UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TREBALL FI DE GRAU

Grau en Enginyeria Electrònica Industrial i Automàtica

DISSENY I IMPLEMENTACIÓ D'UN ROBOT MÒBIL CAPAÇ DE RESOLDRE LABERINTS.

Memòria – Annex

Autor: Marta Real Vial
Director: Sebastián Tornil Sin
Departament ESAII

Convocatòria: Gener 2019

ÍNDEX

Agraïments

Vull agrair a la meva família i amics l'ajuda i el suport en aquest projecte i durant tots els anys de carrera.

També vull donar les gràcies al meu tutor, Sebastián Tornil, pels consells que m'ha donat i l'ajuda que m'ha oferit.

"Knowing the mouse might one day find a way to heat up the spaghetti... It fills you with determination." - Undertale, 2015

Resum

Cada any se celebra a diferents regions del món la competició Micromouse, on múltiples equips dissenyen i implementen petits robots mòbils i autònoms que han d'aconseguir resoldre un laberint el més ràpid possible. En aquest projecte s'ha dissenyat i construït un robot, al que s'ha anomenat CEBA, i s'han emprat com a base les normes de la Micromouse perquè també pugui resoldre laberints similars als de la competició. Aquesta memòria reflecteix el procés de creació de la CEBA, tant el disseny i selecció d'elements del robot, com per exemple els sensors, els motors o l'estructura feta a partir d'impressió 3D; així com la construcció d'aquest.

Mitjançant una Raspberry Pi i el llenguatge de programació Python, s'ha implementat un codi capaç de recórrer i resoldre els laberints a la vegada que mostra per la pantalla de l'ordinador el mapa del camí explorat. El codi es divideix en dues fases: la d'exploració i la de resolució. En la primera es fa un recorregut per prendre dades de la distribució del laberint per aplicar la solució en la segona fase. Per altra banda, també s'han construït les peces necessàries per muntar els diferents laberints on s'han realitzat les proves. Aquestes proves han servit per comprovar el funcionament correcte del robot i aplicar millores tant en el robot com en el codi.

Resumen

Cada año se celebra en diferentes regiones del mundo la competición Micromouse, donde múltiples equipos diseñan e implementan pequeños robots móviles y autónomos que tienen que conseguir resolver un laberinto lo más rápido posible. En este proyecto se ha diseñado y construido un robot, al que se ha llamado CEBA, y se han usado como base las normas de la Micromouse para que también pueda resolver laberintos similares a los de la competición. Esta memoria refleja el proceso de creación de CEBA, tanto el diseño y selección de los elementos del robot, como por ejemplo los sensores, los motores o la estructura hecha a partir de impresión 3D; así como la construcción de éste.

Mediante una Raspberry Pi y el lenguaje de programación Python, se ha implementado un código capaz de recorrer y resolver los laberintos a la vez que muestra por la pantalla del ordenador el mapa del camino explorado. El código se divide en dos fases: la de exploración y la de resolución. En la primera se hace un recorrido para tomar datos de la distribución del laberinto para así en la segunda fase aplicar la solución encontrada a partir de estos datos. Por otra parte, también se han construido las piezas necesarias para montar los distintos laberintos donde se han realizado las pruebas. Estas pruebas han servido para comprobar el correcto funcionamiento del robot y aplicar mejoras tanto en el robot como en el código.

Abstract

Every year the Micromouse competition takes place in different regions of the world, where multiple teams design and implement small mobile and autonomous robots that have to solve a maze as quick as possible. In this project a robot has been designed and built, which has been named CEBA, and the rules of the Micromouse have served as a basis to solve mazes similar to those of the competition. This report reflects the creation process of CEBA, both the design and the selection of the robot's elements, like the sensors, the motors or the 3D printed structure as well as the construction of it.

Through a Raspberry Pi and the programming language Python, it has been implemented a code capable of navigate and solve the mazes at the same time that shows a map of the explored path on the computer's screen. The code is divided into two stages, the exploration and the resolution, where in the first stage the robot goes through the maze while collecting data of it's distribution, in order to apply the solution found from the data in the second stage. Moreover, it also has been created the necessary parts to build several mazes to carry out the tests. These tests have been used to check the proper functioning of the robot and to apply improvements both to the robot and to the code.

1 Introducció

Aquest projecte es basa en una aplicació concreta del que és conegut com a SLAM (*Simultaneous Localization And Mapping* o localització i mapejat simultanis), on es realitza l'exploració i el mapejat de l'entorn. En aquest cas, l'exploració de l'entorn té un objectiu final: trobar el camí òptim dins el recorregut realitzat anteriorment dins un laberint. Tenint en compte aquest concepte, el projecte es pot dividir en dues fases: una primera seria en la que es realitza l'SLAM, on el robot fa un recorregut no necessàriament exhaustiu en el laberint una primera vegada fins a trobar la sortida; i una segona fase, on s'analitzen les dades obtingudes i el robot resol el laberint arribant a la sortida pel camí més curt trobat a partir de l'exploració. Alguns factors de l'exploració i el mapejat s'han simplificat el màxim possible per facilitar acomplir l'objectiu final, com per exemple que l'entorn sigui sempre un laberint amb una entrada i sortides fixes, encara que els passadissos dins d'aquest variïn.

Així com hi ha competicions de sumo o de velocitat en l'àmbit de la robòtica, també hi ha competicions en laberints, ja sigui de resolució d'aquests com d'aconseguir extreure un element (com una pilota) de dins del laberint. La categoria on l'objectiu és el de resoldre el més ràpid possible el laberint s'anomena "Micromouse" i es realitzen diferents competicions arreu del món amb normes similars respecte tant al robot que competeix com al laberint. Tant la normativa estàndard com la història de la competició es poden consultar a la pàgina web: *http://www.micromouseonline.com/* [1]

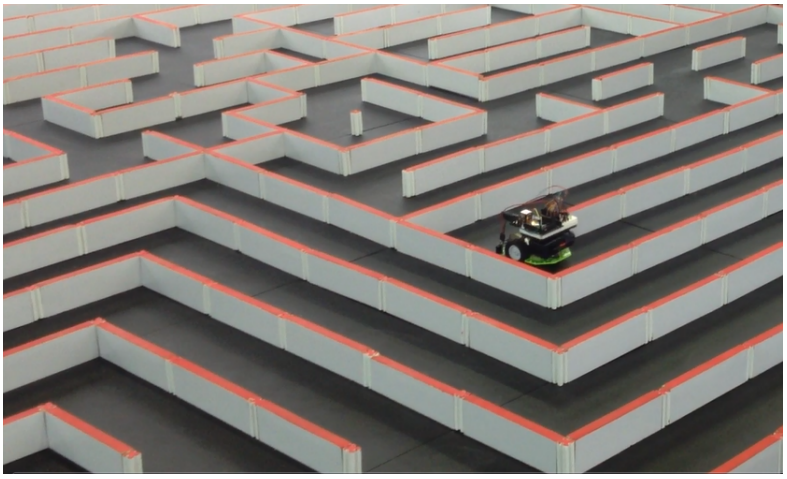

Figure 1: Robot participant en una competició de Micromouse. [2]

1.1 Objectiu del treball

L'objectiu d'aquest projecte és el de dissenyar i implementar un robot mòbil autònom capaç de resoldre laberints el més ràpid possible basant-se en les normes de la competició Micromouse. Per acomplir aquest objectiu s'han de dur a terme una sèrie de tasques:

1. Disseny del robot:
    - Seleccionar la placa i el llenguatge de programació en el qual s'executarà el programa principal.
    - Seleccionar els sensors que obtindran la informació necessària per a la navegació del robot.
    - Triar els motors adients perquè el robot es mogui i dissenyar el sistema de tracció.
    - Dissenyar l'estructura del robot.
    - Triar fonts d'alimentació per tal que el robot sigui autònom.
2. Software implementat:
    - Establir un sistema de comunicació per monitoritzar el robot amb l'ordinador a temps real.
    - Crear i implementar un codi capaç d'emprar les dades rebudes pels sensors per navegar i visualitzar en temps real el recorregut del robot pel laberint.
3. Construcció del laberint:
    - Dissenyar i construir el laberint com a banc de proves per comprovar que tant el robot és capaç de navegar-hi com de resoldre'l.
4. Proves experimentals:
    - Realització de proves del robot dins el laberint per comprovar el funcionament d'aquest.
    - Anàlisi del resultat de les proves.

2 Disseny del robot

A continuació s'explicaran les diferents parts que conformen el robot i el procés que s'ha seguit per seleccionar-les o dissenyar-les.

2.1 Selecció de la placa i del llenguatge de programació

La placa emprada pel control del robot és una Raspberry Pi 3 model B (figura 2), un ordenador d'una sola placa que s'utilitza habitualment per aplicacions d'esbarjo o per projectes de petita escala. La selecció d'aquest tipus de placa també ve donada pel seu preu reduït (varia entre 30€i 35€) i la seva facilitat d'utilització. A la taula 1 es presenten les característiques bàsiques d'aquest model:

Especificacions	Raspberry Pi 3B
CPU	64bit-Quad Core 1.2GHz Broadcom BCM2837
Memòria	1 GB
Consum energètic	800 mA
Nombre de pins	40 GPIO pins
Connectivitat de xarxa	Wi-fi, Bluetooth, Ethernet
Nombre de ports USB 2.0	4
Emmagatzematge intern	MicroSD
Alimentació	5 V per MicroUSB

Table 1: Característiques Raspberry Pi 3B. [3]

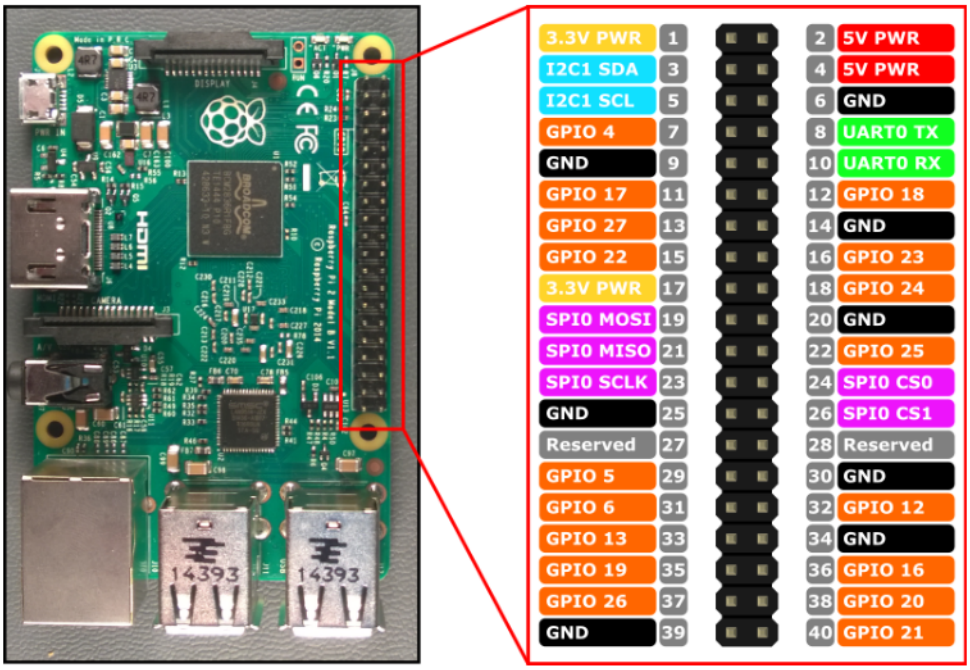

Figure 2: Pin-out GPIO Raspberry Pi.[4]

Per altra banda, el llenguatge de programació amb el que es farà el codi de control i visualització serà Python, perquè és un llenguatge amb el qual ja estic familiaritzada i que dins la Raspberry ja estan instal·lats diversos programes d'escriptura i execució de Python.

Python és un llenguatge de programació d'alt nivell, de codi obert i visualment de fàcil lectura amb una comunitat bastant activa que fa més fàcil trobar ajuda o exemples d'aplicacions ja creades.

2.2 Selecció de sensors

Per tal que el robot pugui detectar si té parets del laberint properes tant al davant com a ambdós costats i a quina distància es troben, haurà d'anar dotat de sensors tant a la part frontal com a cada costat.

Els sensors seleccionats són HC-SR04 com el que apareix en la imatge 3. El seu funcionament es basa en ultrasons, amb els quals detecta si hi ha un objecte proper i a quina distància està mitjançant un pols d'ultrasons que emet quan s'activa amb el senyal trigger. La distància a la qual es troba l'objecte detectat es calcula amb el temps que tarda en rebre l'eco del senyal que s'ha enviat. A la figura 4 s'esquematitza el funcionament dels HC-SR04. El càlcul es realitza amb la següent funció:

$$Distancia = Velocitat * \frac{Temps}{2}$$

$$Distancia = 34300 * \frac{Temps}{2}$$

$$Distancia = Temps * 17150$$

On la velocitat es considera 34300 cm/s pel fet que es pren la velocitat del so per aire com a constant en aquest cas. El temps es divideix entre 2, ja que és el temps d'anada i tornada entre el sensor i l'objecte que detecta.

Les característiques que s'han tingut en compte a l'hora de seleccionar aquest sensor han estat les següents [6]:

- Voltatge d'alimentació: +5 VDC.
- Rang de mesura: 2 cm a 400 cm.
- Resolució: 0.3 cm.
- Freqüència d'ultrasò: 40 KHz.

Figure 3: Sensor d'ultrasons HC-SR04.

A diferència d'altres models, l'HC-SR04 és el que compta amb un rang més adient a la funció que desenvoluparà dins el laberint. A més, és compatible amb els pins GPIO de la Raspberry, cosa que en facilita la utilització.

Inicialment s'havia plantejat d'utilitzar una càmera i analitzar les imatges per tal fer el recorregut, però es va desestimar la idea degut a la dificultat extra que suposaria el processament d'imatges i pel fet que totes les imatges serien molt similars, ja que només es captarien imatges de parets del laberint.

Una alternativa al sensor d'ultrasons són els sensors d'infrarojos. Aquests utilitzen un làser d'infrarojos per detectar la distància a la qual es troben els objectes que tenen al davant. Els problemes que presenta aquest tipus de sensors és el fet que la seva distància de detecció és més petita que la d'un sensor d'ultrasons, la seva precisió és molt baixa, pel

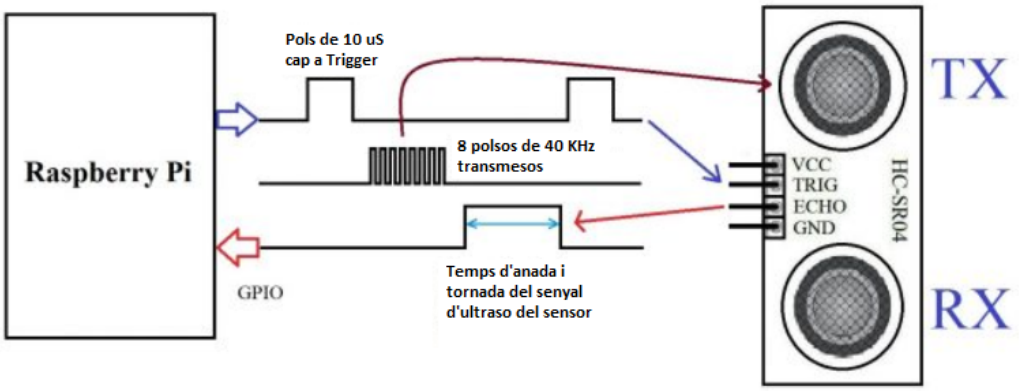

Figure 4: Pin-out i esquema de funcionament del HC-SR04.[5]

que no es pot saber la distància exacta a la qual es troba l'objecte detectat; i depèn de la superfície del material o color de l'objecte on rebota la llum infraroja.

Els tres sensors d'ultrasons aniran col·locats i soldats a una placa de baquelita, el mòdul dels sensors, i connectats a la Raspberry mitjançant la mateixa placa. Aquest mòdul anirà situat al davant de tot del robot, de manera que els sensors puguin captar de forma més precisa la distància frontal del robot respecte la paret més propera. L'esquema de connexió del mòdul dels sensors es pot observar a la figura 7 de l'apartat de disseny electrònic.

2.3 Selecció de motors

Al principi es va plantejar la possibilitat d'utilitzar motors pas a pas, ja que proporcionen un major control sobre el seu moviment , però això a la vegada complica i afegeix molt codi que ralentitza el funcionament del robot. A més tot i que tenen més potència, la seva velocitat és bastant més reduïda. Per tant, finalment s'han emprat 2 motors DC (figura 5) que ja estan dissenyats per petits robots mòbils comercials. Tenen un reductor de velocitat de 48:1 per tal d'incrementar el par proporcionat a la vegada que es disminueix la velocitat. A més, el seu control és més senzill a l'hora de programar perquè només s'ha de donar una instrucció cada vegada que es vulgui fer un canvi de sentit, no constantment perquè les rodes girin com és el cas dels motors pas a pas.

Figure 5: Motors DC amb reductora.

2.4 Disseny electrònic

El control dels motors es fa a través del driver L293D, que permet connectar-hi fins a 4 motors DC que girin en un sol sentit o 2 que girin en ambdós sentits. Aquest a la vegada està connectat a pins GPIO de la Raspberry, que és des d'on s'indica en quin sentit han de girar els motors en cada moment en funció de la informació proporcionada pels sensors d'ultrasons. Per modificar la potència i velocitat es fa a través del voltatge proporcionat en les entrades de PWM/Enable, pins 1 i 9 del xip, que en aquest cas és un voltatge constant de 6 V. En el següent esquema de la figura 6 es mostren les connexions del driver L293D, on Vcc1 correspon als 5 V proporcionats per la Raspberry i Vcc2 al voltatge provinent de la bateria de piles.

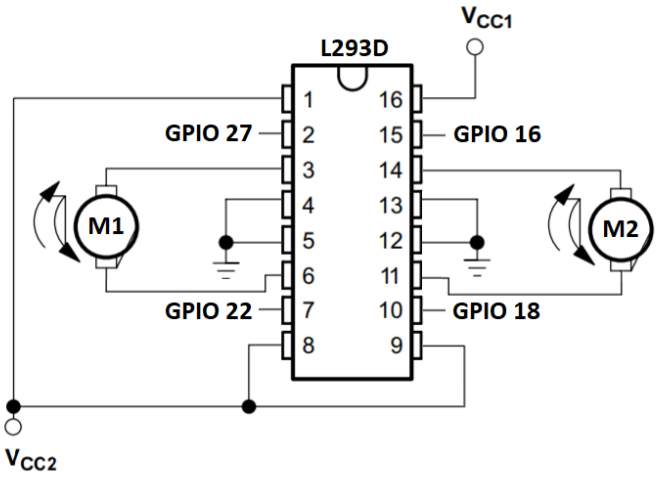

Figure 6: Esquema de connexió del driver L293D. (Font pròpia)

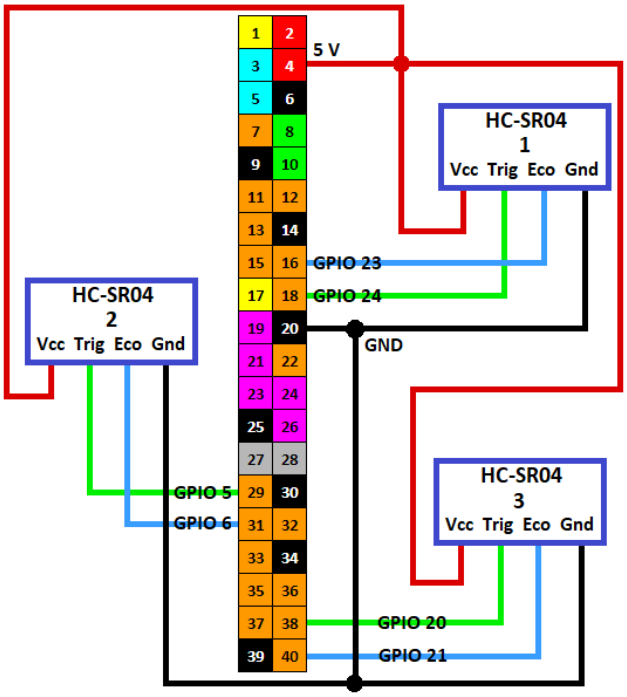

Figure 7: Esquema de connexió dels sensors d'ultrasons a la Raspberry. (Font pròpia)

Per altra banda els 3 sensors d'ultrasons HC-SR04 estan connectats mitjançant el mòdul dels sensors als pins GPIO de la Raspberry com es mostra a l'esquema de la figura 7. En aquest esquema es poden veure els pins de la Raspberry enumerats per ordre juntament amb el mateix codi de colors que presenta la figura 2 a més del nombre GPIO que correspon als pins utilitzats pels sensors. L'alimentació del circuit prové tota del mateix pin degut a la falta de pins que proporcionin 5 V, ja que l'altre, pin 2, està reservat pel driver dels motors. A més, d'aquesta forma es simplifica el cablejat en el mòdul dels sensors. Els sensors de l'esquema estan numerats, essent l'1 el sensor frontal, el 2 el sensor de l'esquerra i el 3 el de la dreta.

2.5 Estructura del Robot

El robot està format per dues plaques dissenyades específicament per aquest projecte i impreses amb impressora 3D, situades una damunt l'altra. Tant en l'espai que queda al mig com a sobre o davall de les plaques hi aniran repartits tots els elements del robot. A sobre de la placa superior hi anirà la Raspberry Pi juntament amb el mòdul dels sensors, com es pot observar a la imatge 8. Sota aquesta mateixa placa, estarà enganxada la bateria que alimenta la Raspberry pi. A sobre de la placa inferior estarà col·locada la bateria de piles que alimenta els motors juntament amb el driver dels motors. Finalment, a sota de la placa inferior es trobaran els motors de les rodes de darrere acoblats a aquestes. La unió entre ambdues plaques es fa mitjançant 4 cargols volumètrics, de tal manera que s'en pot ajustar la distància.

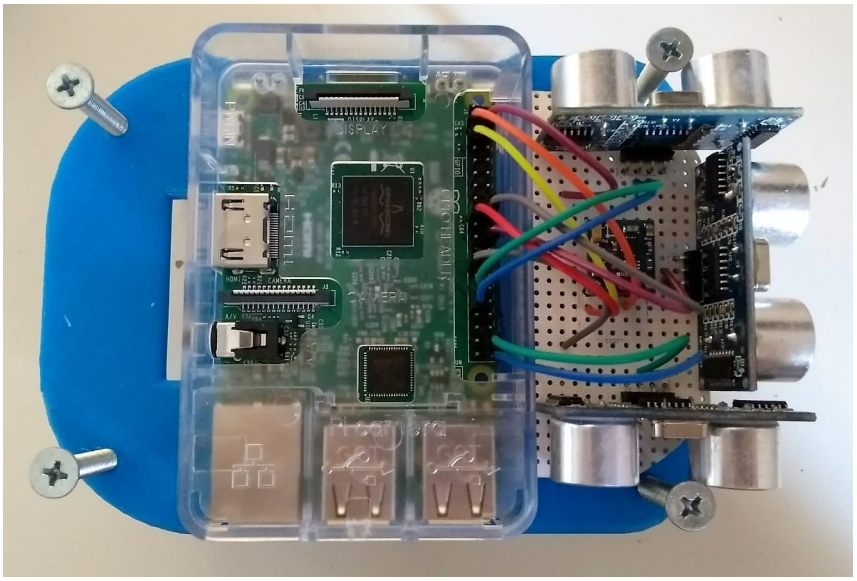

Figure 8: Placa superior vista des d'adalt.

En el disseny de rodes, s'han plantejat tres possibles opcions: quatre rodes, tres rodes o dues rodes.

Amb quatre rodes s'ha de tenir en compte tant la tracció del darrere com el control de direcció de les rodes del davant. Per tant, a més dels motors DC continus, s'ha d'afegir un servo per fer canviar de direcció al robot, fet que implica afegir més elements en el control de moviment del robot. Però per altra banda, aquesta opció és estable i no hi ha problemes d'equilibri en cas que un costat pesi més que l'altre.

Amb dues rodes s'elimina la complicació d'afegir un servo per la direcció, ja que només hi ha les rodes amb motors i aquestes són les que proporcionen el gir. Però d'aquesta forma, s'ha de tenir en compte que la distribució de pes ha de ser equilibrada per evitar que el robot tombi.

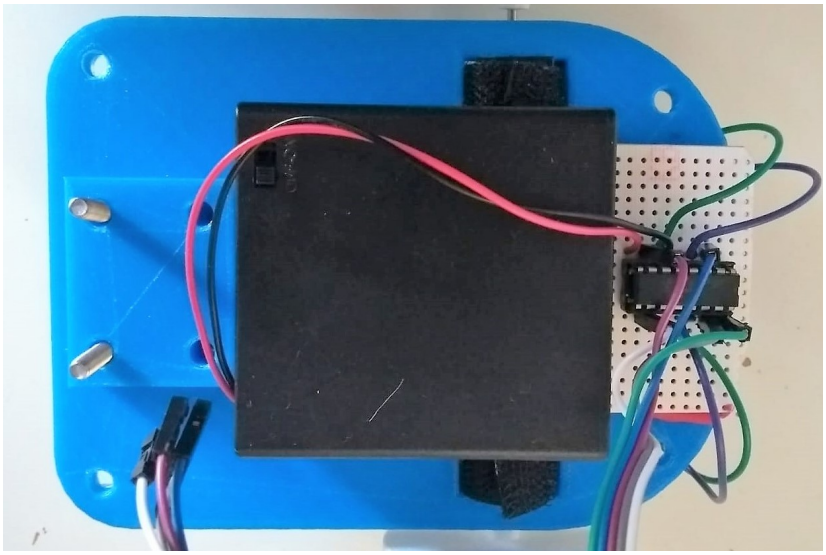

Figure 9: Placa inferior vista des d'adalt.

Finalment, per tal de simplificar el control de moviment i que a la vegada sigui estable, s'ha optat per utilitzar tres rodes. Les dues situades a la part de darrere s'encarreguen de la tracció i dels girs, i una més petita i multi-direccional, figura 10, al davant que serveix per a fer de suport i que el robot no tombi cap endavant. Per realitzar els canvis de direcció, es giraran les dues rodes a la vegada en sentit contrari, de tal manera que girarà sobre ell mateix i s'estalviarà espai a l'hora de moure's pel laberint.

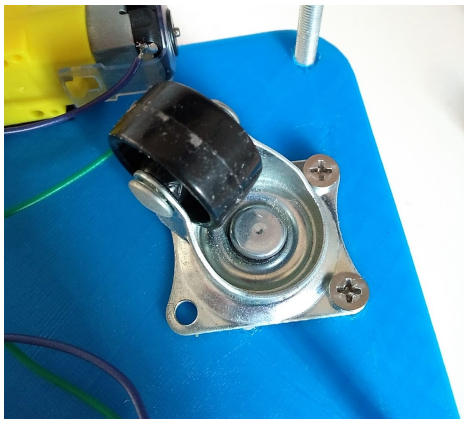

Figure 10: Roda multi-direccional situada al davant del robot.

Les dues rodes encarregades de la tracció estaran impreses en 3D també per poder adaptar-les el millor possible als motors seleccionats, figura 11. Com que el material d'impressió 3D no és adient per fer la tracció, s'han afegit gomes a les rodes com es pot observar a les figures 12 i 13.

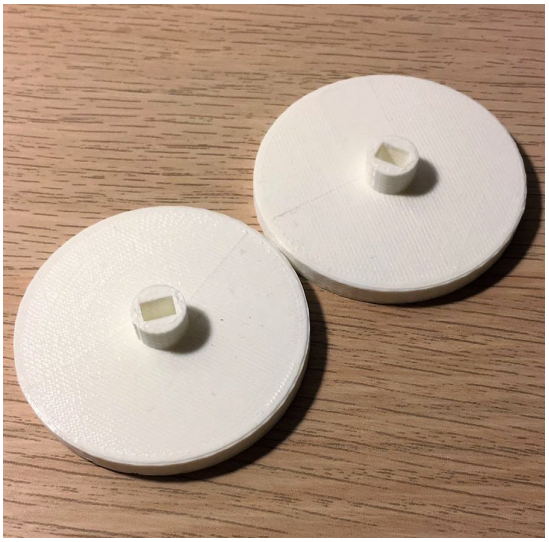

Figure 11: Rodes definitives del robot.

A les imatges 12 i 13 es pot observar el resultat final del robot una vegada acabat de muntar:

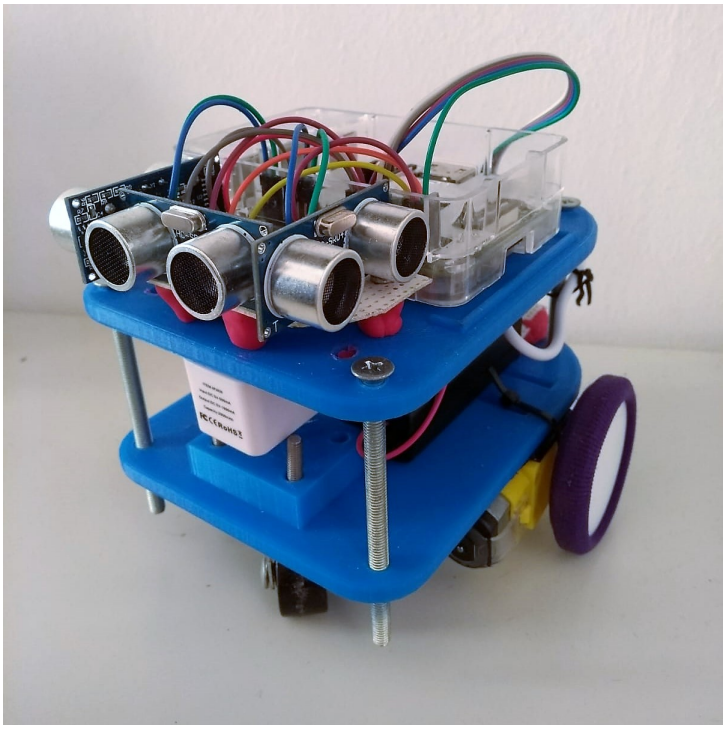

Figure 12: Robot muntat vist de cara.

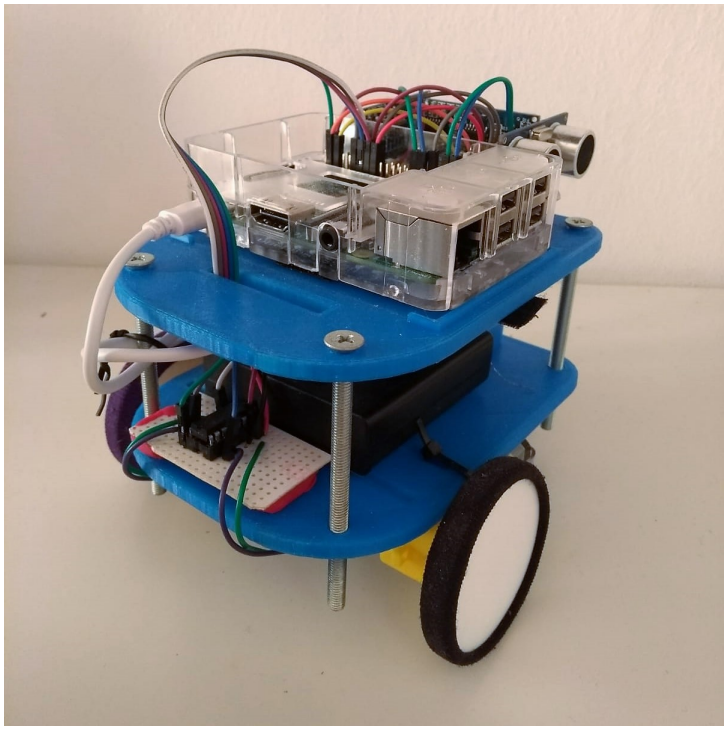

Figure 13: Robot muntat vist des del darrere.

2.6 Alimentació

El robot consta de dues fonts d'alimentació principals, una per alimentar la Raspberry i l'altre per alimentar els motors a través del driver. Tant al mòdul dels sensors com del driver dels motors l'alimentació provendrà directament dels pins de la Raspberry de 5 V, 3.3 V i GND.

Aquesta separació de bateries i per tant la necessitat de tenir-ne dues prové de l'exigència que té el mateix driver dels motors de tenir dues alimentacions separades, com s'ha vist anteriorment a la figura 6. La que prové de la Raspberry proporcionant 5 V s'encarrega d'alimentar la part lògica del circuit de L293D, i la potència dels motors prové d'una bateria formada per un porta-piles amb 4 piles AA de 1.5 V cada una.

L'alimentació de Vcc1 té un rang molt petit de voltatges, que van entre 4,5 i 7 V, en canvi, l'alimentació de Vcc2 pot variar dins un rang molt major, de 4,5 a 36 V. Com que a més voltatge a Vcc2 més potència i velocitat tendran els motors, s'ha optat per proporcionar només 6 V en total, que ja és suficient en termes de velocitat del robot, i així emfatitzar més en la part del control del robot a l'hora que es mogui dins el laberint.

A part d'això, aquesta bateria de piles també proporciona la terra (GND) al driver L293D als pins 4, 5, 12 i 13.

La Raspberry està alimentada amb una bateria portàtil per telèfons mòbils, també anomenades Power Banks, que proporciona 5 V a 1000 mA/h mitjançant l'entrada MicroUSB de la mateixa Raspberry. Com que el consum de la Raspberry és lleugerament superior als mA/hora que proporciona la Power Bank, per assegurar un bon funcionament, cal que aquesta bateria estigui del tot carregada a l'hora de realitzar proves amb el robot. D'aquesta manera, tot i que el corrent proporcionat no dura tant, es pot assegurar un bon rendiment de la Raspberry durant les proves.

Figure 14: Porta-piles.

Figure 15: Power Bank.

3 Software implementat

3.1 Comunicació

La comunicació entre la Raspberry del robot i l'ordinador es realitzarà mitjançant VNC Viewer, un programa que via Wi-fi permet visualitzar dispositius externs i controlar-los remotament, en aquest cas la interfície gràfica de la Raspberry. Per tant la visualització tant del codi com del recorregut que va registrant el robot es veurà directament a la pantalla corresponent a la Raspberry des de la pantalla de l'ordinador, a la vegada que la Raspberry vagi executant el programa de navegació del robot pel laberint.

Els únics inconvenients amb aquest tipus de connexió són que es requereix una connexió estable de Wi-Fi per part d'ambdós aparells i que s'estableix mitjançant la IP de la Raspberry, i si es canvia de xarxa Wi-fi, aquesta IP que té assignada pot canviar, per tant primer s'ha de comprovar la IP actual i després fer la connexió amb l'ordinador.

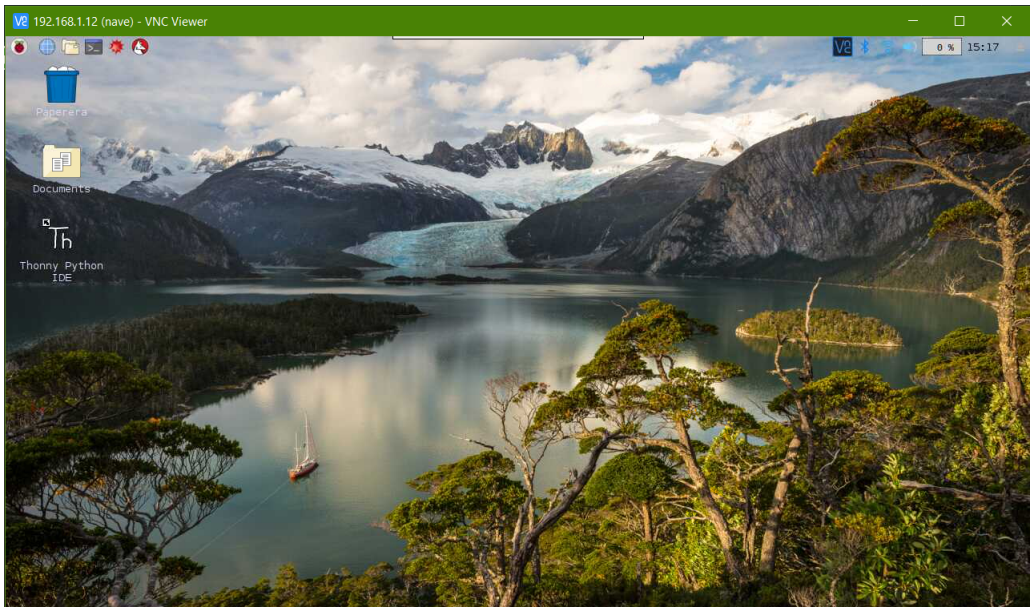

Figure 16: Pantalla de la Raspberry vista des de l'ordinador mitjançant VNC Viewer.

3.2 Codi

En aquest apartat s'explicaran les diferents parts del codi així com els algoritmes que conformen el codi complet mostrant la presa de decisions del robot en cada situació i la resolució del laberint. Com anteriorment s'ha mencionat, el llenguatge de programació emprat és Python i el programa on s'implementa s'anomena Thonny Python IDE, que és una consola de Python com la que es mostra en la imatge 17, que ja ve instal·lada dins la mateixa Raspberry.

3.2.1 Control sensor HC-SR04

El codi pel control dels sensors d'ultrasons es basa en el seu funcionament del càlcul de distància en funció del temps que tarda en rebre el senyal d'eco de tornada. Aquí es mostra l'estructura del codi bàsic, que es va repetint en diverses ocasions en el codi final:

```
def sensor():

    gpio.output(trigger, False)  #Primer s'activa el senyal de trigger durant
    time.sleep(2*10**-6)         # un petit període de temps per tal de
```

```
gpio.output(trigger, True)    # començar a mesurar la distància.
time.sleep(10*10**-6)         #"trigger" correspon al pin GPIO on s'ha
gpio.output(trigger, False)   # connectat el trigger del sensor.

start = time.time()           #Es defineixen dos variables de temps.
end = time.time()

while gpio.input(echo) == 0: #S'estableix el temps d'anada de la senyal echo.
     pass                    #Per evitar que el programa es quedi encallat si el
     i += 1                  # sensor no reb una senyal de tornada es posa un
     if i>=5000:             # límit dins del loop de while.
         break
start = time.time()          #"echo" correspon al pin GPIO on s'ha
                             # connectat l'echo del sensor.

while gpio.input(echo) == 1: #S'estableix el temps de tornada de la
     pass                    # senyal echo.
end = time.time()

t = end-start                #"t" és la diferència entre el temps registrat
                             # d'anada i tornada.
m = t*17150                  #"m" és la mesura final de la distància en cm.
return m                     #Es retorna el valor de "m".
```

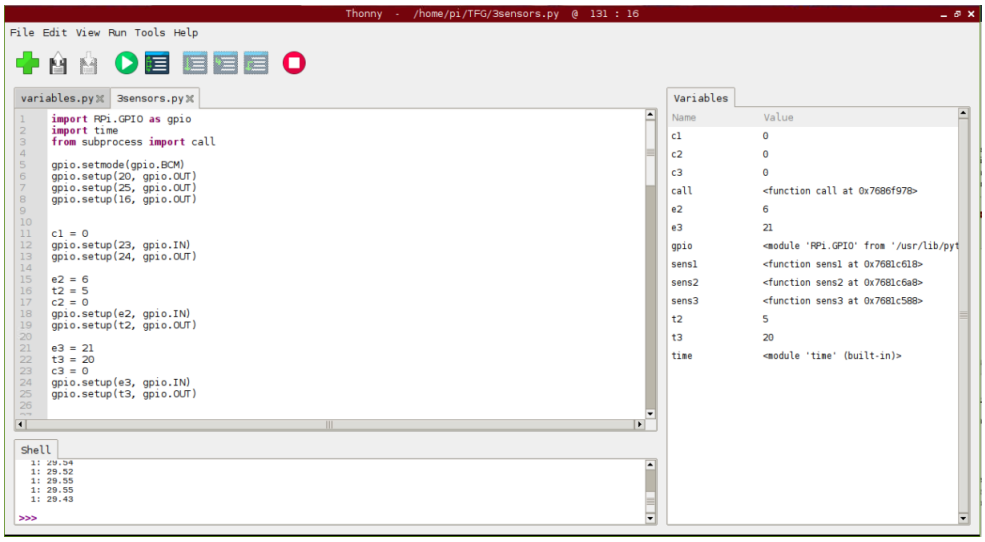

Figure 17: Consola del programa Thonny Python IDE.

3.2.2 Control motors

El control dels dos motors es realitza mitjançant quatre pins GPIO de la Raspberry on en funció de si estan en mode HIGH o LOW es determina en quin sentit gira el motor o si s'ha de parar. En cap cas es poden posar ambdós pins en HIGH, ja que a part de què en el programa no tendria sentit, podria fer malbé els motors. A continuació es mostra un exemple de cada combinació possible en un sol motor:

```
md1 = 27        #md1 és el pin GPIO 27 connectat al pin 2 del driver del motor.
md2 = 22        #md2 és el pin GPIO 22 connectat al pin 7 del driver del motor.

#Configuració per que vagi en el sentit de les agulles de rellotge,
#en aquest cas cap endavant:
```

```
gpio.output(md1, gpio.HIGH)
gpio.output(md2, gpio.LOW)

#Configuració per que vagi en el sentit contrari de les agulles de rellotge,
#en aquest cas cap enrere:
gpio.output(md1, gpio.LOW)
gpio.output(md2, gpio.HIGH)

#Configuració per aturar el motor:
gpio.output(md1, gpio.LOW)
gpio.output(md2, gpio.LOW)
```

3.2.3 Visualització

Per la visualització en directe del camí explorat s'emprarà una eina de Python anomenada "Turtle", que serveix per dibuixar figures simples mitjançant comandaments dins el mateix codi. Aquesta eina s'anomena així degut a la forma de tortuga que té el cursor, que és el que es va movent en funció de les instruccions que apareixen el codi, com per exemple que dibuixi línies rectes, cercles, que canviï de color, o que giri cap a qualsevol direcció. A més, també permet modificar altres coses a part del cursor, com el fons del dibuix o la finestra on es mostra el resultat.

El mapa del laberint s'actualitzarà cada vegada que es topi amb una paret i es realitzi un gir, i es representarà amb línies rectes que seran els passadissos. Ja que el laberint està format per quadrats de la mateixa mesura, els passadissos dibuixats seran proporcionals al nombre de quadrats recorreguts cada vegada. A més, una vegada trobat el camí òptim es podrà veure també per pantalla amb un color diferent a sobre del recorregut dibuixat durant l'exploració.

A continuació es mostra un exemple senzill de codi de Turtle amb el seu resultat gràfic:

```
t = turtle.Turtle()          #Primer s'assigna la variable "t" al cursor
t.color("red")               #Es canvia el color del cursor a vermell
t.ht()                       #S'amaga el cursor agilitzant així el procés de dibuix
t.pensize(3)                 #S'estableix l'ample de les línies

t.fd(100)                    #El dibuixa una línia recta de mida 100
t.right(120)                 #Es gira el cursor cap a la dreta 120º
t.fd(100)
t.right(120)
t.fd(100)
```

Figure 18: Resultat del codi de Turtle anterior.

3.2.4 Algoritme d'exploració

El codi general es pot dividir en dues parts, la d'exploració i la de resolució del laberint. Aquesta primera part s'encarrega de les decisions que pren el robot en tot moment en funció dels inputs que rep pels sensors d'ultrasons. A més, aquí també s'hi inclouen l'emmagatzemament de dades i la visualització a temps real del recorregut que fa.

El laberint està dividit en caselles o quadrats, que conformen una matriu quadrada on cada casella té un nombre assignat:

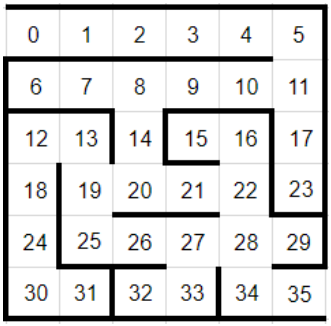

Figure 19: Exemple de laberint de 6x6 amb caselles numerades.

A mesura que el robot es va movent dins del laberint, es van guardant les caselles per les quals ha passat per ordre en què les ha recorregut dins d'un vector. Aquest vector serà el que una vegada acabada la fase d'exploració s'emprarà per trobar la solució al laberint.

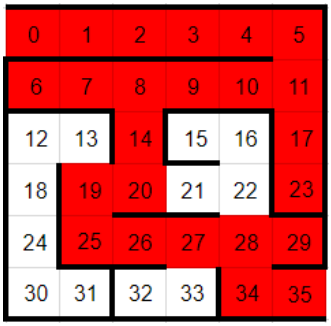

Figure 20: Exemple de laberint de 6x6 amb recorregut fet pel robot.

El vector de sortida en l'exploració de la figura 20 serà:

$$\text{vec} = [0,1,2,3,4,5,11,17,23,17,11,10,9,8,7,6,7,8,14,20,19,25,26,27,28,29,28,34,35]$$

Per altra banda, s'ha afegit un sistema d'orientació per saber en tot moment en quina direcció es troba mirant el robot. Aquest sistema està basat en els punts cardinals, on utilitzant l'exemple de la figura 19, el nord es troba a l'esquerra del laberint, el sud a la dreta, l'est a sobre i l'oest abaix. D'aquesta forma, si es canvien les dimensions del laberint, també s'han de canviar alguns paràmetres dins del codi, ja que quan es dirigeix cap a l'oest o l'est s'han de sumar o restar nombres a mesura que baixa o puja per una columna dins de la matriu. Per exemple, en el cas presentat a la figura 19 es sumen o resten 6 quan es canvia de fila, però si el laberint fos de 10x10 s'haurien de sumar o restar 10 cada vegada.

En el diagrama de flux de les figures 21 i 22 es plasma la presa de decisions del robot en tot moment i les accions que duu a terme en funció d'aquestes decisions. La idea bàsica de l'estratègia d'exploració és que el robot seguirà recte fins a trobar-se amb una paret a menys de 4 cm. Una vegada s'hagi topat amb una paret al davant, s'analitzen les possibilitats de moviment en funció de les parets que detecti a prop de cada costat. En funció del nombre de parets i d'on estiguin situades, el robot girarà cap a un costat o tornarà enrere. Els blocs

A, B i C són simplificacions del diagrama on es representen diverses accions com un sol bloc. Per exemple, el bloc A correspon a les accions d'establir una nova orientació i guardar la distància a la pròxima paret frontal.

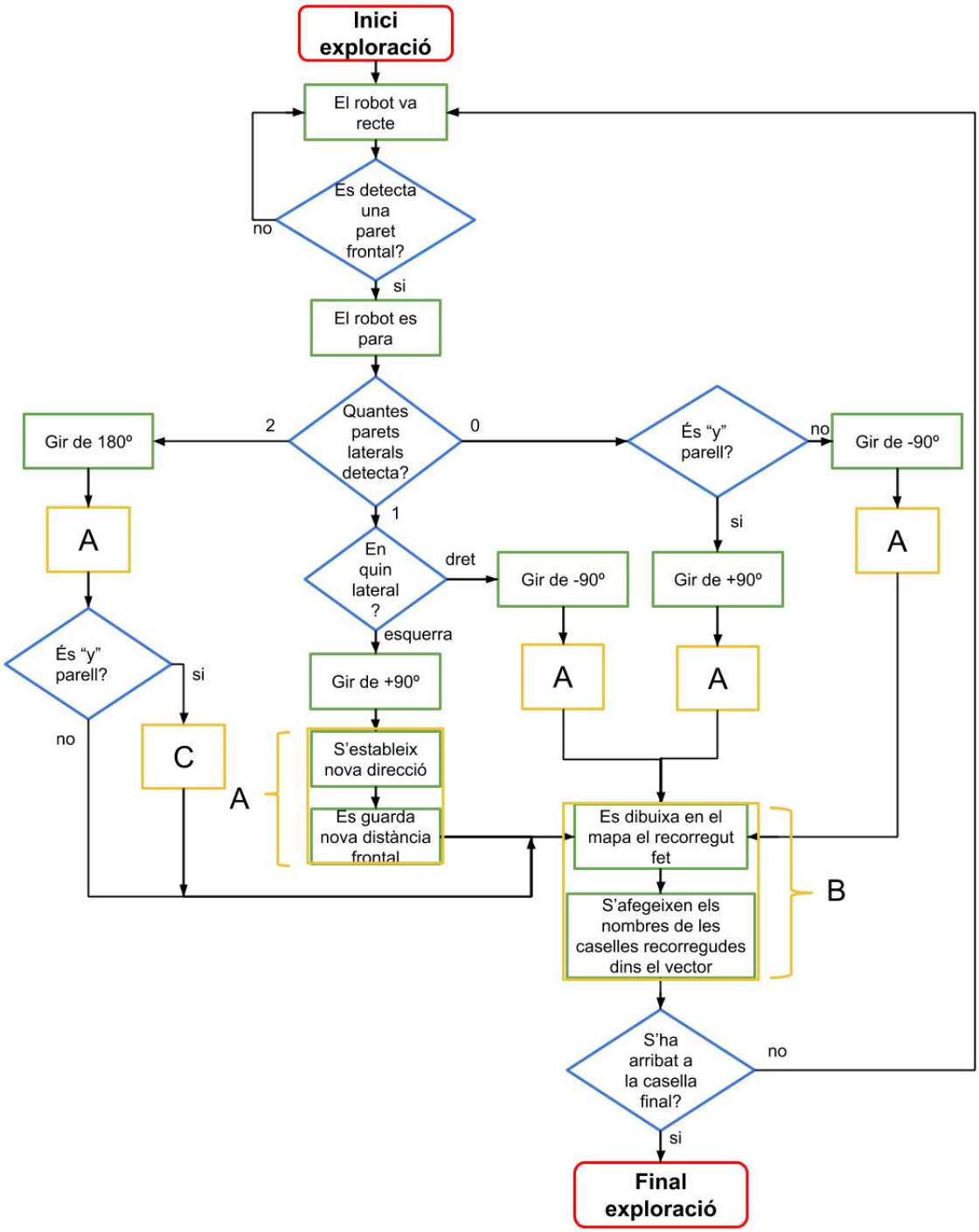

Figure 21: Diagrama de flux de l'algoritme d'exploració.

Hi ha dues situacions on s'ha deixat a l'atzar què farà el robot, el que es representa en el diagrama de flux com a "És "y" parell?", on y és una variable aleatòria que pot anar de l'1 al 100. D'aquesta forma hi ha un 50% de possibilitats que prengui una decisió o l'altra. En el cas que no tengui cap paret als laterals serveix perquè no sempre giri cap al mateix costat quan es trobi en aquesta situació.

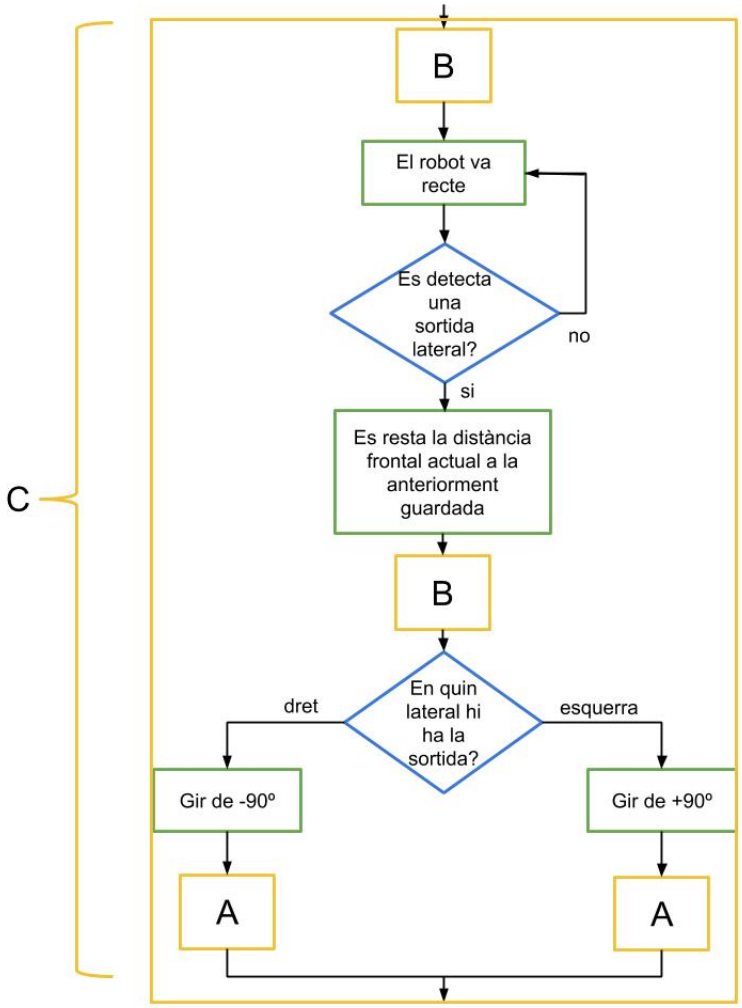

Figure 22: Diagrama de flux de l'algoritme d'exploració (segona part, bloc C).

Aquestes decisions es prenen en la funció principal "main()":

```
def main():
    global ori, f
    m = detec()  #La funció detec() capta diferents mesures del sensor
                 # frontal i proporciona la mediana.
    a = 0        #Variable que determina si hi ha una paret l'esquerra del robot.
    b = 0        #Variable que determina si hi ha una paret a la dreta del robot.
    w = 15       #Distància a la que es determina si hi ha una paret als costats.
    if 1.99 < m < 4.00 or m > 1000:  #Si el sensor frontal capta entre 2 i 4 cm de
                                     # distància, es procedeix a decidir que fa el robot.
        gpio.output(22, gpio.LOW)    #S'aturen els motors.
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.LOW)
        gpio.output(17, gpio.LOW)
        direc(ori, f)                #Funció que guarda les dades dins el vector.
        time.sleep(0.5)
        ceba.fd(f*25)                #Es dibuixa el camí fet per el passadís.
        a = sense(w)                 #Es determina si hi ha una paret a l'esquerra.
        b = sensd(w)                 #Es determina si hi ha una paret a la dreta.
        if a == 1 and b == 1:        #Cas en que hi ha dues parets laterals.
            z = volta(ori,f)         #Gir de 180º per sortir del camí sense sortida.
```

```
            ori = z[0]                  #Es guarda la nova orientació del robot.
            if z[1] < 500:
                f = math.floor((z[1])/18)    #Es calcula el nombre de caselles que
            else:                       # té el següent passadís en funció de la distància
                f = 0                   # captada després del gir (z[1]).
        elif a == 0 and b == 1:         #Cas en que hi ha una paret a la dreta.
            z = gire(ori)               #Gir a l'esquerra.
            ori = z[0]
            if z[1] < 500:
                f = math.floor((z[1])/18)
            else:
                f = 0
        elif a == 1 and b == 0:         #Cas en que hi ha una paret a l'esquerra.
            z = gird(ori)               #Gir a la dreta.
            ori = z[0]
            if z[1] < 500:
                f = math.floor((z[1])/18)
            else:
                f = 0
        elif a == 0 and b == 0:         #Cas en que no hi ha parets al voltant.
            z = gira(ori)               #Gir aleatori.
            ori = z[0]
            if z[1] < 500:
                f = math.floor((z[1])/18)
            else:
                f = 0
    else:
        gpio.output(22, gpio.HIGH)      #Si no es capta cap paret frontal el robot
        gpio.output(27, gpio.LOW)       # segueix recte.
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOW)
```

Per altra banda en el cas que es trobi en un camí sense sortida, es dóna la possibilitat que en comptes d'esperar a trobar una paret frontal una vegada ja ha realitzat el gir de 180º, busqui amb una sortida en un dels dos costats per girar cap aquell camí. La idea de fer això és per evitar que si es troba entre dos camins sense sortida, pugui sortir d'aquest i no es quedi encallat allà tot el temps. Aquest procediment alternatiu és el que conforma el bloc C, detallat en la segona part del diagrama de flux. A la figura 23 es representen dues situacions on l'aleatorietat després del gir de 180º resulta útil, on la part que està en blau es resol mitjançant el bloc C i en canvi la part en verd es resol només si no s'apliqués el bloc C, ja que si no podria no arribar a sortir del laberint si fa el gir en la casella 32.

Figure 23: Diferents situacions en què es pot trobar el robot.

Funció corresponent al bloc C del diagrama de flux de la figura 22:

```
def noucami(ori,j):                 #S'introdueix la orientació després del gir de
                                    # 180º i la distància guardada.
```

```
    gpio.output(22, gpio.HIGH)      #El robot va recte
    gpio.output(27, gpio.LOW)
    gpio.output(18, gpio.HIGH)
    gpio.output(17, gpio.LOW)
    h = False                       #Variable que determina si s'ha fet el gir o no
    j = math.floor(sens()/18)       #Es calcula el nombre de caselles fins a la
    k = 0                           # següent paret.
    w = 19                          #Distància a la que es determina si hi ha
                                    # una sortida als costats.
    while h == False:               #Mentre no s'hagi efectuat cap gir:
        a = sense(w)                #Es comprova si el sensor de l'esquerra capta
                                    # una sortida.
        b = sensd(w)                #Es comprova si el sensor de la dreta capta
                                    # una sortida.
        if a == 0:                  #Cas en que hi ha una sortida a l'esquerra.
            time.sleep(0.4)         #Temps on el robot va recte abans de girar.
            g = math.floor(sens()/18)   #Es calculen les caselles que queden en el
                                    # passadís abans de girar.
            k = j-g                 #Es calculen les caselles que s'han recorregut
                                    # per passadís.
            direc(ori,k)            #Es guarden les caselles recorregudes en el vector.
            z = gire(ori)           #Es realitza el gir.
            h = True                #S'estableix que ja s'ha fet el gir.
        elif b == 0:                #Cas en que hi ha una sortida a la dreta.
            time.sleep(0.4)
            g = math.floor(sens()/18)
            k = j-g
            direc(ori,k)
            z = gird(ori)
            h = True
    return z                #Es retorna un vector de dos elements amb la nova
                            # orientació i la distància guardada després del gir.
```

Una vegada s'ha realitzat un gir, s'estableix una nova orientació, es dibuixa en el mapa el que s'ha recorregut entre el gir anterior i el gir que acaba de fer i es guarden els nombres de les caselles respectives. Tant el dibuix en el mapa com els nombres de les caselles guardades depenen de la distància frontal que es guarda al final de cada gir i de l'orientació a la qual es troba el robot abans de fer el gir, com s'ha mencionat abans. D'aquesta manera tant la llargada de la línia dibuixada com la quantitat de nombres afegits al vector es determinen amb la distància; i la direcció a la qual es dibuixa la línia i quins nombres són els afegits depèn de l'orientació. En cas que es detecti que ha arribat a la casella de sortida, l'exploració es dóna per acabada, i en cas que torni a la casella d'entrada, s'esborrarà el camí guardat i tornarà a començar.

En la següent funció es realitza el gir cap a la dreta juntament amb les accions vinculades al gir que s'efectuen just després:

```
def gird(ori):

    gpio.output(22, gpio.LOW)       #El robot es mou cap enrere durant 0.15s.
    gpio.output(27, gpio.HIGH)
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.HIGH)
    time.sleep(0.15)

    gpio.output(22, gpio.LOW)       #El robot gira cap a la dreta durant 0.7s
    gpio.output(27, gpio.HIGH)      # per girar 90º.
    gpio.output(18, gpio.HIGH)
    gpio.output(17, gpio.LOW)
```

```
    time.sleep(0.7)
    gpio.output(22, gpio.HIGH)      #El robot gira cap a l'esquerra durant 0.15s
    gpio.output(27, gpio.LOW)       # per rectificar la roda multi-direccional.
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.HIGH)
    time.sleep(0.15)
    gpio.output(22, gpio.LOW)       #Es para el robot momentàniament per tenir
    gpio.output(27, gpio.LOW)       # un gir més exacte.
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.LOW)

    ceba.right(90)                  #El cursor de dibuix es gira 90º a la dreta.
    orii = bruixd(ori)              #S'estableix la nova direcció després de sumar
                                    # +1 al nombre de la orientació anterior.
    x = sens()                      #Es guarda la distància frontal fins a la
                                    # següent paret.
    return orii, x                  #Es retorna un vector de dos elements amb la
                                    # nova orientació i la distància guardada.
```

Finalment, els nombres corresponents a les caselles recorregudes es guarden en el vector d'exploració mitjançant la següent funció:

```
def direc(ori, f):              #S'introdueixen la orientació i el nombre de
                                #caselles recorregudes
    global pos, vec, end        #"pos" és la posició del nombre dins el vector,
                                # "vec" és el vector i "end" és la variable que
                                # determina si s'ha acabat l'exploració.
    for i in range(f):          #S'afegeixen tants nombres com caselles recorregudes
        if ori == 1:            #Si la orientació és "nord" (1):
            pos += 1            #S'avança una posició
            a = vec[pos-1]-1    #Es resta -1 al nombre anterior del vector
            vec.append(a)       #S'afegeix el nombre calculat al vector
        elif ori == 4:          #Si la orientació és "oest" (4):
            pos += 1
            b = vec[pos-1]+4    #Es suma +4 al nombre anterior del vector
            vec.append(b)
        elif ori == 3:          #Si la orientació és "sud" (3):
            und += 1
            c = vec[pos-1]+1    #Es suma +1 al nombre anterior del vector
            vec.append(c)
        elif ori == 2:          #Si la orientació és "est" (2):
            und += 1
            d = vec[pos-1]-4    #Es resta -4 al nombre anterior del vector
            vec.append(d)
        if vec[pos] >= 15 or vec[pos] < 0: end = True #En cas que s'arribi a la
                                # casella final (15) o es surti del laberint per
                                # on s'ha entrat, l'exploració es dóna per acabada.
    print(vec)              #Es mostra per pantalla el resultat actual del vector
```

En aquesta funció s'han d'anar canviant els alguns nombres en relació a les dimensions del laberint. En el cas que apareix a la funció es tracta d'un laberint de 4x4 i per tant es suma o resta 4 si es canvia de posició dins la mateixa columna, i el final del laberint és la casella 15.

3.2.5 Algoritme de resolució del laberint

La segona part del codi és on es realitza l'anàlisi de les dades recollides durant l'exploració i per tant la resolució del laberint en funció d'aquestes dades. Una vegada obtingut un camí directe cap a la sortida, s'executa per part del robot la solució trobada des de l'entrada fins a la sortida.

Per fer la resolució el que es fa és eliminar tots els camins sense sortida i loops que presenta el vector de navegació prèviament obtingut, que en aquest cas es representen amb la repetició de nombres dins del vector. Per tant, s'analitza el vector i si en algun moment es troben dos nombres iguals, s'elimina tot el que hi hagi entremig deixant només el nombre en qüestió. Per exemple, si el vector inicial és [0,1,5,6,7,8,7,6,10], el vector final després d'haver analitzat quedaria com a [0,1,5,6,10], on ja no hi ha les xifres sobrants. Aquest mètode però no assegura que s'obtingui el camí òptim a la sortida, ja que analitza els nombres per ordre i pot ser que durant el procés d'eliminació de loops, s'elimini una sèrie de nombres corresponents a un camí més curt que el que queda finalment. A més, el fet que només s'analitzi el que ha recorregut el robot fa que la solució trobada no sigui necessàriament l'òptima del laberint, sinó que com a molt serà l'òptima dins del que ha arribat a explorar el robot.

En el següent codi s'implementa l'anàlisi i l'eliminació de camins sense sortida:

```
def elim(vec):                  #S'introdueix el vector "vec" a analitzar".
    for r in range(5):
        d = []                  #Es crea el vector en blanc "d".
        for t in range(5):
            ah = vec            #Es fa una còpia de "vec" a "ah".
            g = 0
            for w in range(len(vec)):
                d.append(vec[w])        #S'afegeix el nombre actual a "d".
                for i in range(w):      #Es recorren tots els nombres.
                    if ah[i] == vec[w] and g == 0:  #Si dos nombres són iguals es
                        g = 1           # procedeix a eliminar els nombres
                                        # d'entremig.
                        h = w-i         #"h" és el nombre de xifres que s'han
                                        # d'eliminar del vector "d".
                        if h > 1:
                            for j in range(h):
                                d.pop(i)    #S'elimina el nombre en la posició "i"
                                            # del vector "d".
                        else:
                            d.pop(i)
            vec = d                 #S'actualitza el vector "vec".
            d = []                  #S'esborra el contingut del vector "d"
    return vec                      #Es retorna el vector ja analitzat.
```

El motiu pel qual en el codi anterior s'analitza unes quantes vegades el vector és per assegurar que s'eliminen tots el loops i camins sense sortida que hi hagi. Si cada vegada que passa pel codi s'elimina només un loop, s'eviten problemes derivats de la reducció de mida del vector, per tant s'ha d'assegurar que ho faci diverses vegades per eliminar tots els possibles camins innecessaris. Utilitzant l'exemple anterior de les figures 19 i 20, a continuació a la figura 24 es pot veure el resultat de la solució del laberint en funció del que ha recorregut el robot:

I el vector generat com a solució serà:

$$\text{vec} = [0,1,2,3,4,5,11,10,9,8,14,20,19,25,26,27,28,34,35]$$

Una vegada s'obté el vector final, es procedeix a què el robot realitzi el recorregut del camí en funció d'aquest, com es mostra en el següent codi. En aquesta ocasió però no emprarà els sensors per saber si té parets al voltant, sinó que s'efectuaran moviments seqüencials temporitzats per cada casella del camí obtingut.

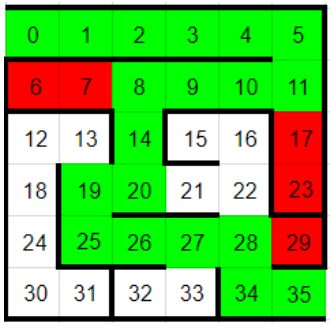

Figure 24: Exemple de laberint de 6x6 amb la solució trobada.

Per fer aquest recorregut el que fa és comparar seqüencialment els nombres del vector amb el nombre en la posició anterior per veure la diferència numèrica entre ells. En funció d'aquesta diferència i l'orientació a la qual es troba el robot en tot moment, aquest es mou en una direcció o una altra fins a arribar a la sortida del laberint. Per exemple, si el robot es troba mirant en direcció sud i la diferència entre el nombre de la posició actual en el vector i l'anterior és de +4 (en cas que el laberint sigui de 4x4), el robot efectuarà un gir cap a la dreta i avançarà una posició en direcció oest.

```
def solu(vec):                  #S'introdueix a la funció el vector de solució "vec".
  ori = 3                       #S'estableix la orientació inicial a sud (3).
  for r in range(len(vec)-1):
     if vec[r]-1==vec[r+1]:     #Si la diferència entre el nombre de la posició
         ceba.right(90)         # actual i l'anterior és de -1, el robot es col·loca
         ceba.fd(25)            # en posició nord (1).
         ceba.left(90)          #També es dibuixa el recorregut en el mapa.
         nord(ori)              #Aquesta funció efectua el gir en funció de la orientació.
         if ori != 1:           #Si ha hagut de girar, s'estableix la nova orientació
             ori = 1            # i s'avança una casella.
             nord(ori)
     elif vec[r]+4==vec[r+1]:   #Si la diferència entre el nombre de la posició
         ceba.fd(25)            # actual i l'anterior és de +4, el robot es col·loca
         oest(ori)              # en posició oest (4).
         if ori != 4:
             ori = 4
             oest(ori)          #Aquesta funció efectua el gir en funció de la orientació.
     elif vec[r]+1==vec[r+1]:   #Si la diferència entre el nombre de la posició
         ceba.left(90)          # actual i l'anterior és de +1, el robot es col·loca
         ceba.fd(25)            # en posició sud (3).
         ceba.right(90)
         sud(ori)
         if ori != 3:
             ori = 3
             sud(ori)           #Aquesta funció efectua el gir en funció de la orientació.
     elif vec[r]-4==vec[r+1]:   #Si la diferència entre el nombre de la posició
         ceba.left(180)         # actual i l'anterior és de -4, el robot es col·loca
         ceba.fd(25)            # en posició est (2).
         ceba.left(180)
         est(ori)
         if ori != 2:
             ori = 2
             est(ori)           #Aquesta funció efectua el gir en funció de la orientació

  gpio.output(22, gpio.LOW)     #Una vegada efectuat el moviment, el robot es para
  gpio.output(27, gpio.LOW)     # fins a decidir el pròxim moviment.
  gpio.output(18, gpio.LOW)
```

```
  gpio.output(17, gpio.LOW)
```

Tot l'algoritme de resolució s'executa mitjançant la funció "final()" una vegada que es determina que l'exploració ha acabat:

```
def final(vec):                 #S'introdueix el vector final "vec"
    dio1 = elim1(vec)           #S'aplica l'eliminació de loops i camins sense
                                #sortida i es guarda com a nou vector "dio1".
    dio2 = elim2(vec)           #S'aplica l'eliminació amb l'ordre dels nombres
                                # del vector girats i es guarda com a nou vector "dio2".
    if len(dio1)<=len(dio2):    #Es selecciona el vector resultant més curt.
        jojo = dio1
    else: jojo = dio2
    print(jojo)                 #Es mostra per pantalla el vector final "jojo".
    ceba.color("red")           #Es canvia el color de dibuix a vermell.
    ceba.up()                   #S'aixeca el cursor per tal d'evitar que dibuixi
    ceba.pensize(3)             #Es canvia el gruix de les línies de dibuix perquè
                                # siguin més petites.
    ceba.setposition(-200,100)  #Es retorna el cursor a la posició inicial.
    ceba.setheading(270)        #Es col·loca el cursor en la orientació inicial.
    ceba.down()                 #Es baixa el cursor perquè torni a dibuixar.
    solu(jojo)                  #Es realitza la funció "solu" amb el vector
                                # final "jojo".
```

4 Construcció i disseny del laberint

Per dissenyar i construir el laberint s'han tingut en compte les normes de l'anteriorment mencionada Micromouse. En aquesta competició es defineixen que els camins a recórrer dins el laberint seran de 18 cm d'ample i ha d'ocupar una matriu de 16x16 quadrats. Inicialment es realitzaran proves en laberints més petits i una vegada es provi que el robot pot resoldre els més petits s'anirà augmentant la mida i la complexitat dels laberints.

4.1 Construcció

El laberint construït és desmuntable i consta de tres tipus de peces principals: les parets, les bases i les pinces. Les parets estan formades per taules quadrades de fusta de 18x18 cm que s'uneixen entre elles mitjançant les pinces d'estendre roba. Les pinces són dues pinces d'estendre unides amb un espai al mig, com es pot observar a la figura 25, que per tant permeten arribar a posar fins a 4 fustes perpendiculars juntes. Les bases són petites fustes en forma quadrada amb 4 petites fustes cilíndriques posades perpendicularment a la base. Aquestes s'usen perquè l'estructura del laberint sigui més sòlida i es situen a cada extrem inferior de la paret individual, i si s'uneixen vàries parets, poden compartir base fins a 4 parets individuals. Degut a que les bases lleven espai de navegació i gir al robot, només s'empraran en cas que sigui estrictament necessari, com per aguantar parets que estiguin soles.

El fet que sigui desmuntable ajuda a canviar la configuració del laberint sempre que es vulgui i ampliar o reduir la mida si fos necessari. A més, simplifica bastant la tasca de transportar-lo si aquest fos el cas. Per altra banda, aquest tipus d'estructures són fràgils i si es dóna el cas que el robot xoqués amb un paret, podria moure o fins i tot desmuntar part del laberint.

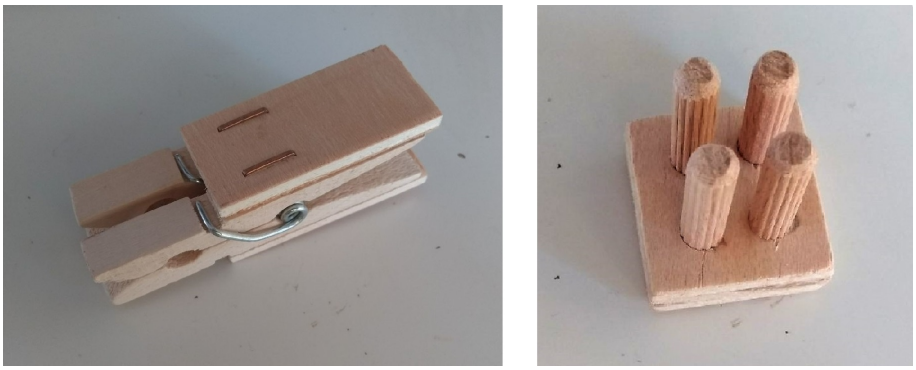

Figure 25: Pinça que uneix dues o més parets i base que fa de suport inferior a la paret.

4.2 Disseny

En el disseny del laberint com a banc de proves s'han considerat algunes simplificacions respecte a la normativa de Micromouse [1] i altres factors que afecten a la disposició del laberint:

- Així com l'objectiu de la competició és el d'arribar a un quadrat de dimensions fixes que ocupa 4 caselles col·locat en el centre del laberint, en el cas d'aquest projecte s'ha decidit que el destí final del robot sigui una sortida al costat oposat de l'entrada. Aquesta decisió s'ha pres perquè simplifica part del codi a l'hora de decidir quan ha acabat el robot la primera exploració del laberint i propicia la realització de laberints més petits que no els que marca la normativa de la competició.
- Tots els laberints usats com a banc de proves tendran una paret exterior que els envoltarà havent-hi només una entrada i una sortida.

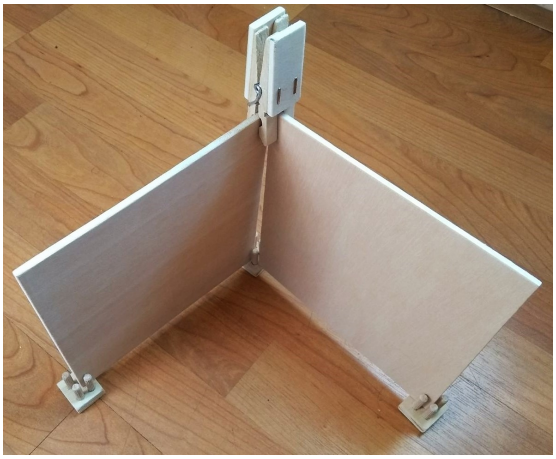

Figure 26: Estructura simple amb dues parets.

- També, com ja s'ha mencionat en la introducció, s'han fixat en un lloc concret tant l'entrada com la sortida del laberint, estant l'entrada en el punt més nord-est del laberint i la sortida en el punt més sud-oest. Aquesta disposició dels punts cardinals ve donada pel mateix codi on es considera que el robot sempre comença mirant en direcció sud tenint la paret exterior direcció est. A la figura 27 es poden veure tant la disposició dels punts cardinals del sistema d'orientació com la casella d'entrada, marcada en verd, i la casella de sortida, marcada en blau.
- Degut a un tema de falta d'espai i de peces suficients, no s'empraran laberints majors a 6x6 quadrats de 18 cm, essent així més petits dels reglamentaris en les competicions, que són de 16x16 quadrats.
- Per evitar que el laberint es pugui resoldre amb l'estratègia de sempre girar cap a la dreta o sempre girar cap a l'esquerra, s'ha procurat afegir loops dins de l'estructura del laberint, que fan que si es segueix una d'aquestes estratègies el robot es quedi donant voltes al loop tot el temps. Un exemple de loop està marcat en vermell a la figura 27.
- No només es muntarà un laberint fixe, sinó que aprofitant que és desmuntable, una vegada resolt un d'ells, es modificarà tant en mida com els passadissos interiors, sempre tenint en compte les consideracions anteriors.

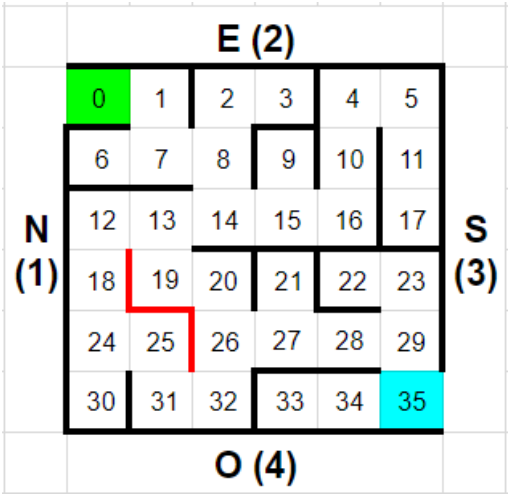

Figure 27: Esquema de laberint de 6x6.

A les imatges següents (28, 29 i 30) es mostren alguns exemples de laberints de diferents mides i complexitat:

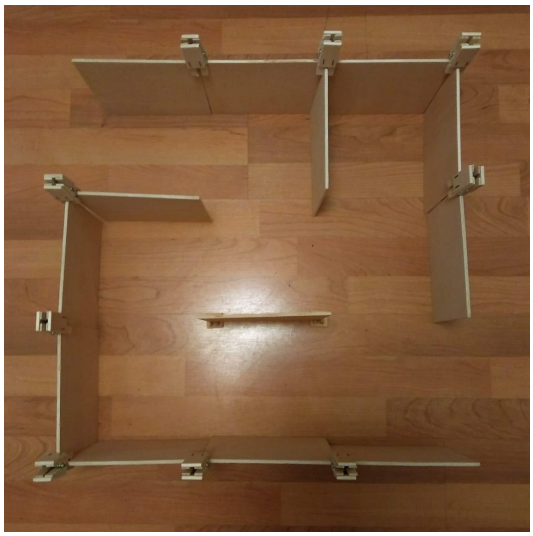

Figure 28: Exemple laberint de 3x3.

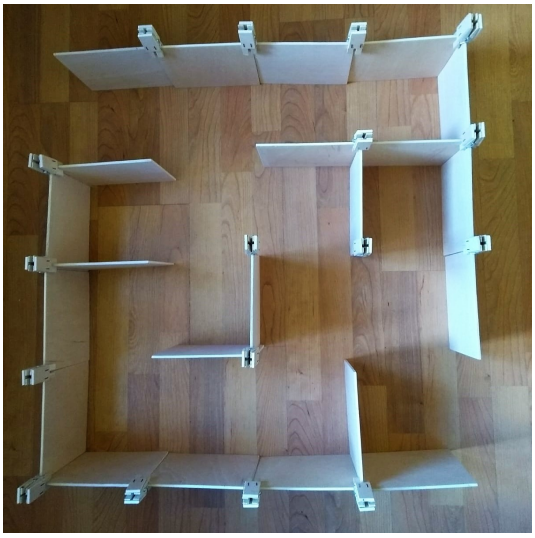

Figure 29: Exemple laberint de 4x4.

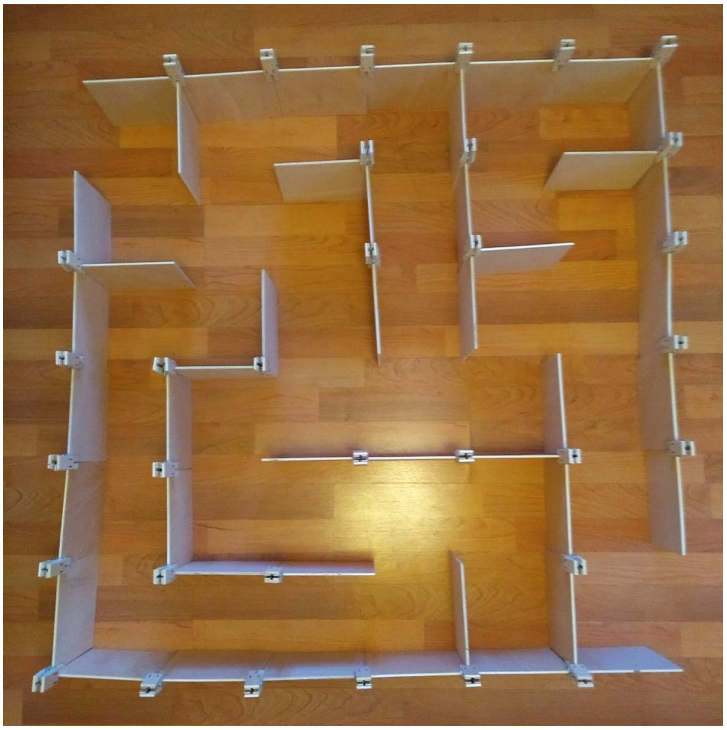

Figure 30: Exemple laberint de 6x6.

5 Resultats experimentals

5.1 Proves realitzades

En aquesta secció es reflecteixen els passos seguits per confirmar que el robot dissenyat compleix l'objectiu marcat inicialment en el projecte i les modificacions i millores que s'han hagut d'aplicar per aconseguir-ho.

5.1.1 Proves inicials

Abans de testejar el robot en un laberint complet, el que s'ha fet són proves per comprovar que el sistema funciona correctament a petita escala. D'aquesta manera es pot saber si la tracció funciona correctament, que el funcionament dels sensors no falla i que les funcions bàsiques de l'algoritme d'exploració es realitzen com toca.

Al llarg de la construcció del robot i de la implementació del codi s'han anat comprovant el funcionament individual de cada element introduït, però per verificar el correcte funcionament del conjunt del robot s'han realitzat les següents proves:

1. Comprovació del funcionament dels sensors d'ultrasons i el moviment dels motors mitjançant un programa per evitar obstacles:

   En aquesta prova el que s'ha volgut comprovar és el funcionament de la relació entre el que el robot percep a través dels sensors i els moviments que fa una vegada es topa amb un objecte, ja sigui al davant com als costats. En aquest cas no s'ha emprat cap estructura del laberint, sinó que s'ha deixat que el robot es passegi dins una habitació mentre evita qualsevol obstacle que aparegui davant seu. Mitjançant aquestes proves s'han pogut corregir alguns problemes amb el gir del robot, la fixació de les rodes i establir una distància adient d'aturada quan el robot té un objecte al davant.

2. Comprovació del moviment del robot dins del laberint:

   Una vegada el robot ja es mou evitant objectes, s'ha comprovat que es pugui moure correctament a través del laberint. Amb aquest objectiu, es fa recórrer el robot per un petit passadís amb cantons, com es pot veure en la imatge 31.

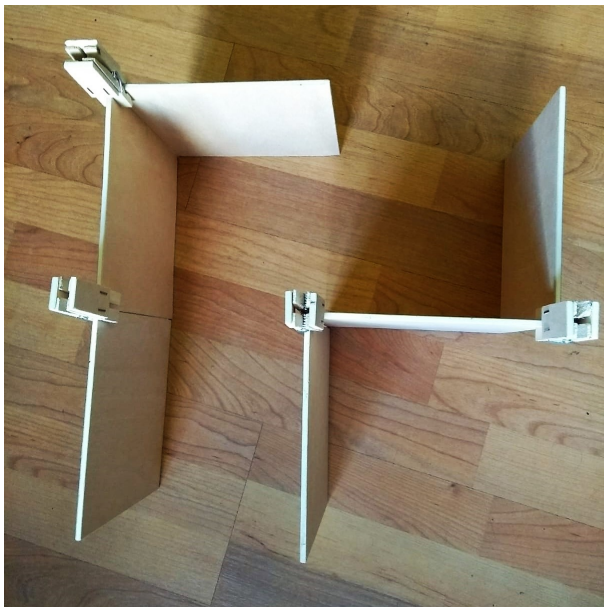

Figure 31: Passadís per proves bàsiques.

   Gràcies a aquesta prova s'ha constatat que els sensors d'ultrasons proporcionen lectures falses en cas que estiguin dins un espai petit, degut al rebot de les ones sonores dins

de les parets del laberint. Per solucionar aquest problema s'han afegit millores tant en el codi com en el robot. Primer de tot en comptes d'usar la primera lectura del sensor frontal per decidir si hi ha una paret propera al davant, s'ha realitzat un filtratge de lectures realitzant 7 lectures quasi simultànies, s'han ordenat de major a menor i s'ha donat per bona la mesura que queda al centre de la llista. També dins el codi s'ha limitat el marge de distància, ja que moltes vegades la lectura falsa era menor a 1 cm. Al robot, el que s'ha afegit són unes "ulleres" al sensor frontal, uns petits tubs de paper col·locats al voltant de l'emissor i el receptor d'ultrasons del sensor per redirigir el senyal i disminuir l'efecte de rebot.

Per altra banda, també ha servit per acabar d'ajustar coses com els temps de gir dels motors i les decisions que pren el robot en funció de les parets que es troben al seu costat.

3. Comprovació de la visualització i l'emmagatzemament del recorregut:

   A més, s'ha hagut de comprovar que mentre el robot circula dins del laberint, també va guardant el camí per allà on passa correctament i que es mostri el camí per la pantalla de l'ordinador a la vegada que fa el recorregut. Utilitzant el mateix passadís de la imatge 31, s'ha pogut veure com funcionava una versió més senzilla de l'algoritme d'exploració i així ajustar paràmetres del codi de cara a obtenir un millor resultat a l'hora de plasmar el camí explorat en un laberint complet.

4. Comprovació d'actuació en camins sense sortida:

   Pel fet que l'algoritme en cas que el robot es trobi en un camí sense sortida és lleugerament diferent d'altres situacions de gir, s'ha testejat per separat per veure si aquest aconsegueix sortir d'un doble camí sense sortida amb sortides només als laterals, com ja s'ha mencionat en l'apartat de Codi. Gràcies a aquestes proves s'han pogut ajustar els temps de reacció per als girs i corregir errors a l'hora de dibuixar el recorregut en el mapa de visualització.

5.1.2 Proves finals

Com a proves finals s'han considerat aquelles en què s'hagi usat tot el codi en conjunt dins un laberint i s'hagin dut a terme diferents funcions en la mateixa prova. S'ha començat amb laberints petits i senzills i s'ha anat augmentant la dificultat a mesura que s'ha anat perfeccionant el codi i en alguns casos l'estructura del robot.

Per començar, s'han fet proves amb laberints 4x4. A continuació es veu el procés d'exploració i resolució del laberint de la figura 32 a partir de la visualització del mapa del recorregut i els vectors generats a partir de l'exploració:

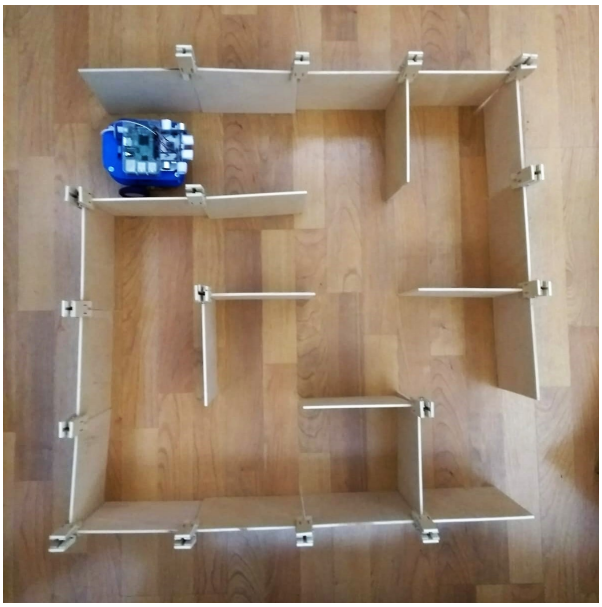

Figure 32: Laberint 4x4 per proves.

Figure 33: Mapa generat del laberint.

El vector final de l'exploració d'aquest laberint és el següent:

$$\text{vec} = [0, 1, 2, 6, 9, 13, 12, 8, 4, 5, 6, 7, 3, 7, 6, 5, 4, 8, 12, 13, 14, 13, 9, 10, 11, 15]$$

Una vegada el vector d'exploració s'ha analitzat, s'extreu d'aquest el vector amb la solució del laberint:

$$\text{vec} = [0, 1, 2, 6, 10, 11, 15]$$

En aquest cas, s'ha donat la situació que l'exploració ha estat exhaustiva, ja que tot i no ser necessari, el robot ha recorregut totes les caselles del laberint.

A partir d'aquesta prova s'ha pogut millorar el vector resultant de la solució afegint al codi una funció similar a la d'anàlisi del vector d'exploració on s'analitza en vector en

sentit contrari, i una vegada s'obté aquest segon resultat, es comparen el dos i s'aplica el camí que sigui més curt. Tot i aquest canvi, es pot donar el cas que el camí resultant de sortida tampoc sigui l'òptim del laberint. A continuació a les figures 34 i 35 es pot veure una comparació dels resultats de cada funció d'anàlisi:

Figure 34: Solució en verd amb codi inicial.

Figure 35: Solució en vermell amb el nou codi de resolució.

Un factor que ha afectat bastant a la navegació del robot dins del laberint és que al ser un recorregut més llarg, els petits errors de moviment es van acumulant, fent que en alguns casos el robot no vagi recte pels passadissos i es vagi xocant contra les parets, arribant a desmuntar el laberint en algunes ocasions. En les proves inicials, si hi havia un petit desviament no afectava gaire a l'hora de moure's pel passadís, ja que no li donava temps a acumular tant d'error, però a mesura que es va movent per dins del laberint el desviament pot arribar a ser massa gran i provocar errors tant de lectura de distàncies com de mapejat. S'ha aconseguit reduir l'error afegint una subjecció de fusta entre els motors per mantenir-los el més recte possibles i fixant millor les rodes als motors. A més, per contrarestar l'efecte de desviament provocat per la rodeta multi-direccional, s'ha afegit un gir molt curt en sentit contrari del gir que acaba de realitzar, de manera que la rodeta multi-direccional es recol·loca i no afegeix error al moviment següent. Aquesta modificació es pot observar a l'exemple de gir de l'apartat de Codi.

Finalment s'han realitzat proves amb laberints de 6x6, com la que es mostra a continuació, on el robot realitza vàries exploracions dins el mateix laberint (figura 36).

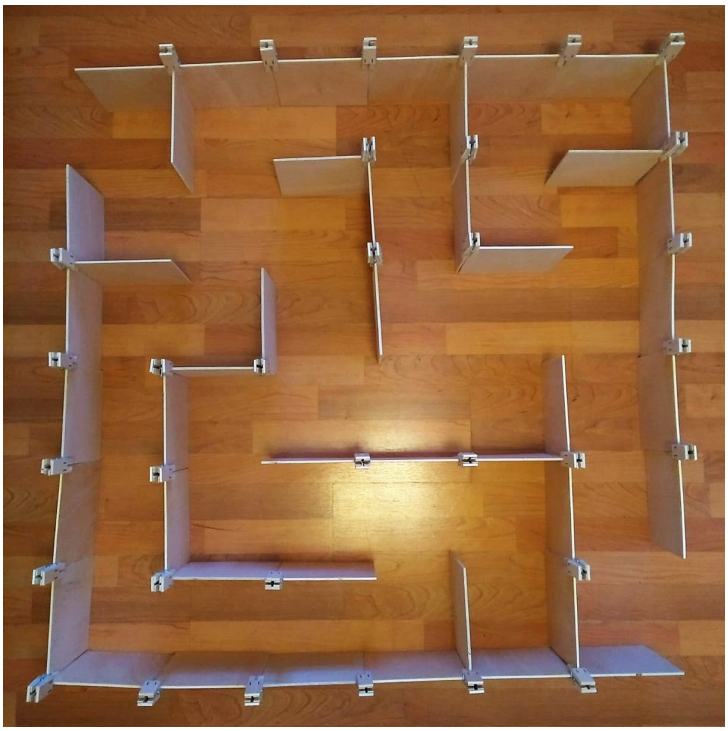

Figure 36: Laberint 6x6 per proves.

A la primera i segona exploració, figures 37 i 38, que s'ha realitzat no han estat exhaustives, però tot i això s'ha aconseguit que el robot obtingués la mateixa solució malgrat anar per diferents camins. En el cas d'aquest laberint es dóna el cas que hi ha fins a tres solucions possibles igual de curtes.

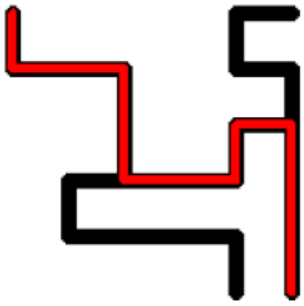

Figure 37: Primera exploració del laberint amb la seva solució.

Després de diverses proves s'ha obtingut una exploració exhaustiva del laberint on a la solució s'ha produït un error. L'error està marcat amb un cercle verd a la figura 39 i és degut al fet que el nombre de vegades que s'ha analitzat el vector final en la funció d'eliminació no ha estat suficient i no s'ha eliminat els nombres corresponents a aquest camí sense sortida. Per tant, a l'haver augmentat la mida del laberint el nombre de loops o camins a eliminar és major i s'ha de tenir en compte a l'hora d'utilitzar la funció "elim()" mostrada a l'apartat de Codi. Una vegada fixat l'error, s'han fet més proves fins a comprovar que si es realitza

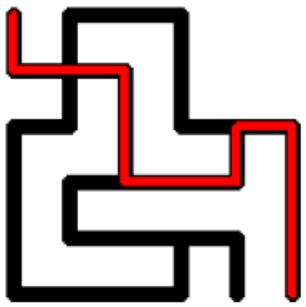

Figure 38: Segona exploració del laberint amb la seva solució.

un recorregut exhaustiu en l'exploració no es produeixi l'error comentat una altra vegada a la resolució, com es pot veure a la figura 40.

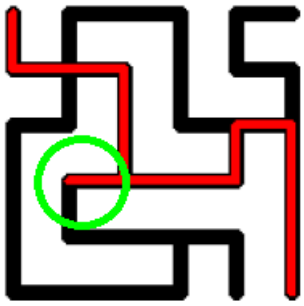

Figure 39: Exploració exhaustiva amb error a la resolució.

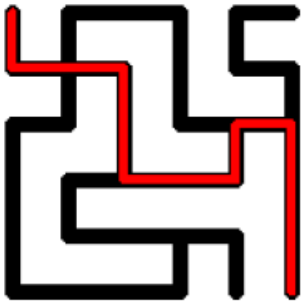

Figure 40: Exploració exhaustiva amb resolució corregida.

5.2 Problemes i consideracions

En aquesta secció es comentaran tant els problemes principals amb els quals m'he anat trobant durant la realització del projecte i quines mesures s'han pres com algunes consideracions respecte als problemes presentats.

Per començar, un factor que ha afectat i en gran part limitat la presa de decisions del robot és el fet que no faci girs si no es topa amb una paret de front. Aquest fet fa que depenent de la disposició del laberint, no pugui arribar a passar per certs camins i inclús en alguna ocasió no pugui trobar la casella final. Aquest és un problema de plantejament de la mateixa estratègia d'exploració i per tant de tot el codi, que tot i adonar-me'n, no es va trobar una alternativa factible a part d'afegir un element d'aleatorietat on el robot en certes ocasions no busca per una paret frontal, sinó per una sortida lateral després de fer un gir de 180º.

Per altra banda, inicialment es volia afegir una brúixola digital al sistema de sensors per tal que es pogués orientar correctament en tot moment i realitzar uns girs més precisos sense dependre del temps de gir establert. El problema va ser que una vegada la brúixola va estar soldada a la placa dels sensors, aquesta va deixar de funcionar correctament per motiu del camp magnètic tant de la Raspberry com d'altres elements del robot, que afectaven a les mesures. A partir d'aquí, tot i utilitzar programes per calibrar-la correctament no es va aconseguir eliminar les interferències. Per tant, finalment es va optar per no emprar-la tot i seguir soldada a la placa, i es va prioritzar que el robot es pogués moure tot i fer girs aproximats.

Un altre problema relacionat amb els sensors ha estat la pèrdua ocasional del senyal "echo" dels ultrasons que feia que el programa es quedés encallat esperant el senyal de resposta. Això feia que en moltes proves, el robot seguís recte tot i tenir la paret al davant. Aquest problema va ser bastant difícil de detectar, ja que pareixia que era la mateixa Raspberry la que es quedava encallada, però una vegada es va trobar la font del problema, es va poder solucionar fàcilment afegint un comptador en el moment d'espera del senyal, que una vegada finalitzat fa que el programa continuï correctament. La solució ja està implementada en el codi de mostra dels sensors HC-SR04 en l'apartat de Codi.

Com anteriorment s'ha mencionat, un problema derivat del disseny propi del robot són els errors en el disseny de la tracció, que han provocat que en alguns moments el robot acumuli error de desviació, no vagi del tot recte i col·lisioni amb les parets del laberint. Tot i haver-se intentat reduir al màxim la deriva, el robot segueix presentant petits problemes de direcció.

En relació a l'estructura del laberint, tot i que es menciona la utilització de les bases, els petits quadrats de fusta amb quatre barres sortints, només se'n fa ús en el cas de parts individuals perquè el puguin aguantar dretes. El motiu és que les bases limitaven l'amplària dels passadissos del laberint i el robot no tenia prou espai per maniobrar.

A part de tot això, al robot s'hi podrien haver afegit alguns components electrònics més per complementar la informació que mostra per la pantalla de l'ordinador, com per exemple LEDs que indiquessin en quina fase es troba, exploració o resolució, o algun component sonor per indicar que ja ha acabat l'exploració. A més, s'ha trobat a faltar interruptors o polsadors per iniciar ambdues fases i així no haver de dependre del temps d'espera entre fases.

6 Costos Econòmics

En aquest apartat es contemplen els costos econòmics que ha implicat el projecte tant en relació als materials adquirits per fer-ho com el temps dedicat.

6.1 Costos de materials

A continuació es presenten els materials emprats i el seu cost:

Materials	Quantitat/Preu unitat (€)	Cost total (€)
Kit Raspberry Pi Model 3B*	1 / 45,95	45,95
Sensors d'ultrasons	3 / 2,05	6,15
Placa de baquelita	1 / 5,30	5,30
Motors	2 / 5,49	10,98
Socket L293D	1 / 0,20	0,20
Portapiles	1 / 3,61	3,61
Power Bank	1 / 14,99	14,99
Rodeta multidireccional	1 / 2,54	2,54
PLA impressió 3D**	350 g / 9.99	9.99
Cargols volumètrics (paquet)	1 / 2,60	2,60
Materials laberint***	- / 39,87	39,87
Eines emprades****	- / 53,05	53,05
COST FINAL		195,23

Table 2: Costos dels materials emprats.

*Dins del Kit estan inclosos materials com els cables emprats, el driver L293D i la carcassa de la Raspberry a part de la mateixa Raspberry.

**Aquí es compta el material (PLA) usat per fer totes les peces 3D en grams i el preu del conjunt de les peces.

***Dins materials dels laberints s'han inclòs tots els diferents tipus de fusta comprats, com la planxa usada per les parets, les pinces d'estendre, les varilles per fer les bases o la cola i grapes per fusta.

****Aquí s'inclouen totes les eines usades tant per fer les peces del laberint com per soldar les plaques de baquelita.

6.2 Costos d'enginyeria

Tot i no ser un projecte destinat a la producció i venda del prototip dissenyat i construït, s'han afegit els costos de producció que tendria en termes de temps dedicat a cada una de les parts que han conformat la realització del projecte.

Parts del projecte	Hores dedicades/Preu hora (€)	Cost total (€)
Disseny	116 / 20	2320
Construcció	114 / 20	2280
Programació	256 / 20	5120
Redacció de la memòria	119 / 20	2380
TOTAL	HORES: 605	COST: 12100

Table 3: Temps dedicat i preu per hora.

Dins de disseny s'hi inclou el temps dedicat a la selecció de sensors i compra d'aquests sensors, així com la recerca d'informació respecte a la competició Micromouse i d'altres robots petits autònoms, juntament amb el disseny de l'estructura del robot.

A construcció es comptabilitzen les hores dedicades al muntatge del robot, com la soldadura de les plaques de sensors i motors, les hores d'impressió de les peces en 3D i la

col·locació de tots els elements que conformen el robot. A més també s'han afegit les hores invertides a fer totes les peces de fusta del laberint.

Com a programació s'han comptat les hores d'investigació respecte al codi, a les llibreries emprades i al funcionament de la Raspberry, les proves individuals dels sensors, i la implementació de totes les petites parts del codi fins a arribar a obtenir el definitiu.

El temps dedicat a fer les proves del robot dins del laberint així com les modificacions que s'han hagut de dur a terme a partir d'aquestes s'han repartit entre la construcció i la programació.

7 Impacte mediambiental

Per aquest projecte s'han emprat tant materials reutilitzats de treballs realitzats durant el grau com comprat components a propòsit.

Dins l'àmbit de la impressió 3D de cada vegada més s'estan utilitzant materials reciclats per fer el fil d'impressió, com botelles i altres plàstics llençats al mar o residus de materials orgànics [7]. Per altra banda, també està en augment la creació i venda de màquines que reciclen el mateix material ja usat en altres impressions, per tant, tot i que el plàstic emprat en el meu projecte no prové de material reciclat, es podria reciclar i reutilitzar en altres impressions si fos necessari.

Donada la versatilitat de la Raspberry, aquesta es podria utilitzar en molts altres projectes, donant-li una vida útil bastant llarga. Com es tracta d'un petit ordinador, a l'hora de llençar-la s'hauria de portar a un punt de deixalles electròniques, com altres aparells electrònics i electrodomèstics, també anomenats RAEE (*Residus d'Aparells Elèctrics i Electrònics*)[8].

El problema dels RAEE és que tot i la seva catalogació com a residus perillosos, deguda al fet que molts contenen substàncies tòxiques, com plom o mercuri, només es recicla correctament el 30% d'aquests. Gran part d'aquests residus acaba en països poc desenvolupats, majoritàriament a l'Àfrica i l'Àsia, on no existeix cap tipus de regulació en aquest àmbit i surt més a compte econòmicament a les empreses europees i nord-americanes exportar-hi les deixalles fent-les passar com a "productes de segona mà", que no reciclar-les [8]. A més, per culpa de l'obsolescència programada, que acurça la vida útil dels aparells electrònics [9], hi ha de cada vegada més producció, venda i finalment generació de RAEE, fet que fa més urgent l'aplicació de mesures per eliminar aquest problema tant ambiental com social. Per altra banda, en molts casos en els països desenvolupats el reciclatge o tractament dels RAEE implica la seva crema, augmentant així l'alliberament de diòxid de carboni a l'atmosfera.

Finalment, les piles emprades són recarregables, i que per tant poden ser reutilitzades una mitjana de 300 cicles cada una. Tant les piles com la bateria Power Bank poden ser reciclades si es depositen en contenidors habilitats, i una vegada tractades es pot arribar reutilitzar a entre un 50% i un 75% dels seus materials [10].

8 Conclusions

Per començar cal comentar que s'han assolit en gran mesura tots els objectius marcats a l'inici del projecte, ja que s'ha aconseguit dissenyar i construir un robot seleccionant i/o dissenyant tots els elements que el conformen; s'ha implementat un codi que fa que el robot sigui capaç de navegar pels laberints muntats, resoldre'ls tot i que no sempre amb la millor solució i de mentre mostrar per pantalla el recorregut; i finalment s'han realitzat proves per verificar i millorar el seu funcionament.

Per altra banda, s'ha de dir que gran part dels problemes que han sorgit a l'hora de fer les proves deriven del disseny del robot i per tant de la decisió inicial de fer l'estructura del robot de zero, ja que tenia com a opció comprar una estructura de robot mòbil i així poder-me centrar més en la part de software. Això ha fet que molt temps dedicat al disseny i millora del robot es podrien haver invertit a aprofundir en l'estratègia d'exploració i la resolució dels laberints.

Per aquest motiu, consider que el treball es podria ampliar en alguns aspectes de cara al futur. Una possible ampliació seria un algoritme per tal que sabés en quines caselles ha estat durant l'exploració i que prengués les decisions de recorregut basant-se en aquest coneixement, de tal manera que es realitzés sempre una exploració exhaustiva del laberint.

A part de l'àmbit dels laberints, aquest projecte també es podria extrapolar a aplicacions més rellevants, en general per trobar sortides ràpides dins un entorn explorat, ja sigui per terrenys difícils o perillosos com per optimitzar processos on actuen robots mòbils, que de cada vegada és més comú, sobretot en l'àmbit industrial.

A més, aquest projecte m'ha servit per aprofundir els meus coneixements sobretot en la part de software, tant per haver après a utilitzar la Raspberry Pi com per haver millorat el meu domini del llenguatge Python. També m'ha servit per aprendre a planificar-me millor respecte a projectes de llarga durada.

En conclusió, consider que tot i que el projecte es pot expandir i aplicar-hi millores, ha resultat satisfactori amb l'objectiu global plantejat inicialment.

References

[1] HARRISON, Peter, *Micromouse Online - Everything for Micromouse and Line Follower Robots.* Micromouse Online [en línia]. [Darrera consulta: 21/11/2018] Disponible a: http://www.micromouseonline.com/

[2] MALDONADO, Daniela, *Micromouse*, Los Andes [en línia], 2014. [Darrera consulta: 4/12/2018] Disponible a: http://portfolios.uniandes.edu.co/gallery/33882580/Micromouse

[3] *Raspberry Pi 3 Model B*, Raspberry Pi Org [en línia]. [Darrera consulta: 6/11/2018] Disponible a: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[4] *Raspberry y GPIO*, Prometec [en línia] [Darrera consulta: 6/11/2018] Disponible a: https://www.prometec.net/rpi-gpio/

[5] LIGO, George, *Interfacing HC-SR04 Ultrasonic Sensor with PIC Microcontroller*, Electrosome [en línia], 2018. [Darrera consulta: 17/10/2018] Disponible a: https://electrosome.com/hc-sr04-ultrasonic-sensor-pic/

[6] Datasheet HC-SR04. [Darrera consulta 17/10/2018] Disponible a: http://raspoid.com/download/datasheet/HCSR04

[7] C.,Lucía, *10 iniciativas que combinan la impresión 3D y la ecología*, 3D natives [en línia], 2016. [Darrera consulta: 9/12/2018] Disponible a: https://www.3dnatives.com/es/top-10-de-iniciativas-que-combinan-la-impresion-3d-y-la-ecologia-19012016/

[8] DOMÈNECH, Gemma, *Deixalles electròniques: el progrés que intoxica el planeta*, El MónPlaneta [en línia], 2017. [Darrera consulta: 9/12/2018] Disponible a: https://elmon.cat/monplaneta/reportatges/deixalles-electroniques-el-progres-que-intoxica-el-planeta

[9] DANNORITZER, Cosima, *La Fàbrica del Sol: Destí dels residus elèctrics i electrònics*, Ajuntament de Barcelona [en línia], 2014. [Darrera consulta: 9/12/2018] Disponible a: http://ajuntament.barcelona.cat/lafabricadelsol/ca/content/desti-dels-residus-electrics-i-electronics

[10] COMAS, Víctor, *La recogida y reciclaje de pilas y baterías avanza por buen camino*, Interempresas [en línia], 2012. [Darrera consulta: 9/12/2018] Disponible a: https://www.interempresas.net/Reciclaje/Articulos/100393-La-recogida-y-reciclaje-de-pilas-y-baterias-avanza-por-buen-camino.html

A ANNEX I

A.1 Codi complet

```
#PREPARACIÓ PER AL PROGRAMA:
import RPi.GPIO as gpio          #Declaració de llibreries necessàries per al
import time                      # programa.
from subprocess import call
import smbus
import QMCBRU
import turtle
import copy
import math
import random

gpio.setmode(gpio.BCM)       #S'estableix el sistema d'enumeració dels pins GPIO.

ceba = turtle.Turtle()       #S'assigna la variable "ceba" al cursor de dibuix.

ceba.color("black")          #Es canvia el color del cursor a negre.
ceba.pensize(10)             #S'estableix la mida de les línies
ceba.ht()                    #S'amaga el cursor agilitzant així el procés de dibuix.
ceba.up()
ceba.setposition(-200,100)   #Es col·loca el cursor en la posició inicial
ceba.down()
pos = 0                      #Es crea la variable global "pos" que determina la posició dins
                             # el vector de guardat de dades "vec".
vec = [0]                    #S'inicia el vector "vec".
ori = 3                      #S'estableix "sud" com a orientació inicial.

gpio.setup(22, gpio.OUT)     #S'assignen els pins GPIO corresponents als motors.
gpio.setup(27, gpio.OUT)
gpio.setup(18, gpio.OUT)
gpio.setup(17, gpio.OUT)

gpio.setup(23, gpio.IN)      #S'assignen els pins GPIO corresponents al sensor
gpio.setup(24, gpio.OUT)     # frontal.

gpio.setup(6, gpio.IN)       #S'assignen els pins GPIO corresponents al sensor
gpio.setup(5, gpio.OUT)      # de l'esquerra.

gpio.setup(21, gpio.IN)      #S'assignen els pins GPIO corresponents al sensor
gpio.setup(20, gpio.OUT)     # de la dreta.

gpio.output(22, gpio.HIGH)   #Es posen en marxa els motors per anar endavant.
gpio.output(27, gpio.LOW)
gpio.output(18, gpio.HIGH)
gpio.output(17, gpio.LOW)

#FUNCIÓ PRINCIPAL DE LA FASE D'EXPLORACIÓ:
def main():
   global ori, f
   m = detec()  #La funció detec() capta diferents mesures del sensor
                # frontal i proporciona la mediana.
   a = 0        #Variable que determina si hi ha una paret l'esquerra del robot.
   b = 0        #Variable que determina si hi ha una paret a la dreta del robot.
   w = 15       #Distància a la que es determina si hi ha una paret als costats.
   if 1.99 < m < 4.00 or m > 1000:  #Si el sensor frontal capta entre 2 i 4 cm de
```

```
                                    # distància, es procedeix a decidir que fa el robot.
        gpio.output(22, gpio.LOW)   #S'aturen els motors.
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.LOW)
        gpio.output(17, gpio.LOW)
        direc(ori, f)               #Funció que guarda les dades dins el vector.
        time.sleep(0.5)
        ceba.fd(f*25)               #Es dibuixa el camí fet per el passadís.
        a = sense(w)                #Es determina si hi ha una paret a l'esquerra.
        b = sensd(w)                #Es determina si hi ha una paret a la dreta.
        if a == 1 and b == 1:       #Cas en que hi ha dues parets laterals.
            z = volta(ori,f)        #Gir de 180º per sortir del camí sense sortida.
            ori = z[0]              #Es guarda la nova orientació del robot.
            if z[1] < 500:
                f = math.floor((z[1])/18)    #Es calcula el nombre de caselles que
            else:                   # té el següent passadís en funció de la distància
                f = 0               # captada després del gir (z[1]).
        elif a == 0 and b == 1:     #Cas en que hi ha una paret a la dreta.
            z = gire(ori)           #Gir a l'esquerra.
            ori = z[0]
            if z[1] < 500:
                f = math.floor((z[1])/18)
            else:
                f = 0
        elif a == 1 and b == 0:     #Cas en que hi ha una paret a l'esquerra.
            z = gird(ori)           #Gir a la dreta.
            ori = z[0]
            if z[1] < 500:
                f = math.floor((z[1])/18)
            else:
                f = 0
        elif a == 0 and b == 0:     #Cas en que no hi ha parets al voltant.
            z = gira(ori)           #Gir aleatori.
            ori = z[0]
            if z[1] < 500:
                f = math.floor((z[1])/18)
            else:
                f = 0
    else:
        gpio.output(22, gpio.HIGH)  #Si no es capta cap paret frontal el robot
        gpio.output(27, gpio.LOW)   # segueix recte.
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOW)

#FUNCIÓ DE GIR A LA DRETA:
def gird(ori):

    gpio.output(22, gpio.LOW)       #El robot es mou cap enrere durant 0.15s.
    gpio.output(27, gpio.HIGH)
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.HIGH)
    time.sleep(0.15)

    gpio.output(22, gpio.LOW)       #El robot gira cap a la dreta durant 0.7s
    gpio.output(27, gpio.HIGH)      # per girar 90º.
    gpio.output(18, gpio.HIGH)
    gpio.output(17, gpio.LOW)
    time.sleep(0.7)
```

```
    gpio.output(22, gpio.HIGH)      #El robot gira cap a l'esquerra durant 0.15s
    gpio.output(27, gpio.LOW)       # per rectificar la roda multi-direccional.
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.HIGH)
    time.sleep(0.15)
    gpio.output(22, gpio.LOW)       #Es para el robot momentàniament per tenir
    gpio.output(27, gpio.LOW)       # un gir més exacte.
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.LOW)

    ceba.right(90)                  #El cursor de dibuix es gira 90º a la dreta.
    orii = bruixd(ori)              #S'actualitza l'orientació.
    x = sens()                      #Es guarda la distància frontal fins a la
                                    # següent paret.
    return orii, x                  #Es retorna un vector de dos elements amb la
                                    # nova orientació i la distància guardada.

#FUNCIÓ DE GIR A L'ESQUERRA:
def gire(ori):

    gpio.output(22, gpio.LOW)       #El robot es mou cap enrere durant 0.15s.
    gpio.output(27, gpio.HIGH)
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.HIGH)
    time.sleep(0.15)

    gpio.output(22, gpio.HIGH)      #El robot gira cap a l'esquerra durant 0.7s
    gpio.output(27, gpio.LOW)       # per girar 90º.
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.HIGH)
    time.sleep(0.7)
    gpio.output(22, gpio.LOW)       #El robot gira cap a la dreta durant 0.15s
    gpio.output(27, gpio.HIGH)      # per rectificar la roda multi-direccional.
    gpio.output(18, gpio.HIGH)
    gpio.output(17, gpio.LOWH)
    time.sleep(0.15)
    gpio.output(22, gpio.LOW)       #Es para el robot momentàniament per tenir
    gpio.output(27, gpio.LOW)       # un gir més exacte.
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.LOW)
    ceba.left(90)                   #El cursor de dibuix es gira 90º a l'esquerra.
    orii = bruixe(ori)              #S'actualitza l'orientació.
    x = sens()                      #Es guarda la distància frontal fins a la
                                    # següent paret.
    return orii, x                  #Es retorna un vector de dos elements amb la
                                    # nova orientació i la distància guardada.

#FUNCIÓ DE GIR ALEATORI:
def gira(ori):
    y = random.randint(1,100)       #Es genera un nombre aleatori i en funció
                                    # d'aquest, es gira cap a la dreta o cap a
    if y%2 == 0:                    # l'esquerra.
       h = gird(ori)
    else:
       h = gire(ori)
    return h                #Es retorna el vector generat després del gir amb
                            # la nova orientació i la distància guardada.
```

```
#FUNCIÓ DE GIR DE 180º
def volta(ori,f):
    gpio.output(22, gpio.HIGH)  #El robot gira durant 1s per fer un gir de 180º.
    gpio.output(27, gpio.LOW)
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.HIGH)
    time.sleep(1)
    ceba.left(180)              #El cursor gira 180º.
    orii = bruixv(ori)          #S'actualitza l'orientació.
    x = sens()                  #Es guarda la distància frontal fins a la
                                # següent paret.
    y = random.randint(1,100)   #Es genera un nombre aleatori.
    if y%2 == 0:                #En funció del nombre aleatori es seguirà una
                                # estratègia o una altra.
        direc(ori,f)            #Es guarden les caselles recorregudes.
        wry = noucami(orii,v)   #Funció on es busquen sortides i no parets.
        return wry
    else:
        return orii,x

#FUNCIÓ DE GIR PER SORTIDA:
def noucami(ori):                   #S'introdueix la orientació després del gir de
                                    # 180º i la distància guardada.
    gpio.output(22, gpio.HIGH)      #El robot va recte.
    gpio.output(27, gpio.LOW)
    gpio.output(18, gpio.HIGH)
    gpio.output(17, gpio.LOW)
    h = False                       #Variable que determina si s'ha fet el gir o no.
    j = math.floor(sens()/18)       #Es calcula el nombre de caselles fins a la
    k = 0                           # següent paret.
    w = 19                          #Distància a la que es determina si hi ha
                                    # una sortida als costats.
    while h == False:               #Mentre no s'hagi efectuat cap gir:
        a = sense(w)                #Es comprova si el sensor de l'esquerra capta
                                    # una sortida.
        b = sensd(w)                #Es comprova si el sensor de la dreta capta
                                    # una sortida.
        if a == 0:                  #Cas en que hi ha una sortida a l'esquerra.
            time.sleep(0.45)         #Temps on el robot va recte abans de girar.
            g = math.floor(sens()/18)   #Es calculen les caselles que queden en el
                                    # passadís abans de girar.
            k = j-g                 #Es calculen les caselles que s'han recorregut
                                    # per passadís.
            direc(ori,k)            #Es guarden les caselles recorregudes en el vector.
            z = gire(ori)           #Es realitza el gir.
            h = True                #S'estableix que ja s'ha fet el gir.
        elif b == 0:                #Cas en que hi ha una sortida a la dreta.
            time.sleep(0.4)
            g = math.floor(sens()/18)
            k = j-g
            direc(ori,k)
            z = gird(ori)
            h = True
    return z                #Es retorna un vector de dos elements amb la nova
                            # orientació i la distància guardada després del gir.

#FUNCIÓ DEL SENSOR D'ULTRASONS FRONTAL:
def sens():
```

```
    i = 0                    #Variable per el comptador.
    gpio.output(24, False)   #Primer s'activa el senyal de trigger durant
    time.sleep(2*10**-6)     # un petit període de temps per tal de
    gpio.output(24, True)    # començar a mesurar la distància.
    time.sleep(10*10**-6)    #"24" correspon al pin GPIO on s'ha
    gpio.output(24, False)   # connectat el trigger del sensor.

    s = time.time()          #Es defineixen dos variables de temps.
    e = time.time()

    while gpio.input(23) == 0:   #S'estableix el temps d'anada de la senyal echo.
       pass                      #Per evitar que el programa es quedi encallat si el
       i += 1                    # sensor no rep una senyal de tornada es posa un
       if i >=5000:              # límit dins del loop de while.
           break                 #"23" correspon al pin GPIO on s'ha
    s = time.time()              # connectat l'echo del sensor.

    while gpio.input(23) == 1:   #S'estableix el temps de tornada de la
       pass                      # senyal echo.
    e = time.time()
    t = e-s                      #"t" és la diferència entre el temps registrat
                                 # d'anada i tornada.
    m1 = t*17150                 #"m1" és la mesura final de la distància en cm.
    return m1                    #Es retorna el valor de "m1".

#FUNCIÓ DEL SENSOR D'ULTRASONS DE L'ESQUERRA:
def sense(w):
    i = 0                    #Variable per el comptador.
    gpio.output(5, False)    #Primer s'activa el senyal de trigger durant
    time.sleep(2*10**-6)     # un petit període de temps per tal de
    gpio.output(5, True)     # començar a mesurar la distància.
    time.sleep(10*10**-6)    #"5" correspon al pin GPIO on s'ha
    gpio.output(5, False)    # connectat el trigger del sensor.

    ss = time.time()         #Es defineixen dos variables de temps.
    ee = time.time()

     while gpio.input(6) == 0:   #S'estableix el temps d'anada de la senyal echo.
       pass                      #Per evitar que el programa es quedi encallat si el
       i += 1                    # sensor no rep una senyal de tornada es posa un
       if i >=5000:              # límit dins del loop de while.
           break                 #"6" correspon al pin GPIO on s'ha
    ss = time.time()             # connectat l'echo del sensor.

    while gpio.input(6) == 1:    #S'estableix el temps de tornada de la
       pass                      # senyal echo.
    ee = time.time()

    t = ee-ss                    #"t" és la diferència entre el temps registrat
                                 # d'anada i tornada.
    m = t*17150                  #"m" és la mesura final de la distància en cm.

    if m < w:                    #Si la distància mesurada és menor a la
        a = 1                    # distància que determina si hi ha una paret
    else:                        # (w), s'estableix que hi ha una paret.
        a = 0

    return a                     #Es retorna si s'ha detectat una paret lateral o no.
```

```
#FUNCIÓ DEL SENSOR D'ULTRASONS DE LA DRETA:
def sensd(w):
   i = 0                    #Variable per el comptador.
   gpio.output(20, False)   #Primer s'activa el senyal de trigger durant
   time.sleep(2*10**-6)     # un petit període de temps per tal de
   gpio.output(20, True)    # començar a mesurar la distància.
   time.sleep(10*10**-6)    #"20" correspon al pin GPIO on s'ha
   gpio.output(20, False)   # connectat el trigger del sensor.

   ss = time.time()         #Es defineixen dos variables de temps.
   ee = time.time()

    while gpio.input(21) == 0:  #S'estableix el temps d'anada de la senyal echo.
      pass                      #Per evitar que el programa es quedi encallat si el
      i += 1                    # sensor no rep una senyal de tornada es posa un
      if i >=5000:              # límit dins del loop de while.
          break                 #"21" correspon al pin GPIO on s'ha
   ss = time.time()             # connectat l'echo del sensor.

   while gpio.input(21) == 1:   #S'estableix el temps de tornada de la
      pass                      # senyal echo.
   ee = time.time()

   t = ee-ss                    #"t" és la diferència entre el temps registrat
                                # d'anada i tornada.
   m = t*17150                  #"m" és la mesura final de la distància en cm.

   if m < w:                    #Si la distància mesurada és menor a la
       b = 1                    # distància que determina si hi ha una paret
   else:                        # (w), s'estableix que hi ha una paret.
       b = 0

   return b                     #Es retorna si s'ha detectat una paret lateral o no.

#FUNCIÓ
def detec():
    m1 = sens()             #Es prenen 7 mesures casi simultànies del sensor frontal
    time.sleep(0.0005)
    m2 = sens()
    time.sleep(0.0005)
    m3 = sens()
    time.sleep(0.0005)
    m4 = sens()
    time.sleep(0.0005)
    m5 = sens()
    time.sleep(0.0005)
    m6 = sens()
    time.sleep(0.0005)
    m7 = sens()
    v = [m1,m2,m3,m4,m5,m6,m7]  #S'introdueixen totes les mesures dins el vector "v".
    sv = sorted(v)    #S'ordenen les mesures dins el vector de més petita a més gran.
    d = sv[3]            #Es selecciona la mesura que queda en el centre del vector.

    return d                    #Es retorna la mesura seleccionada.

#FUNCIÓ DE GUARDAT DE LES CASELLES RECORREGUDES:
def direc(ori, f):              #S'introdueixen la orientació i el nombre de
```

```
                                #caselles recorregudes
    global pos, vec, end        #"pos" és la posició del nombre dins el vector,
                                # "vec" és el vector i "end" és la variable que
                                # determina si s'ha acabat l'exploració.
    for i in range(f):          #S'afegeixen tants nombres com caselles recorregudes
        if ori == 1:            #Si la orientació és "nord" (1):
            pos += 1            #S'avança una posició
            a = vec[pos-1]-1    #Es resta -1 al nombre anterior del vector
            vec.append(a)       #S'afegeix el nombre calculat al vector
        elif ori == 4:          #Si la orientació és "oest" (4):
            pos += 1
            b = vec[pos-1]+4    #Es suma +4 al nombre anterior del vector
            vec.append(b)
        elif ori == 3:          #Si la orientació és "sud" (3):
            und += 1
            c = vec[pos-1]+1    #Es suma +1 al nombre anterior del vector
            vec.append(c)
        elif ori == 2:          #Si la orientació és "est" (2):
            und += 1
            d = vec[pos-1]-4    #Es resta -4 al nombre anterior del vector
            vec.append(d)
        if vec[pos] >= 15 or vec[pos] < 0: end = True #En cas que s'arribi a la
                                # casella final (15) o es surti del laberint per
                                # on s'ha entrat, l'exploració es dóna per acabada.
    print(vec)              #Es mostra per pantalla el resultat actual del vector

#FUNCIÓ D'ACTUALITZACIÓ D'ORIENTACIÓ CAP A LA DRETA:
def bruixd(ori):
    ori = ori + 1       #Es suma +1 a l'orientació actual.
    if ori == 5:        #Si el nombre de l'orientació supera 4, aquest passa a 1.
        ori = 1
    return ori          #Es retorna l'orientació actualitzada.

#FUNCIÓ D'ACTUALITZACIÓ D'ORIENTACIÓ CAP A L'ESQUERRA:
def bruixe(ori):
    ori = ori - 1       #Es resta -1 a l'orientació actual.
    if ori == 0:        #Si el nombre de l'orientació baixa de 1, aquest passa a 4.
        ori = 4
    return ori          #Es retorna l'orientació actualitzada.

#FUNCIÓ D'ACTUALITZACIÓ D'ORIENTACIÓ PER GIR DE 180º:
def bruixv(ori):
    ori = ori + 2       #Es suma +2 a la orientació actual.
    if ori == 5:        #Si el nombre de l'orientació és 5, aquest passa a 1.
        ori = 1
    elif ori == 6:      #Si el nombre de l'orientació és 6, aquest passa a 2.
        ori = 2
    return ori          #Es retorna l'orientació actualitzada.

#FUNCIÓ DE RESOLUCIÓ DEL LABERINT:
def solu(vec):                  #S'introdueix a la funció el vector de solució "vec".
  ori = 3                       #S'estableix l'orientació inicial a sud (3).
  for r in range(len(vec)-1):
     if vec[r]-1==vec[r+1]:     #Si la diferència entre el nombre de la posició
         ceba.right(90)         # actual i l'anterior és de -1, el robot es col·loca.
         ceba.fd(25)            # en posició nord (1).
         ceba.left(90)          #També es dibuixa el recorregut en el mapa.
         nord(ori)              #Aquesta funció efectua el gir en funció de l'orientació.
```

```
            if ori != 1:            #Si ha hagut de girar, s'estableix la nova orientació
                ori = 1             # i s'avança una casella.
                nord(ori)
        elif vec[r]+4==vec[r+1]:    #Si la diferència entre el nombre de la posició
            ceba.fd(25)             # actual i l'anterior és de +4, el robot es col·loca.
            oest(ori)               # en posició oest (4).
            if ori != 4:
                ori = 4
                oest(ori)           #Aquesta funció efectua el gir en funció de l'orientació.
        elif vec[r]+1==vec[r+1]:    #Si la diferència entre el nombre de la posició
            ceba.left(90)           # actual i l'anterior és de +1, el robot es col·loca.
            ceba.fd(25)             # en posició sud (3).
            ceba.right(90)
            sud(ori)
            if ori != 3:
                ori = 3
                sud(ori)            #Aquesta funció efectua el gir en funció de l'orientació.
        elif vec[r]-4==vec[r+1]:    #Si la diferència entre el nombre de la posició
            ceba.left(180)          # actual i l'anterior és de -4, el robot es col·loca.
            ceba.fd(25)             # en posició est (2).
            ceba.left(180)
            est(ori)
            if ori != 2:
                ori = 2
                est(ori)            #Aquesta funció efectua el gir en funció de l'orientació.

    gpio.output(22, gpio.LOW)       #Una vegada efectuat el moviment, el robot es para.
    gpio.output(27, gpio.LOW)       # fins a decidir el pròxim moviment.
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.LOW)

#FUNCIÓ D'ANÀLISI DEL VECTOR FINAL:
def elim1(vec):                     #S'introdueix el vector "vec" a analitzar".
    for r in range(5):
        d = []                      #Es crea el vector en blanc "d".
        for t in range(5):
            ah = vec                #Es fa una còpia de "vec" a "ah".
            g = 0
            for w in range(len(vec)):
                d.append(vec[w])        #S'afegeix el nombre actual a "d".
                for i in range(w):      #Es recorren tots els nombres.
                    if ah[i] == vec[w] and g == 0:  #Si dos nombres són iguals es
                        g = 1           # procedeix a eliminar els nombres
                                        # d'entremig.
                        h = w-i         #"h" és el nombre de xifres que s'han
                                        # d'eliminar del vector "d".
                        if h > 1:
                            for j in range(h):
                                d.pop(i)    #S'elimina el nombre en la posició "i"
                                            # del vector "d".
                        else:
                            d.pop(i)
            vec = d                 #S'actualitza el vector "vec".
            d = []                  #S'esborra el contingut del vector "d"
    return vec                      #Es retorna el vector ja analitzat.

#FUNCIÓ D'ANÀLISI DEL VECTOR FINAL INVERTIT:
def elim2(vec):                      #S'introdueix el vector "vec" a analitzar".
```

```
    for r in range(5):
        d = []                  #Es crea el vector en blanc "d".
        vec = vec[::-1]         #S'inverteix l'ordre dels nombres dins el vector.
        for t in range(5):
            ah = vec            #Es fa una còpia de "vec" a "ah".
            g = 0
            for w in range(len(vec)):
                d.append(vec[w])        #S'afegeix el nombre actual a "d".
                for i in range(w):      #Es recorren tots els nombres.
                    if ah[i] == vec[w] and g == 0:  #Si dos nombres són iguals es
                        g = 1           # procedeix a eliminar els nombres
                                        # d'entremig.
                        h = w-i         #"h" és el nombre de xifres que s'han
                                        # d'eliminar del vector "d".
                        if h > 1:
                            for j in range(h):
                                d.pop(i)    #S'elimina el nombre en la posició "i"
                                            # del vector "d".
                        else:
                            d.pop(i)
            vec = d                 #S'actualitza el vector "vec".
            d = []                  #S'esborra el contingut del vector "d".
    vec = vec[::-1]                 #Una vegada acabat el procés, es tornen a posar
                                    # els nombres en l'ordre corresponent.
    return vec                      #Es retorna el vector ja analitzat.

#FUNCIÓ DE MOVIMENT CAP A SUD PER LA RESOLUCIÓ:
def sud(ori):
    if ori == 3:                        #Si l'orientació és "sud", s'avança una casella.
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOW)
        time.sleep(0.8)
    elif ori == 4:                      #Si la direcció és "oest" es gira cap a l'esquerra.
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.LOW)
        gpio.output(17, gpio.HIGH)
        time.sleep(0.7)
        gpio.output(22, gpio.LOW)
        gpio.output(27, gpio.HIGH)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOWH)
        time.sleep(0.15)
    elif ori == 2:                      #Si la direcció és "est" es gira cap a la dreta.
        gpio.output(22, gpio.LOW)
        gpio.output(27, gpio.HIGH)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOW)
        time.sleep(0.7)
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.LOW)
        gpio.output(17, gpio.HIGH)
        time.sleep(0.15)
    gpio.output(22, gpio.LOW)           #Es paren momentàniament els motors.
    gpio.output(27, gpio.LOW)
```

```
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.LOW)
    time.sleep(0.1)

#FUNCIÓ DE MOVIMENT CAP A EST PER LA RESOLUCIÓ:
def est(ori):
    if ori == 2:                    #Si l'orientació és "est", s'avança una casella.
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOW)
        time.sleep(0.8)
    elif ori == 3:                  #Si la direcció és "sud" es gira cap a l'esquerra.
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.LOW)
        gpio.output(17, gpio.HIGH)
        time.sleep(0.7)
        gpio.output(22, gpio.LOW)
        gpio.output(27, gpio.HIGH)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOWH)
        time.sleep(0.15)
    elif ori == 1:                  #Si la direcció és "nord" es gira cap a la dreta.
        gpio.output(22, gpio.LOW)
        gpio.output(27, gpio.HIGH)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOW)
        time.sleep(0.7)
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.LOW)
        gpio.output(17, gpio.HIGH)
        time.sleep(0.15)
    gpio.output(22, gpio.LOW)       #Es paren momentàniament els motors.
    gpio.output(27, gpio.LOW)
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.LOW)
    time.sleep(0.1)

#FUNCIÓ DE MOVIMENT CAP A NORD PER LA RESOLUCIÓ:
def nord(ori):
    if ori == 1:                    #Si l'orientació és "nord", s'avança una casella.
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOW)
        time.sleep(0.8)
    elif ori == 2:                  #Si la direcció és "est" es gira cap a l'esquerra.
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.LOW)
        gpio.output(17, gpio.HIGH)
        time.sleep(0.7)
        gpio.output(22, gpio.LOW)
        gpio.output(27, gpio.HIGH)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOWH)
```

```
        time.sleep(0.15)
    elif ori == 4:                  #Si la direcció és "oest" es gira cap a la dreta.
        gpio.output(22, gpio.LOW)
        gpio.output(27, gpio.HIGH)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOW)
        time.sleep(0.7)
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.LOW)
        gpio.output(17, gpio.HIGH)
        time.sleep(0.15)
    gpio.output(22, gpio.LOW)       #Es paren momentàniament els motors.
    gpio.output(27, gpio.LOW)
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.LOW)
    time.sleep(0.1)

#FUNCIÓ DE MOVIMENT CAP A OEST PER LA RESOLUCIÓ:
def oest(ori):
    if ori == 4:                    #Si l'orientació és "oest", s'avança una casella.
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOW)
        time.sleep(0.8)
    elif ori == 1:                  #Si la direcció és "nord" es gira cap a l'esquerra.
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.LOW)
        gpio.output(17, gpio.HIGH)
        time.sleep(0.7)
        gpio.output(22, gpio.LOW)
        gpio.output(27, gpio.HIGH)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOWH)
        time.sleep(0.15)
    elif ori == 3:                  #Si la direcció és "sud" es gira cap a la dreta.
        gpio.output(22, gpio.LOW)
        gpio.output(27, gpio.HIGH)
        gpio.output(18, gpio.HIGH)
        gpio.output(17, gpio.LOW)
        time.sleep(0.7)
        gpio.output(22, gpio.HIGH)
        gpio.output(27, gpio.LOW)
        gpio.output(18, gpio.LOW)
        gpio.output(17, gpio.HIGH)
        time.sleep(0.15)
    gpio.output(22, gpio.LOW)       #Es paren momentàniament els motors.
    gpio.output(27, gpio.LOW)
    gpio.output(18, gpio.LOW)
    gpio.output(17, gpio.LOW)
    time.sleep(0.1)

#FUNCIÓ PRINCIPAL DE LA FASE DE RESOLUCIÓ:
def final(vec):                 #S'introdueix el vector final "vec"
    dio1 = elim1(vec)           #S'aplica l'eliminació de loops i camins sense
                                #sortida i es guarda com a nou vector "dio1".
```

```
    dio2 = elim2(vec)            #S'aplica l'eliminació amb l'ordre dels nombres
                                 # del vector girats i es guarda com a nou vector "dio2".
    if len(dio1)<=len(dio2):     #Es selecciona el vector resultant més curt.
        jojo = dio1
    else: jojo = dio2
    print(jojo)                  #Es mostra per pantalla el vector final "jojo".
    ceba.color("red")            #Es canvia el color de dibuix a vermell.
    ceba.up()                    #S'aixeca el cursor per tal d'evitar que dibuixi
                                 # mentre es resitua.
    ceba.pensize(3)              #Es canvia el gruix de les línies de dibuix perquè
                                 # siguin més petites.
    ceba.setposition(-200,100)   #Es retorna el cursor a la posició inicial.
    ceba.setheading(270)         #Es col·loca el cursor en la orientació inicial.
    ceba.down()                  #Es baixa el cursor perquè torni a dibuixar.
    solu(jojo)                   #Es realitza la funció "solu" amb el vector
                                 # final "jojo".

f = math.floor(sens()/18)        #Es calcula el nombre de caselles que té el
                                 # robot inicialment al davant.
end = False                      #Es declara la variable que determina si la fase
                                 # d'exploració ha acabat o no.
#BUCLE PRINCIPAL:
while True:
   try:
      main()                     #Es crida la funció principal main() fins que
                                 # s'acaba l'exploració.
      if end == True:            #Una vegada acabada la primera fase, es paren
          gpio.output(22, gpio.LOW) # els motors durant 5s i es procedeix a executar
          gpio.output(27, gpio.LOW) # la fase de resolució amb la funció final().
          gpio.output(18, gpio.LOW)
          gpio.output(17, gpio.LOW)
          time.sleep(5)
          final(wry)
          break
   except KeyboardInterrupt:
      break
gpio.cleanup()
```

A.2 Esquemes elèctrics

A continuació es presenten els esquemes elèctrics que conformen el robot:

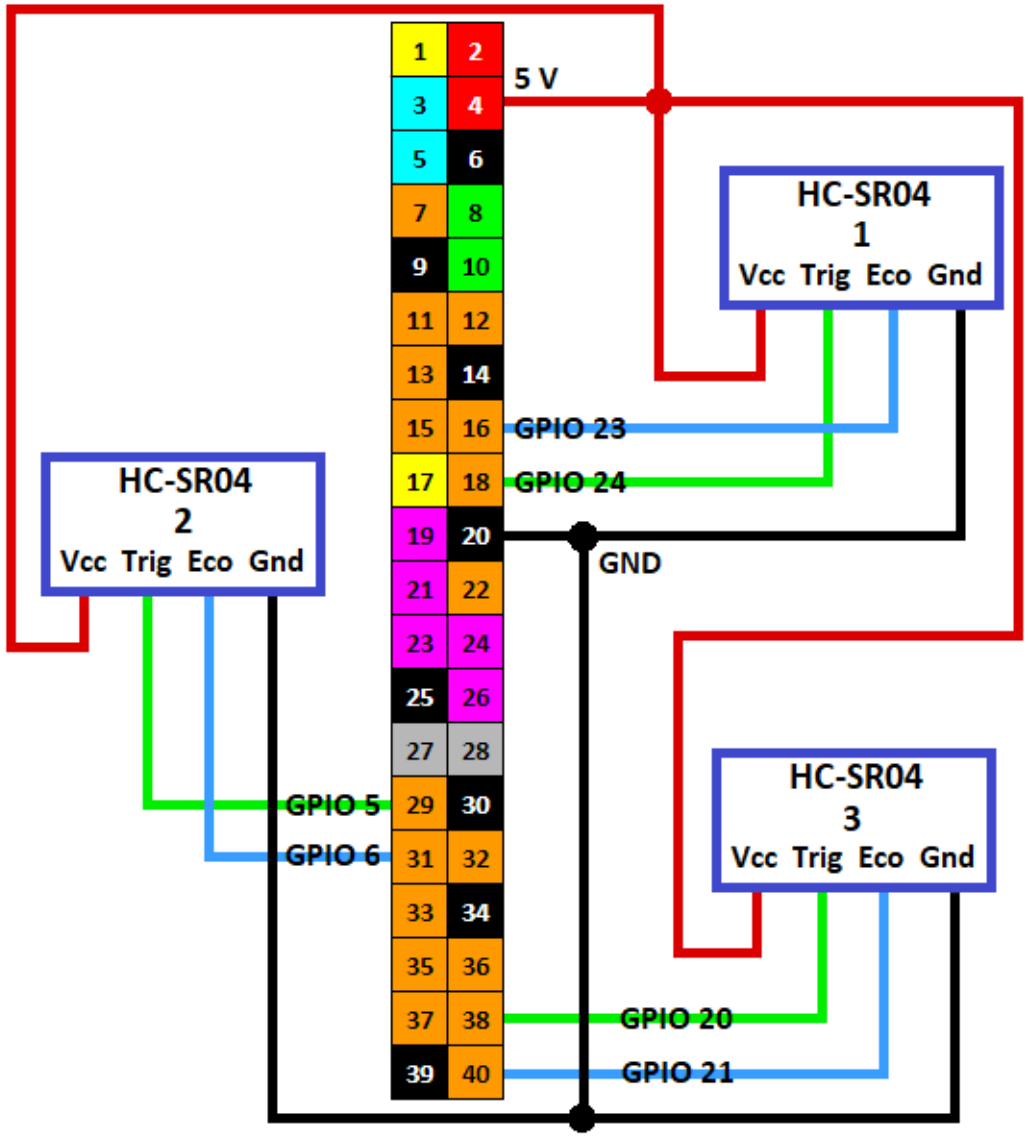

Figure 41: Esquema de connexió dels sensors d'ultrasons a la Raspberry. (Font pròpia)

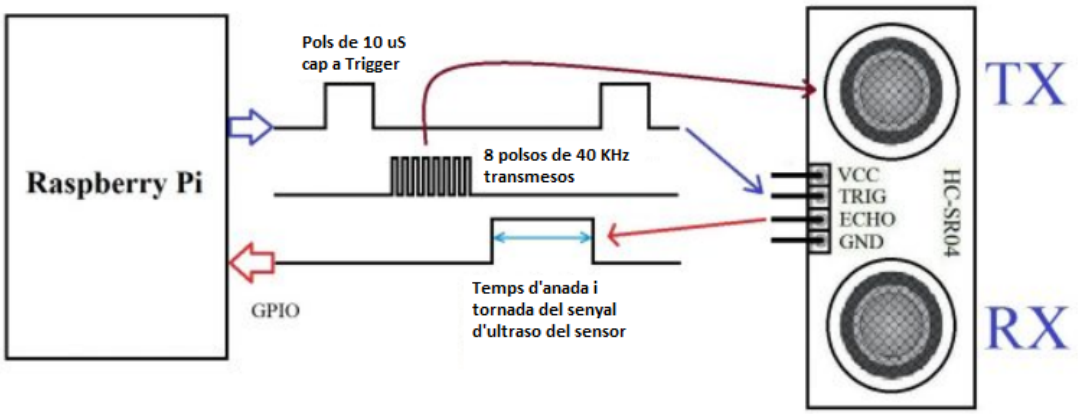

Figure 42: Pin-out i esquema de funcionament del HC-SR04.[6]

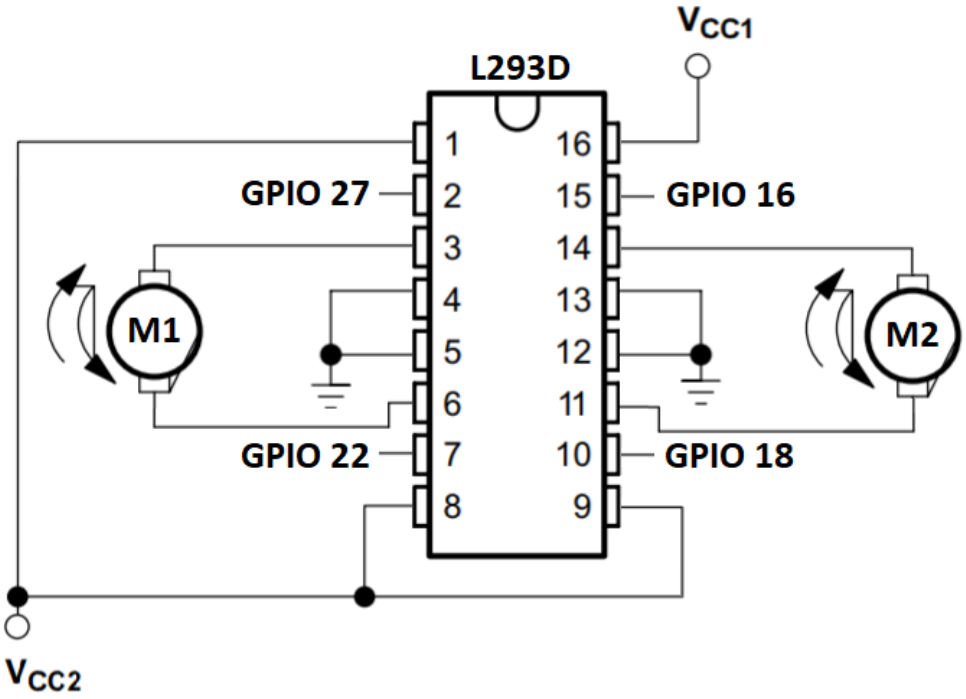

Figure 43: Esquema de connexió del driver L293D. (Font pròpia)

A.3 Models de les peces 3D

A continuació es presenten els models creats amb el programa Fusion 360 per imprimir les peces fetes amb impressió 3D.

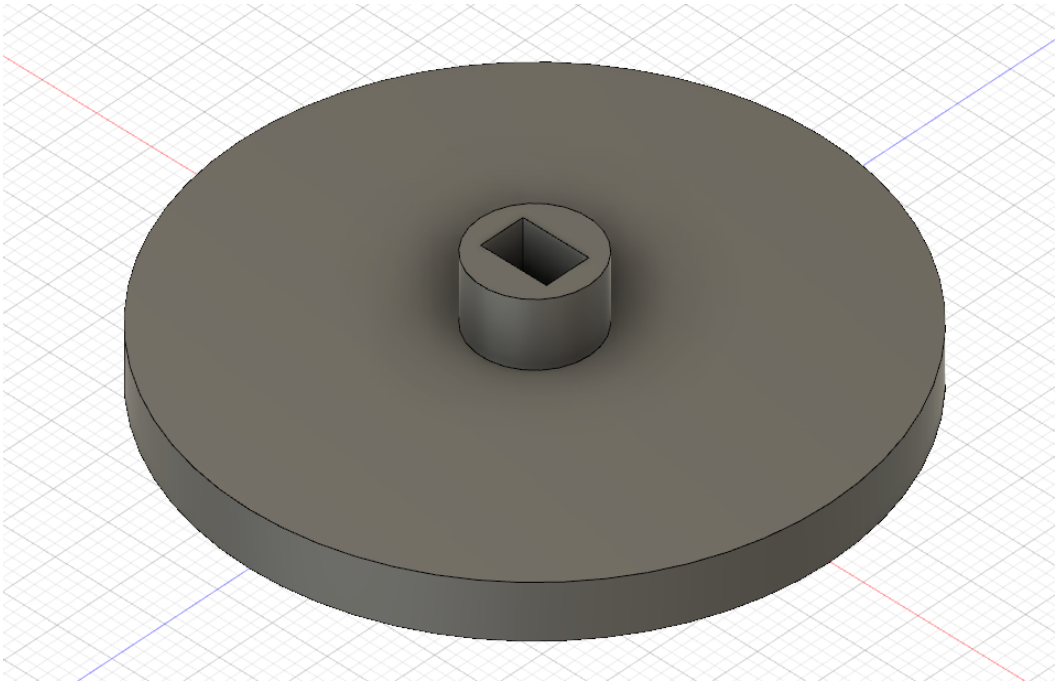

Figure 44: Model 3D de les rodes.

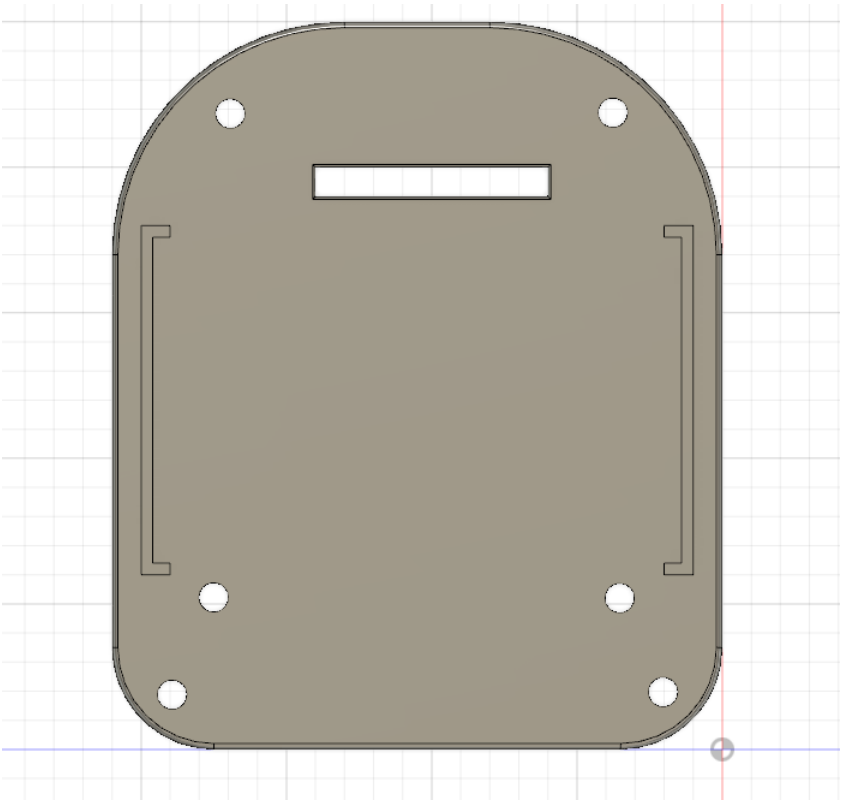

Figure 45: Model 3D de la placa superior, vista superior.

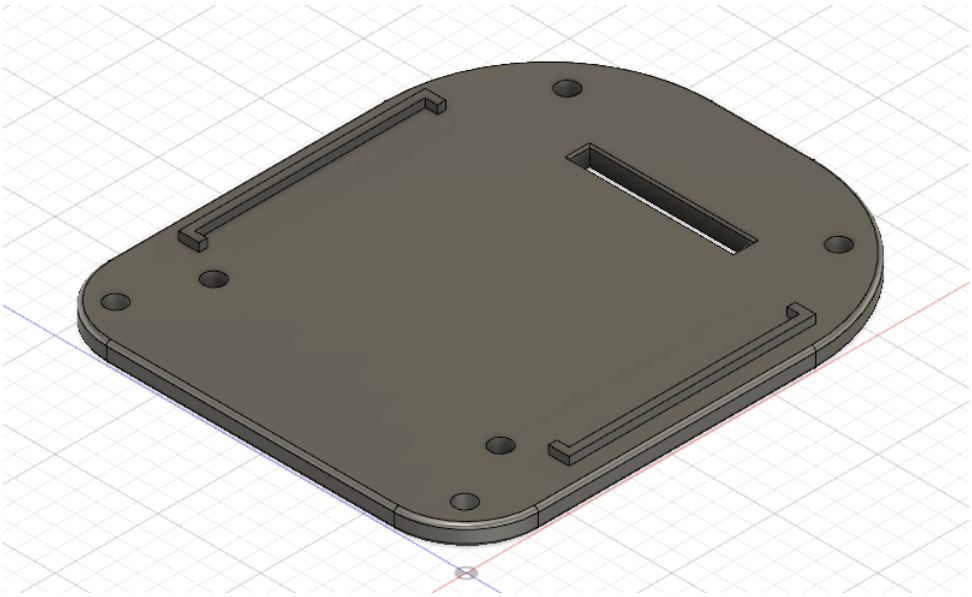

Figure 46: Model 3D de la placa superior, vista inclinada.

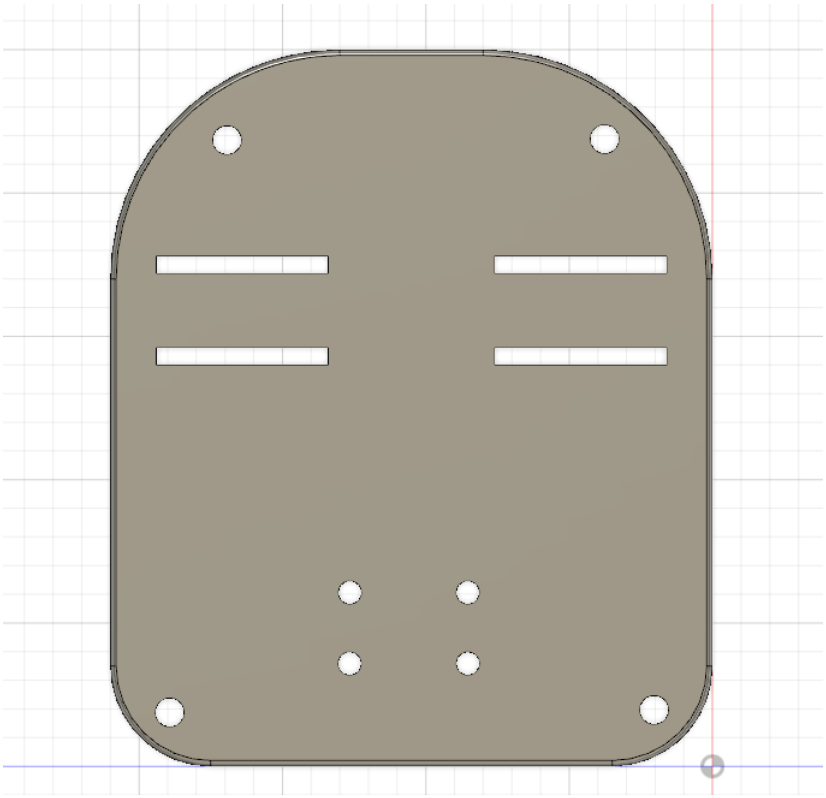

Figure 47: Model 3D de la placa inferior, vista superior.

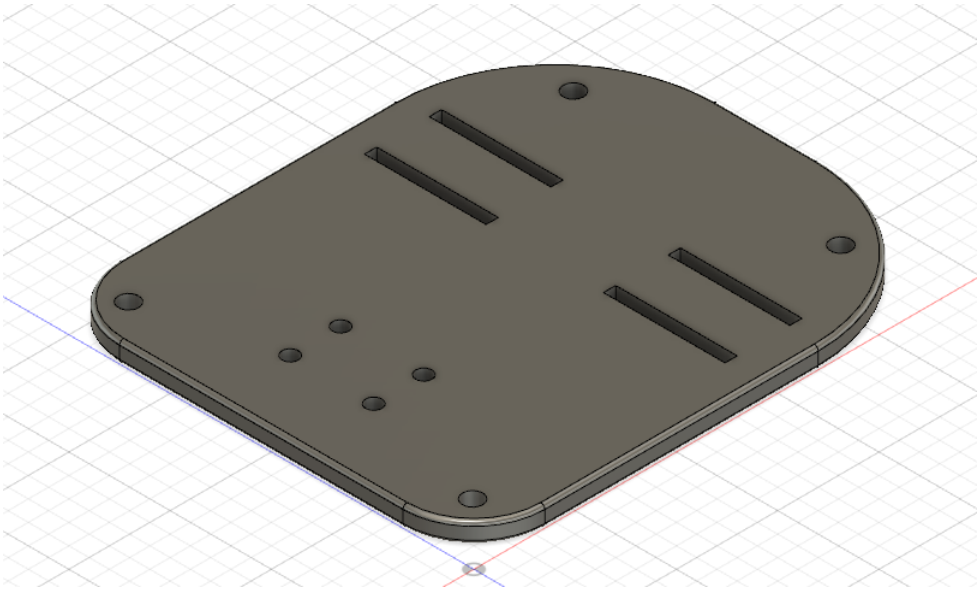

Figure 48: Model 3D de la placa inferior, vista inclinada.