Programació d'un robot movil

Assignatura de Robòtica

Grup 10G:

E. Boronat Rosselló, M. Perelló Nieto

27 de juny de 2012

Índex

Índex de figures

Capítol 1

Introducció

El repte a resoldre en aquesta pràctica es disenyar un robot mobil capaç de moures al llarg del laberint de la figura 1. Les linies negres representen les parets del laberint. Els quadrads verds simbolitzen els encreuaments per fer-los detectables peral robot. Les franjes grises als extrems del laberint simbolitzen la entrada ila sortida.

Per afrontar aquesta tasca s'ha utilitzat un robot construït apartir del set lego mindstorm nxt 2.0 dels que és disposen al laboratori i que en la secció 2 se'n fa una descripció detallada de la configuració mecànica i la sensorització que se li ha instal·lat.

Com a aproximació per superar el laberint ens hem decantat per una arquitectura reactiva per dissenyar el robot. El sistema s'explica més extensament a l'apartat 3. Com a objectiu té trobar una franja gris en algun moment i el que tractara es de moures al llarg del laberint sense parar mentre no es trobi am algun obstacle o la sortida del laberint. Actuant de forma diferent segons els obstacles que trobi siguin negres (parets) o verds (encreuaments de camins). Presentarem l'implementació d'un algorisme de navegació que va generant un mapa a mesura que el robot es mou i que guia el robot a l'hora d'explorar el laberint. D'alguna forma es segueix l'esquema representat en el diagrama de la figura 2.

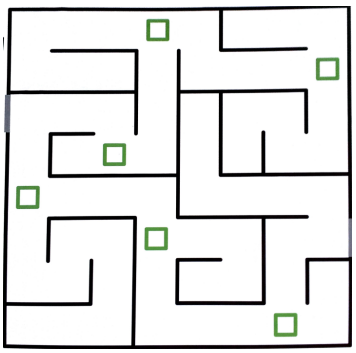


Figura 1.1: Mapa

El laberit a superar, que es mostra en la figura 3 te unes dimensions de X per Y i espodria dividir en cel·les quadrades de 20x20 cm. No obstant nosaltres percevem, i així ho implmentem per al robòt com un grafo d'obstacles cadascun dels quals esta relacionat amb altres obstacles. El que fa el nostre robot principalment es un algorisme de cerca voraz en que explorar les conexions del graph mantenint només en memoria la que li permet al final arribar al seu objectiu, la sortida.

Capítol 2

Descripció mecànica del robot

En aquesta secció explicarem quines decisions s'han pres a l'hora de construir el robot, el tipus de rodes, el tipus de sensors i les mides que s'han otorgat al robot per tal de moures adecuadament per les diferents posicions del laberint. En la figura 2.1 és poden veure les mides del robot.

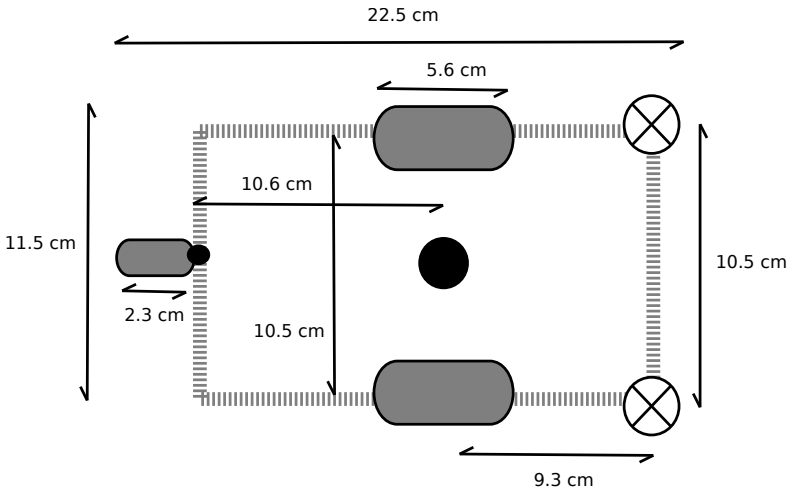


Figura 2.1: Mides del robot.

2.1 Motorització

A l'hora de dissenyar el robot es va optar per realitzar-lo amb dues rodes diferencials i una tercera roda "castor". Aquesta estructura ens dona la aventatge de poder rotar sobre si mateix, i d'aquesta manera no haver de maniobrar per agafar les curves e interseccions.

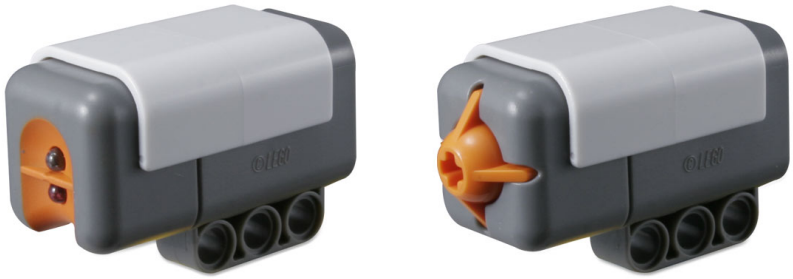
2.2 Sensorització

Per afrontar el repte del laberint hem decidit utilitzar dos sensors de llum NXT a la part davantera del robot. Aquets sensor estan totalment preparats per treballar amb el set de Lego. La seva sortida, directament accessible des de l'entorn de programació retorna un valor enter entre 0 i 100. El sensor bàsicament serveix per detectar blanc o negre. I fent que qualsevol color intermedi es trobe en un valor intermedi. En concret els valors amb que hem pogut calibrar per al nostre robot han estat.

Negre	**Verd i Gris**	**Blanc**
≥ 33	48 ± 5	≥ 60

Taula 2.1: Valors captats pels sensors segons el color.

Aquest sensor te la particularitat que just al salt entre blanc i negre dona presicament el valor intermedi i per tant la senyal que retorna es trova entre la franja del que podriem considerar verd o gris. De fet, qualsevol color que no sigue blanco negre es trovaria codficiat en aquesta franja pel sensor. Per tant un detall important a tenir en compte a l'hora d'implmentar les rutines de sensorització del robot.



(a) Sensor de llum. (b) Interruptor de posta en marxa.

Figura 2.2: Diferents periferics que s'han connectat al robot

Capítol 3

Descripció del programa

La nostra primera idea, i que hem seguit fins a completar el projecte, era mirar de fer un robot que no tingues una representació complicada del món i que no tractes de fer les coses dificils. Tot i això al principi haviem concebut el robot com dues capes de control diferents, una per al movimenti una altraper a la sensorització i decisió, però això complicava molt les coses ja que al funcionar en paral·lel com a taskes intedependents feia que fos molt difícil sincronitzar-les i donar el poder de control sobre una o l'altra al moment adequat. A més aquesta aproximació acabava convertint-se en un model molt complex. Després de diverses revisions vam acabar conceven un programa que tindria principalment dues funcions que s'executarien una despres de l'altra ciclicament. Aquestes serien Avançar i Percebre. La primera es una funció relativament senzilla i d'alguna forma en diriem que instintiva ja que la seva tasca principal es fer avançar el robot fins quees topi amb algun obstacle, matenint-se als centres dels passadis i rectificant la trajectoria si es desvia cap a una paret. La funció de percepció es més compleza, però la seva principal missió és saber on es troba el robot quan ha deixat de morues, que ha de fer i decidir com el rebot reaccióna davant de l'obstacle. Tot i que com a representació teorica considerem que el robot te dues funcions principals que desenvolupen aquestes tasques, en realitat n'hi ha algunes més que realitzen diverses tasques a petició de lesdues principals i per tant podriem dividir el programa en 3 moduls amb les seves respectives funscions especifiques.

Acció, percepció, reacció, navegació.

Per tant el programa implementat per fer navegar el robot a quedat reduït a una sola tasca integrada dins el main del programa. De totes formes però el primer que s'ha implementat es una segona tasca que no interfereix amb el main i que serveix principalment com a eina de debugat mostrant per pantalla els valors de les variables i l'estat intern del robot i on creu que es troba.

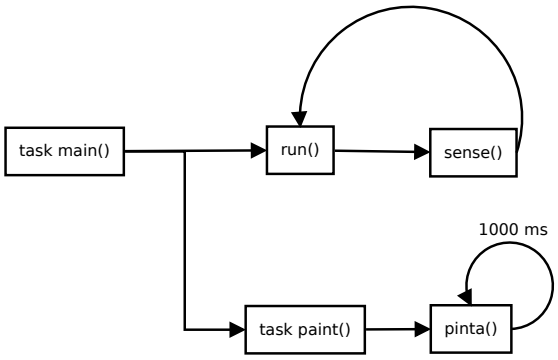


Figura 3.1: Diagrama d'estats del main.

3.1 Reacció del robot

La primera cosa que el robot prova de fer en activar-se l'interruptor es moures. La funció encarragada de fer-ho és la que es mostra a la figura 3.2.

Aquesta funció es una implementació d'un controlador proporcional discret que seguiria l'esquema de control de la figura 3.3. El que aquest controlador mira d'aconseguir es que els dos sensor vegin el mateix color si es Negre. Es considera l'error com la diferencia entre l'entrada del sensor dret i la del sensor esquerre.

D'aquesta manera sol·lucionem diversos problemes d'una sola forma. El primer es que quan acabem un moviment sapiguem que som perpendiculars a la paret, el segon que si ens desviem del centre del passadis ho poguem detectar facilment i recuparar el centre del passadis sense mecanismes més complicats utilitzant la funció de gir tal i com s'utilitza quant troba una paret de front o un quadre verd.

En essencia aquesta funcio fa avançar el robot mentre es trobi sobre blanc. Quan es trobi sobre negre, mirara d'aliniar-s'hi quedant perpendicular a la paret. La velocitat dels motors es en funció de l'error dels sensors, per tant el seu moviment es banstant precis i ràpid a l'hora d'aliniarse cap a la paret. Això explica les 3 primeres condicions del controlador. La tersera, que va en funció de les mostres consecutives de verd que s'han trobat es fa servir per poder discernir un canvi de negre a blanc o de blanc a negre d'un quadrad o sortida. Això es fa per que just en el canvi d'un color a l'altre el sensor llegeix un valor intermig que es molt proper al verd i el robot s'aturava pensant que trovaba verd. Per això en lloc d'aturarse a la primera ho fa quan detectar 10 mostres de verd. D'aquesta maenra quan s'arriba una mica torçat a una cruilla o la sortida, el robot s'hi alinia una mica ajudant a que els moviments següents siguin més precissos.

La funció run() retorna per poder executar el sense() quan s'ha quedat amb els dos sensors a negre i no ha rotat més de 40 tics o quan ha recollit 10 mostres consecutives de verd. Si al aliniar-se a una paret rota més de 40 tics la funció retorna despres de fer un gir de 90 graus en el sentit contrari de l'alineació per recuperar el centre del passadis, però al retornar no es cridara el sense i tornar a comenaçar a executar-se el run un altre cop fins a un obstacle real.

Cada cop que el robot topa amb una paret o amb una intersecció (quadrat

verd) s'atura i cedeix el control a la funció 'sense', aquesta funció avalua totes les dades amb les que compta, i decideix en quin estat es troba.
Un cop decidit l'estat, realitza una acció o una altra, demanant al mapa tota la informació necessària per explorar o realitzar els canvis oportuns.

En la figura 3.4 es pot veure el diagrama de execució d'aquesta funció.

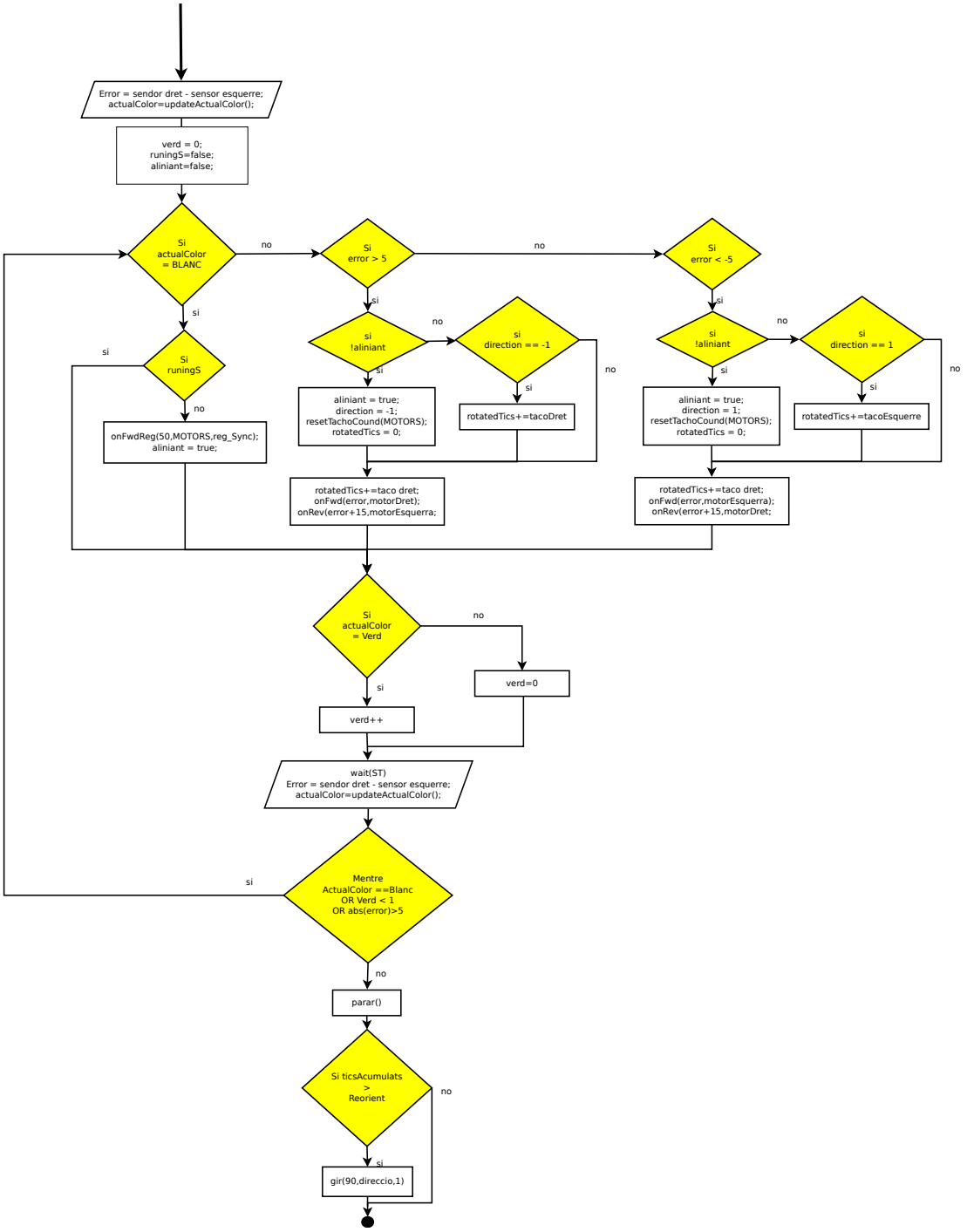


Figura 3.2: Diagrama d'estats de la funció run.

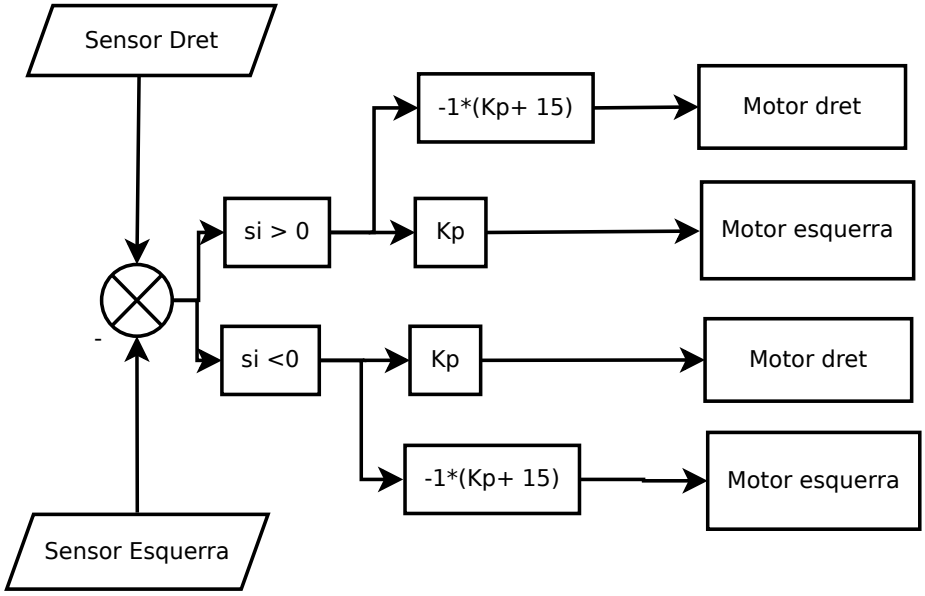


Figura 3.3: Diagrama de control del robot mentre avança.

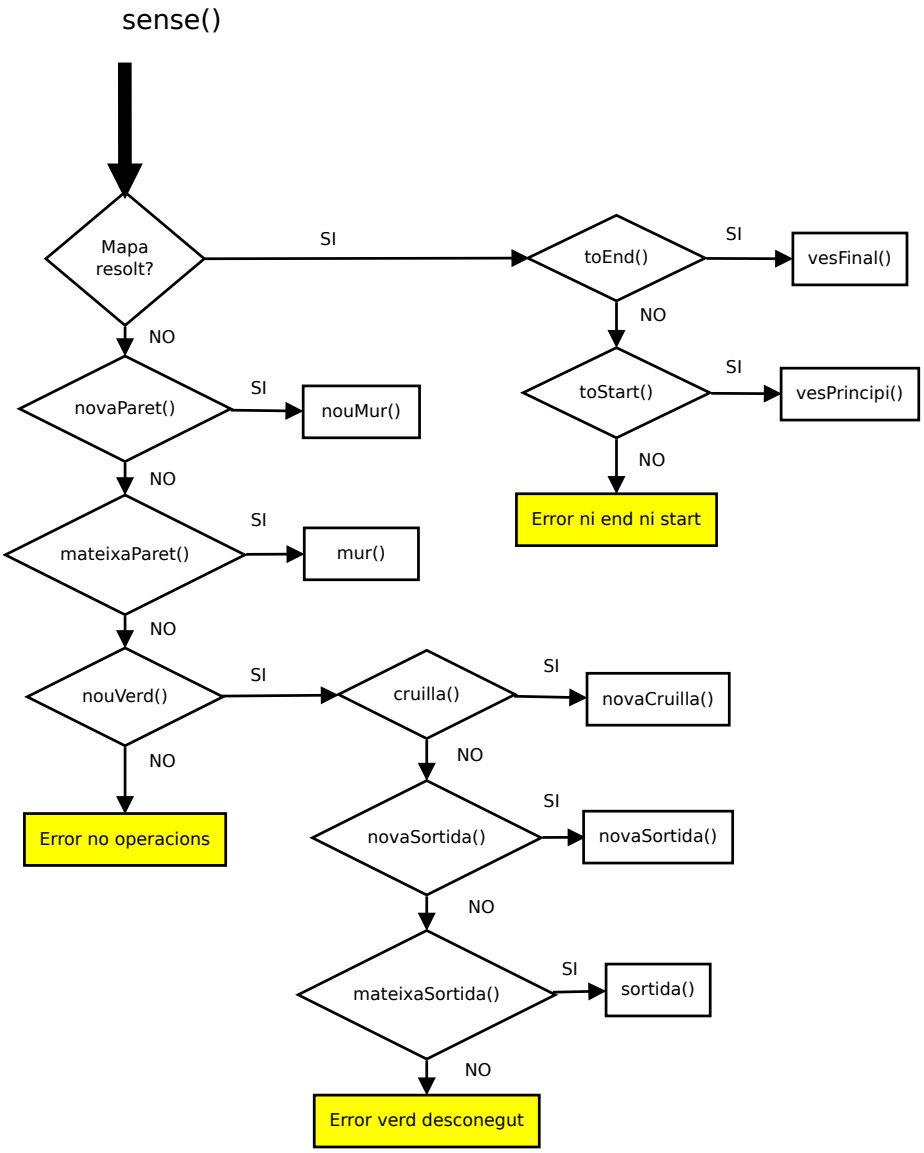


Figura 3.4: Diagrama d'estats de la funció sense.

3.2 Pantalla

Gracies a la pantalla que ens proporciona el NXC de lego, podem visualitzar moltes dades de l'estat en el que es troba el nostre robot. Tota aquesta informació ens facilita molt la tasca de comprovar el bon funcionament, i revisar els errors que surgeixen.

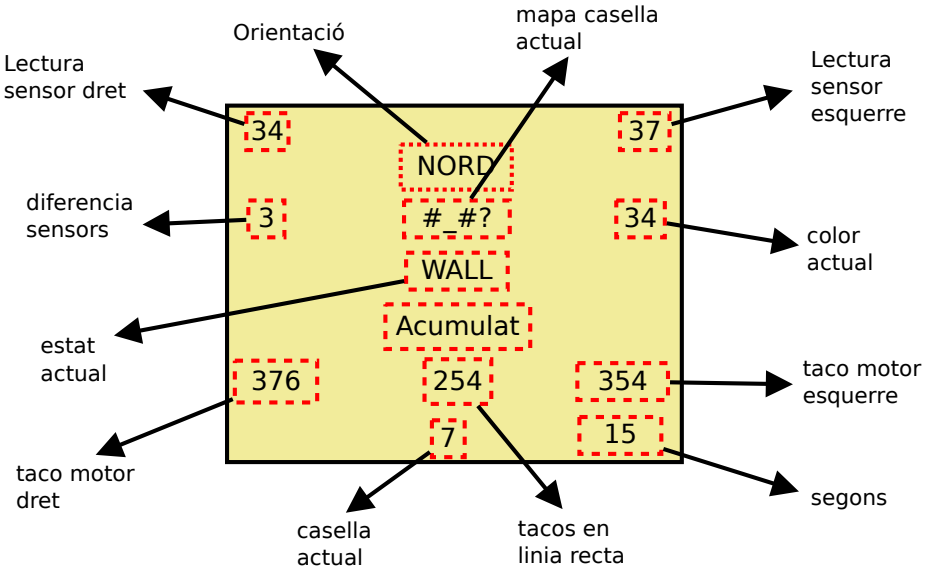


Figura 3.5: Descripció dels elements de la pantalla.

La informació que hem considerat necesaria ha estat la següent:

- Lectura del nivell de llum del sensor dret i esquerre.
- Diferencia de nivells de lectura dels dos sensors, per crear l'error i poder guiar el robot amb el nostre algorithme.
- Color actual, que és el color menys lluminos sota dels dos sensors.
- Lectures dels dos tacometres dels motors; dret i esquerra; que ens permeten comprovar que està contant correctament.
- Nombre de tics dels tacometres en linia recta, sense contar girs ni reajustaments amb les parets.
- Segons que porta engegat el robot.
- La orientació en la que es troba el robot; nord, est, sud i oest. Prenent com a nord la primera orientació que se li dona al engegar-lo.
- Representació del mapa de la casella actual; aquesta representació consta de quatre símbols que representen l'estat de cada orientació de la casella en el següent ordre: nord, oest, sud i est. I el simbols que pot pendre són:

: Orientació amb una paret (o amb un camí sense sortida, un cop s'ha explorat).

? : Orientació inexplorada.

_ : Orientació amb un camí.

- : Error (mai hauria de ser-hi).

- Estat d'execució en el que es troba el robot, el qual es pot trobar en els següents:

STOP
Amb els motors parats.

RUN
Moviment en linia recta.

CRUILLA
Al trobar-se amb una cruïlla (quadrat verd).

WALL
S'ha trobat amb una paret en la mateixa casella on es trobava.

NEWWALL
S'ha trobat amb una paret i venia d'una altre casella.

EXIT
Ha trobat la sortida del laberint en una banda de la casella on es trobava.

NEWEXIT
Ha trobat la sortida del laberint i venia d'una altre casella.

TOSTART
Amb el mapa memoritzat està decidint el camí per tornar al principi del laberint.

TOEND
Amb el mapa memoritzat està decidint el camí per anar al final del laberint.

ALINIAR
Ha tocat una paret lateral mentre anava endavant i s'està ajustant un altre cop al centre del passadís.

POU
Ha explorat totes les direccions d'una casella i ha vist que no te sortida, així que torna enrere i marca la casella com una paret.

SORTIDA

STOP

STOP

- Estats d'execució en els que apareix un error:

ORINEG
Al escollir direcció no ha trobat cap explorable.

ORISAME
Al intentar explorar, l'orientació inicial i final resulten ser la mateixa.

NOOP
El robot es troba en una situació en la que no es pot identificar l'estat actual.

STARTEND
Un cop memoritzat el mapa el robot es troba en una situació en la que no es pot identificar l'estat actual.

UNKGREEN
Amb els sensors en un color diferent del negre i blanc no sap identificar l'estat.

ERR_X
Error al accedir a una posició de memòria del mapa incorrecte.

ERR_Y
Error al accedir a una orientació del mapa incorrecte.

UNKNOW
Error desconegut.

3.3 Representació del mapa

La primera aproximació que es va fer en l'intent de representar el mapa en la memòria del robot va ser la de dividir el mapa en una matriu de 8x8 amb un total de 64 caselles (veure figura 3.6). Aquestes caselles tindrien cadascuna unes orientacions amb paret i sense.

Els problemes que vam veure que tindríem van ser els de mesurar correctament totes les distancies que es recorrien, per identificar els canvis de casella, així que es va optar per trobar una nova solució.

Tot seguit vam observar que moltes de les caselles no ens servien per decidir la direcció, ja que només eren posicions intermitges entre caselles en les que sí s'havia de prendre decisions de gir, per tant la aproximació va ser la d'eliminar totes aquestes caselles i quedar-nos només amb les que ens aportaven informació rellevant, d'aquesta manera ja no es veia com una matriu, sinó un graf amb diferents nodes, cadascun d'ells amb informació sobre totes les seves direccions (veure figura 3.7). Amb aquesta aproximació es reduïa el nivell màxim de nodes a 40, i només havíem de vigilar si en cada moment ens trobarem en una mateixa casella, o havíem recorregut una distancia d'una casella, lo qual ens indicaria que hem sortit de la anterior, i hem entrat en una nova.

Per representar aquesta aproximació el mapa es pot veure com un arbre binari, amb arrel la casella inicial, i com a nodes fills els diferents camins. El camí de l'esquerra es sempre el més prioritari, i el de la dreta es recorre en segon lloc, per tant tindríem l'ordre Nord, Oest, Sud, Est, on el primer és el més prioritari i per tant va a un fill esquerra. Un cop amb aquesta representació, només cal recorre l'arbre en profunditat, i tancant aquells nodes que ja han estat visitats (aquesta representació es pot veure en la figura 3.8). Al explorar en profunditat i esborrar els nodes en els que hem realitzat "backtraking"aquest arbre es pot anar creant sobre un array amb quatre posicions.

Per tant finalment s'ha creat una matriu de 4x40 (es pot veure en la figura 3.9), en la qual hi ha les 40 caselles màximes que es poden arribar a recórrer,

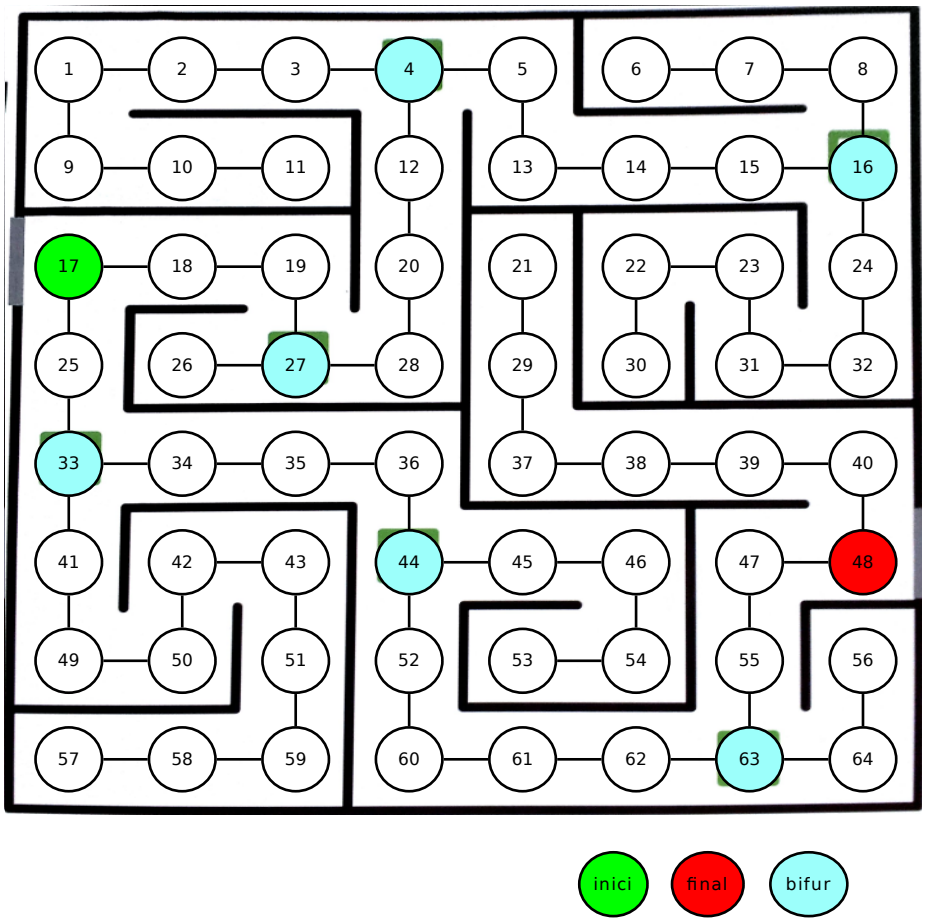


Figura 3.6: Primera aproximació de la representació del mapa

i les quatre orientacions diferents que tenen, amb l'estat en el que es poden trobar:

-1 : Hi ha una paret, o s'ha explorat i no hi ha sortida.

0 : No ha estat explorat encara

1 : No hi ha cap paret.

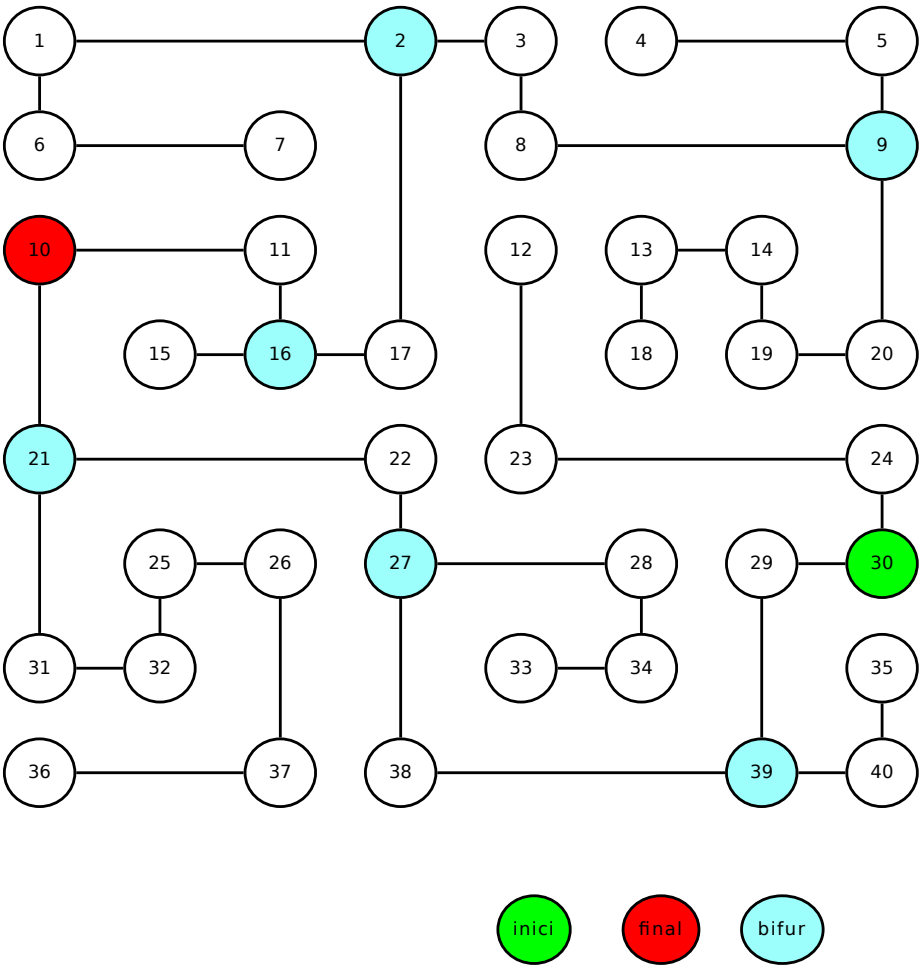


Figura 3.7: Optimització de la representació del mapa.

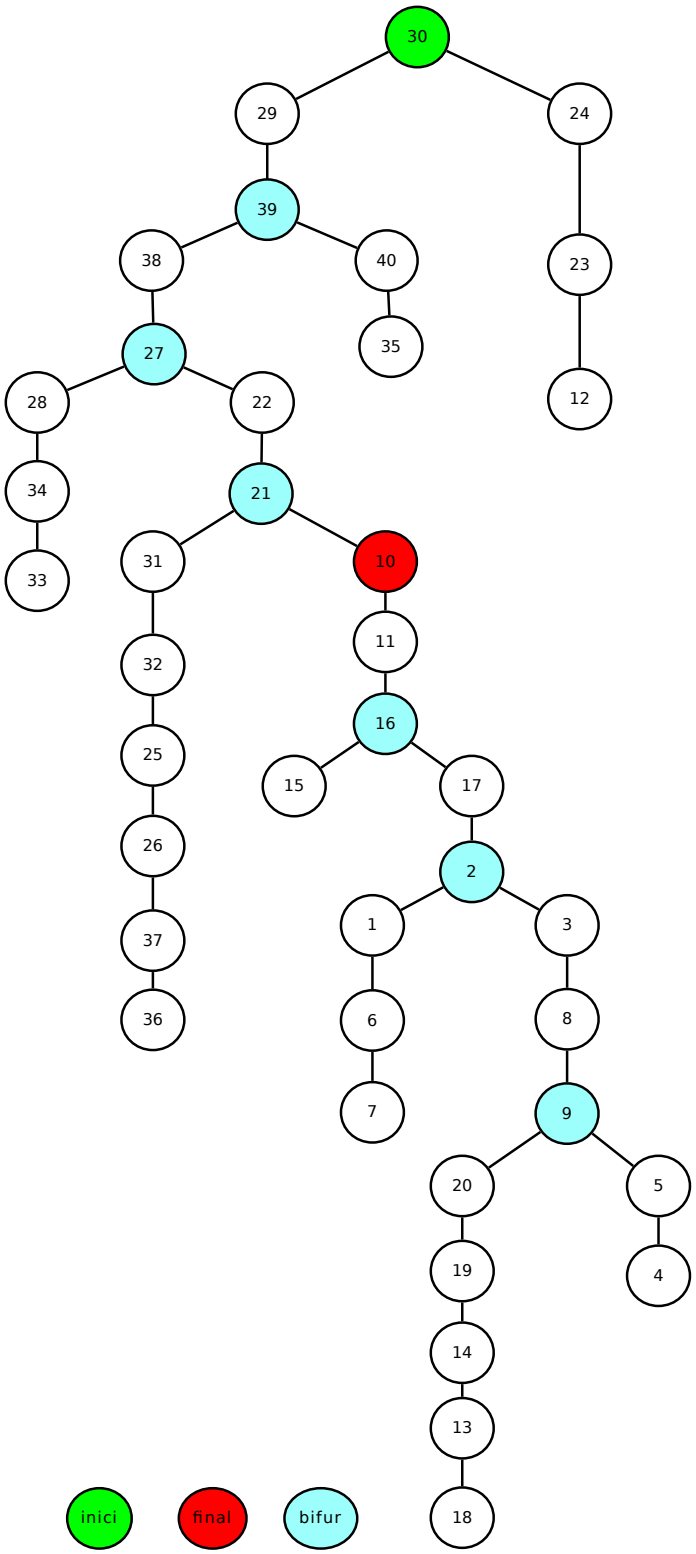


Figura 3.8: Representació en arbre del mapa optimitzat.

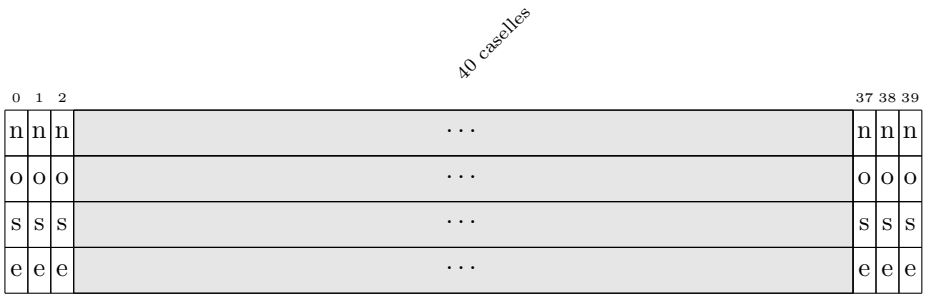


Figura 3.9: Estructura per emmagatzemar el mapa, on les lletres 'n', 'o', 'e' i 's' són els estats de cada orientació en la casella, amb valors -1, 0, i 1 (paret, desconegut i obert respectivament).

3.4 Exploració del mapa

Un cop tenim clara la representació del mapa en el robot, el següent pas es com s'inicialitza, es recorre, i es troba el camí optim de l'inici del laberint al final. L'algoritme que hem utilitzat per explorar tot el mapa és el següent:

Explora :

1 Si no s'ha explorat encara el Nord

- Explora el Nord.

2 Si no s'ha explorat encara l'Oest

- Explora l'Oest.

3 Si no s'ha explorat encara el Sud

- Explora el Sud.

4 Si no s'ha explorat encara l'Est

- Explora l'Est.

5 Si ja s'ha explorat tot

- Esborra la casella.
- Marca un pou a la casella anterior.
- Torna a la casella anterior.
- Explora.

Això ens ha permès explorar totes les direccions possibles per totes les caselles, ja que si algo no ha estat explorat al final es tornarà a la casella i s'explorarà (sempre i quan no s'hagi trobat la sortida abans). També ens permet marcar els camins sense sortida com a pous, i per tant mai més seran explorats. Dintre de la exploració de cada orientació es fa el següent:

1 Si xoques amb una paret de la mateixa casella.

- Marcala com a paret.
- Explora.

2 Si xoques amb una paret d'una nova casella.

- Crea una nova casella i marca la direcció actual com a paret.
- Marca la direcció contraria com a camí.
- Marca la orientació actual de la casella anterior com a camí.
- Explora.

3 Si xoques amb un quadrat verd.

- Crea una nova casella.
- Explora.

4 Si xoques amb la sortida.

- Crea la casella de sortida.
- Marca el camí optim per arribar.
- Tanca tots els estats com a explorat.
- Espera l'avís per anar del principi al final.

Per tant podem veure en la figura 3.10, totes les caselles per les que passarà el robot per trobar la sortida, havent-lo de cara a l'esquerra (i per tant amb el nord al seu front). Recorrerà primer totes les caselles que pugui cap al seu nord, i en el moment que no pugui recorrerà oest, sud i est. En aquest exemple es poden veure dos pous, un en la intersecció 5, i un altre en la intersecció 13.

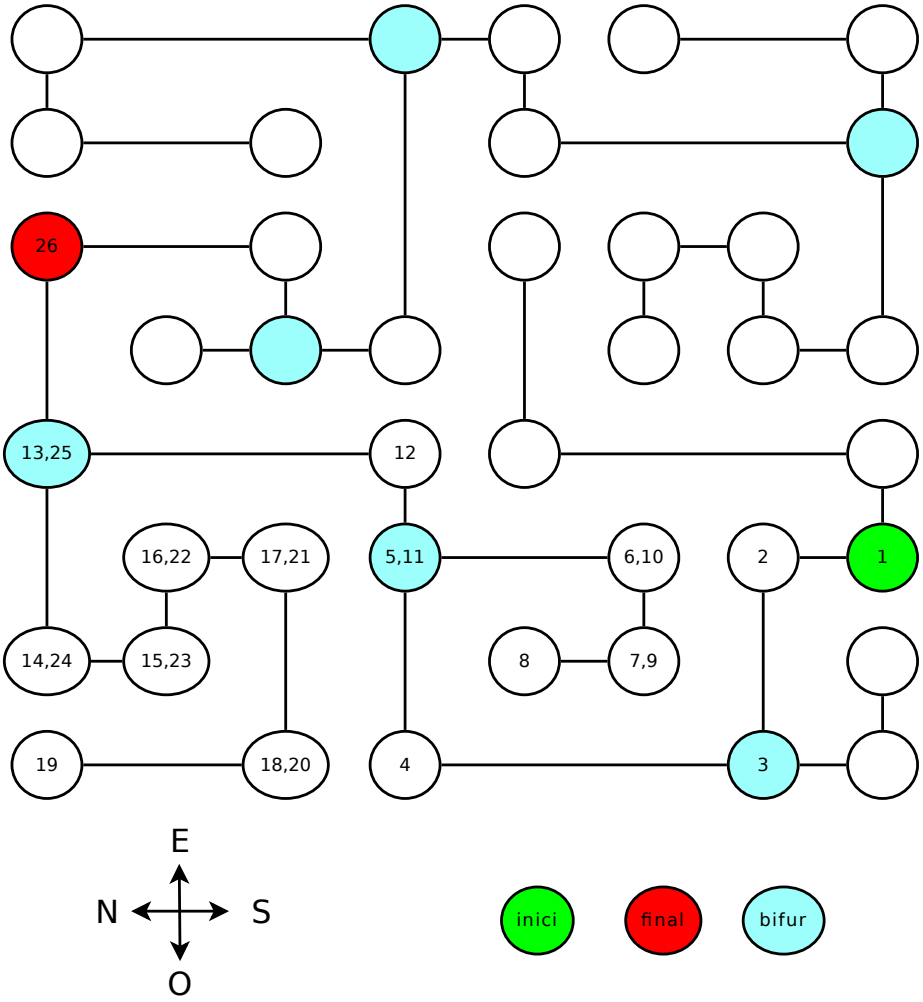


Figura 3.10: Exemple en l'ordre d'exploració del mapa.

Un cop s'ha trobat la sortida, el robot observar des de el final fins al principi quins camins hi ha inexplorats, i els marca com a parets, d'aquesta manera

només queden les caselles amb la orientació d'origen i destí. Els pous ja han estat marcats anteriorment com a parets, per tant ja no ens han de preocupar. En la figura 3.11 es pot veure quines caselles queden en el mapa del robot amb les quals pot arribar de manera òptima.

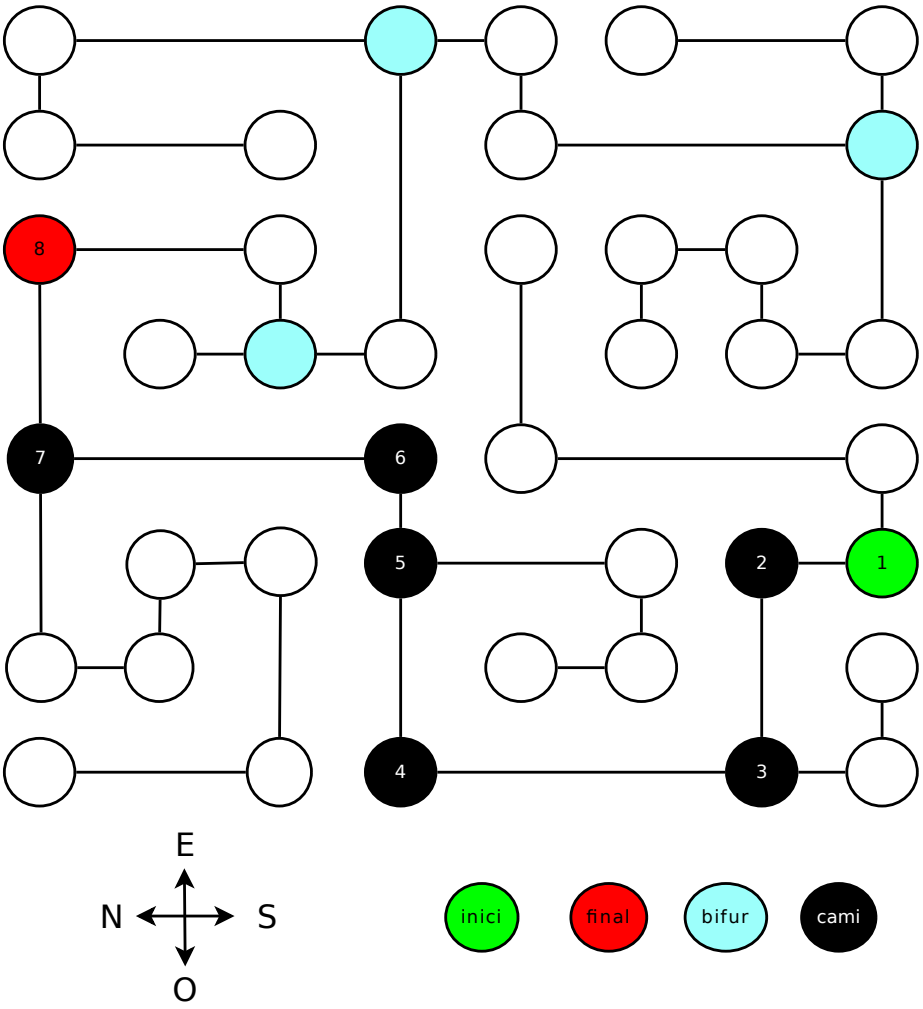


Figura 3.11: Exemple de la representació del camí optim.

3.5 Funcions importants comentades

3.5.1 run()

Aquesta funció es l'encarregada del moviment del robot a traves del laberint. Es una tasca totalment instintiva. Un cop el robot s'ha proposat moures, la funció el fa abançar fins topa amb algun obstacle. Pel cami la tasca s'encarrega de mantenir el robot en línea recta mentre no trobi cap obstacle. Si algun dels sensors detectar negre el robot tractara d'alinear-si tractant de minimitzar

l'error entre tots dos sensors. De forma senzilla s'ha implementat un control proporcional. Si en el proces d'alineació el robot gira més d'una certa proporció considerara que s'ha desviat durant el desplaçament i retornar al centre del passadis, cosa que fara que la funció de percepció no entri en joc i el robot es tornarà a posar e marxa fins a un obstacle real del laberint. Moment en que retorna i atura el robot fins que alguna funció de moviment s'activi.

3.5.2 gir(int graus, int sentit)

Aquesta funció s'encarrega de fer girar el robot el nombre més o menys exacte de graus que se li indica, i en quin sentit ho ha de fer. El paràmetre rev serveix per indicar-li si al moment de fer el gir s'ha de desplaçar una mica enrere o ho ha de fer sobre si mateix.

3.5.3 throwError(int error)

Aquesta funció només es crida quan succeeix un error inesperat, en aquest cas el mapa es veu corromput, i per tant el que fa es parar el robot, avisar amb varis tons d'error i posar l'estat actual amb l'error. I per ultim inicialitza el mapa de nou per poder seguir explorant amb un mapa net, ja que en cas contrari el mapa podria impedir-nos trobar el final.

3.5.4 int mapExploreDirection()

Aquesta funció es l'encarregada d'explorar una casella del mapa, el que fa es retornar la primera orientació que veu inexplorada. En cas que no existeixin caselles inexplorades vol dir que es troba en un camí sense sortida i per tant elimina aquesta casella, marca la casella anterior amb una paret en aquesta direcció, i retorna la orientació que portarà al robot a la casella anterior.

3.5.5 int mapNewCell(int paret, int ori)

Aquesta funció es crida quan s'ha accedit a una casella nova, i se li ha de dir amb la variable paret si és una paret o una intersecció. En el cas de que sigui una paret marca aquesta orientació com a paret. En els dos casos es marca la orientació actual de la casella anterior com a camí, i la orientació contraria de la casella actual també. Un cop s'ha marcat d'aquesta manera el mapa, es demana a la funció mapExploreDirection que ens digui quin camí seguir en aquesta casella.

3.5.6 int mapWall(int ori)

Aquesta funció es crida quan s'ha topat amb una paret i ens trobem a la mateixa casella, el que fa es marcar aquesta orientació com a una paret i demanar quina es la següent casella a explorar en la situació actual.

3.5.7 int mapEndCell(int newCell, int ori)

Aquesta funció es crida quan s'ha arribat a la marca de final de laberint. Se li ha d'indicar si es una casella nova o es trobaba a la mateixa casella. En cas

de que sigui nova s'afegeix aquesta nova casella. En ambdos casos es tanquen amb parets totes les direccions que no han estat explorades fins ara, i d'aquesta manera es genera el camí optim desde la casella inicial fins a la final. Tot seguit s'ajusten la resta de variables per indicar que el mapa ja ha estat explorat i es posa en mode per tornar a recórrer el mapa.

3.5.8 int mapSelectDirection(int nori)

Aquesta funció només es crida un cop el mapa ha estat explorat, i el que fa es seleccionar la orientació a escollir tenint en compte que no volem anar per la de provinença, per tant explorarà de principi a final, o de final a principi.

3.5.9 int mapOrToDegrees(int oriStart, int oriEnd)

Aquesta funció tradueix una rotació amb una orientació d'origen i una de destí, en els graus positius que ha de girar el robot per orientar-se.

3.5.10 int detectExit()

Aquesta funció comprova si la casella en la que es troba és un quadrat verd o es la sortida del laberint, mitjançant una serie de moviments i comprovacions que tenen en compte les diferents mides de cada casella.

3.5.11 int orienta(int actualOri, int finalOri, int rev)

Aquesta funció orienta el robot cap al destí indicat, utilitzant la funció que li indica els graus a girar, aquest gir pot ser sobre si mateix o retrocedint una mica. Això se li indica mitjançant la variable rev, i és degut a la situació en la que es troba en un quadrat verd, o si en canvi es una paret.

3.5.12 void sense()

Aquesta funció decideix en quin estat es troba el robot actualment, i per consegüent quina es la acció que hauria de realitzar. Detecta si es troba en una casella nova, o en la mateixa, si és una intersecció o camí final, i si está explorant el mapa, o ja l'ha explorat i ha d'anar al principi o al final. Es pot veure el diagrama de flux en la secció 3.1 on s'explica realment cada cas.

3.5.13 task paint()

Aquesta es la segona tasca que s'executa en el programa, i periòdicament pinta per pantalla totes les dades necessaries per comprovar l'estat del robot. Bàsicament es una ajuda per nosaltres, però no es necessaria per el bon funcionament del robot. Ara mateix s'executa a una freqüència de 1Hz, el que ens permet no col·lapsar el microcontrolador del NXC i permetre veure l'estat en cada casella. Es pot veure en la secció 3.2 tota la informació que ens mostra.

Capítol 4

Codi comentat

```
/**
* SENSORITZACI
**/

#define RS Sensor(IN_1)
#define LS Sensor(IN_4)
#define RT MotorTachoCount(OUT_A)
#define LT MotorTachoCount(OUT_C)
#define SB Sensor(IN_2)

/**
* ACTUADORS
**/
#define MOTORS 4
#define VOLUM 3
#define LOOP false

/**
* PARAMETRES CONTROLADOR
**/
#define Kproportional 1
#define ST 25


/**
* PARAMETRES DEL ROBOT
**/
#define DEG_OFFSET 7
#define D_RODA 56
#define D_VEHIC 114
#define AMPLADA 2000        // amplada en centimetres
    d'una casella
#define TICKS_AMPLADA  300  // si no aban a tot aixo
    abans de trobar una paret es que es troba a la
```

```
    mateix cantonada
#define REORIENT 45     //si gira m s d'aix  es que
    anava tor at pel passadis i no es una paret

/**
* VALORS DELS COLORS
**/
#define WHITE 60
#define GREEN 48
#define BLACK 30
/**
* state : Variable on es guarda l'estat actual d'
    execuci  del robot
* 0 = STOP          (el robot est  aturat)
* 1 = RUN           (el robot va totalment recte, si
    trova una parat si alinia)
* 2 = CRUILLA         el robot es trova en una
    cru lla)
* 3 = PARET         (el robot ha trobat una paret)
* 4 = PARET2   (el robot habia trobat una paret i
    torna a una altra)
* 5 = PARET3   (el robot troba un cul de sac)
* 6 = ALINEAR  (el robot tracta de recuperar el centre
     del passadis)
* 7 = RETURN   (el robot esta tornant al inici)
* 8 = SOLVING  (el robot esta anant del principi al
    final)
*/
#define STOP    0
#define RUN     1
#define CRUILLA 2
#define WALL    3
#define NEWWALL 4
#define EXIT    5
#define NEWEXIT 6
#define TOSTART 7
#define TOEND   8
#define ALINIAR 9
#define POU   10
#define SORTIDA 69


#define ORINEG    -1
#define ORISAME   -2
#define NOOP    -3
#define STARTEND  -4
#define UNKGREEN  -5
#define ERR_X   -6
#define ERR_Y   -7
#define UNKNOW    -9
```

```
75
76
77 /**
78 * VARIABLES
79 **/
80
81 //flux control
82 int mazesolved;              //indica si el robot ha
      trobat la sortida o no.
83 int state;                   //indica que esta fent el
      robot.
84 int direction;
85 int actualColor;
86
87 int toInici = 0;
88 int ticksAcumulats = 0;
89
90 //variables del controlador
91 int error;
92 int speed = 50;
93
94
95 /**
96 * FUNCTIONS
97 **/
98 // Para el robot
99 void parar()
100 {
101     Off(MOTORS);
102 }
103
104 void mapInit();
105
106 void throwError(int error)
107 {
108   parar();
109   PlayToneEx(1000,3000,VOLUM,LOOP);
110   Wait(3000);
111   state = error;
112   PlayToneEx(1200,5000,VOLUM,LOOP);
113   parar();
114   Wait(5000);
115
116   // FIXME: d'aquesta manera mai parara
117   // Torna a comenar com si estes a la
118   // primera casella, per poder seguir i
119   // acabar trobant el final.
120   mapInit();
121 }
122
```

```
123 /*
124 * posa l'estat a UNKNOW i para el robot
125 */
126 void debug(int error)
127 {
128   parar();
129   PlayToneEx(1000,3000,VOLUM,LOOP);
130   Wait(3000);
131   state = error;
132   PlayToneEx(1200,5000,VOLUM,LOOP);
133   parar();
134   Wait(5000);
135 }
136
137 void victory()
138 {
139   PlaySound(3);
140   Wait(5000);
141 }
142
143 // Actualitza estat actual dels sensors donant el
    valor del m s petit o del verd.
144 int updateActualColor()
145 {
146  int actualColor;
147      if ((RS<GREEN+5)&&(RS>GREEN-5))
148         {actualColor=GREEN;}
149      else if ((LS<GREEN+5)&&(LS>GREEN-5))
150         {actualColor=GREEN;}
151      else if (RS<LS)
152          {actualColor=RS;}
153      else
154          {actualColor=LS;}
155      return actualColor;
156 }
157 // Retorna els tics de motor que ha de fer el robot
    per girar els graus d'entrada
158 int deg2ticks (int degrees)
159 {
160  return (degrees-DEG_OFFSET)*D_VEHIC/D_RODA;
161 }
162
163 // Comprova si al surtir del run ho ha fet be o sa
    parat per un canvi de blanc a negre
164 int checkSense(int actualColor)
165 {
166  if (updateActualColor()==actualColor)
167  {
168     return 1 ;
169  }
```

```
 else
 {
     return 0;
 }
}
//Fa girar el robot els graus donats en el setit donat
    (1 positio -1 negatiu)
int gir(int degrees,int direction,int rev)
{
  int ticks = deg2ticks(degrees);
  int gspeed = direction*50;
  if (rev)
  {
  OnRevSync(OUT_AC,50,OUT_REGMODE_SYNC);
  Wait(500);
  }
  ResetTachoCount(OUT_AC);
  OnFwd(OUT_A, gspeed);
  OnRev(OUT_C, gspeed);
  int runA = 1;
  int runC = 1;
  while (runA || runC){
     if ((runA == 1) && (abs(MotorTachoCount(OUT_A)) >
         ticks))
     {
      runA = 0;
      Off(OUT_A);
     }

     if ((runC == 1) && (abs(MotorTachoCount(OUT_C)) >
         ticks))
     {
      runC = 0;
      Off(OUT_C);
     }
 }
}


// Aban a recte fins que arriba a un encreuament o a
   una paret frontal
void run()
{
  error = RS-LS;
  actualColor=updateActualColor();
  int aliniant = 0;
  int rotatedTics=0;
  int initR=0;
  int initL=0;
  int runningS = 0;
```

```
216   int verd = 0;
217
218   while (((abs(error)>5)||(actualColor>=WHITE-7))&&(
          verd<10))
219   {
220
221     if (actualColor>WHITE-7)
222     {
223       if (!runningS)
224       {
225         ticksAcumulats = ticksAcumulats + RT;
226         runningS=1;
227         OnFwdReg(MOTORS, speed, OUT_REGMODE_SYNC);
228       }
229     }
230     else if (error>5)
231     {
232       if (aliniant==0) // negre a l'esquerra
233       {
234         ticksAcumulats = ticksAcumulats + RT;
235         //ficar variabla Miquel
236         ResetTachoCount(OUT_AC);
237         direction = -1;
238         aliniant=1;
239         //debug();
240       }
241       else if (direction==-1)
242       {
243          rotatedTics=rotatedTics+RT;
244       }
245       OnFwd (OUT_A,error);
246       OnRev (OUT_C,error+15);
247       runningS=0;
248     }
249     else if (error<-5) // negre a la dreta girar la
          roda esquerra
250     {
251       if (aliniant==0)
252       {
253         ticksAcumulats = ticksAcumulats + RT;
254         //ficar variable Miquel
255         ResetTachoCount(OUT_C);
256         aliniant=1;
257         direction=1;
258         //debug();
259       }
260       else if (direction == 1)
261       {
262         rotatedTics=rotatedTics+(LT);
263       }
```

```
264       OnFwd (OUT_C,-error);
265       OnRev (OUT_A,-error+15);
266       runningS=0;
267     }
268     Wait(ST);
269     error = RS-LS;
270     actualColor=updateActualColor();
271     if (actualColor==GREEN)
272     {
273       verd = verd+1;
274     }
275     else
276     {
277       verd = 0;
278     }
279   }
280   ticksAcumulats = ticksAcumulats + RT;
281   parar();
282   if (abs(rotatedTics)>REORIENT)
283   {
284     state = ALINIAR;
285     PlayToneEx(440, 500,VOLUM,LOOP);
286     //debug();
287     gir(90,direction,1);
288   }
289 }
290
291
292 /*
293  /'\_/`\
294 /\       \      __      _____
295 \ \ \__\ \    /'__`\   /\ '__`\
296  \ \ \_/\ \  /\ \L\.\_\ \ \L\ \
297   \ \_\\ \_\\ \__/.\_\\ \ ,__/
298    \/_/ \/_/ \/__/\/_/  \ \ \/
299                         \ \_\
300                          \/_/
301 */
302
303 /**
304 Resum de les funcions per crear el mapa: header.
305 llegir tambe resum de la part map saved.
306
307 // funcio per inicialitzar el mapa
308 void mapInit();
309
310 // funcio al trobar una paret (no al corretgir
      direccio)
311 // o un quadrat verd
312 // paret = 1, si es una paret
```

```
313 // paret = 0, si es un quadrat verd
314 // ori = orientacio actual
315 // retorna la nova direccio a seguir
316 int mapNewCell(int paret, int ori);
317
318 // funcio que es crida quan es choca contra
319 // una pared de la mateixa casella
320 // ori = orientacio actual
321 // retorna la nova direccio a seguir
322 int mapWall(int ori)
323
324 // funcio que es crida quan s'arriba a la sortida del
    mapa
325 // necesita saber si es ve d'una altra casella o s'
    estava
326 // fent un gir.
327 // Ho prepara tot per tornar a l'inici.
328 // newCell = 1, si es venia d'una altra casella
329 // ori = orientacio actual
330 void mapEndCell(int newCell, int ori);
331
332
333 **/
334
335 /**
336 * orient : Variable l'ordre esta forat
337 * 0 = NORD          (el robot est aturat)
338 * 1 = OEST           (el robot va totalment recte, si
    trova una parat si alinia)
339 * 2 = SUD         el robot es trova en una crulla)
340 * 3 = EST         (el robot ha trobat una paret)
341 */
342 // FIXME : posar a 0
343 int orient = 0;
344 #define NORD  0
345 #define OEST  1
346 #define SUD   2
347 #define EST     3
348
349 #define TANCAT    -1
350 #define OBERT   1
351 #define DESCONEGUT  0
352
353 #define MAX_CELLS 25
354
355 //int tmp[];
356 //int map[MAX_CELLS][4];
357
358 int mapN[MAX_CELLS];
359 int mapE[MAX_CELLS];
```

```
360 int mapS[MAX_CELLS];
361 int mapO[MAX_CELLS];
362
363 // FIXME: es un header
364 int mapGoToEnd(int ori);
365
366 int mapGetValue(int x, int y)
367 {
368   switch (y)
369   {
370     case 0:
371       return mapN[x];
372       break;
373     case 1:
374       return mapO[x];
375       break;
376     case 2:
377       return mapS[x];
378       break;
379     case 3:
380       return mapE[x];
381       break;
382     default:
383       throwError(233);
384       break;
385   }
386 }
387
388 int mapPutValue(int x, int y, int val)
389 {
390   if (x < 0)
391   {
392     throwError(ERR_X);
393   }
394   else
395   {
396     switch (y)
397     {
398       case 0:
399         mapN[x] = val;
400         break;
401       case 1:
402         mapO[x] = val;
403         break;
404       case 2:
405         mapS[x] = val;
406         break;
407       case 3:
408         mapE[x] = val;
409         break;
```

```
410       default:
411         throwError(ERR_Y);
412         break;
413     }
414   }
415 }
416
417 int mapPos = 0;
418 int mapFinalPos = -1;
419
420 /*
421 * Funcio privada que inicialitza una casella
422 * i la posa tota inexplorada.
423 */
424 void mapClearCell(int pos)
425 {
426   //FIXME : = 0
427   for (int i = 0; i < 4; i++)
428   {
429     mapPutValue(pos,i,DESCONEGUT);
430
431
432     //tmp = map[pos];
433     //tmp[i] = -1;
434     //map[pos] = tmp;
435
436     // FIXME:
437     //map[pos][i] = -1;
438   }
439 }
440
441 /*
442 *  Funcio que es crida un cop s'ha arribat
443 *  al final del mapa, i que fixa la ruta
444 *  que ha seguit fins aquest punt.
445 */
446 void mapFixEnd()
447 {
448   mapFinalPos = mapPos;
449
450   for (int i = 0; i < mapPos; i++)
451   {
452     for (int j = 0; j < 4; j++)
453     {
454       // si no s'havia explorat es tanca
455       if (mapGetValue(i,j) == DESCONEGUT) {mapPutValue
              (i,j,TANCAT);}
456     }
457   }
458
```

```
459   mapPutValue(1,SUD,OBERT);
460   orient = NORD;
461   mapPos = 2;
462 }
463
464 /*
465 * Funcio per inicialitzar el mapa
466 */
467 void mapInit()
468 {
469   mapPos = 2;
470
471   // es marquen totes les caselles com
472   // desconegudes (un zero en totes direccions)
473   //ArrayInit(cell, 0, 4);
474   //ArrayInit(map, cell, MAX_CELLS);
475
476   for (int i = 0; i < MAX_CELLS; i++)
477   {
478     mapClearCell(i);
479   }
480 }
481
482 void mapInit2()
483 {
484   mapPos = 2;
485
486   // es marquen totes les caselles com
487   // desconegudes (un zero en totes direccions)
488   //ArrayInit(cell, 0, 4);
489   //ArrayInit(map, cell, MAX_CELLS);
490
491   for (int i = 0; i < MAX_CELLS; i++)
492   {
493     mapClearCell(i);
494   }
495
496   mapPutValue(0,NORD,OBERT);
497   mapPutValue(0,OEST, TANCAT);
498   mapPutValue(0,SUD,OBERT);
499   mapPutValue(0,EST,TANCAT);
500
501   mapPutValue(1,NORD,TANCAT);
502   mapPutValue(1,OEST, TANCAT);
503   mapPutValue(1,SUD,OBERT);
504   mapPutValue(1,EST,OBERT);
505
506
507   mapPutValue(2,NORD,TANCAT);
508   mapPutValue(2,OEST, OBERT);
```

```
509   mapPutValue(2,SUD,TANCAT);
510   mapPutValue(2,EST,OBERT);
511
512   mapPutValue(3,NORD,OBERT);
513   mapPutValue(3,OEST, OBERT);
514   mapPutValue(3,SUD,TANCAT);
515   mapPutValue(3,EST,TANCAT);
516
517   mapPutValue(4,NORD,TANCAT);
518   mapPutValue(4,OEST, TANCAT);
519   mapPutValue(4,SUD,OBERT);
520   mapPutValue(4,EST,OBERT);
521
522   mapPutValue(5,NORD,OBERT);
523   mapPutValue(5,OEST, OBERT);
524   mapPutValue(5,SUD,TANCAT);
525   mapPutValue(5,EST,TANCAT);
526
527   mapFixEnd();
528 }
529
530 /*
531 * Funcio privada que indica la
532 * seguent direccio a provar
533 */
534 int mapExploreDirection()
535 {
536   int i;
537   // Si hi ha alguna posici  per explorar la retorna
538   for (i = 0; i < 4; i++)
539   {
540     if (mapGetValue(mapPos,i) == DESCONEGUT)
541     {
542       return i;
543     }
544   }
545
546   //debug(POU);
547
548   // Si hi ha alguna posici  per tornar a l'inici la
        retorna
549   // i marca el cami com a pou
550   if (mapPos > 1)
551   {
552     for (i = 0; i < 4; i++)
553     {
554       if (mapGetValue(mapPos,i) == OBERT)
555       {
556         mapPutValue(mapPos-1,(i+2)%4,TANCAT);
557
```

```
558         mapClearCell(mapPos);
559
560         mapPos = mapPos -2;
561         return i;
562       }
563     }
564   }
565
566   return -1;
567 }
568
569 /*
570 * Funcio que es crida quan es toca una paret
571 * d'una casella nova, o es toca un quadrat verd
572 *
573 * paret = 1 si es una paret,
574 *       = 0 si es quadrat verd
575 * ori = orientacio del robot
576 */
577 int mapNewCell(int paret, int ori)
578 {
579   // coordenada d'on vens
580   int nori = (ori+2)%4;
581   // casella nova
582   int newMapPos = mapPos+1;
583
584   //mapClearCell(newMapPos);
585
586   // si es una paret marca aquesta orientacio
587   // com impossible
588   if (paret == 1)
589   {
590     mapPutValue(newMapPos,ori, TANCAT);
591   }
592
593   // Si la direccio d'on vinc es zero
594   // vol dir que aquella direccio no
595   // es un pou
596   if (mapGetValue(newMapPos,nori) == DESCONEGUT)
597   {
598     // la marquem com provinencia
599     mapPutValue(newMapPos,nori,OBERT);
600     // marquem tambe la casella
601     // anterior com a provinent
602     mapPutValue(mapPos,ori,OBERT);
603   }
604
605   mapPos = newMapPos;
606
607   // retornem la nova direccio per explorar
```

```
608   // excloent de la que venim
609   return mapExploreDirection();
610 }
611
612 /*
613 * Funcio que es crida quan es choca contra
614 *   una paret de la mateixa casella.
615 */
616 int mapWall(int ori)
617 {
618   mapPutValue(mapPos,ori,TANCAT);
619   return mapExploreDirection();
620 }
621
622
623
624 /*
625 * Funcio que es crida quan s'arriba a la
626 * Casella final, se li ha de dir si
627 * es ve d'una altra casella o s'ha trobat girant.
628 * newCell = 1 si es ve de "lluny"
629       = 0 si es la mateixa casella
630 */
631 int mapEndCell(int newCell, int ori)
632 {
633
634   if (newCell == 0)
635   {
636     mapPutValue(mapPos,ori,OBERT);
637   }
638   else
639   {
640     // coordenada d'on vens
641     int nori = (ori+2)%4;
642     // casella nova
643     int newMapPos = mapPos+1;
644
645     mapPutValue(newMapPos,ori,OBERT);
646
647
648     // Si la direccio d'on vinc es zero
649     // vol dir que aquella direccio no
650     // es un pou
651     if (mapGetValue(newMapPos,nori) == DESCONEGUT)
652       // la marquem com provinencia
653       mapPutValue(newMapPos,nori,OBERT);
654       // marquem tambe la casella
655       // anterior com a provinent
656       mapPutValue(mapPos,ori,OBERT);
657
```

```
658     mapPos = newMapPos;
659   }
660
661   mapFixEnd();
662   //victori();
663
664   return mapGoToEnd(NORD);
665 }
666
667
668 /**
669  /'\_/`\
670 /\       \      __     _____          ____     __
671 \ \ \__\ \   /'__`\  /\ '__`\       /',__\  /'__`\
672  \ \ \_/\ \ /\ \L\.\_\ \ \L\ \     /\__, `\/\ \L\.\_\
673   \ \_\\ \_\\ \__/.\_\\ \ ,__/     \/\____/\ \__/.\_\
674    \/_/ \/_/ \/__/\/_/ \ \ \/       \/___/  \/__/\/_/
675                         \ \_\
676                          \/_/
677 **/
678
679 /*
680   Resum de les funcions, un cop creat el mapa,
681
682 // Indica la orientacio  a seguir per arribar al
    principi
683 // soposant que es troba en la casella final i s'ha
    posat
684 // mapPos = mapFinalPos, o s'acaba d'arribar al final.
685 // S'ha de cridar consecutivament cada cop que s'
    arriva
686 // a una paret o a una interseccio
687 int mapGoToStart(int ori);
688
689 // El mateix que la anterior pero soposant que s'ha
    cridat
690 // per primer cop desde la casella inicial.
691 // mapPos = 0
692 int mapGoToEnd(int ori);
693
694 */
695
696 /*
697 * Funcio privada per escollir la direccion
698 * No s'ha d'utilitzar mai desde fora.
699 */
700 int mapSelectDirection(int nori)
701 {
702   int i;
703   // Mirem quin es la direccio
```

```
704   // que no es la de provinencia
705   for (i = 0; i < 4; i++)
706   {
707     if (i != nori && mapGetValue(mapPos,i) == OBERT)
708       return i;
709   }
710 
711   return -1;
712 }
713 
714 
715 /*
716 * Aquesta funcio porta el robot cap al principi
717 * sempre i cuan s'hagi comenat a cridar
718 * desde el primer cop amb la variable global
719 * mapPos = mapFinalPos;
720 */
721 int mapGoToStart(int ori)
722 {
723   int dir = mapSelectDirection(ori+2%4);
724   mapPos = mapPos - 1;
725   return dir;
726 }
727 
728 /*
729 * Aquesta funcio porta el robot cap al final
730 * sempre i cuan s'hagi comenat a cridar
731 * desde el primer cop amb la variable global
732 * mapPos = 0;
733 */
734 int mapGoToEnd(int ori)
735 {
736   int dir = mapSelectDirection(ori+2%4);
737 
738   return dir;
739 }
740 
741 
742 /*
743 * Funcio que tradueix la orientacio actual i la
744 * orientacio a la que es vol estar als
745 * graus d'un gir per el costat mes curt
746 */
747 int mapOrToDegrees(int oriStart, int oriEnd)
748 {
749   int degrees = (oriEnd - oriStart);
750   if (degrees == -1)
751   {
752     degrees = 3;
753   }
```

```
  else if (degrees == -3)
  {
    degrees = 1;
  }
  else if (degrees == -2)
  {
    degrees = 2;
  }

  return degrees*90;
}


/**
 /'\_/`\
/\       \      __      _____          __
\ \ \__\ \   /'__`\  /\ '__`\       /'__`\
 \ \ \_/\ \ /\ \L\.\_\ \ \L\ \     /\  __/
  \ \_\\ \_\\ \__/.\_\\ \ ,__/     \ \____\
   \/_/ \/_/ \/__/\/_/ \ \ \/       \/____/
                 \ \_\
                  \/_/
**/

/**
 ____
/\  _`\
\ \,\L\_\      __      ___       ____     __
 \/_\__ \    /'__`\ /' _ `\    /',__\  /'__`\
   /\ \L\ \ /\  __/ /\ \/\ \ /\__, `\/\  __/
   \ `\____\\ \____\\ \_\ \_\\/\____/\ \____\
  \/_____/ \/____/ \/_/\/_/ \/___/  \/____/

**/


int esSortida = 0;

/*
* Reinicia el contador de ticks
*/
void resetTicks()
{
  ticksAcumulats = 0;
  ResetTachoCount(OUT_AC);
}

//This function explores if a green detected in sense
    is a square or the goal Line
// return 0 if is a green square
```

```
803 // return 1 if is the goal
804 int detect_exit()
805 {
806     int count = 0;
807     int newGreen= 0;
808     int greenCount = 25; //Nombre de mostres seguides
        que si s n verdes podem considerar que estem
        en un quadrad
809     int noGreenCount = 25; //Nombre de mostres
        seguides que si no apareix verd podrem pensar
        que som a la sortida
810     OnFwdReg(MOTORS,50,OUT_REGMODE_SYNC);
811     while ((updateActualColor() == GREEN) && newGreen
        < greenCount)
812     {
813     newGreen = newGreen + 1;
814     Wait(ST);
815     }
816     if (newGreen < greenCount)
817     {
818         while ((count < noGreenCount) && (
            updateActualColor()!=GREEN))
819         {
820             count=count+1;
821             Wait(ST);
822         }
823         if (count < noGreenCount)
824         {
825       while(count<noGreenCount)
826       {
827         count=count+1;
828         Wait(ST);
829       }
830       parar();
831       return 0;
832         }
833     parar();
834         return 1;
835     }
836     else
837     {
838     parar();
839         return 0;
840     }
841 }
842
843 /*
844 * Retorna si venim d'una altre casella
845 */
846 int lluny()
```

```
847 {
848   return ticksAcumulats > TICKS_AMPLADA;
849 }
850
851 /*
852 * retorna si s'ha resolt el mapa
853 */
854 int mapaResolt()
855 {
856   return (mapFinalPos != -1);
857 }
858
859 /*
860 * retorna si anem cap a la sortida
861 */
862 int toEnd()
863 {
864   return (toInici != 1);
865 }
866
867 /*
868 * retorna si anem cap a l'entrada
869 */
870 int toStart()
871 {
872   return (toInici == 1);
873 }
874
875 /*
876 * retorna si es una sortida de la mateixa casella
877 */
878 int mateixaSortida()
879 {
880   return ! lluny();
881 }
882
883 /*
884 * retorna si es una sortida d'una nova casella
885 */
886 int novaSortida()
887 {
888   return lluny();
889 }
890
891 /*
892 * retorna si estem en una paret de la mateixa casella
893 */
894 int mateixaParet()
895 {
896   int sensor = updateActualColor();
```

```
897   return ((sensor < GREEN) && ! lluny());
898 }
899
900 /*
901 * retorna si estem en una paret d'una nova casella
902 */
903 int novaParet()
904 {
905   int sensor = updateActualColor();
906   return ((sensor < GREEN) && lluny());
907 }
908
909 /*
910 * retorna si estem en un quadrat verd
911 */
912 int nouVerd()
913 {
914   int sensor = updateActualColor();
915   return (sensor == GREEN);
916 }
917
918
919 /*
920 * Orienta el robot cap a la orientacio
921 * que li diguem
922 */
923 void orienta(int actualOri, int finalOri,int rev)
924 {
925   int sentit = 1;
926   int graus;
927
928   resetTicks();
929
930   /*if (actualOri == finalOri)
931   {
932     throwError(ORISAME);
933   }*/
934
935   if (finalOri == -1)
936   {
937     throwError(ORINEG);
938   }
939
940   graus = mapOrToDegrees(actualOri, finalOri);
941
942   if (graus == 270)
943   {
944     graus = 90;
945     sentit = -1;
946   }
```

```
947
948   gir(graus,sentit,rev);
949   if (updateActualColor()==GREEN)
950   {
951     OnFwdReg(MOTORS,50,OUT_REGMODE_SYNC);
952     while(updateActualColor()==GREEN)
953     {
954       Wait(ST);
955     }
956     parar();
957   }
958
959
960   orient = finalOri;
961 }
962
963 /*
964 * Funcio que escull quina accio executar
965 * segons la posicio en la que es troba
966 */
967 void sense()
968 {
969   int finalOri;
970   int graus;
971   int actualOri = orient;
972   PlayToneEx(880,500,VOLUM,LOOP);
973
974   if (!mapaResolt())
975   {
976
977     if (novaParet())
978     {
979       state = NEWWALL;
980       finalOri = mapNewCell(1, actualOri);
981       orienta(actualOri, finalOri,1);
982     }
983     else if (mateixaParet())
984     {
985       state = WALL;
986       finalOri = mapWall(actualOri);
987       orienta(actualOri, finalOri,1);
988     }
989     else if (nouVerd())
990     {
991
992       if (detect_exit() == 0)
993       {
994         state = CRUILLA;
995         finalOri = mapNewCell(0, actualOri);
996         orienta(actualOri, finalOri,0);
```

```
997         }
998         else if (novaSortida())
999         {
1000          state = NEWEXIT;
1001          finalOri = mapEndCell(1, actualOri);
1002          parar();
1003          while(!SB){}
1004          orienta(NORD,finalOri,0);
1005          mapPos = mapPos + 1;
1006        }
1007        else if (mateixaSortida())
1008        {
1009          state = EXIT;
1010          finalOri = mapEndCell(0, actualOri);
1011          parar();
1012          while(!SB){}
1013          orienta(NORD,finalOri,0);
1014          mapPos = mapPos + 1;
1015        }
1016        else
1017        {
1018          throwError(UNKGREEN);
1019        }
1020      }
1021      else
1022      {
1023        throwError(NOOP);
1024      }
1025    }
1026    else
1027    {
1028      if (toStart())
1029      {
1030        state = TOSTART;
1031        finalOri = mapGoToStart(actualOri);
1032        orienta(actualOri, finalOri,1);
1033      }
1034      else if (toEnd())
1035      {
1036        state = TOEND;
1037        finalOri = mapGoToEnd(actualOri);
1038        if (updateActualColor() == GREEN)
1039        {
1040          detect_exit();
1041          if (mapPos == mapFinalPos)
1042          {
1043            victory();
1044            parar();
1045            while(!SB){}
1046            mapPos = 2;
```

```
1047            orienta(NORD,finalOri,0);
1048            mapPos = mapPos + 1;
1049          }
1050          else
1051          {
1052            orienta(actualOri, finalOri,0);
1053          }
1054
1055        }
1056        else
1057        {
1058          orienta(actualOri, finalOri,1);
1059        }
1060        mapPos = mapPos + 1;
1061      }
1062      else
1063      {
1064        throwError(STARTEND);
1065      }
1066    }
1067
1068    state = RUN;
1069  }
1070  /**
1071   ____
1072  /\  _`\
1073  \ \,\L\_\      __      ___       ____     __           __
1074   \/_\__ \    /'__`\ /' _ `\    /',__\  /'__`\       /'__
        `\
1075     /\ \L\ \ /\  __/ /\ \/\ \ /\__, `\/\  __/      /\
          __/
1076     \ `\____\\ \____\\ \_\ \_\\/\____/\ \____\     \ \
          ____\
1077    \/_____/ \/____/ \/_/\/_/ \/___/  \/____/      \/____
          /
1078
1079  **/
1080
1081
1082  /**
1083  * TASKS
1084  **/
1085
1086  /*Es dedica a treure per pantalla l'estat de les
        variables del robot i altres dades*/
1087  /*d'utilitat per entendre que esta fent el robot. No
        se si serviar exactament com estat*/
1088  /*ara ...*/
1089  task paint()
1090  {
```

```
int it = 0;

while (TRUE)
{
 ClearScreen();

 NumOut(80,LCD_LINE1,LS);
 NumOut(10,LCD_LINE3,error);
 NumOut(10,LCD_LINE1,RS);
 NumOut(80,LCD_LINE3,actualColor);

  switch(orient)
 {
  case NORD:
       TextOut (40, LCD_LINE1, "NORD");
       break;
  case EST:
       TextOut (40, LCD_LINE1, "EST");
       break;
  case SUD:
       TextOut (40, LCD_LINE1, "SUD");
       break;
  case OEST:
       TextOut (40, LCD_LINE1, "OEST");
       break;
  default:
       TextOut (40, LCD_LINE1, "-----");
       break;
 }

 if (mapPos >= 0)
 {
  for (int i = 0; i < 4; i++)
  {
    switch (mapGetValue(mapPos,i))
    {
      case -1:
        TextOut((40+i*5), LCD_LINE2, "#");
        break;
      case 0:
        TextOut((40+i*5), LCD_LINE2, "?");
        break;
      case 1:
        TextOut((40+i*5), LCD_LINE2, "_");
        break;
      default:
        TextOut((40+i*5), LCD_LINE2, "-");
        break;
    }
  }
```

```
1141    }
1142    else
1143    {
1144      TextOut((40), LCD_LINE2, "eeee");
1145    }
1146
1147      switch(state)
1148     {
1149      case STOP:
1150           TextOut (40, LCD_LINE4, "STOP");
1151           break;
1152      case RUN:
1153           TextOut (40, LCD_LINE4, "RUN");
1154           break;
1155      case WALL:
1156           TextOut (40, LCD_LINE4, "WALL");
1157           break;
1158      case NEWWALL:
1159           TextOut (40, LCD_LINE4, "NEWWALL");
1160           break;
1161      case EXIT:
1162           TextOut (35, LCD_LINE4, "EXIT");
1163           break;
1164      case NEWEXIT:
1165           TextOut (40, LCD_LINE4, "NEWEXIT");
1166           break;
1167      case TOSTART:
1168           TextOut (40, LCD_LINE4, "TOSTART");
1169           break;
1170      case TOEND:
1171           TextOut (40, LCD_LINE4, "TOEND");
1172           break;
1173      case ALINIAR:
1174           TextOut (40, LCD_LINE4, "ALINIAR");
1175           break;
1176      case SORTIDA:
1177           TextOut (40, LCD_LINE4, "SORTIDA");
1178           break;
1179    case ORINEG:
1180           TextOut (40, LCD_LINE4, "ORINEG");
1181           break;
1182    case ORISAME:
1183           TextOut (40, LCD_LINE4, "ORISAME");
1184           break;
1185    case CRUILLA:
1186           TextOut (40, LCD_LINE4, "CRUILLA");
1187           break;
1188    case NOOP:
1189           TextOut (40, LCD_LINE4, "NOOP");
1190           break;
```

```
1191    case STARTEND:
1192           TextOut (40, LCD_LINE4, "STARTEND");
1193           break;
1194    case UNKGREEN:
1195           TextOut (40, LCD_LINE4, "UNKGREEN");
1196           break;
1197    case POU:
1198           TextOut (40, LCD_LINE4, "POU");
1199           break;
1200    case ERR_X:
1201      TextOut (40, LCD_LINE4, "ERR_X");
1202      break;
1203    case ERR_Y:
1204      TextOut (40, LCD_LINE4, "ERR_Y");
1205      break;
1206      case UNKNOW:
1207           TextOut (40, LCD_LINE4, "UNKNOW");
1208           break;
1209      default:
1210           NumOut (50, LCD_LINE4, state);
1211           break;
1212     }
1213     TextOut(34,LCD_LINE5,"Acumulat");
1214     NumOut(40,LCD_LINE6,ticksAcumulats);
1215
1216     NumOut(10,LCD_LINE7,RT);
1217     NumOut(75,LCD_LINE7,LT);
1218
1219     NumOut(40,LCD_LINE8,mapPos);
1220     NumOut(86,LCD_LINE8,it);
1221
1222
1223     it++;
1224     Wait(1000);
1225    }
1226 }
1227
1228 /*Menetre no hagi trobat la sortida el robot es
       dedicara a explorar. Aix */
1229 /*consisteix en correr pel laberint fins que trobe
       parets frontals o quadres verds*/
1230 /*que indiquen encreuament de camins. Llavors fa un
       sense per veure si es una paret*/
1231 /*un quadre verd o la sortida i actua en
       conseq  ncia.*/
1232 task main()
1233 {
1234   mapInit();
1235   //mapInit2();
1236     state = STOP;
```

```
1237      SetSensorLight(IN_4);
1238      SetSensorLight(IN_1);
1239      SetSensorTouch(IN_2);
1240      StartTask(paint);
1241      while (!SB)
1242      {}
1243      mazesolved = 0;
1244      while(true)
1245      {
1246          state = RUN;
1247          run();
1248
1249      if (updateActualColor() < WHITE -7)
1250      {
1251        sense();
1252      }
1253
1254      }
1255  }
```

Capítol 5

Conclusions

Durant la realització de la practica ens hem adonat de diversos detalls que fan que el problam sigui més complicat del que és. Un d'ells és el format en que esta marcats els encreuaments: Un requadre verd ambl'interior blanc. Pel que a nosaltres afecta. El fet que el quadre fos buit de dins ha fet que que de forma molt aleatoria hi hagi vegades que el robot no respongui be. El motiu es que per saber que es tracta d'un encreuament i no una sortida el robot un cop ha detectat ver (o gris) ha d'avançar fins que sigui blanc i despres moures una certa distancia, contabilitzada amb mostres temporals a la nostra implmetació, fins que es trobi un altre verd. Aquest fet, que tots hem d'implementar d'alguna forma seria molt més facil i faria més consistent el robots si fos un quadre completamente ple. A més a més això permetria no haver de pedre temps en un fet com aquest i concentrar-se més en un bon algoris-me de control per al robot.

Una altra dificultat ha estat les diverses particularitats del llenguatge que s'utilitza. Que a més també porten de corcoll a tots. No seria una mala idea proposar que entre alumnes i professors és generes un recull de preguntes i respostes freqüents que a cada curs es vaig completant i que permetria no pedre tantes hores tractant de saber el que espot fer i no i d'alguna forma ajudaria a condensar el temps amb el que considerem que era la part importat del robot. Control i navegació, no pas tant com codificar. Nosaltres proposem com a primera pregunta i resposta:

Q1 : Quant es reseteja un tacometre?

R1 : Cada cop que s'atura el motor i cada cop que s'arrenca ja sigui utilitzant onFwd() o onFwdReg().

Q2 : Per que tot i utilit-zar onFwdReg el robot no va recte?

R2 : Vigila de l'ordre no sigui en un loop, ja que si s'executa molt sovint al regulacio no funciona.

Hem aplicat d'alguna forma els algorismes de control que s'han mostrat a teoria i ens han permes entendrels una mica millor. I a pesar de totes les petites dificultats que ens hem anat trovant pensem que hem aconseguit un bon funcionament del robot i que acompleix tots els requisits de la practica. El

següent nivell ja seria fer que el robot explores tots els camins possibles i que tries per sol·lucionar el laberint el més curt.

Bibliografía

[1] D Benedettelli. **Programming lego nxt robots using nxc**. *Available: bricxcc. sourceforge. net/nbc/nxcdoc/*, 2007.

[2] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Jean-christophe Zufferey, Dario Floreano, and Alcherio Martinoli. **The e-puck, a Robot Designed for Education in Engineering**. *Robotics*, 2006.

[3] John Hansen. *Not eXactly C (NXC)*. 2007.