RECTOS
        if ( resulSensor.StatusPacket[6] > 110 && estado == ESTADO_PAREDDERECHA) {   // SI SE VA A CHOCAR EN PARED DERECHA

            halLcdClearLine(8);
            char der1[14] = "derecha pared";
            halLcdPrintLine(der1, 8 ,0);

            stop(); //antes de rotar, nos detenemos
            move_rotation(900); //procedemos a rotar el robot

            do {
                //leemos los valores de los sensores y los imprimimos
                resulSensor = leer_sensor();

                sprintf(numeroIzquierda, "%03d", resulSensor.StatusPacket[5]); // Convertir a cadena
                sprintf(numeroDerecha, "%03d", resulSensor.StatusPacket[7]); // Convertir a cadena
                sprintf(numeroMedio, "%03d", resulSensor.StatusPacket[6]); // Convertir a cadena

                halLcdPrintLine(numeroDerecha, 1 ,0);
                halLcdPrintLine(numeroMedio,4,0);
                halLcdPrintLine(numeroIzquierda,7,0);

            } while (resulSensor.StatusPacket[6] != 0); //rotaremos hasta que el frontal detecte como lejos la pared
            //} while (resulSensor.StatusPacket[7] > 110);

            halLcdClearLine(8);
            char der12[14] = "derecha SALIR";
            halLcdPrintLine(der12, 8 ,0);

            //seguimos moviendonos hacia delante
            move_up(VELOCIDADESTANDAR);
        }
```

Figure 3: Estat Paret Dreta: Es choca amb paret frontal

Aquest primer cas correspon quan el sensor frontal detecta una paret massa aprop (valor del sensor frontal major a 110), pivotarem el robot cap a l'esquerra fins que detecti com a lluny aquesta paret frontal (resulSensor.StatusPacket[6] != 0). Una vegada ha completat el moviment de rotació, reprendem la marxa cap endavant.

```
347         //CASO DONDE NOS ALEJAMOS MUCHO DE LA PARED DERECHA, RECTIFICAMOS GIRANDO A LA DERECHA PARA VOLVERNOS A ACERCAR A ESTA
348         if ( resulSensor.StatusPacket[7] == 0 && estado == ESTADO_PAREDDERECHA) {   // SI LA PARED DERECHA ESTA MUY LEJOS
349
350             halLcdClearLine(8);
351             char der2[5] = "giro";
352             halLcdPrintLine(der2, 8 ,0);
353             //detenemos el movimiento
354             stop();
355
356             do {
357                 //empezamos a girar a la derecha
358                 move_up_custom(300,900);
359                 resulSensor = leer_sensor();
360
361                 if (resulSensor.StatusPacket[6] > 110) {  // si se va a comer una pared que salga asap
362                     break;
363                 }
364
365                 sprintf(numeroIzquierda, "%03d", resulSensor.StatusPacket[5]); // Convertir a cadena
366                 sprintf(numeroDerecha, "%03d", resulSensor.StatusPacket[7]); // Convertir a cadena
367                 sprintf(numeroMedio, "%03d", resulSensor.StatusPacket[6]); // Convertir a cadena
368
369                 halLcdPrintLine(numeroDerecha, 1 ,0);
370                 halLcdPrintLine(numeroMedio,4,0);
371                 halLcdPrintLine(numeroIzquierda,7,0);
372
373             } while (resulSensor.StatusPacket[7] < 90); //giraremos a la derecha siempre que la distancia del sensor derecho sea menos de 90
374
375             move_up(VELOCIDADESTANDAR);
376
377         }
```

Figure 4: Estat Paret Dreta: S'allunya massa de la paret dreta

Per un altre part, hem de contemplar el cas que el nostre robot, a l'anar recte, es pugui allunyar massa de la paret dreta. Es per això que quan el nostre sensor dret detecti massa lluny la paret, específicament, quan la distància sigui 0, rectificarem la marxa girant cap a la dreta sempre que no estigui suficientment a prop d'aquesta (resulSensor.StatusPacket[7] < 90). Una vegada hem acabat de reajustar el moviment, tornem a moure'ns recte. Aquest cas també ens servirà per poder girar quan arribem al límit de la nostre paret de referència; al seguir avançant el sensor dret no detectarà res ja que haurem sobrepassat el límit de la paret dreta, es per això que aquest començarà a girar a la dreta, tornant-se a alieanar amb la paret dreta.

```
379         //CASO DONDE LA PARED DERECHA ESTA DEMASIADO CERCA, ROTAMOS EL EJE A LA IZQUIERDA Y SEGUIMOS RECTOS
380         if ( resulSensor.StatusPacket[7] > 90 && estado == ESTADO_PAREDDERECHA) {   // SI LA DERECHA ESTA MUY CERCA
381
382             halLcdClearLine(8);
383             char der3[14] = "derecha cerca";
384             halLcdPrintLine(der3, 8 ,0);
385
386             stop();
387
388             do {
389
390                 move_rotation_left(900);  //empezamos a movernos a la izquierda
391                 resulSensor = leer_sensor();
392
393                 if (resulSensor.StatusPacket[6] > 110) {  //si al girar a la izquierda vemos que se va a chocar con una pared frontal, paramos
394                     break;
395                 }
396
397                 sprintf(numeroIzquierda, "%03d", resulSensor.StatusPacket[5]); // Convertir a cadena
398                 sprintf(numeroDerecha, "%03d", resulSensor.StatusPacket[7]); // Convertir a cadena
399                 sprintf(numeroMedio, "%03d", resulSensor.StatusPacket[6]); // Convertir a cadena
400
401                 //imprimimos
402                 halLcdPrintLine(numeroDerecha, 1 ,0);
403                 halLcdPrintLine(numeroMedio,4,0);
404                 halLcdPrintLine(numeroIzquierda,7,0);
405
406
407             } while (resulSensor.StatusPacket[7] > 90); //seguiremos girando siempre que la distancia con la pared derecha sea mayor que 90
408
409             //una vez completado el giro, procedemos a ir rectos
410             move_up(VELOCIDADESTANDAR);
411
412         }
413 }}
```

Figure 5: Estat Paret Dreta: S'apropa massa a la paret dreta

Per últim; se'ns pot donar el cas de que el robot estigui massa aprop de la paret dreta (resulSensor.StatusPacket[7] $>$ 90), així que per evitar-nos qualsevol problema, rotarem a la esquerra fins que la distància sigui menor que 90. Una vegada hem sortit del bucle, reanudarem la marxa tirant cap endevant.

1.7 Problemes que han sorgit i com s'han solucionat.

Quan vam començar a fer el projecte, vam tenir diferents idees per evitar col·lisions i fer l'alineament amb la paret. En fer el gir per evitar la paret, primer intentàvem anar marxa enrere, però, al no estar completament alineats, ens xocàvem. Per evitar això, primer vam implementar el codi de l'ESTATRECTE, que funcionava parcialment. Quan es trobava en un carreró, el sensor detectava l'obstacle tan lluny i el gir era tan ràpid que acabava col·lisionant. Per això, vam optar per fer un enfocament diferent. En comptes de girar, pivotem fins a no tenir cap obstacle davant nostre. Com que no teníem un coneixement precís dels obstacles, preferim pivotar només en aquest cas i fer un moviment menys òptim, però aconseguir un posicionament més ideal.

Un altre problema va ser decidir amb quines velocitats configuràvem els moviments dels nostres motors, ja que hi havia casos on es desincronitzava amb el robot, fent que es xoqués amb la paret en algunes ocasions. Finalment, després de diversos testeigs, vam decidir que 900 seria la velocitat ideal.

1.8 Conclusions

Amb aquest projecte finalitzem l'assignatura de programació d'arquitectures encastades, destacant la importància de fer un bon codi per comunicar-se amb l'arquitectura i facilitar un desenvolupament posterior més senzill. En aquesta última pràctica, hem pogut aplicar tot el codi de la pràctica 4 a un problema real, utilitzant tots els recursos disponibles per trobar la manera més òptima de moure el robot. Això ens ha

permès posar en pràctica tots els conceptes apresos durant l'assignatura de manera molt creativa.

2 Programa comentat

2.1 main.c

```
#include <msp432p401r.h>
#include "msp.h"
#include "lib_PAE.h"
#include "msp.h"
#include <stdbool.h>
#include "lib_p4_robot.h"
typedef uint8_t byte;

#define TXD2_READY (UCA2IFG & UCTXIFG)

#define ESTADO_RECTO 0
#define ESTADO_PAREDDERECHA 1
#define ESTADO_PAREDIZQUIERDA 2
#define ESTADO_ROTAR 3

#define VELOCIDADESTANDAR 700       // PROBADO CON 500
#define ROTACION 200                // 200 ESTA BIEN




struct RxReturn leer_sensor();
void Activa_TimerA2_TimeOut();
void Desactiva_TimerA2_TimeOut();

typedef struct RxReturn{
    byte StatusPacket[16];    // CAMBIO : [9]
    byte timeout;
};

/* funcions per canviar el sentit de les comunicacions */
void Sentit_Dades_Rx(void)
{ //Configuració del Half Duplex dels motors: Recepció
    P3OUT &= ~BIT0; //El pin P3.0 (DIRECTION_PORT) el posem a 0 (Rx)
}
void Sentit_Dades_Tx(void)
{ //Configuració del Half Duplex dels motors: Transmissió
    P3OUT |= BIT0; //El pin P3.0 (DIRECTION_PORT) el posem a 1 (Tx)
}
/* funció TxUACx(byte): envia un byte de dades per la UART 0 */
void TxUACx(uint8_t bTxdData)
```

```
{
    while(!TXD2_READY); // Espera a que estigui preparat el buffer de transmissió
    UCA2TXBUF = bTxdData;
}



/**
 * Método con el que inicializamos UART
 */
void Init_UART(void)
{
    UCA2CTLW0 |= UCSWRST; //Fem un reset de la USCI, desactiva la USCI
    UCA2CTLW0 |= UCSSEL__SMCLK; //UCSYNC=0 mode asíncron
    UCA2MCTLW = UCOS16; // Necessitem sobre-mostreig => bit 0 = UCOS16 = 1
    //Ara fixarem el nostre prescaler. En aquest cas, el nostre UART és de 8MHz com hem vist anteriorment.
    //Com utilitzem un un SMCLK a una freqüència de 24MHz, l'hem de divir de tal manera d'aconseguir 8MHz.
    //Per tant, si 8MHz=(24MHz/prescaler), prescaler=(24MHz/8MHz)=3
    UCA2BRW = 3;

    //volem un baud rate de 500 kbps i fem sobre-mostreig de 16 mostres
    //el rellotge de la UART ha de ser de 8MHz (ja que 500000*16= 8MHz).
    UCA2MCTLW |= (0x00 << 8); //UCBRSx, part fractional del baud rate

    //activamos pines de la uart
    P3SEL0 |= BIT2 | BIT3; //I/O funció: P1.3 = UART0TX, P1.2 = UART0RX
    P3SEL1 &= ~ (BIT2 | BIT3);
    //activamos que sea de salida
    P3DIR |= BIT0;
    Sentit_Dades_Rx(); //ejecutamos metodo para poder cambiar el sentido del flujo de datos
    P3SEL0 |= BIT2 | BIT3; //I/O funció: P1.3 = UART0TX, P1.2 = UART0RX
    P3SEL1 &= ~ (BIT2 | BIT3);

    UCA2CTLW0 &= ~UCSWRST;
    EUSCI_A2->IFG &= ~EUSCI_A_IFG_RXIFG; // Clear eUSCI RX interrupt flag
    EUSCI_A2->IE |= EUSCI_A_IE_RXIE;

}

//TxPacket() 3 paràmetres: ID del Dynamixel, Mida dels paràmetres, Instruction byte. torna la mida del "Return pack
byte TxPacket(byte bID, byte bParameterLength, byte bInstruction, byte Parametros[16])
{
    byte bCount,bCheckSum,bPacketLength;
    byte TxBuffer[32];


    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;
    }



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

    for(bCount = 0; bCount < bParameterLength; bCount++) //Comencem a generar la trama que hem d'enviar
    {
        TxBuffer[bCount+5] = Parametros[bCount];
    }
    bCheckSum = 0;
    bPacketLength = bParameterLength+4+2;
    for(bCount = 2; bCount < bPacketLength-1; bCount++) //Càlcul del checksum
    {
        bCheckSum += TxBuffer[bCount];
    }
    TxBuffer[bCount] = ~bCheckSum; //Escriu el Checksum (complement a 1)
    for(bCount = 0; bCount < bPacketLength; bCount++) //Aquest bucle és el que envia la trama al Mòdul Robot
    {
        TxUACx(TxBuffer[bCount]);
    }
    while((UCA2STATW & UCBUSY)); //Espera fins que s'ha transmès el últim byte
    Sentit_Dades_Rx(); //Posem la línia de dades en Rx perquè el mòdul Dynamixel envia resposta
    return(bPacketLength);
}

void init_interrupciones()
{
    // Configuracion al estilo MSP430 "clasico":
    // --> Enable Port 4 interrupt on the NVIC.
    // Segun el Datasheet (Tabla "6-39. NVIC Interrupts", apartado "6.7.2 Device-Level User Interrupts"),
    // la interrupcion del puerto 1 es la User ISR numero 35.
    // Segun el Technical Reference Manual, apartado "2.4.3 NVIC Registers",
    // hay 2 registros de habilitacion ISER0 y ISER1, cada uno para 32 interrupciones (0..31, y 32..63, resp.),
    // accesibles mediante la estructura NVIC->ISER[x], con x = 0 o x = 1.
    // Asimismo, hay 2 registros para deshabilitarlas: ICERx, y dos registros para limpiarlas: ICPRx.

    //Habilitamos el timer 0 (posicion 8 )
```

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

    //Habilitamos el timer 1 (posicion 10 )
    NVIC->ICPR[0] |= BIT(10); //Primero, me aseguro de que no quede ninguna interrupcion residual pendiente para es
    NVIC->ISER[0] |= BIT(10); //y habilito las interrupciones del puerto

    //Habilitamos el timer 2 (posicion 12 )
    NVIC->ICPR[0] |= BIT(12); //Primero, me aseguro de que no quede ninguna interrupcion residual pendiente para es
    NVIC->ISER[0] |= BIT(12); //y habilito las interrupciones del puerto

    //Habilitamos el uart (posicion 18 )
    NVIC->ICPR[0] |= BIT(18); //Primero, me aseguro de que no quede ninguna interrupcion residual pendiente para es
    NVIC->ISER[0] |= BIT(18); //y habilito las interrupciones del puerto

}

void init_timers(void){

   //Para testear los ejercicios, hemos comentado el codigo. Borrad y comentad para ir ejecutando las
    //partes que querais. Hemos dejado descomentado inicialmente la segunda parte

    /**
     *
     * PRIMERA PARTE DE LA PRÃCTICA
     *
    ***/


   //ConfiguraciÃ³n a 10Khz
   //Timer A1, used for robot
   //Divider = 1; CLK source is SMCLK; clear the counter; MODE is up

    TIMER_A1->CTL = TIMER_A_CTL_ID__1 | TIMER_A_CTL_SSEL__ACLK | TIMER_A_CTL_CLR
            | TIMER_A_CTL_MC__UP;

   //el clock utilizado es el SMCLK, el qual funciona a 24MHz. Nuestra base de tiempo en esta primera parte
   //es de 10Khz, es por eso que si fijamos el timer a 240, obtendremos esos 10Khz, ya que 24MHz/240 = 10Khz
   TIMER_A1->CCR[0] = 240;     // 10 kHz
   TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0




   //Timer A2, para contar segundos

   //Divider = 1; CLK source is ACLK; clear the counter; MODE is up
```

```
   TIMER_A2->CTL = TIMER_A_CTL_ID__1 | TIMER_A_CTL_SSEL__ACLK | TIMER_A_CTL_CLR
              | TIMER_A_CTL_MC__UP;
   //el clock utilizado para esta segunda parte es el ACLK, el qual funciona a 32768 tics por segundo.
   //En este caso, al querer una freqüència de 1Hz, simplemente fijamos el timer A1 32768-1, ya que al dividir
   //esto nos daria 1Hz
   TIMER_A2->CCR[0] = 32768 - 1; // Configuración para 1 Hz
   TIMER_A2->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0


}

/**
 * main.c
 */

static uint8_t tiempoRotacion;


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

    //iniciamos todos los metodos antes de iniciar el main
    init_ucs_24MHz();
    Init_UART();

    init_interrupciones();
    __enable_interrupts();
    init_timers();

    volatile uint32_t i;
    uint32_t temps;
    uint8_t linea = 0;
    struct RxReturn resulSensor;
    //uint8_t resulSensorInt;

    halLcdInit();

    char numeroDerecha[3];         // variable para tener valor del sensor como string
    char numeroMedio[3];           //
    char numeroIzquierda[3];       //
    char Derecha[8] = "DERECHA";   // fila 0
    char Medio[6] = "MEDIO";       // fila 3
    char Izquierda[10] = "IZQUIERDA";  // fila 6

    // CODIGO PARA IMPRIMIR NUESTROS NOMBRES

    char nombre[19] = "PAE Andres y Adria";
```

```
halLcdClearScreen(0);

halLcdPrintLine(Derecha, 0 ,0);  // Imprimir texto derecha
halLcdPrintLine(Medio, 3 ,0);    // Imprimir texto medio
halLcdPrintLine(Izquierda, 6 ,0);  // Imprimir texto izquierda

halLcdPrintLine(nombre, 8 ,0);      // nombre

move_up(VELOCIDADESTANDAR); //empezamos a movernos rectos
//esta variable se encargarà de guardar el estado actual de movimiento de nuestro robot
uint8_t estado = ESTADO_RECTO; //En el caso del comienzo, el estado inicial será el movimiento recto hacia dela

//prueba de mover hacia delante, parar, mover hacia atras, parar, rotar a la derecha, parar, mover a la izquier
while(1){

    resulSensor = leer_sensor();

    sprintf(numeroIzquierda, "%03d", resulSensor.StatusPacket[5]); // Convertir a cadena
    sprintf(numeroDerecha, "%03d", resulSensor.StatusPacket[7]); // Convertir a cadena
    sprintf(numeroMedio, "%03d", resulSensor.StatusPacket[6]); // Convertir a cadena

    halLcdPrintLine(numeroDerecha, 1 ,0);
    halLcdPrintLine(numeroMedio,4,0);
    halLcdPrintLine(numeroIzquierda,7,0);


    /* *****
     *
     * ESTADO INICIAL
     *
     * ******
     */
    if ( resulSensor.StatusPacket[6] > 30 && estado == ESTADO_RECTO ) {   //si el sensor frontal(en el estado r

        stop();
        char estado1[15] = "Estado Inicial";
        halLcdPrintLine(estado1, 8 ,0);
        tiempoRotacion = 0; //variable con el que regularemos el tiempo de giro del robot

        Activa_TimerA1_TimeOut(); //activamos timer
        //para hacer el giro a la izquierda antes de que se choque con la pared, lo regularemos mediante dos co
        do {

            move_up_custom(1000,200);   //empezamos a girar a la izquierda
            resulSensor = leer_sensor();
```

```
        sprintf(numeroIzquierda, "%03d", resulSensor.StatusPacket[5]); // Convertir a cadena
        sprintf(numeroDerecha, "%03d", resulSensor.StatusPacket[7]); // Convertir a cadena
        sprintf(numeroMedio, "%03d", resulSensor.StatusPacket[6]); // Convertir a cadena

        halLcdPrintLine(numeroDerecha, 1 ,0);
        halLcdPrintLine(numeroMedio,4,0);
        halLcdPrintLine(numeroIzquierda,7,0);

        if (resulSensor.StatusPacket[6] > 110) {  //si al estar girando vemos que se acerca demasiado a la
                        break;
                    }
        //el tiempo de duración del giro estará regulado por un timer, el cual, si el tiempo de rotación es
        if (tiempoRotacion > 1) {
            break;
        }

    } while (resulSensor.StatusPacket[7] < 80); // esto se repetirá siempre que la distancia del sensor der

    //Desactiva_TimerA1_TimeOut();

    //una vez hemos acabado el giro, seguiremos el camino yendo recto
    move_up(VELOCIDADESTANDAR);
    estado = ESTADO_PAREDDERECHA; //Ahora nuestro estado cambia a pared derecha
}




/* *****
 *
 * ESTADO PARED DERECHA
 *
 * ******
 */
//SE VA A CHOCAR CONTRA LA PARED FRONTAL, ROTAMOS SU EJE HASTA QUE SE ALINIE EN PARALELO CON LA PARED Y SEG
if ( resulSensor.StatusPacket[6] > 110 && estado == ESTADO_PAREDDERECHA) {   // SI SE VA A CHOCAR EN PARED

    halLcdClearLine(8);
    char der1[14] = "derecha pared";
    halLcdPrintLine(der1, 8 ,0);

    stop(); //antes de rotar, nos detenemos
    move_rotation(900); //procedemos a rotar el robot

    sprintf(numeroIzquierda, "%03d", resulSensor.StatusPacket[5]); // Convertir a cadena
    sprintf(numeroDerecha, "%03d", resulSensor.StatusPacket[7]); // Convertir a cadena
    sprintf(numeroMedio, "%03d", resulSensor.StatusPacket[6]); // Convertir a cadena
```

```
    halLcdPrintLine(numeroDerecha, 1 ,0);
    halLcdPrintLine(numeroMedio,4,0);
    halLcdPrintLine(numeroIzquierda,7,0);

    do {
        //leemos los valores de los sensores y los imprimimos
        resulSensor = leer_sensor();

    } while (resulSensor.StatusPacket[6] != 0); //rotaremos hasta que el frontal detecte como lejos la pare

    //seguimos moviendonos hacia delante
    move_up(VELOCIDADESTANDAR);
}



//CASO DONDE NOS ALEJAMOS MUCHO DE LA PARED DERECHA, RECTIFICAMOS GIRANDO A LA DERECHA PARA VOLVERNOS A ACE
if ( resulSensor.StatusPacket[7] == 0 && estado == ESTADO_PAREDDERECHA) {   // SI LA PARED DERECHA ESTA MUY

    halLcdClearLine(8);
    char estado3[12] = "Pared Lejos";
    halLcdPrintLine(estado3, 8 ,0);
    //detenemos el movimiento
    stop();

    do {
        //empezamos a girar a la derecha
        move_up_custom(300,900);
        resulSensor = leer_sensor();

        sprintf(numeroIzquierda, "%03d", resulSensor.StatusPacket[5]); // Convertir a cadena
        sprintf(numeroDerecha, "%03d", resulSensor.StatusPacket[7]); // Convertir a cadena
        sprintf(numeroMedio, "%03d", resulSensor.StatusPacket[6]); // Convertir a cadena

        halLcdPrintLine(numeroDerecha, 1 ,0);
        halLcdPrintLine(numeroMedio,4,0);
        halLcdPrintLine(numeroIzquierda,7,0);

        if (resulSensor.StatusPacket[6] > 110) {  // si se va a comer una pared que salga asap
            break;
        }

    } while (resulSensor.StatusPacket[7] < 90); //giraremos a la derecha siempre que la distancia del senso

    move_up(VELOCIDADESTANDAR);
```

```
        }

        //CASO DONDE LA PARED DERECHA ESTA DEMASIADO CERCA, ROTAMOS EL EJE A LA IZQUIERDA Y SEGUIMOS RECTOS
        if ( resulSensor.StatusPacket[7] > 90 && estado == ESTADO_PAREDDERECHA) {   // SI LA DERECHA ESTA MUY CERCA

            halLcdClearLine(8);
            char estado4[12] = "Pared Cerca";
            halLcdPrintLine(estado4, 8 ,0);

            stop();

            do {

                move_rotation_left(900);  //empezamos a movernos a la izquierda
                resulSensor = leer_sensor();

                sprintf(numeroIzquierda, "%03d", resulSensor.StatusPacket[5]); // Convertir a cadena
                sprintf(numeroDerecha, "%03d", resulSensor.StatusPacket[7]); // Convertir a cadena
                sprintf(numeroMedio, "%03d", resulSensor.StatusPacket[6]); // Convertir a cadena

                halLcdPrintLine(numeroDerecha, 1 ,0);
                halLcdPrintLine(numeroMedio,4,0);
                halLcdPrintLine(numeroIzquierda,7,0);

                if (resulSensor.StatusPacket[6] > 110) {  //si al girar a la izquierda vemos que se va a chocar con
                    break;
                }

            } while (resulSensor.StatusPacket[7] > 90); //seguiremos girando siempre que la distancia con la pared

            //una vez completado el giro, procedemos a ir rectos
            move_up(VELOCIDADESTANDAR);

        }
}}

static byte Byte_Recibido = 0;
static byte DatoLeido_UART;
static uint8_t tiempoGlobal;

void EUSCIA2_IRQHandler (void)
{ //interrupcion de recepcion en la UART A0

    EUSCI_A2->IFG &=~ EUSCI_A_IFG_RXIFG; // Clear interrupt
    UCA2IE &= ~UCRXIE; //Interrupciones desactivadas en RX
    DatoLeido_UART = UCA2RXBUF;
    Byte_Recibido=1;
```

```
    UCA2IE |= UCRXIE; //Interrupciones reactivadas en RX

}
void Activa_TimerA1_TimeOut(void){
    TIMER_A1->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0 ???
    TIMER_A1->CCR[0] = 0;
}
void Desactiva_TimerA1_TimeOut(void){
    TIMER_A1->CCTL[0] &= ~TIMER_A_CCTLN_CCIE;
}


void Activa_TimerA2_TimeOut(void){
    TIMER_A2->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Interrupciones activadas en CCR0 ???
}
void Desactiva_TimerA2_TimeOut(void){
    TIMER_A2->CCTL[0] &= ~TIMER_A_CCTLN_CCIE;
}



/**
 * Metodo con el cual reiniciamos el valor del timer (tiempoGlobal) para el metodo recibir
 */
void Reset_Timeout(void){
    tiempoGlobal = 0;
}
/**
 * Metodo que comprueba que si el tiempo es mayor que el valor fijado, devuelve 1, si no 0.
 */
byte TimeOut(uint8_t tiempoMaximo){

    if (tiempoMaximo < tiempoGlobal){
        return 1;
    }

    return 0;

}

void TA0_0_IRQHandler (void){
    TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG; //Hem de netejar el flag de la interrupció
    //Codi de la IRQ
}
/**
 * SUbrutina que cada vez que será llamada, aumenta un valor global
 */
```

```
void TA1_0_IRQHandler (void)
{
    TIMER_A1->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG; //Hem de netejar el flag de la interrupció
    //Codi de la IRQ
    tiempoGlobal++;

}
void TA2_0_IRQHandler (void)
{
    TIMER_A2->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG; //Hem de netejar el flag de la interrupció
    tiempoRotacion++;
}

struct RxReturn RxPacket(void) {
    struct RxReturn respuesta;

    byte bCount, bLenght, bChecksum;
    byte Rx_time_out=0;
    Sentit_Dades_Rx(); //Ponemos la linea half duplex en Rx
    Activa_TimerA1_TimeOut();
    for(bCount = 0; bCount < 4; bCount++) //bRxPacketLength; bCount++)
    {
        Reset_Timeout();
        Byte_Recibido=0; //No_se_ha_recibido_Byte();
        while (!Byte_Recibido) //Se_ha_recibido_Byte())
        {
            Rx_time_out=TimeOut(1000); // tiempo en decenas de microsegundos
            if (Rx_time_out)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();
    }//fin del for

    if (!Rx_time_out) { // Continua llegint la resta de bytes del Status Packet

        // leemos los bytes siguientes, el numero de bytes correspondiendo a la length
        // que se recibe en la 3a posicion
        for(bLenght = 0; bLenght < respuesta.StatusPacket[3]; bLenght++) {
            Reset_Timeout();
            Byte_Recibido=0;
            while (!Byte_Recibido)
            {
                Rx_time_out=TimeOut(1000); // tiempo en decenas de microsegundos
                if (Rx_time_out)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+bLenght] = DatoLeido_UART; //Get_Byte_Leido_UART();
        }//fin del for

        byte checkSum = 0;


        for(bCount = 2; bCount < 4+respuesta.StatusPacket[3]-1; bCount++) {
            checkSum += respuesta.StatusPacket[bCount];
        }

    }

    // Desactivamos el timer
    Desactiva_TimerA1_TimeOut();

    return respuesta;
}
/**
 * Método para poder mover el motor, donde cada parametro indica el movimiento a realizar
 */
void move_motor(byte id_motor, byte sentido, uint32_t velocidad){
    byte Parametros[3];
    Parametros[0]=32; //el 32 corresponde al move speed
    Parametros[1]=velocidad; //este parametro correspondrá a la velocidad pasada por nosotros
    Parametros[2]=((sentido<<2)|(velocidad>>8)); // como el registro moving speed H tiene el valor del Turn Directi
                                                 // ejecutamos esta or para asignar cada valor donde corresponde
    TxPacket(id_motor,3,3,Parametros); //una vez tenemos los parametros, los enviamos. Al ser write, correspnde al

    RxPacket(); // CAMBIO
}
/**
 * Método para poder leer los valores del sensor
 */
struct RxReturn leer_sensor(){
    byte Parametros[2];
    Parametros[0]=26; //sensor left
    Parametros[1]=3; //numeros de sensores totaales
    TxPacket(100,2,2,Parametros); //los enviamos
    return RxPacket(); //recibimos la info y la devolvemos
}
```

2.2 lib_p4_robot.h

```
#ifndef lib_p4_robot_LCD_H_
#define lib_p4_robot_LCD_H_
```

```
#include <stdio.h>
#include <stdint.h>
typedef uint8_t byte;

void move_motor(byte id_motor, byte sentido,uint32_t velocidad);
void move_up(uint32_t velocidad);
void move_down (uint32_t velocidad);
void move_rotation(uint32_t velocidad);
void move_rotation_custom(uint32_t velocidadDerecha, uint32_t velocidadIzquierda);
void move_rotation_inverse(uint32_t velocidad);
void move_rotation_inverse_custom(uint32_t velocidadDerecha, uint32_t velocidadIzquierda);
void move_rotation_right(uint32_t velocidad);
void move_rotation_right_custom(uint32_t velocidadDerecha, uint32_t velocidadIzquierda);
void move_rotation_left(uint32_t velocidad);
void move_rotation_left_custom(uint32_t velocidadDerecha, uint32_t velocidadIzquierda);
void stop();

#define MOTOR_LEFT 4
#define MOTOR_RIGHT 2
#define SENSOR 100

#endif /* lib_p4_robot_LCD_H_ */
```

2.3 lib_p4_robot.c

```
/*
 * lib_p4_robot.c
 *
 *  Created on: 6 may. 2024
 *      Author: vadri
 */
#include "lib_p4_robot.h"

void move_up(uint32_t velocidad){
    move_motor(MOTOR_RIGHT,1,velocidad);//motor derecho
    move_motor(MOTOR_LEFT,0,velocidad);//motor izquierdo
}

void move_up_custom(uint32_t velocidadDerecha, uint32_t velocidadIzquierda){
    move_motor(MOTOR_RIGHT,1,velocidadDerecha);//motor derecho
    move_motor(MOTOR_LEFT,0,velocidadIzquierda);//motor izquierdo

}
```

```
void move_down (uint32_t velocidad){
    move_motor(MOTOR_RIGHT,0,velocidad);//motor derecho
    move_motor(MOTOR_LEFT,1,velocidad);//motor izquierdo
}
void move_rotation(uint32_t velocidad){
    move_motor(MOTOR_RIGHT,1,velocidad);//motor derecho
    move_motor(MOTOR_LEFT,1,velocidad);//motor izquierdo
}

void move_rotation_custom(uint32_t velocidadDerecha, uint32_t velocidadIzquierda){
    move_motor(MOTOR_RIGHT,1,velocidadDerecha);//motor derecho
    move_motor(MOTOR_LEFT,1,velocidadIzquierda);//motor izquierdo
}

void move_rotation_inverse(uint32_t velocidad){
    move_motor(MOTOR_RIGHT,0,velocidad);//motor derecho
    move_motor(MOTOR_LEFT,0,velocidad);//motor izquierdo
}

void move_rotation_inverse_custom(uint32_t velocidadDerecha, uint32_t velocidadIzquierda){
    move_motor(MOTOR_RIGHT,0,velocidadDerecha);//motor derecho
    move_motor(MOTOR_LEFT,0,velocidadIzquierda);//motor izquierdo
}


void move_rotation_right(uint32_t velocidad){
    move_motor(MOTOR_LEFT,0,velocidad);//motor izquierdo
}

void move_rotation_left(uint32_t velocidad){
    move_motor(MOTOR_RIGHT,1,velocidad);//motor derecho
}

void stop(){
    move_motor(MOTOR_RIGHT,0,1);//motor derecho
    move_motor(MOTOR_LEFT,0,1);//motor izquierdo
}
```

3 Diagrames de flux

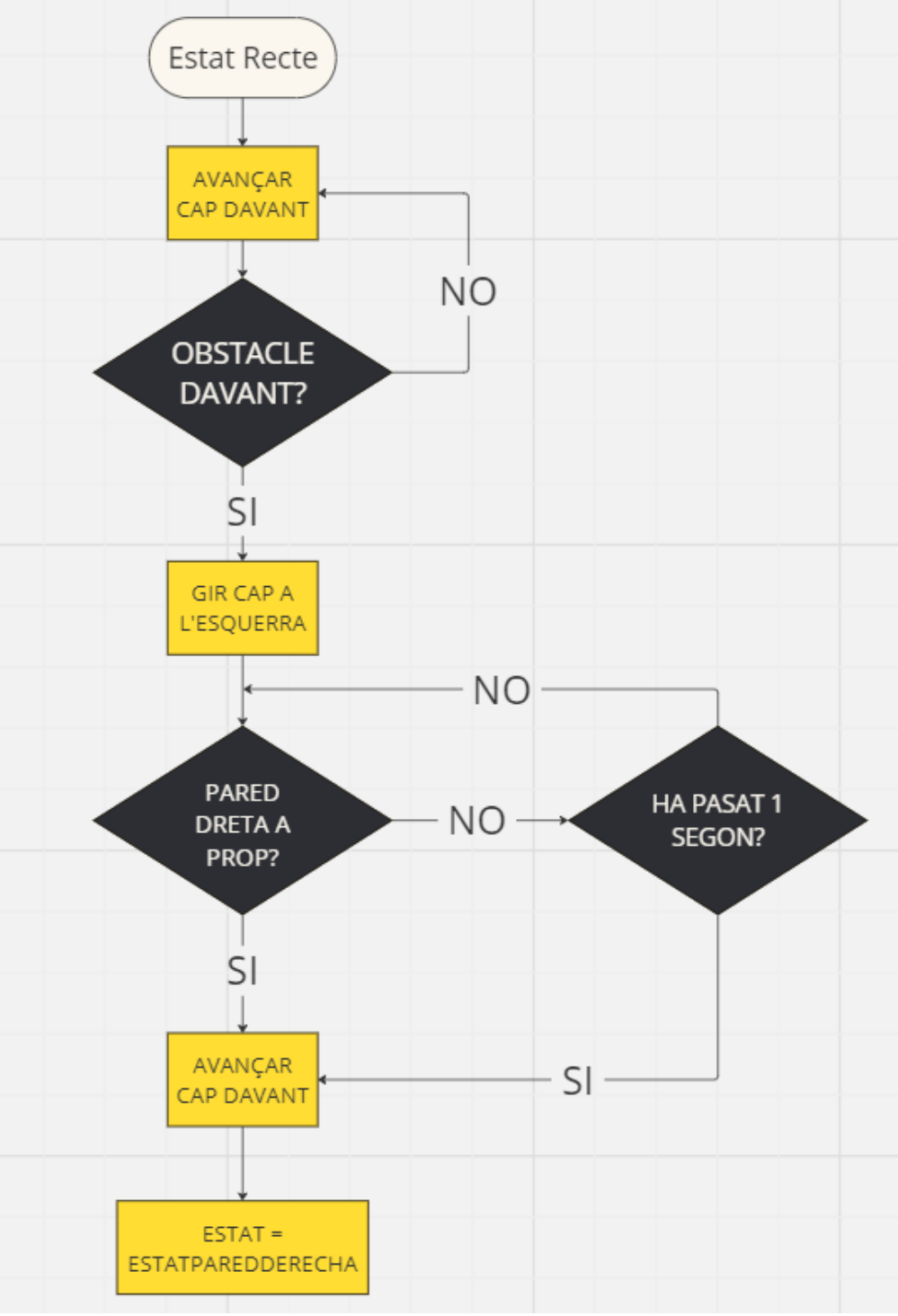


Figure 6: Diagrama Flux estado recto

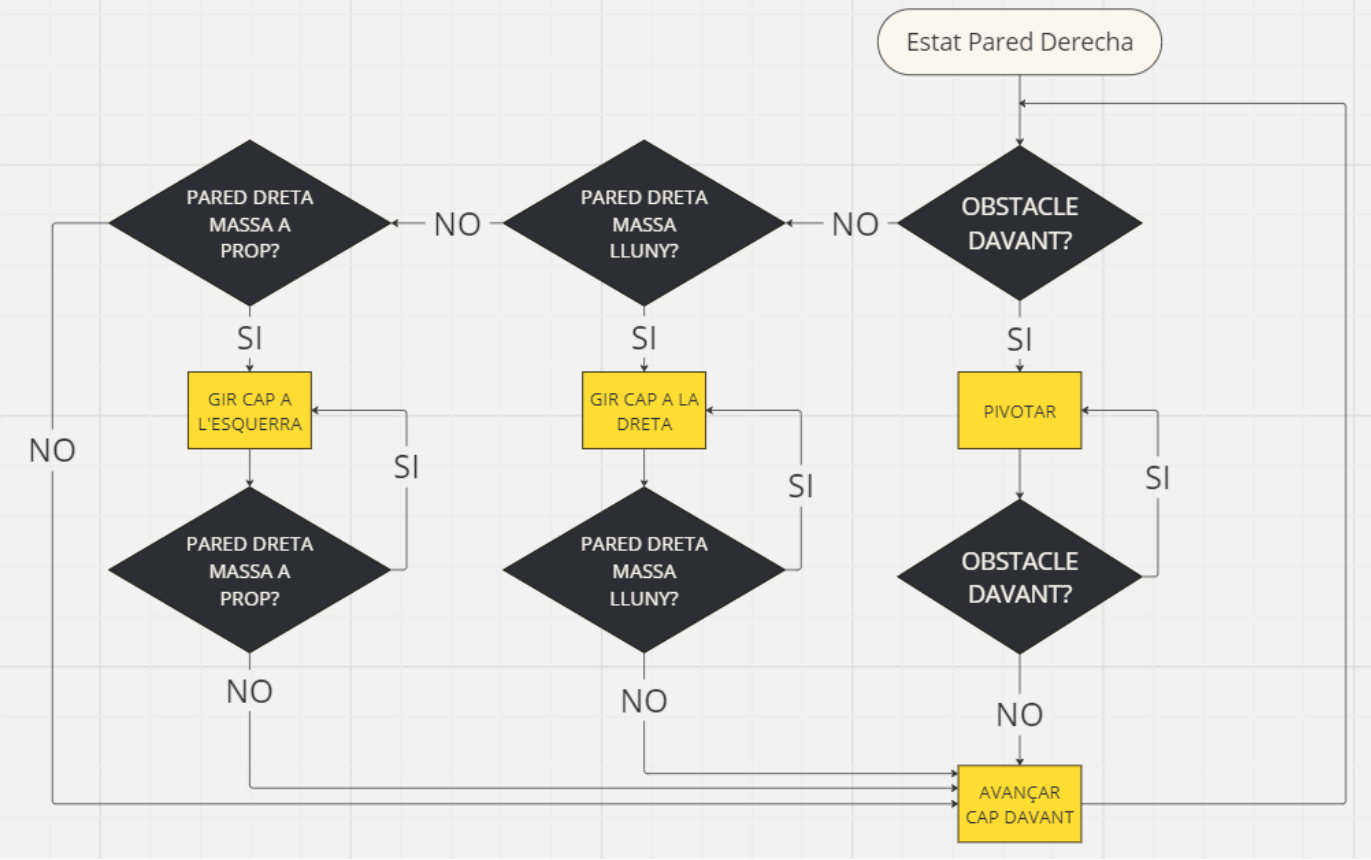


Figure 7: Diagrama Flux estado pared derecha