UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

TREBALL FINAL DE GRAU

TÍTOL: Adaptació de kits d'aprenentatge de circuits electrònics a l'ensenyament d'enginyeria

TITULACIÓ: Grau en Enginyeria de Sistemes de Telecomunicació

AUTOR: Vicenç Nebot Parra

DIRECTOR: Francesc Josep Robert Sanxis

DATA DIPÒSIT: 03/07/2025

TÍTOL: Adaptació de kits d'aprenentatge de circuits electrònics a l'ensenyament d'enginyeria

TITULACIÓ: Grau en Enginyeria de Sistemes de Telecomunicació

AUTOR: Vicenç Nebot Parra

DIRECTOR: Francesc Josep Robert Sanxis

DATA DIPÒSIT: 03/07/2025

Resum

Moltes empreses dedicades al sector de fabricació de circuits electrònics compten amb kits complets amb els quals pots muntar diversos aparells electrònics, la qual cosa és molt adient per perfils de clients interessats a iniciar-se en el món de l'electrònica i el disseny de circuits integrats. La pregunta que se'ns obre a continuació és la següent, poden aquests kits d'aprenentatge ser vàlids per a un perfil d'estudiant més avançat i que busca un aprenentatge i anàlisi profunda en l'àmbit d'enginyeria?

L'objectiu d'aquest projecte és transformar un kit d'aprenentatge per a principiants en una eina d'ensenyament per a estudiants d'enginyeria, aplicant la metodologia de l'assignatura de Circuits i Sistemes Digitals, que consta en fer projectes amb 5 etapes diferenciades, especificacions, plantejament, desenvolupament, test i prototipatge. El material utilitzat, tant software com hardware, són recursos íntegrament disponibles per als alumnes a través de les llicències de la universitat o descàrrega gratuïta.

En aquest treball s'ha fet una anàlisi del kit d'aprenentatge escollit i s'han realitzat un seguit de projectes progressius llistats a continuació: Disseny d'un sensor de distància frontal, disseny d'un sensor de distància panoràmic, disseny d'un sistema d'evasió d'obstacles i el disseny d'un robot controlat remotament per Bluetooth. Tots aquests projectes junts són els que constitueixen el robot sencer.

Amb tots els recursos, explicacions i demostracions que s'aporten en aquest projecte, els estudiants d'enginyeria seran capaços de fer funcionar un sistema electrònic complex de manera didàctica. Una vegada executat el projecte, els alumnes seran capaços d'implementar més funcionalitats al robot, així com millorar les funcionalitats actuals.

TITLE: Adaptation of electronic circuit learning kits for engineering education

DEGREE: Bachelor's Degree in Telecommunications Systems Engineering

AUTHOR: Vicenç Nebot Parra

DIRECTOR: Francesc Josep Robert Sanxis

SUBMISSION DATE: 03/07/2025

Abstract

Many companies dedicated to the manufacturing of electronic circuits offer complete kits with which various electronic devices can be assembled. This is highly suitable for customer profiles interested in getting started in the world of electronics and integrated circuit design. The question that arises is the following: can these learning kits be valid for a more advanced student profile seeking deep learning and analysis in the field of engineering?

The aim of this project is to transform a beginner learning kit into a teaching tool for engineering students, applying the methodology used in the subject Digital Circuits and Systems, which involves completing projects through five distinct stages: specifications, planning, development, testing, and prototyping. The materials used, both software and hardware, are entirely available to students through university licenses or free downloads.

In this paper, an analysis of the selected learning kit has been carried out, and a series of progressive projects have been developed, listed as follows: design of a front distance sensor, design of a panoramic distance sensor, design of an obstacle avoidance system, and the design of a robot remotely controlled via Bluetooth. All these projects together constitute the complete robot.

With all the resources, explanations, and demonstrations provided in this project, engineering students will be able to operate a complex electronic system in a didactic way. Once the project is completed, students will be capable of implementing additional functionalities in the robot, as well as improving the existing ones.

Als meus pares i els meus germans que m'han ajudat durant tota la carrera.
A la meva parella per acompanyar-me sempre i batejar el robot com a *Wall-e.*
Al meu tutor del projecte per guiar-me en el procés.

ÍNDEX

ÍNDEX DE FIGURES

ÍNDEX DE TAULES

INTRODUCCIÓ

Moltes empreses dedicades al sector de l'educació STEM (Science, Technology, Engineering and Mathematics) desenvolupen i per tant disposen de kits d'aprenentatge i projectes d'exemple basats en plataformes com Arduino, Raspberry Pi, Micro:bit, entre d'altres. Aquests productes són dirigits a estudiants o aficionats de nivells i edats variades i consten tant del material *hardware* per construir l'esquelet del projecte, de la part *software* (el codi font) que controla la lògica del sistema muntat, com també un tutorial associat que explica detalladament com connectar cadascun dels cables perquè funcioni el dispositiu.
Aquests projectes són molt efectius i adequats per a un públic objectiu que simplement vulgui veure una màquina funcionar o inclús també pot servir com a porta d'entrada al món de l'electrònica i disseny de circuits integrats. Per tant, tenim en compte això, la pregunta que em porta a fer el projecte és la següent:

Són suficients els kits d'aprenentatge d'electrònica dissenyats per als aficionats quan es busca un aprenentatge i anàlisi profunda en l'àmbit d'enginyeria?

Tot i trobar-nos en un punt molt prematur del projecte la resposta és clara: No.
Aquests kits no compten amb un codi estructurat per parts que permeti l'anàlisi, optimització i per tant escalabilitat d'aquest, premisses fonamentals que s'han de complir si volem transformar un simple kit d'aprenentatge per a aficionats en una eina útil en l'ensenyament d'enginyeria, o simplement en un projecte d'enginyeria apropiat.

L'objectiu del sistema final és que els estudiants d'enginyeria puguin seguir els passos detallats en aquest projecte per aprendre i dissenyar-ne el seu.
Per tal d'assolir l'objectiu proposat em basaré en l'estructura i progressió que es presenta a Design of Electronic Equipment i DigSys (Departament d'Enginyeria Electrònica - UPC, n.d.). Començant per un projecte simple, a poc a poc afegint-li complexitat per a arribar finalment a les funcionalitats del kit d'aprenentatge, però ara sí, amb el rigor i precisió que exigeix l'ensenyament en l'enginyeria.

CAPÍTOL 1. KIT D'APRENENTATGE ESCOLLIT

1.1. Descripció general

El kit d'aprenentatge escollit és el *M2.0 Metal Chassis Mecanum Wheel Robotic (for Arduino Mega2560)*, de l'empresa Osoyoo (Osoyoo, 2022). Es tracta d'un cotxe petit amb diversos sensors, capaç d'executar vàries funcionalitats (les quals es poden consultar de manera més detallada en els següents punts). Controlable per Wifi o Bluetooth, pot seguir línies i evitar obstacles mitjançant girs gràcies a les *Mecanum wheels*, rodes molt populars que permeten un moviment omnidireccional del cotxe.

Aquest kit està dissenyat per ser compatible amb l'*Arduino Mega 2560*, una placa de desenvolupament molt popular per projectes d'electrònica on el nombre de pins emprats és més alt que en el cas de l'Arduino UNO.

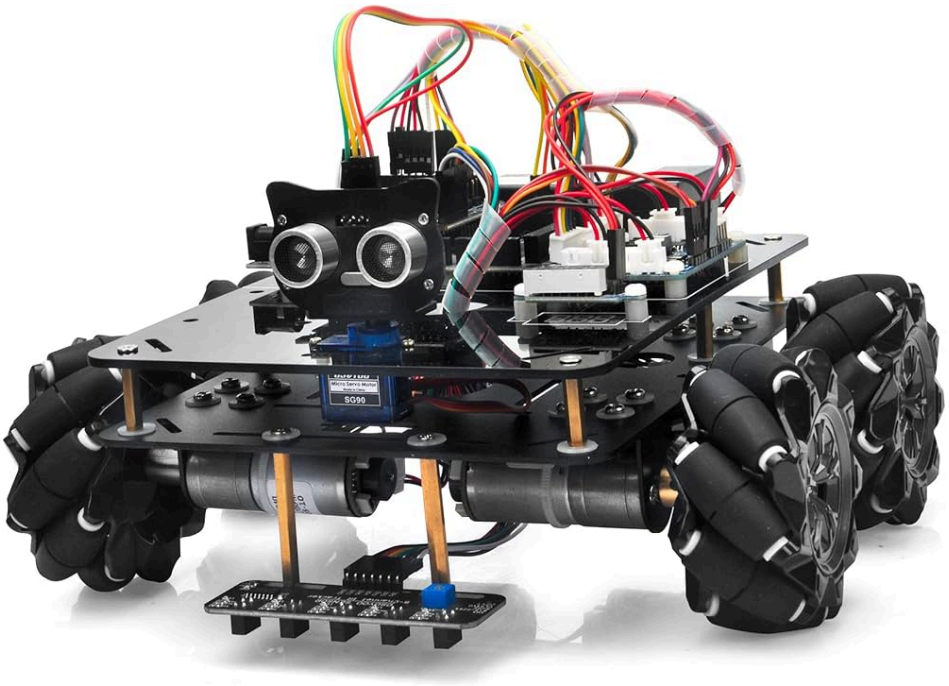

Figura 1.1. Visió general del robot

1.2. Funcionalitats del robot

El robot consta de diverses funcionalitats que es presenten com a mòduls independents, els quals tenen el seu tutorial i codi dedicat, en aquesta secció veurem una per una les funcionalitats que cada *lesson* implementa, recordem que el primer objectiu que tenim en aquest projecte és assolir aquestes funcionalitats adaptades a l'ensenyament en enginyeria.

1.2.1. Moviment omnidireccional

En aquesta primera *lesson* es tracta l'aspecte més fonamental del cotxe robot, més enllà del seu muntatge bàsic, aquesta primera part té l'objectiu de demostrar a l'usuari la varietat de moviments que pot executar, així com moviments rectes, en diagonal, girs en totes les direccions, cap endavant i cap enrere (Generation Robots, n.d.). La sorprenent quantitat de moviments que és capaç d'efectuar venen donats gràcies a un component ja mencionat amb anterioritat, es tracta de les enginyoses Mecanum Wheels, en el següent subapartat entendrem com funcionen.

Amb relació a la part software d'aquesta funcionalitat no consta amb una estructura sòlida amb estats eficientment ordenats, simplement executa les funcions de gir, avenç i marxa enrere amb un *delay* entre cadascuna.

1.2.1.1. Mecanum Wheels

Les Mecanum Wheels són un tipus de rodes especialment dissenyades per a permetre un moviment omnidireccional sense necessitat de girar el vehicle (Taheri & Taghirad, 2015). Concebudes l'any 1973 arriben als anys contemporanis amb protagonisme en la majoria de robots dissenyats per fer moviments d'aquest tipus, ja que permet el desplaçament lateral, diagonal i rotacional. Aquestes rodes són molt útils per a robots que necessitin desplaçaments precisos en espais petits, pel fet que cada roda està controlada de manera independent per un motor, això permet un moviment àgil sense la necessitat d'executar maniobres complexes.

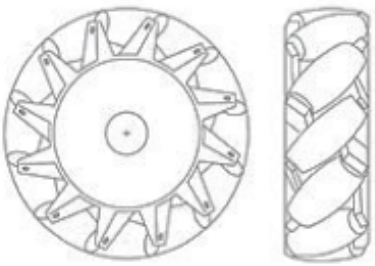

Figura 1.2. Mecanum Wheels

1.2.1.2. Mecanum Wheels: Com funcionen?

Aquestes rodes estan constituïdes per una sèrie de rodets en un eix desplaçat 45° respecte a l'eix de rotació de la roda, com efectivament es pot veure a la figura 1.2 (Grabowiecki, n.d.), l'angle mencionat és imprescindible perquè la suma de les forces entre totes les rodes ocasioni el moviment desitjat. Aquests rodets són en tot moment el punt de contacte de la roda amb el terra.

Per entendre correctament aquesta curiosa arquitectura tractarem cada roda com a un vector de força independent a les altres, d'aquesta manera es podrà veure clarament i entendre les

forces resultants en cada cas. A partir d'aquest punt utilitzarem la següent nomenclatura per a referir-nos a cadascuna de les rodes:

- FR (roda frontal dreta, *front right)*
- FL (roda frontal esquerra, *front left)*
- RR (roda del darrere dreta, *rear right)*
- RL (roda del darrere esquerra, *rear left)*

Per a referir-nos als vectors utilitzarem la lletra V_{xy}, *on xy* serà la referència a la roda corresponent, tal com acabem de definir.

Per tant, ara sí que podem representar els possibles moviments que ens poden donar aquestes quatre rodes. És important destacar que cada roda que fa una força, sigui cap endavant o cap enrere, produirà dos vectors de força, això és degut a l'angle del rodet, que com hem dit abans hi juga un paper molt rellevant.

Evidentment en cada cas, la força total i per tant el moviment resultant que el cos experimentarà (és a dir el robot), serà causat per la suma, constructiva o destructiva, de totes les forces aplicades per les rodes (la suma de tots els vectors). Per qüestions de claredat en la representació, els vectors que es troben en direcció vertical i s'apunten entre ells, no els he posat sobre una mateixa recta, amb l'objectiu d'evitar la sobreposició entre ambdós, aquesta correcció és notable en la figura 1.4, consideri que els vectors es troben sobre la mateixa recta.

Atès que estem treballant en dues dimensions definirem els eixos per poder referir-nos-hi amb precisió

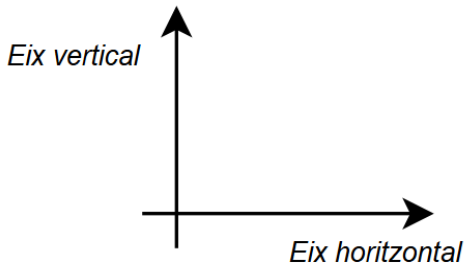

Figura 1.3. Eixos de referència

- Desplaçament cap endavant o cap enrere: Totes les rodes faran força cap endavant o cap enrere, tal com es mostra en la figura 1.4 Trobem una suma destructiva de totes les forces en l'eix horitzontal, succeeix el contrari en l'eix vertical.

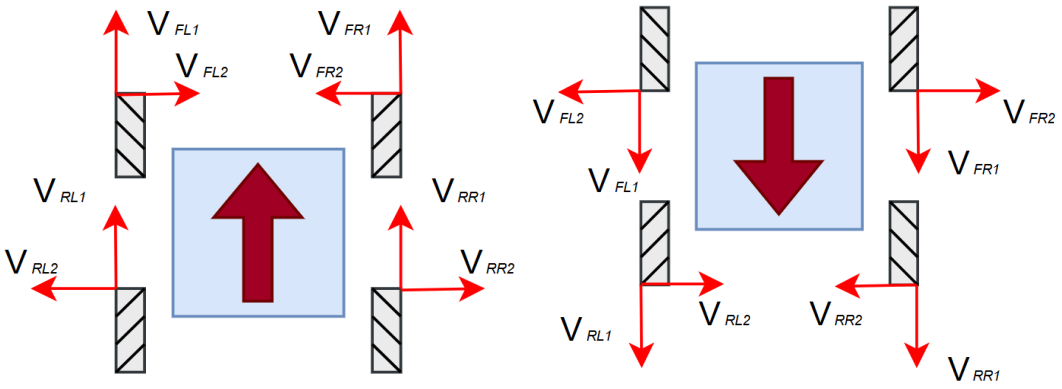

Figura 1.4. Desplaçament cap endavant i cap enrere

- Desplaçament lateral: Les rodes de cada costat donen potència en sentit contrari, en el cas del moviment cap a la dreta veiem com les rodes de la dreta fan força contrària, una cap a l'altra, mentre que les rodes de l'esquerra intenten separar-se. En el cas del moviment cap a l'esquerra és al revés, com es pot apreciar en la figura 1.5. Trobem una suma destructiva de totes les forces en l'eix vertical, a diferència de l'eix horitzontal.

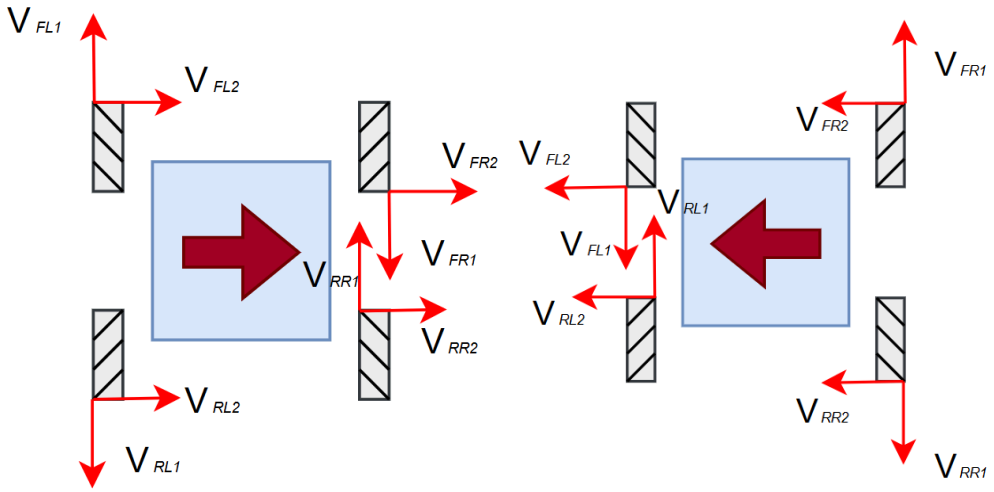

Figura 1.5. Desplaçament cap a la dreta i esquerra

- Desplaçament en diagonal: Només dues de les rodes fan força, una de l'eix davanter i una altra de l'eix del darrere, no seran mai del mateix costat. Es pot apreciar a la figura 1.6. Trobem una suma constructiva en ambdós eixos, cosa que produeix una força amb un angle de $(45 + n \cdot 90)°$ respecte a l'eix horitzontal, on n pren valors de 0 fins a 3.

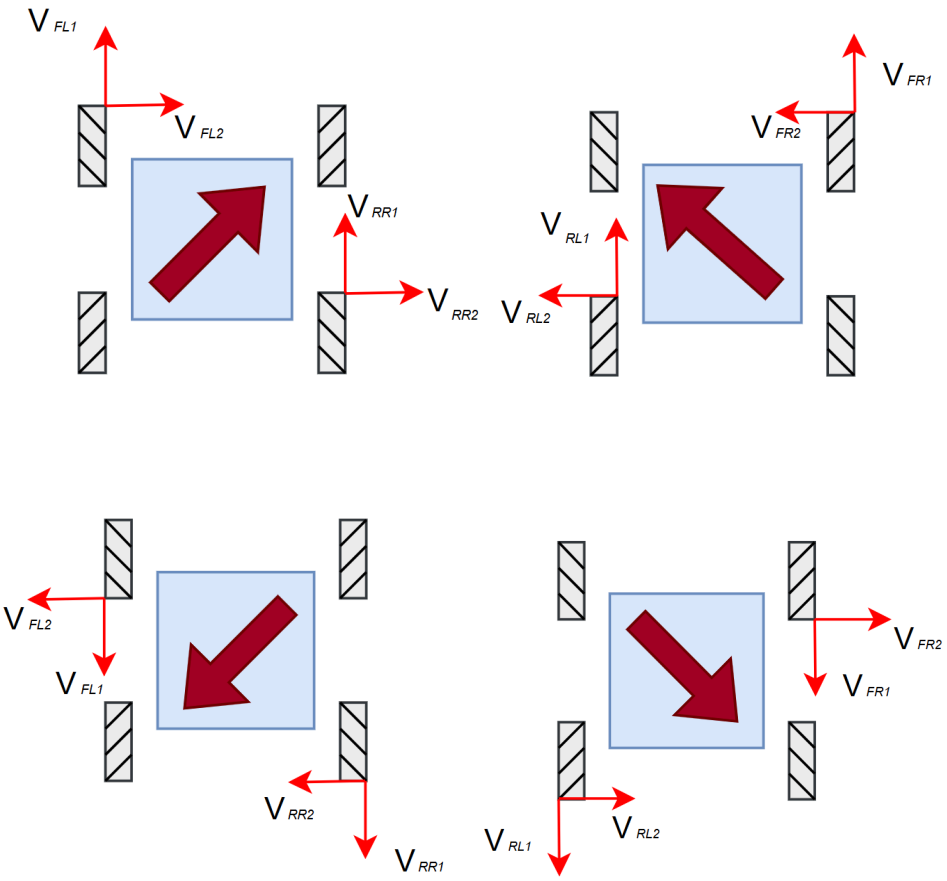

Figura 1.6. Desplaçaments en diagonal

- Gir sobre el mateix eix: En aquest cas s'activen totes les rodes, les rodes de cada costat fan força juntes cap a la mateixa direcció, però en direcció contrària a les rodes del costat contrari, es pot veure en la figura 1.7. Trobem només sumes constructives, però en diferents sentits, això fa que el robot giri sobre el seu eix.

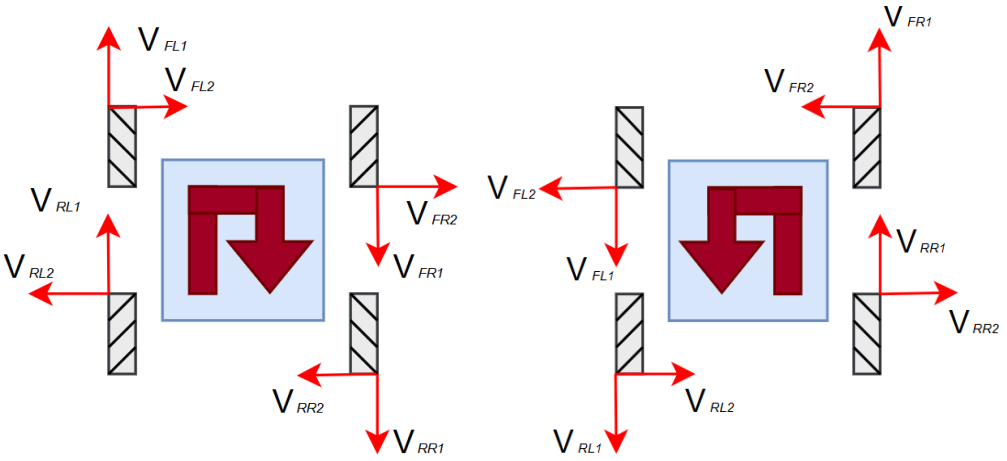

Figura 1.7. Desplaçament de rotació sobre el propi eix

1.2.1.3. Motors DC

Per transformar els impulsos de tensió en parell motor, de manera independent per a cada roda necessitem quatre motors connectats cadascun a una roda diferent, en aquest cas el component utilitzat és el *DC Encoder Motor Robotic Car Speed Encoder for Arduino Raspberry Pi Platform DIY (model 2016012200)*, mostrat en la figura 1.8.

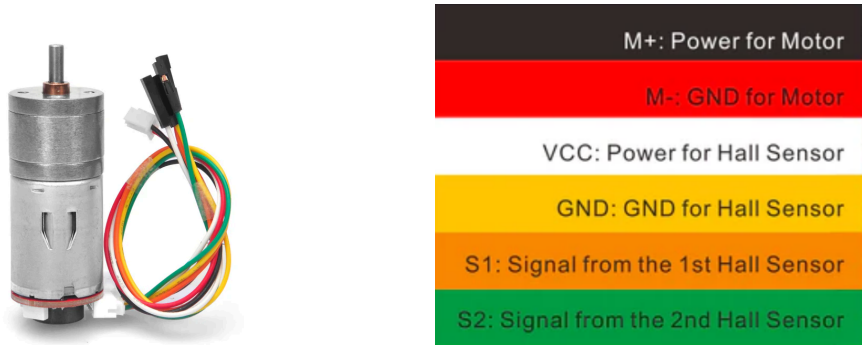

Figura 1.8. DC Encoder Motor Robotic Car Speed Encoder for Arduino

En aquest component hi trobem 6 cables d'entrada, per aquesta funcionalitat només en darem servir dos d'ells, el vermell i el negre, encarregats de donar-li la tensió necessària al motor.

Per a transformar l'energia en forma de tensió a parell motor i per tant poder moure la roda en qüestió, aquest motor usa un mecanisme d'atracció i repulsió de camps magnètics. Per simplificar-ho en dues parts, trobem els estators i els rotors, aquests últims, com el seu nom indica, s'orienten en funció del camp magnètic que els atrau a causa de l'activació de bobines situades en els estators. Aquest efecte de manera continuada provoca que els rotors girin i per tant transfereixin aquest gir a la roda (Zaragoza Maker Space, n.d.). Depenent de quants rotors/estators tingui el motor i l'alimentació que es dugui a terme podrà comptar amb una resolució de rotació major o menor.

Els cables de color blanc, groc, taronja i verd serveixen per determinar la posició de la roda, en el cas d'aquest motor té 150 polsos per volta, é a dir que cada vegada que el motor rep un impuls la roda rota 2,4°, ho podem demostrar amb aquesta senzilla fórmula:

$$Resolució\ de\ rotació\ (°) = \frac{360\ °/volta}{Nombre\ de\ pulsos\ /volta} \quad (1.1)$$

Un dels cables que agafa el senyal provinent del sensor Hall determina si s'ha mogut respecte a l'estat anterior, mentre que l'altre serveix per determinar el sentit de rotació, horari o antihorari, de la roda.

1.2.1.4. Driver

Per al control dels quatre motors es necessita un driver que pugui controlar la direcció, sentit o velocitat d'aquests.
En aquest cas s'opta per utilitzar el 4-Channel H-Bridge Stepper Motor Module Y 2.0 d'Osoyoo, que actuarà d'intermediari entre els motors i l'Arduino.

Aquest driver és capaç de controlar dos *stepper motors* de manera independent o quatre motors DC també de manera independent, precisament aquesta és la característica més fonamental, ja que el nostre projecte consta de quatre rodes controlades de manera independent per quatre motors DC diferents. També és notable en el nom del component la característica H-bridge.

H-bridge és un tipus de circuit electrònic que es caracteritza per canviar la polaritat d'una tensió aplicada a una càrrega, per introduir-ho de manera senzilla podem veure com funciona en les següents figures (Colvin, n. d.).

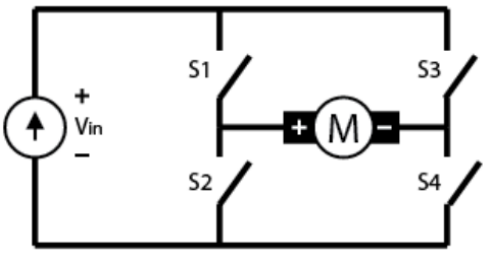

Figura 1.9. Circuit H-bridge

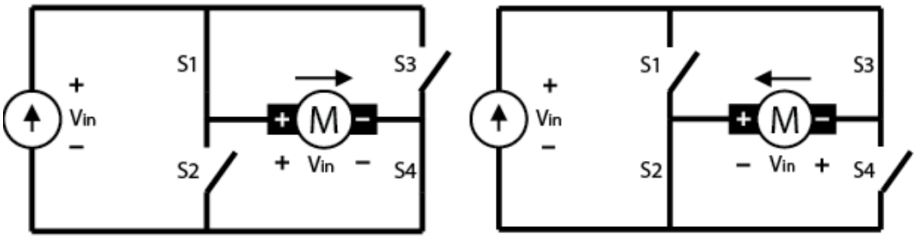

Figura 1.10. Circuits H-bridge polaritzats

Veiem com aquest circuit format per quatre interruptors i connectat a una font de tensió té dos casos d'ús possibles. En el primer cas veiem com els switches S1 i S4 es tanquen a la vegada, això provoca una tensió aplicada a la càrrega, en el moment en que es passi d'aquest estat a obrir els switches S1 i S4 i tancar els switches S2 i S3, immediatament la tensió és revertida i per tant permet al motor operar al revés.

La diferència entre el bloc que utilitzem i el driver tradicional L298 és la quantitat de *DC Motors* que són capaços de controlar a la vegada de manera independent, sent quatre i 2 respectivament. Podem imaginar l'esquema del driver d'*Osoyoo* com a dos L298, esquema del qual podem apreciar a la figura 1.11 (STMicroelectronics, n.d.).

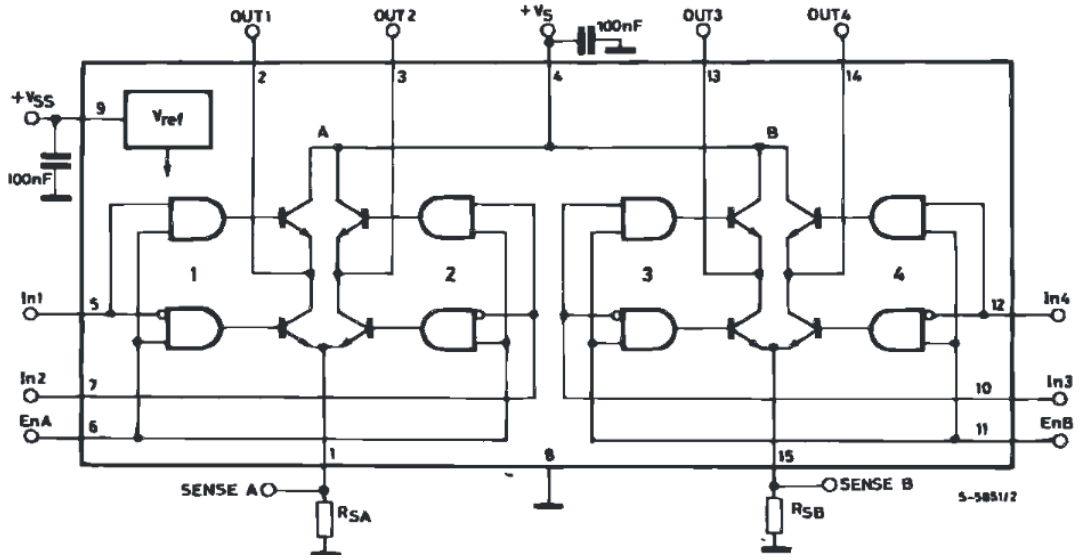

Figura 1.11. L298, esquema del circuit

També és important fer una descripció a efectes pràctics dels ports que té el driver amb el que tractem i les seves funcions. A continuació es pot apreciar una figura representativa.

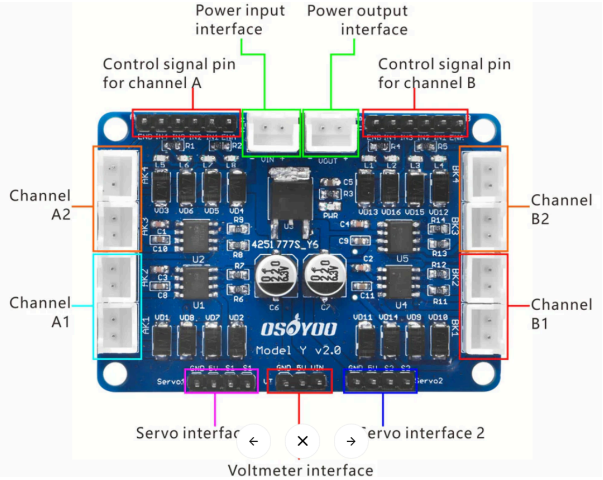

Figura 1.12. 4-Channel H-Bridge Stepper Motor Module Model Y 2.0

1.2.2. Evitar obstacles

En la segona lliçó trobem l'implementació de la funcionalitat d'evitar obstacles. Si volem que un cotxe robot sigui autònom la primera premissa que hem de complir és que sigui capaç de prendre les decisions correctes, en quant a moviment es refereix, en funció dels obstacles o espais lliures que tingui al seu voltant. Per tant la incorporació d'un sensor que permeti processar les dades sobre els diferents obstacles existents és imprescindible, ja que depenent de les dades rebudes el robot podrà, mitjançant software, aprofitar les seves capacitats de moviment i decidir la direcció a on avançar.

1.2.2.1 Sensor ultrasònic

Per implementar la funcionalitat tractada en aquest apartat necessitem l'ús de l'*Ultrasonic sensor module for Arduino V2.1 Robot Car*, mostrat en la figura següent.

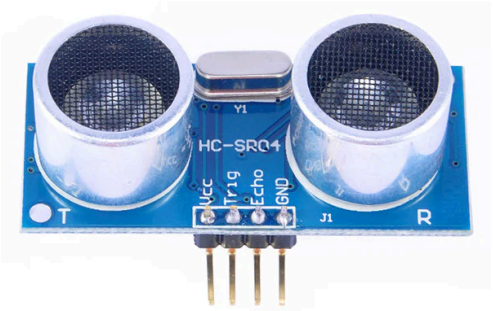

Figura 1.13. Ultrasonic sensor module for Arduino V2.1 Robot Car

El mòdul té quatre pins, dos d'ells són l'alimentació i GND, els altres dos són el pin de Trigger amb el que rep les ordres d'enviar el senyal principal i el pin Echo, on retorna el senyal resultant. Aquest sensor envía una serie pulsos ultrasònics a una freqüència de 40 kHz, quan rep l'eco d'aquestes senyals calcula la diferència temporal que existeix entre que envia la senyal principal i rep l'eco, aquest sensor retorna una senyal ja digital, representada amb l'amplada d'un pols TTL (ETC2, n.d.).

És important ressaltar i entendre, per tots els sensors que implementem en aquest projecte, la importància del format de dades que el sensor ens està enviant, és a dir, si el sensor ens està enviant dades digitals o analògiques, amb la primera opció (el nostre cas) no farà falta convertir el senyal, mentre que si es vol conectar un sensor amb sortida de senyal analògica serà necessari aplicar un *Signal Conditioner*, que pugui convertir el senyal a digital, tenint en compte dades com la resolució del sensor, amplada de dades recollides i la tensió subministrada.

1.2.3. Control remot per Bluetooth

Per a dur a terme aquesta funcionalitat serà imprescindible la utilització del mòdul Bluetooth que tenim a la caixa del kit d'electrònica, ja que adjunt a l'Arduino, serà el terminal que ens permetrà rebre les dades que enviem des d' un terminal extern.

En el nostre cas utilitzem el mòdul HC-01, molt similar al popular HC-05. Aquest mòdul consta de 6 pins, 2 dels quals s'encarreguen de la transmissió i recepció de dades amb l'Arduino (TXD i RXD), l'STATE per a detectar l'estat d'emparellament Bluetooth i el KEY, que indica el mode del mòdul, els 2 restants són el Vcc i GND. Seguidament es pot veure una figura il·lustrativa del mòdul que s'utilitzarà.

Figura 1.14. Mòdul HC-01

1.2. Metodologia i recursos

El projecte a desenvolupar, es durà a terme de manera semblant a la que l'empresa fabricant del kit utilitza amb les *lessons*. Es realitzarà a partir de capítols diferenciats, que representaran les fases del disseny, primerament amb implementacions més senzilles i en funció dels resultats anar avançant a poc a poc cap a dissenys un pèl més complexos que l'anterior. D'aquesta manera aconseguim una progressió molt bona i educativa, perfecte per a l'objectiu clar que tenim, aconseguir que els alumnes d'enginyeria puguin aprendre conceptes complexos a partir d'un kit d'aprenentatge bàsic.

1.3.1 Software i simulacions

La validació de cada part vindrà argumentada amb simulacions. Per aquest projecte necessitem utilitzar vàries eines de simulació, sempre tenint en compte que es tracta d'un projecte que els estudiants de la universitat han de poder seguir. Per tant, s'utilitzaràn eines accessibles per als alumnes, i en conseqüència gratuïtes o amb llicències disponibles per ells mitjançant la universitat.
Proteus és el software elegit per a comprovar el circuit electrònic que volem implementar, així com ens permet incorporar el codi programat per controlar l'Arduino. També necessitem eines per tal de dissenyar els circuits en plaques de circuit imprès (PCB) i la seva implementació física en 3D, per això utilitzarem KiCad. Addicionalment haurem de programar una aplicació per a fer possible la comunicació d'un *smartphone* amb el mòdul Bluetooth *HC-01*, per aquest fi emprarem la plataforma *MIT App Inventor*, ja que proporciona un entorn molt amigable i no és necessari tenir coneixements previs de programació d'aplicacions.

1.2.2 Programació d'una Finite State Machine

Aquest projecte es farà mitjançant el disseny i programació d'una Finite State Machine, un enfocament estructurat per a implementar un sistema capaç d'estar en un de diversos estats, i que pugui saltar entre ells arran d'events o condicions. Aquest tipus de programació es caracteritza pels següents punts:

- Conjunt finit d'estats: Una FSM ha de contenir un nombre finit d'estats, cadascun representa una funció específica dins el sistema. Només pot estar en un estat a la vegada. Això permet que el codi sigui fàcil d'entendre i evita errors.

- Conjunt finit d'esdeveniments: Ha d'haver-hi un conjunt d'esdeveniments finit que desencadenin el canvi d'un estat a l'altre, aquestes condicions poden ser tan externes com internes.

- Estat inicial ben definit: La *Finite State Machine* ha de comptar amb un estat inicial ben definit, en el que es troba en encendre's, això fa que la FSM comenci a funcionar en una condició estable i tingui un comportament predicible.

- Maneig d'interrupcions: Una FSM és capaç de tractar amb interrupcions, redirigint el flux del codi, per després tornar al punt on es trobava.

L'estructura del codi d'una FSM és semblant a la que ens trobem per defecte per programar un Arduino amb les funcions *setup()* i *loop()*, tenim dos blocs clarament diferenciats, el nostre codi en tot moment tindrà aquesta estructura:

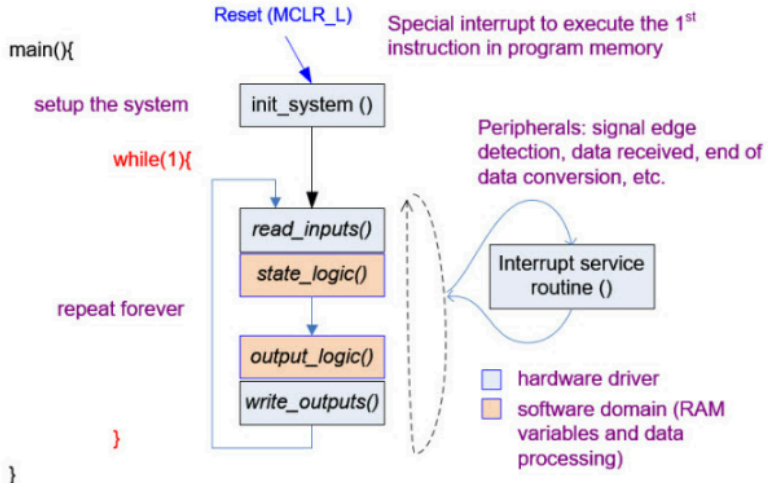

Figura 1.15. Esquema de Finite State Machine

- *Init_system()*: Configuració inicial de la màquina, en aquest punt s'assignen i configuren els pins del microcontrolador, s'inicialitzen les variables i els components externs al microcontrolador, aquesta funció s'executa una vegada. L'única manera que tenim de tornar a executar aquesta funció és fent un reset del sistema.
- *Infinite_loop()*: Aquesta funció és la que conté les cinc funcions que caracteritzen la FSM i el seu comportament.
- *Read_inputs()*: És la funció que s'encarrega de llegir o mostrejar els diferents nivells digitals de tensió a l'entrada dels pins definits com a *inputs*.
- *State_logic()* és la funció que determina la dinàmica dels estats de la màquina, és a dir, cap a quin estat saltarà, o no, la màquina en funció de certes condicions determinades.
- *Output_logic()* conté la lògica de cada estat, el que succeeix en cada estat.

- *Write_outputs()* escriu les variables als pins de sortida, tradueix les variables a nivells digitals de tensió.
- *Interrupt_service_routine()* s'encarrega de gestionar les interrupcions provocades.

Després que s'executi l'*Infinite_loop*, tal com el seu nom indica, es tornarà a executar.

1.3. Plataforma amb les que es treballa

El kit d'aprenentatge amb el qual estem treballant ve implementat en un Arduino Mega 2560. Primerament aquest fet pot semblar un problema, ja que els estudiants de l'EETAC, objectiu final d'aquest projecte, no tenen habilitada la llicència per simular aquest microcontrolador, a diferència de l'Arduino UNO.
Això ens porta a preguntar-nos quines són realment les diferències i què haurem de tenir en compte a l'hora d'adaptar el projecte a l'Arduino UNO (Gudino, 2021).

1.3.1 Arduino Mega 2560

- Microcontrolador: ATmega2560
- Tensió d'alimentació: 5V
- Tensió d'entrada recomanada: 7-12V
- Pins digitals: 54 (15 amb capacitat PWM)
- Pins d'interrupció: 6
- Entrades analògiques: 16
- Memòria flash: 256 KB
- SRAM: 8 KB
- EEPROM: 4 KB
- Velocitat de rellotge: 16 MHz
- Dimensions: 101.52 mm x 53.3 mm
- Pes: 37 grams

Disponible un diagrama de pins de l'Arduino Mega 2560 a l'annex 2.

1.4.2 Arduino UNO

- Microcontrolador: ATmega328P
- Tensió d'alimentació: 5V
- Tensió d'entrada recomanada: 7-12V
- Pins digitales: 14 (6 amb capacitat PWM)
- Pins d'interrupció: 2
- Entrades analògiques: 6
- Memòria flash: 32 KB
- SRAM: 2 KB
- EEPROM: 1 KB
- Velocitat de rellotge: 16 MHz

- Dimensions: 68.6 mm x 53.4 mm
- Pes: 25 grams

Disponible un diagrama de pins de l'Arduino UNO a l'annex 1.

1.4.3 Diferències

Per què és millor utilitzar l'Arduino Mega 2560 abans que l'Arduino UNO per al nostre projecte? Diverses raons ens porten a pensar-ho, però la principal és la quantitat de pins digitals que té l'Arduino MEGA 2560. Aquesta última compta amb 54 pins digitals, a la vegada que amb 15 pins PWM. En un projecte d'un robot on es busca que les rodes puguin funcionar de manera independent perquè sigui escalable, són necessaris molts pins digitals per a controlar tots els elements perifèrics que existeixen al sistema.
Amb l'Arduino UNO no tenim prou pins com per connectar-los tots, i per qüestions de limitacions del hardware (en aquest cas del controlador) no és recomanable connectar dos motors al mateix canal de sortida per dos motius que s'explicaran a continuació. El primer motiu és el més trivial, si connectem les rodes per parelles al mateix canal, perdrem possibles moviments que ens poden interessar en alguna versió millorada del projecte. El segon motiu és que el fabricant ens posa un límit de corrent a cada canal, per tant si connectem dos motors al mateix canal correm el risc, encara que no estiguin treballant a la màxima potència, que s'encallin les rodes i malmetre el hardware.
També cal considerar la quantitat de ports sèrie (UART) que té cadascuna, ja que ens interessa tenir-ne més d'una parella, per monitorar totes les funcions alhora.

Altres avantatges com per exemple la diferència de memòria Flash, RAM o EEPROM també són significatives, ja que si l'objectiu és escalar aquest projecte, és interessant que pugui executar lògica més complexa o llibreries més pesades sense que es vegi sobreestimulat.

En definitiva, per les simulacions amb el *software Proteus*, s'utilitzarà l'Arduino UNO com a exemple i es podrà plasmar perfectament a l'Arduino MEGA2560 que tenim al kit d'aprenentatge, però a mesura que anem afegint-li complexitat al projecte haurem d'anar adaptant la versió simulada a l'Arduino UNO a l'Arduino MEGA 2560, ja que del contrari no serà possible implementar la totalitat del projecte a l'Arduino UNO. A l'Annex 1 i 2 es poden trobar els diagrama de pins de l'Arduino UNO i de l'Arduino MEGA 2560, també es podrà apreciar el diagrama de connexions per aquesta última.

CAPÍTOL 2. ROBOT AMB EVASIÓ D'OBSTACLES

En aquest apartat partirem amb l'Arduino UNO, implementarem un sensor ultrasònic per a detectar la distància als obstacles existents. La metodologia serà per passos, és a dir, primer implementarem un simple sensor frontal i seguidament un sensor que sigui capaç d'escanejar els 180°.

2.1. Sensor frontal

2.1.1. Especificacions

En ambdós casos el sistema funcionarà gràcies a una variable creada anomenada *var_SP_flag,* aquesta variable representa l'ordre de prendre una mostra, ja que està fortament relacionada amb el període de mostreig establert, de moment de manera arbitrària.

var_SP_flag = 0 → No ha passat el període de mostreig, no es pren mesures.
var_SP_flag = 1 → Ha passat el període de mostreig, ordre de prendre mesures.

També s'ha d'implementar un botó que ens permeti engegar el sistema sense que comenci a prendre mesures, és a dir, que es quedi a l'estat *Idle* fins que aquest botó s'activi. Es tracta del botó *Start*, pel qual crearem la variable *var_ST_flag* .

var_ST_flag = 0 → Es manté a l'estat *Idle*
var_ST_flag = 1 → Comença a recórrer els estats

Les dades recollides pel sensor ultrasònic es podran apreciar en el *Serial monitor* de l'Arduino. La implementació tindrà la següent arquitectura:

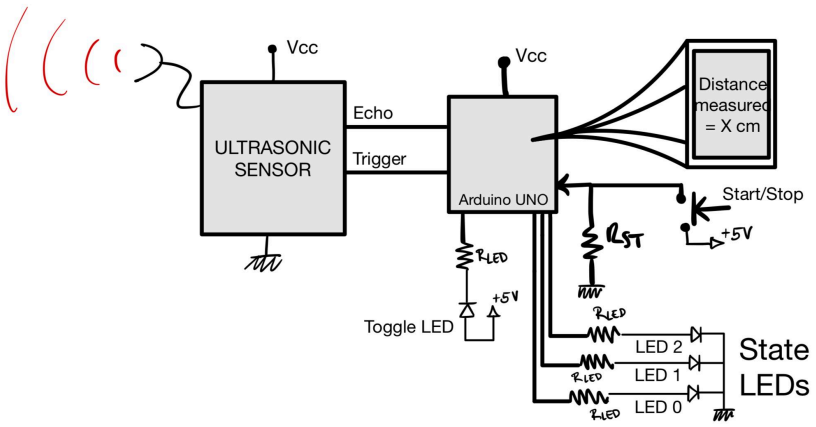

Figura 2.1. Esquema sensor ultrasònic frontal

Seguidament podem veure un diagrama de temps sobre l'actuació de les dues variables introduïdes, també es pot apreciar el Toggle LED, tal com es veu al diagrama, aquest LED s'encén i s'apaga de manera intermitent, segons el període de mostreig.

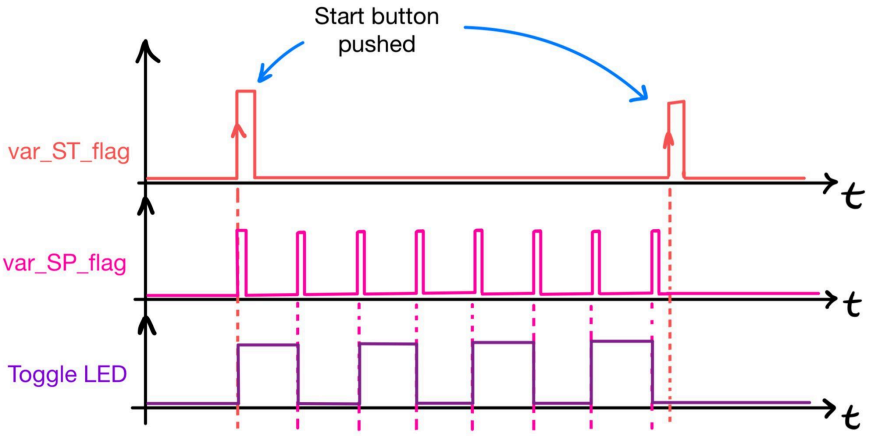

Figura 2.2. Diagrama de temps de les variables del sensor frontal

2.1.2. Planificació del projecte

2.1.2.1. Circuit de l'Arduino

Per implementar el sensor frontal necessitem programar l'Arduino amb la metodologia de Finite State Machine. Per dur a terme això necessitem primerament concebre tots els estats que necessitem per representar tots els processos a realitzar. A continuació s'ha sintetitzat una versió de funcionament del tipus *FSM*, adaptat al projecte amb l'Arduino.

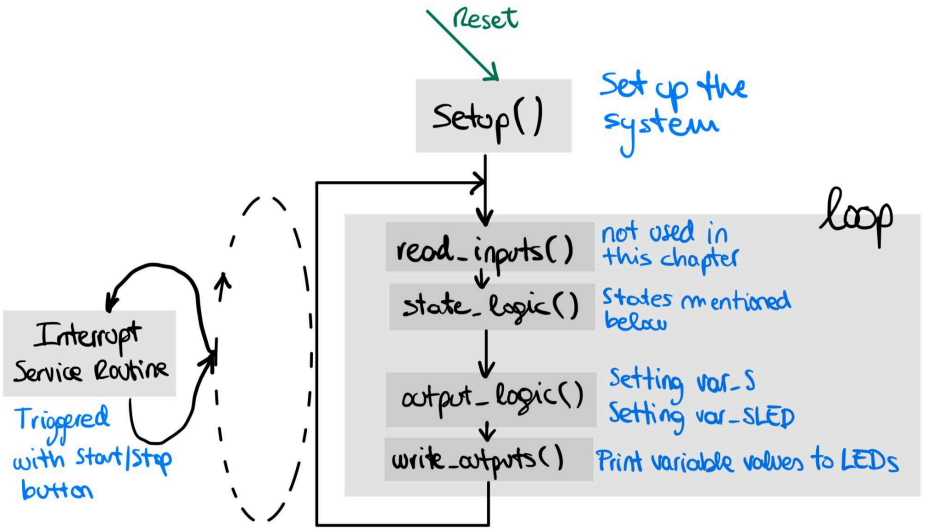

Figura 2.3. FSM adaptada a l'Arduino.

El diagrama d'estats és el que es mostra a la figura 2.4. Tenim un total de 6 estats, explicats i enumerats a continuació.

- Idle: Estat en el qual s'inicia el sistema després d'una inicialització general, espera a que la variable *var_ST_flag* s'activi.
- Set up acquisition timer: Estat per establir el període de mostreig, és a dir, la freqüència amb la qual es prenen mesures.
- Do nothing: Estat per esperar que la variable *var_SP_flag* sigui 1, en cas que no s'activi es manté en aquest estat.
- Measure: S'ordena al sensor ultrasònic que obtingui dades, es calcula la distància a la qual està l'obstacle.
- Transmit: S'envien les dades obtingudes al Serial Monitor.
- Stop acquisition timer: Para el temporitzador iniciat al segon estat.

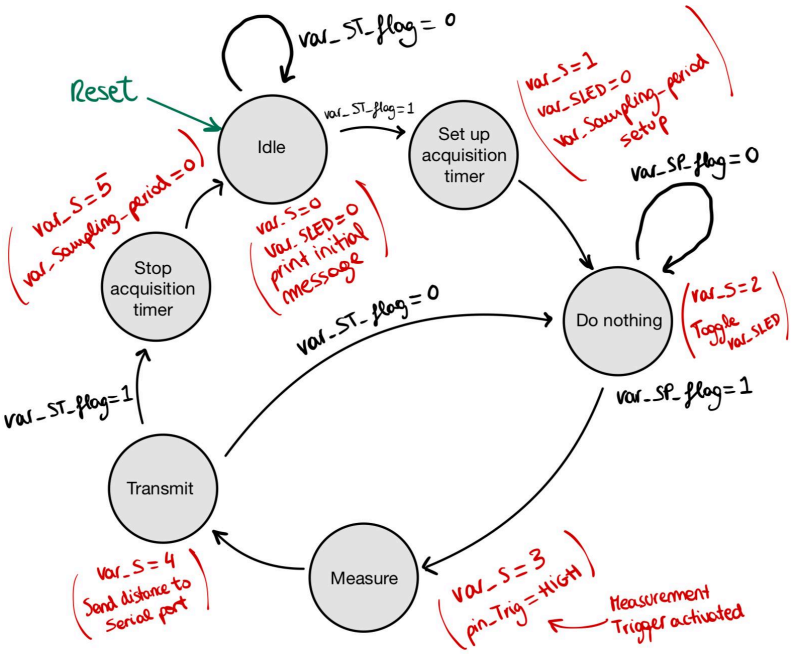

Figura 2.4. Diagrama d'estats pel robot amb sensor frontal

Una vegada el hardware i els estats estan definits podem programar el software de l'Arduino.

2.1.2.2. Software de l'Arduino

Primerament veurem la distribució de les variables RAM:

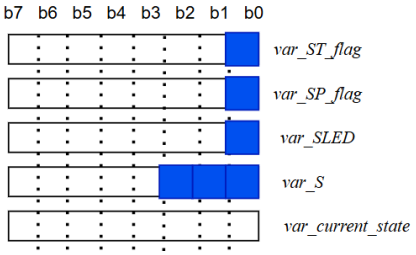

Figura 2.5. Variables RAM del robot amb sensor frontal

Pel que fa a la variable *var_current_state* utilitzarem codi ASCII per a referir-nos a cada estat, per exemple: “A”, “B”, “C”…

Definim també el funcionament de la variable *var_SP_flag*, usarem la funció millis() de l’Arduino per tal de comprovar si ha passat el temps de període determinat. *millis()* és una funció que retorna la quantitat de mil·lisegons que han passat des que l'Arduino és encesa per primera vegada o reiniciada. Per tant, l’esquema a punt per a passar-ho a llenguatge C sobre l’obtenció del període de mostreig seria el que es mostra a la figura 2.6.

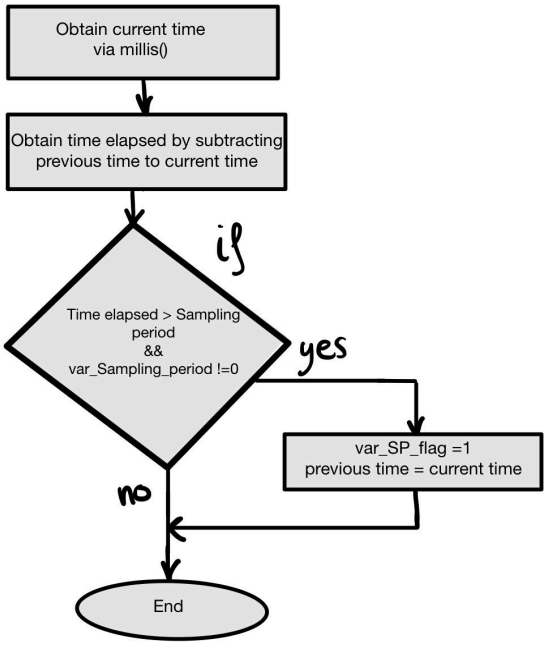

Figura 2.6. Esquema de la funció per a la obtenció de la variable de mostreig

Per poder codificar el codi sense errors i de manera estructurada generarem les taules de la veritat de les funcions *state_logic()* i *output_logic()*, es poden apreciar a continuació.

var_SP_flag	*var_ST_flag*	*var_current state*		*var current state +*
X	0	Idle		Idle 🔄
X	1	Idle		Set up acquisition timer ➡️
X	X	Set up acquisition timer		Do nothing ➡️
0	X	Do nothing		Do nothing 🔄
1	X	Do nothing		Measure ➡️
X	X	Measure		Transmit ➡️
0	X	Transmit		Do nothing ➡️
1	X	Transmit		Stop acquisition timer ➡️
X	X	Stop acquisition timer		Idle ➡️

Taula 2.1. Taula de la veritat state_logic() de sensor frontal

var_current state		*var_S*	*var_SLED*	*Timer*	*Sensor*
Idle		0	0	Deactivated	-
Set up acquisition timer		1	0	Activated	-
Do nothing		2	blinking	Activated	-
Measure		3	blinking	Activated	Trigger ON
Transmit		4	blinking	Activated	-
Stop acquisition timer		5	blinking	Deactivated	-

Taula 2.2. Taula de la veritat output_logic() de sensor frontal

A partir de les taules de la veritat podem crear un esquema per passar directament a llenguatge C amb la màxima claredat.

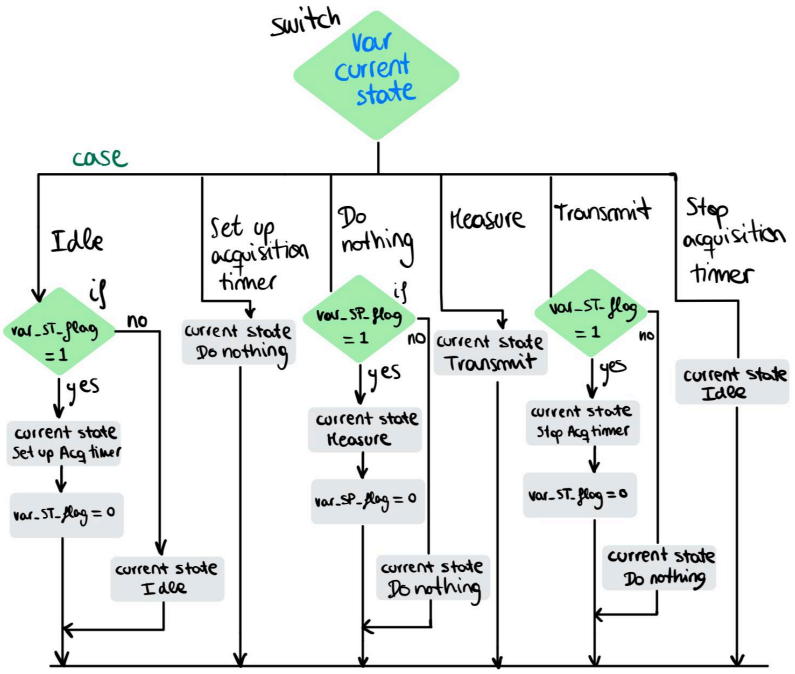

Figura 2.7. Diagrama de flux de la funció state_logic()

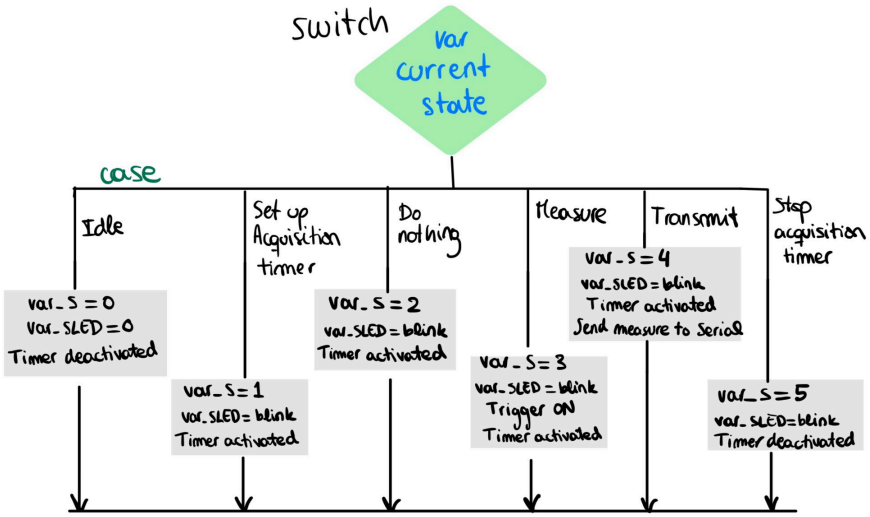

Figura 2.8. Diagrama de flux de la funció output_logic()

Per poder visualitzar la interacció entre hardware i software, integrar les interrupcions de la Interrupt Service Routine i facilitar la implementació de la Finite State Machine al microcontrolador podem concebre un diagrama de software i hardware, de manera que es veurà de forma general la dinàmica del projecte.

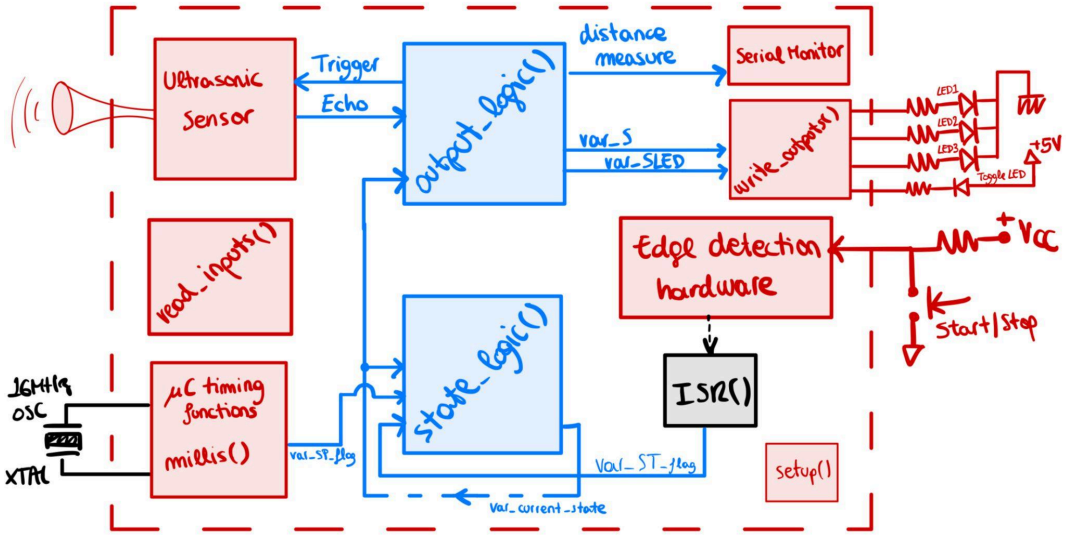

Figura 2.9. Diagrama general hardware/software del sistema de mesura de distància

2.1.3. Desenvolupament i test

2.1.3.1. Hardware

Una vegada tenim clar com ha de ser la implementació de la Finite State Machine podem provar de fer-lo funcionar al software Proteus, a les següents figures podem veure tant el hardware com les variables software que utilitzem.

Pel que fa al Hardware el circuit es veuria de la següent manera:

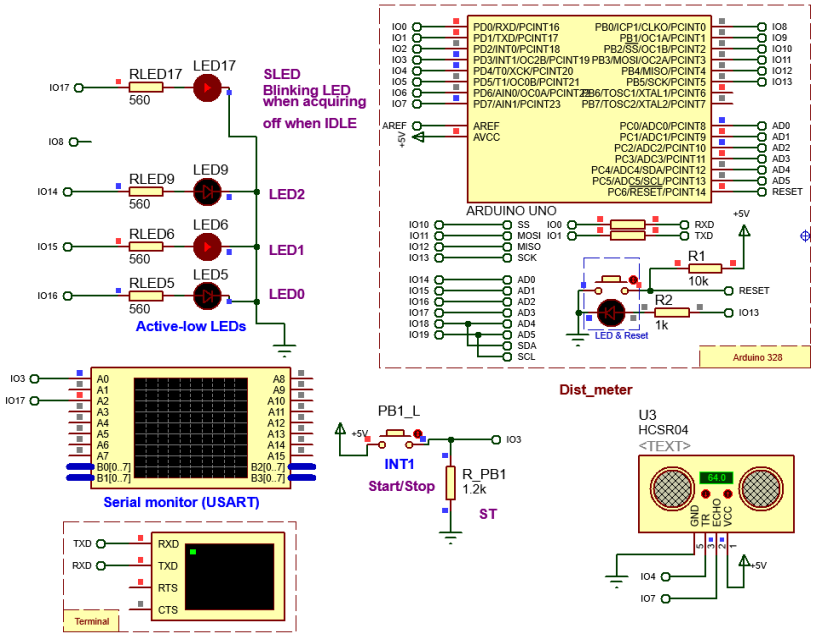

Figura 2.10. Hardware del circuit a Proteus

2.1.3.2. Programari

A continuació podem veure parts clau del codi del qual hem estat concebent en l'apartat anterior, primerament observem la funció d'obtenció del període de mostreig, mitjançant la funció *millis()* de l'Arduino.

```
void update_time_flag(void){

  // From internal CLK:
    var_current_time = millis();
    var_time_elapsed =  var_current_time - var_previous_time;

        if((var_time_elapsed  > var_sampling_period) && var_Sampling_flag != 0) {
                        // is the flag is set is because time elapsed is sampling period
                var_SP_flag = 1;
                        // to start another sampling period cycle
                var_previous_time = var_current_time;
        }
}
```

Figura 2.11. Funció d'obtenció del període de mostreig

Seguidament podem observar també la implementació del *Interrupt Service Routine* a la figura 2.12. Cal remarcar que s'ha hagut d'adaptar el circuit perquè el senyal de la variable *var_ST_flag* reaccioni davant d'un rising edge mitjançant la funció següent:

```
        attachInterrupt(digitalPinToInterrupt(INT1_pin_ST_BUTTON), interrupt_service_routine_INT1, RISING);
        // Interrupt when detecting a rising edge to start and stop operations

void interrupt_service_routine_INT1 (void){

        var_ST_flag = 1;

}
```

Figura 2.12. Setup i funció de l'Interrupt Service Routine

També hem programat la funció que s'encarrega de retornar la distància mesurada, amb el mateix temps de pols i d'espera amb la qual el fabricant ens ho proporciona. Cal destacar que el número pel qual multipliquem la resposta del sensor fa referència a la conversió del temps proporcionat a centímetres, tenint en compte la velocitat del so en l'aire i l'anada i tornada del senyal.

```
//------------------------------------------------------------//
//------------------------------------------------------------//
int watch(){
  long echo_distance=0;
  digitalWrite(pin_Trig,LOW);
  delayMicroseconds(5);
  digitalWrite(pin_Trig,HIGH);
  delayMicroseconds(15);
  digitalWrite(pin_Trig,LOW);
  echo_distance=pulseIn(pin_Echo,HIGH);
  echo_distance=echo_distance*0.01657; //how far away is the object in cm
 //Serial.println((int)echo_distance);
  return round(echo_distance);
}
//------------------------------------------------------------//
//------------------------------------------------------------//
```

Figura 2.13. Codi de la funció encarregada d'enviar el pols i processar la resposta

Seguidament podem veure l'output al Serial Monitor, on veurem la distància que es mesura en cada moment. Amb la simulació amb Proteus la distància detectada és fixa, si es vol canviar s'ha de girar una roda del dispositiu. En el nostre cas ho he forçat per tal que es vegin les variacions. Posteriorment, quan passem a la implementació real del circuit ho comprovarem.

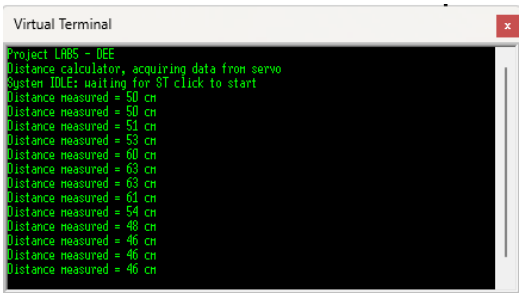

Figura 2.14. Watch Window i Virtual Terminal amb resultats de mesura de distància respectivament

A la figura següent es pot apreciar la dinàmica dels senyals al llarg del temps, es pot apreciar com el senyal de Start/Stop s'activa dues vegades a causa de dos clicks, el senyal del Toggle LED, que indica que el sensor està mesurant segueix la lògica concebuda anteriorment.

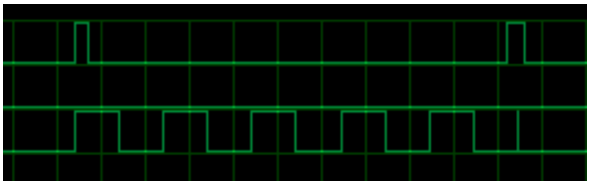

Figura 2.15. Senyals digitals var_ST_flag i var_SLED

Addicionalment podem veure les variables utilitzades i el sistema complet funcionant a la figura 2.16.

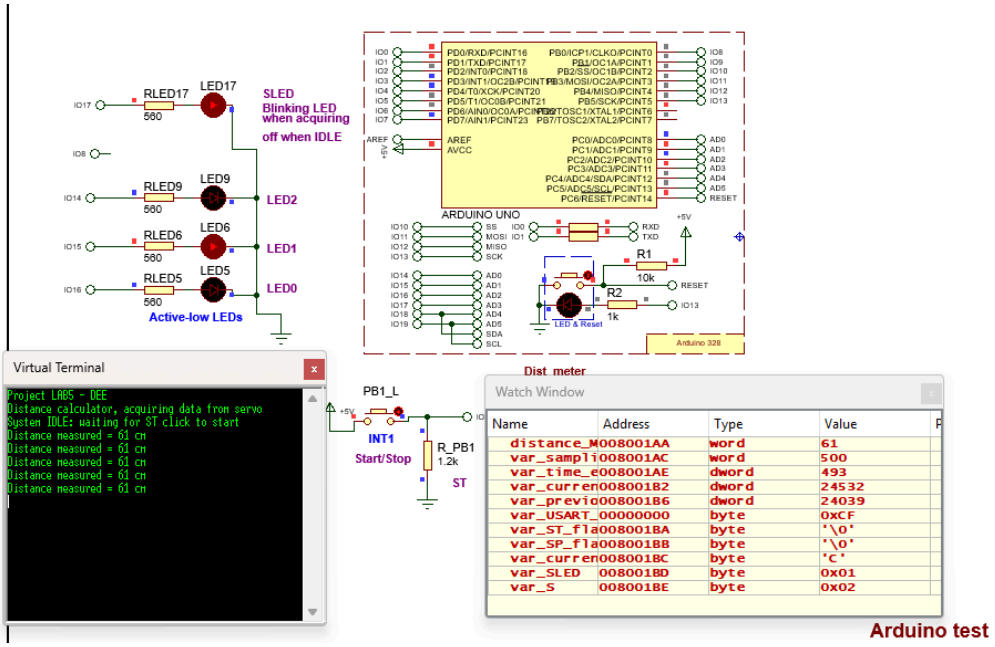

Figura 2.16. Sistema de mesura en funcionament a protus amb la Watch window

Es pot deduir veient la figura 2.16 que estem mesurant amb un període de mostreig de 500 ms, és a dir amb una freqüència de 2 Hz. La simulació ha estat pausada a l'estat Do nothing, codificat com a "C", també relacionat amb l'estat actual veiem la variable *var_S*, que indica el número d'estat en binari, en aquest cas l'estat 2, ja que comencem des de 0.

2.2. Sensor panoràmic

El següent pas en el disseny del sensor que el cotxe utilitzarà per a evitar obstacles és fer una versió del mateix sensor que sigui capaç d'analitzar no només la línia frontal, sinó els 180° frontals. D'aquesta manera permetem que l'obstacle escanegi més superfície, i mirant al futur pròxim del projecte, que sigui més intel·ligent a l'hora de prendre decisions.

2.2.1. Especificacions

Afegirem un *LCD display* que ens permeti veure de manera intuïtiva en quina direcció s'estan trobant els obstacles. Aquest LCD es connectarà a l'Arduino via els ports SCL (Serial data line) i SDA (Serial clock line), utilitzant el protocol I^2C, aquestes dues línies de comunicació bidireccional suporten una comunicació *half-duplex* (Wu, 2022, p. 2).

Per a poder dur a terme aquesta funcionalitat és necessari la implementació d'un motor servo, que serà l'encarregat de moure el sensor ultrasònic. Usarem el que ja ve inclòs al kit d'aprenentatge, es tracta del *Micro Servo SG-90.*

Figura 2.17. Micro Servo SG-90

Aquest servo consta de tres cables, el marró i vermell que fan referència a massa i Vcc, mentre que el taronja és per on rep els senyals PWM que necessita per a discernir entre els angles que li demanem. El pols comença amb un 5% de cicle de treball o *duty cicle* (-90°), i va augmentant a mesura que nosaltres anem augmentant l'angle fins a +90°. La traducció d'aquest senyal la farem mitjançant la funció definida de l'Arduino writeMicroseconds(), on indicarem l'angle amb un nombre, sent 1000 els -90° i 2000 els +90° (Jameco Electronics, n.d.).

En aquest cas afegirem una variable per controlar la freqüència a la qual el servo fa girar el sensor ultrasònic anomenat var_Servo flag.

var_Servo_flag = 0 → Ordre de NO girar el servo.

var_Servo_flag = 1 → Ordre de girar el servo.

Aquesta variable va relacionada amb la var_SP_flag, s'inicialitza i opera de manera similar, aprofitant la funció integrada a l'Arduino millis(). Aquesta variable representa una freqüència que sempre haurà de ser més petita que la freqüència que representa var_SP_flag, més endavant en la planificació d'aquest apartat s'entendrà la importància d'aquesta mesura.

Podem concebre, a grans trets, l'esquema general del circuit a la figura 2.18.

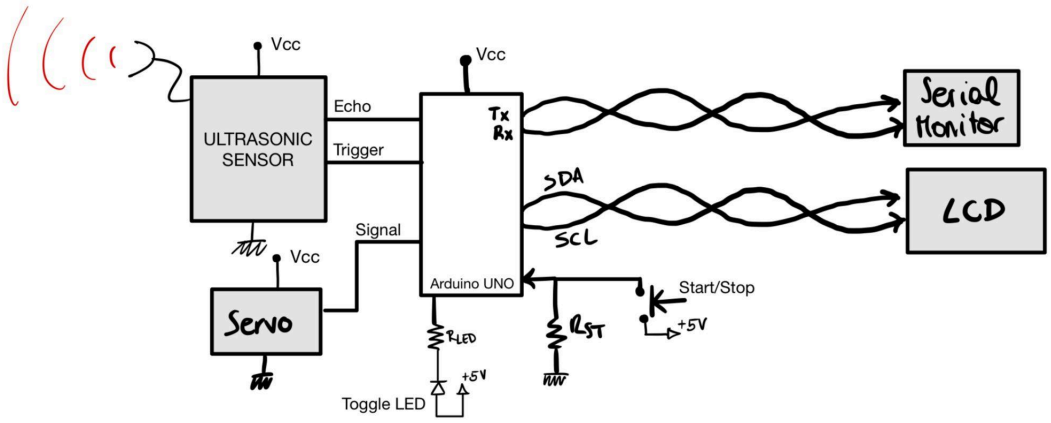

Figura 2.18. Esquema general del robot amb sensor panoràmic

Hi ha diverses actualitzacions respecte al disseny anterior, entre altres, l'eliminació dels tres LEDs indicadors de l'estat, ja que la seva funció la substituirà en gran part el sensor LCD, a més també ens estalviem pins de l'Arduino, cosa que, a causa de la magnitud del projecte agrairem més endavant. Per tant hi ha una reestructuració dels pins.

Una de les novetats en aquesta representació és la pantalla LCD. Aquest component ens servirà per mostrar les mesures que fem a cada cicle, és a dir, en quina direcció detecta l'obstacle.

Per a fer-ho necessitem modificar també el diagrama d'estats anteriorment dissenyat, ja que això ens permetrà fer l'anàlisi d'obstacles correctament.

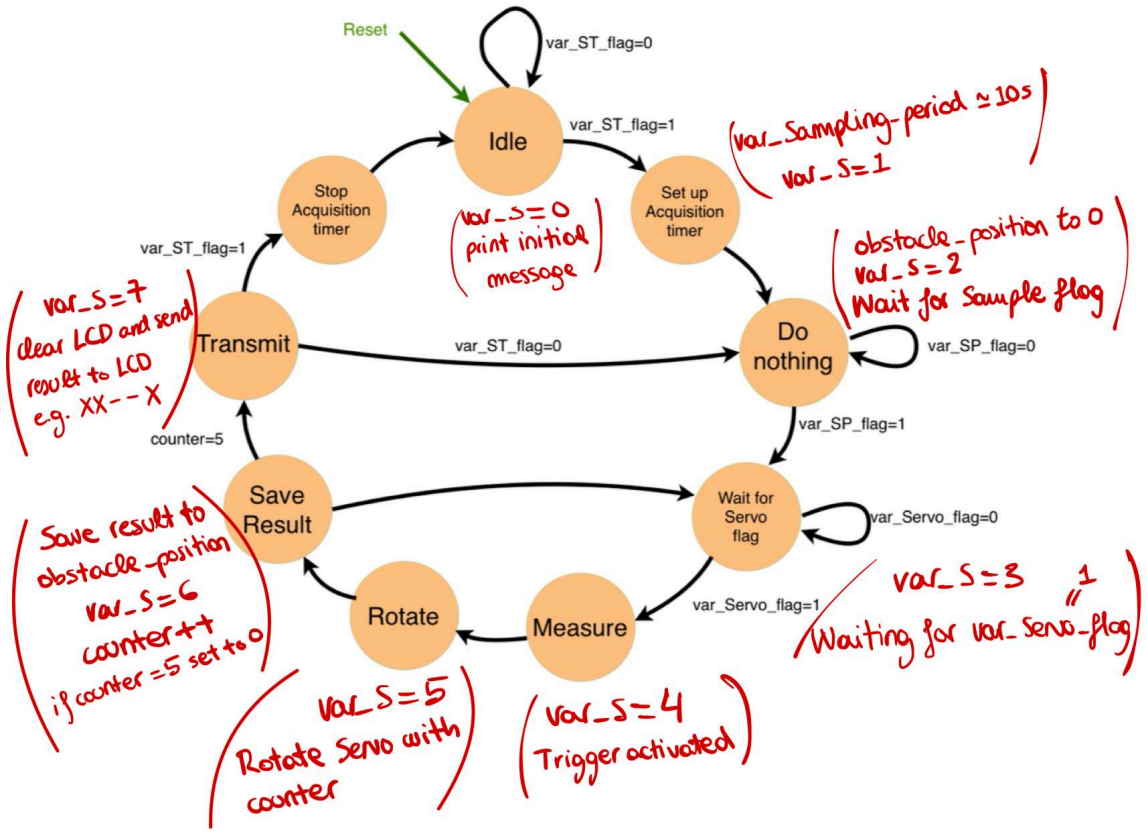

Figura 2.19. Diagrama d'estats del robot amb sensor panoràmic.

En el diagrama de la figura 2.19 podem veure tres estats nous, els quals ens permeten girar el servo que controla la posició del sensor ultrasònic, d'aquesta manera, cada període de mostreig es prendran 5 mostres, una de cada direcció: esquerra, diagonal esquerra, frontal, diagonal dreta i dreta.

Els nous estats són:

- **Wait for Servo flag:** Espera al fet que el flag del servo s'activi per a girar el sensor.
- **Rotate:** Estat per girar el sensor.
- **Save Result:** Guarda el resultat obtingut a una variable, que més endavant s'utilitzarà perquè el sistema pugui prendre decisions.

2.2.2. Planificació del projecte

2.1.2.1 Software de l'Arduino

Les variables RAM són les mateixes que a la figura 2.20, s'afegeix la variable counter i la variable var_Servo_flag, les quals ocupen tres i un bit respectivament. La variable *var_S* passa a ocupar quatre bits a causa de l'augment d'estats.

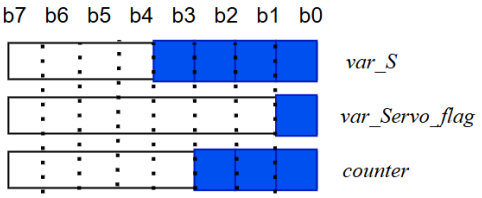

Figura 2.20. Nou flag pel servo, contador per a les rotacions del servo i ampliació de la variable d'estats

La variable *var_SP_flag*, que controla el període de mostreig de cada cicle de cinc rotacions, romà amb el mateix algorisme, ara definim el funcionament de la *var_Servo_flag*.

La *var_Servo_flag* és la variable que controla el període de mostreig de les rotacions del Servo motor, aquesta sempre serà més freqüent que la *var_SP_flag*, ja que es busca que el servo roti cinc vegades dins el període de mostreig global (indicat per *var_SP_flag*). Es pot entendre amb el següent diagrama temporal.

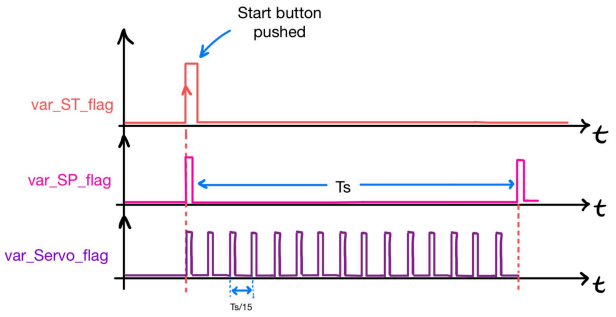

Figura 2.21. Diagrama de temps de les flags de mostreig i rotació var_SP_flag i var_Servo_flag

Per tant, la lògica a seguir per a obtenir aquests flags seria similar a la mostrada a la figura 2.6, tot i que en el condicional canviarem el sampling period per la mateixa variable dividida en 15 parts, d'aquesta manera representarem aquest senyal sempre més freqüent que el període de mostreig global donat per *var_SP_flag*. Es pot veure en la següent il·lustració.

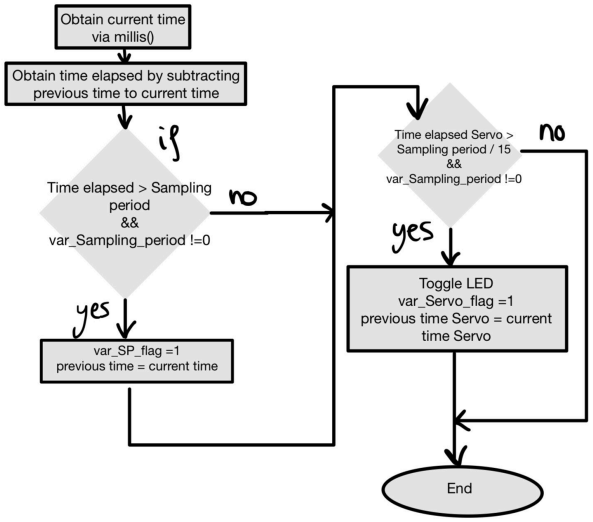

Figura 2.22. Obtenció dels flags de temps de mostreig global i del servo.

Hem canviat el diagrama d'estats, per tant, és necessari modificar també les taules de la veritat, tant de *state_logic()* com de *output_logic()*.

counter	var_Servo_flag	var_SP_flag	var_ST_flag	var_current state		var current state +
X	X	X	0	Idle		Idle 🔄
X	X	X	1	Idle		Set up acquisition timer ➡️
X	X	X	X	Set up acquisition timer		Do nothing ➡️
X	X	0	X	Do nothing		Do nothing 🔄
X	X	1	X	Do nothing		Wait for Servo flag ➡️
X	0	X	X	Wait for Servo flag		Wait for Servo flag 🔄
X	1	X	X	Wait for Servo flag		Measure➡️
X	X	X	X	Measure		Rotate➡️
X	X	X	X	Rotate		Save result ➡️
0, 1, 2, 3 o 4	X	X	X	Save result		Wait for Servo flag ➡️
5	X	X	X	Save result		Transmit ➡️
X	X	X	0	Transmit		Do nothing ➡️
X	X	X	1	Transmit		Stop Acquisition Timer ➡️
X	X	X	X	Stop Acquisition Timer		Idle ➡️

Taula 2.3. Taula de la veritat state_logic() de robot amb sensor panoràmic

var_current state		var_S	var_SLED	Sensor	Servo position
Idle		0	0	-	-90°
Set up acquisition timer		1	0	-	-90°
Do nothing		2	blinking	-	previous position
Wait for Servo flag		3	blinking	-	previous position
Measure		4	blinking	Trigger ON	previous position
Rotate		5	blinking	-	previous position + 45°
Save Result		6	blinking	-	previous position
Transmit		7	blinking	-	previous position
Stop Acquisition Timer		8	blinking	-	-90°

Taula 2.4. Taula de la veritat de la funció output_logic() del robot amb sensor panoràmic

Refem el diagrama de flux de la funció state_logic() a la figura 2.23, per automàticament programar-ho en C.

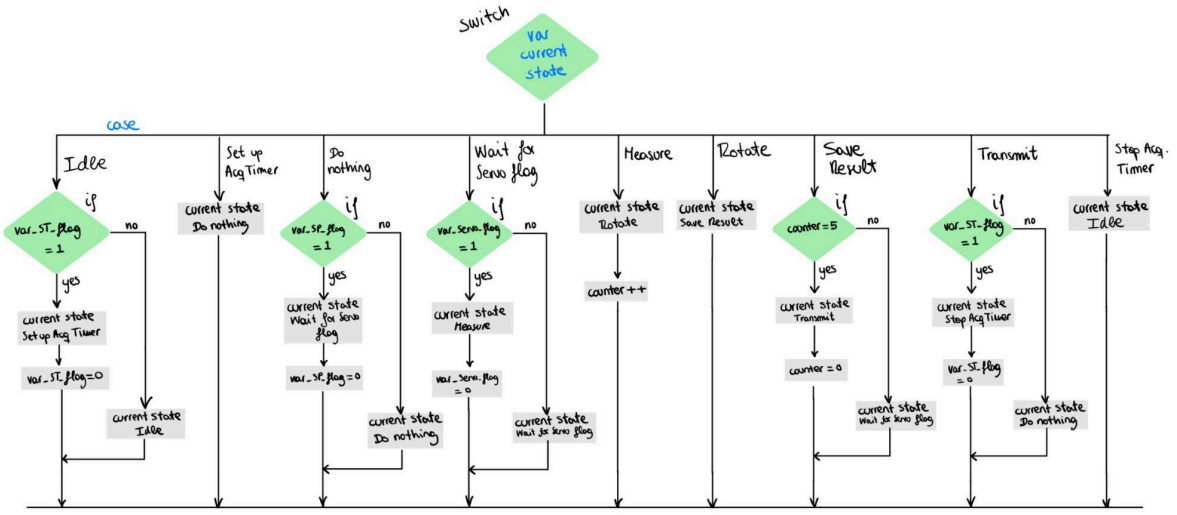

Figura 2.23. Diagrama de flux funció state_logic() de robot amb sensor panoràmic

Seguidament podem representar el diagrama de flux de la funció *output_logic()*, el qual també serà significativament diferent de l'anterior.

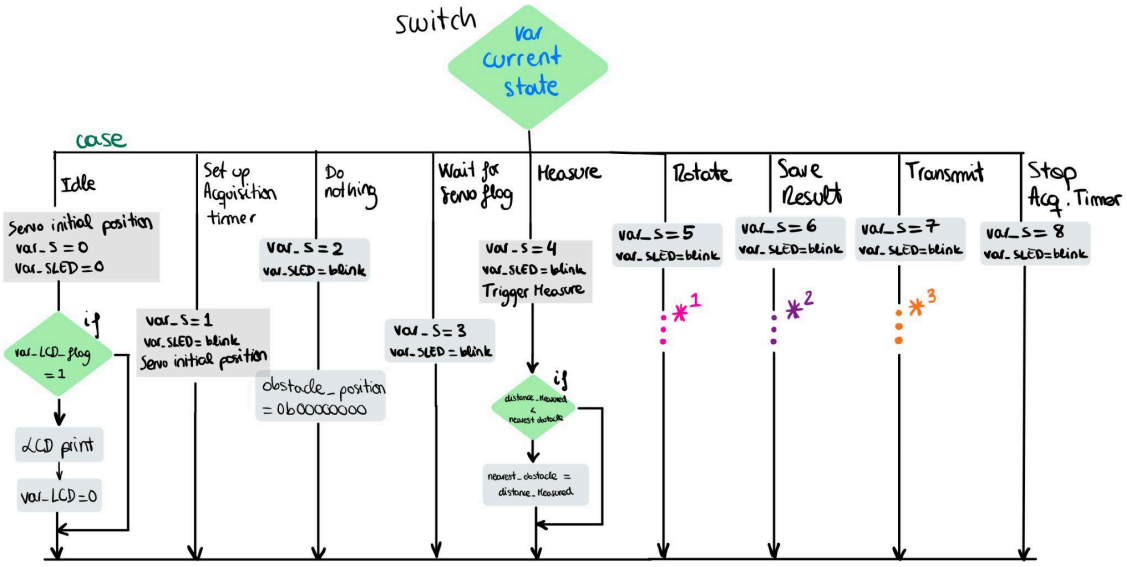

Figura 2.24. Diagrama de flux funció output_logic() de robo amb sensor panoràmic

Es poden apreciar tres asteriscs, aquests fan referència a l'algorisme que s'ha creat per obtenir els obstacles de manera separada i clara en cadascuna de les direccions que el sensor mesura, crearem per cadascun un diagrama de flux que l'exemplifiqui.

Primerament, per a contextualitzar, s'han creat dues variables més, *obstacle_position_operator* i *obstacle_position*, la primera és una variable intermèdia, mentre que la segona conté les posicions on s'ha trobat l'obstacle. Començant pel cas de l'estat *Rotate*, que ve just després de la mesura, aprofita la variable *counter* per a determinar com ha de posicionar-se, és important clarificar que és molt important el format en el qual li enviem la informació al servo, ja que entén els angles amb la funció *writeMicroseconds()*, en el nostre cas volem 5 direccions, tot i que si es volgués més precisió es podria augmentar la resolució.

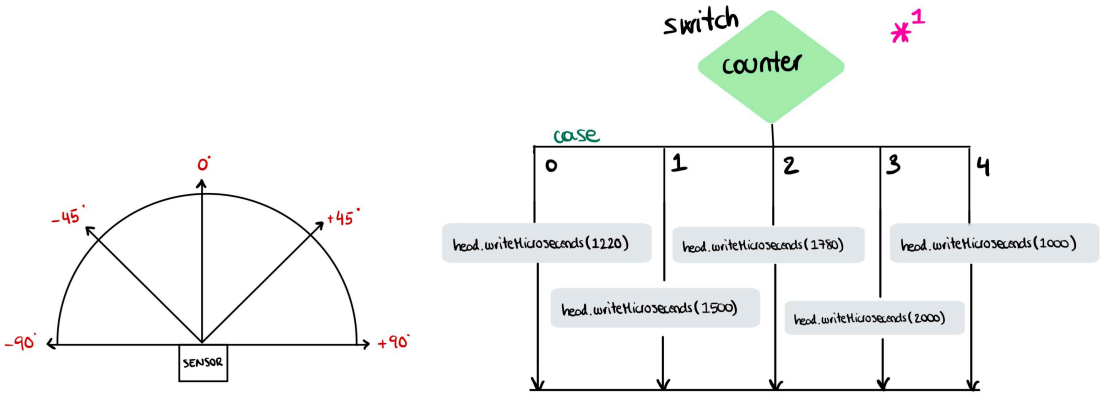

Figura 2.25. Angles de mesura i diagrama de flux de la rotació del servo

Seguidament trobem l'estat *Save Result*, on s'utilitza una de les noves variables mencionades, concretament la variable *obstacle_position* és una variable que servirà com a representació per

saber on s'han trobat els obstacles, en codi binari, només es modificaran els 5 *Least Significant Bits*, cadascun d'ells representa una direcció. Per a posar un exemple, si quan el *counter* ja ha fet el cicle de 5 iteracions i tenim la variable *obstacle_position* = 0b00010110, significa que s'han detectat obstacles a -90°, 0° i 45°. Aquest procediment es realitzarà mitjançant la utilització de l'operador OR (|), es veu de la següent manera:

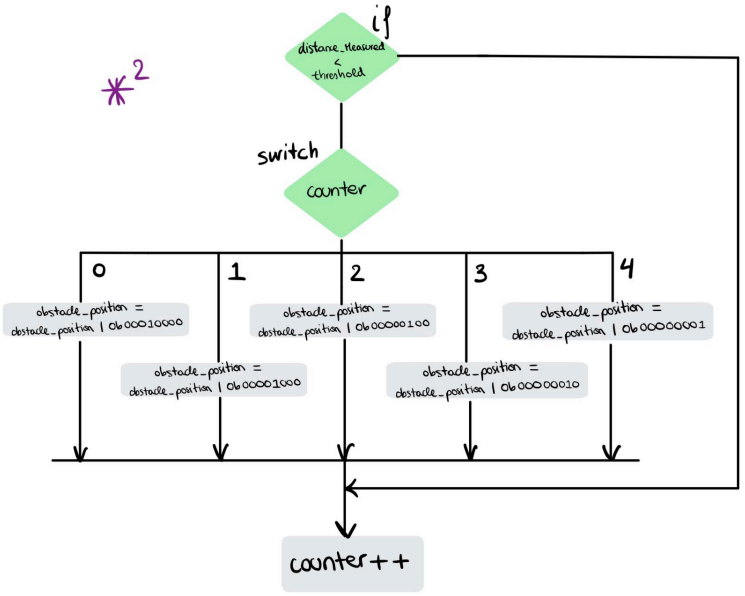

Figura 2.26. Diagrama de flux de l'obtenció de la variable obstacle_position

Per últim, analitzarem l'estat *Transmit*, que s'encarrega d'enviar les dades al LCD display en funció de la variable *obstacle_position*. L'objectiu final és veure-hi de manera molt gràfica, amb creus o guions, les direccions lliures o bloquejades per obstacles. Farem ús de la variable intermèdia *obstacle_position_operator* per tal d'operar amb la variable *obstacle_position*. Utilitzarem l'operador lògic AND per a saber les posicions on hi ha obstacles, després enviarem el caràcter apropiat al LCD display en funció del resultat obtingut. Es pot apreciar el diagrama de flux a la figura 2.27.

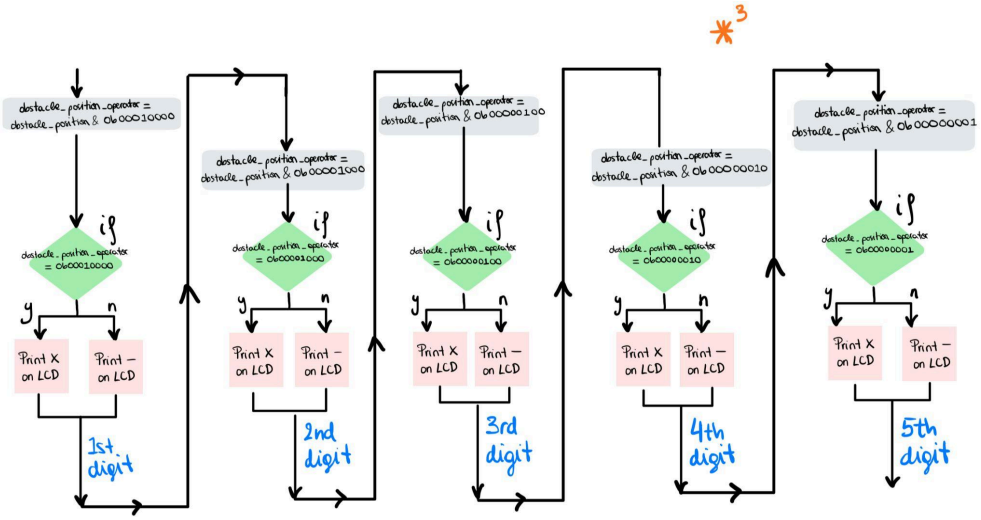

Figura 2.27. Diagrama de flux de l'obtenció del missatge mostrat al LCD display

2.2.3. Desenvolupament i test

2.2.3.1. Hardware

Una vegada executats els passos anteriors podem implementar-ho amb el simulador Proteus, ho farem des del disseny anterior (apreciable a la figura 2.28). Respecte a l'últim disseny s'han d'eliminar i afegir certs components, en aquest cas hi ha menys LEDs, d'altra banda, hem afegit el servo que mou el sensor ultrasònic, un LCD display (component LM016L) i un adaptador per utilitzar el protocol I2C (component PCF8574).

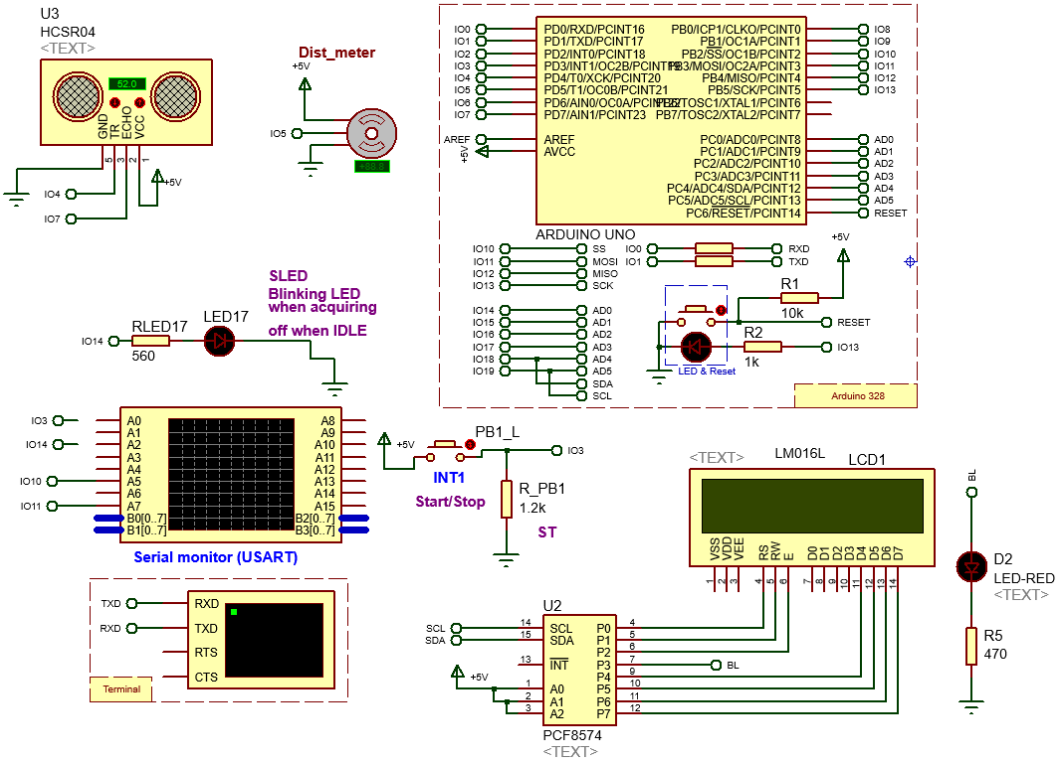

Figura 2.28. Hardware del robot amb sensor panoràmic simulat a Proteus

2.2.3.2. Programari

Quan iniciem la simulació, al Serial Monitor podrem apreciar els dos períodes de mostreig, el que marca les rotacions del motor i el que marca cada grup de cinc mesures. Per tant, compleix la simulació les nostres expectatives? Veiem com el motor rota 5 vegades, amb els períodes de mostreig indicats, seguidament el LCD representa el resultat, com s'ha dit abans, amb creus o guions, en funció de si existeix o no l'obstacle, es pot veure a la figura 2.29 com el disseny compleix amb les expectatives fins al moment.

Cal ressaltar que hem considerat un llindar arbitrari de 50 cm com a límit de decisió en considerar, o no, un obstacle. Més endavant aquest paràmetre es fixarà amb un criteri fonamentat.

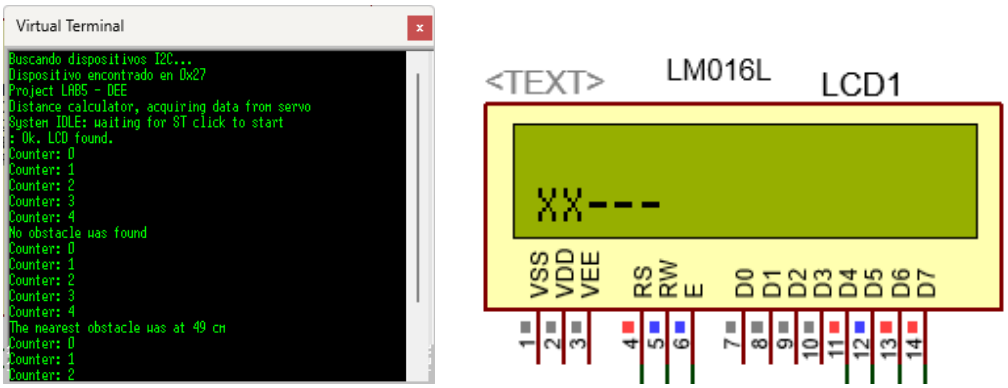

Figura 2.29. Serial monitor i LCD display en simulació del robot amb sensor panoràmic

Podem apreciar també totes les variables que hem mencionat i que per tant utilitzem en aquesta fase en la figura 2.30, en aquesta última figura també es pot apreciar en l'oscil·loscopi com el pols canvia d'amplada en el tercer pols visible, per a indicar-li al servo el canvi d'angle (senyal mostrat en groc). En la figura 2.31 es mostra una il·lustració del codi referent a la lògica de moviment i del servo. En la figura 2.32 es pot apreciar un tros de l'estat d'enviament dels resultats al *LCD display* on clarament s'entén l'evolució del codi i adequació al prèviament concebut.

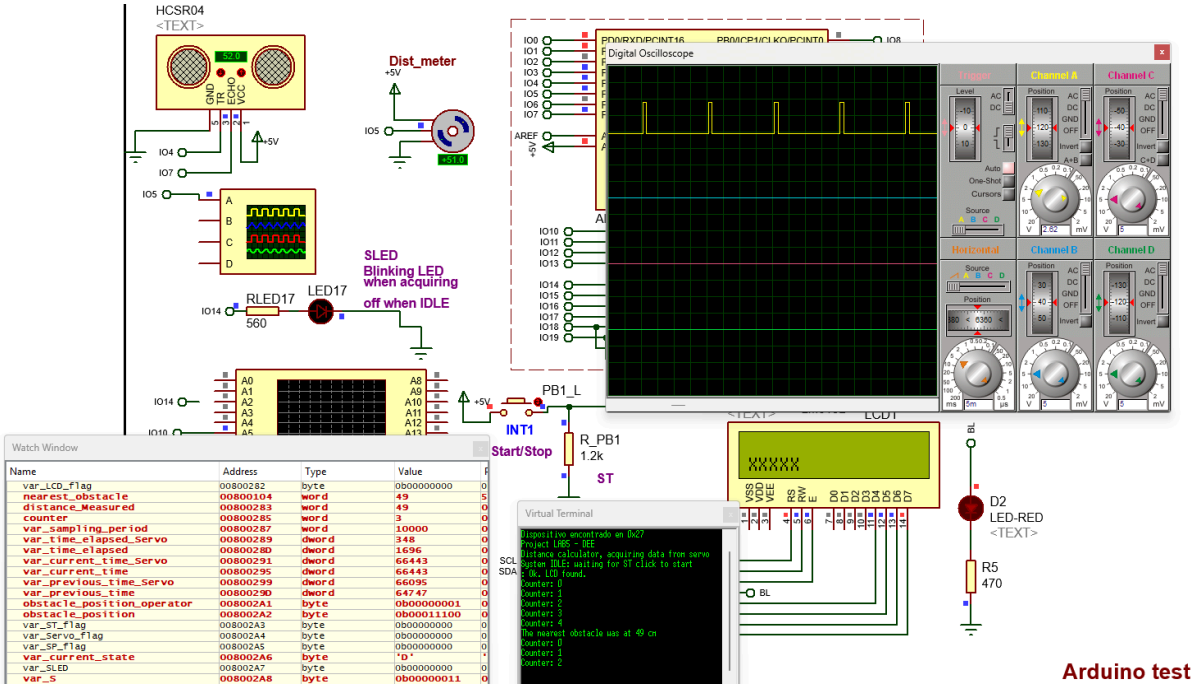

Figura 2.30. Sistema de sensor panoràmic funcionant al complet a Proteus

```
        case Rotate:

        Serial.print("Counter: ");
    Serial.println(counter);

                var_S = 5;
                if (counter == 0) {
                        head.writeMicroseconds(1220);
                }
                else if (counter == 1){
                        head.writeMicroseconds(1500);
                }
                else if (counter == 2){
                        head.writeMicroseconds(1780);
                }
                else if (counter == 3){
                        head.writeMicroseconds(2000);
                }
                else if (counter == 4){
                        head.writeMicroseconds(1000);
                }
                else{

                }

                break;
```

```
case Save_Result:
        var_S = 6;

        if((distance_Measured < threshold) && counter == 0){
            obstacle_position = obstacle_position | 0b00010000;
        }
        if((distance_Measured < threshold) && counter == 1){
            obstacle_position = obstacle_position | 0b00001000;
        }
        if((distance_Measured < threshold) && counter == 2){
            obstacle_position = obstacle_position | 0b00000100;
        }
        if((distance_Measured < threshold) && counter == 3){
            obstacle_position = obstacle_position | 0b00000010;
        }
        if((distance_Measured < threshold) && counter == 4){
            obstacle_position = obstacle_position | 0b00000001;
        }

        counter++;

        break;
```

Figura 2.31. Fragment de codi referent al moviment del servo i guardat de resultats

```
        case TRANSMIT:  // measurement equation
                var_S = 7;
                lcd.clear();
////////////////////////////////////////////////////////////////////////////////////////

                obstacle_position_operator = obstacle_position & 0b00010000;
                if(obstacle_position_operator == 0b00010000){

                        lcd.setCursor(0,1);
                        lcd.print("X");
                }
                else{
                        lcd.setCursor(0,1);
                        lcd.print("-");
                }
////////////////////////////////////////////////////////////////////////////////////////
                obstacle_position_operator = obstacle_position & 0b00001000;
                if(obstacle_position_operator == 0b00001000){

                        lcd.setCursor(1,1);
                        lcd.print("X");
                }
                else{
                        lcd.setCursor(1,1);
                        lcd.print("-");
                }
////////////////////////////////////////////////////////////////////////////////////////
```

Figura 2.32. Fragment de codi referent a l'enviament dels resultats al LCD display

2.2.4. Prototipatge

A partir d'aquest punt ja tenim una base sòlida per portar-ho a la realitat, com estem procedint amb cadascuna de les fases ho realitzarem per passos. Primerament implementarem el prototip sobre una placa de proves, d'aquesta manera podem assegurar que el disseny funciona de la manera esperada. Seguidament passarem els components a una placa PCB universal, en la qual soldarem els components tot comprovant el seu funcionament. A la següent figura es pot apreciar la placa PCB universal amb els components soldats.

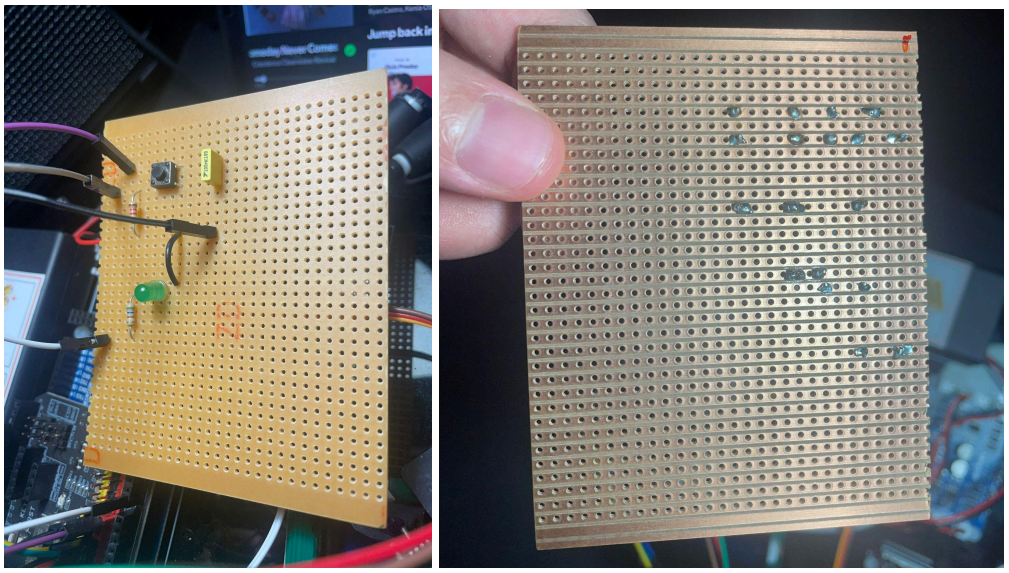

Figura 2.33. Components del robot amb sensor panoràmic externs soldats a la PCB universal

Ara compilarem el codi i el carregarem a l'Arduino, com esperàvem funciona de manera correcta, a la següent figura podem apreciar un exemple del robot en l'estat Idle.

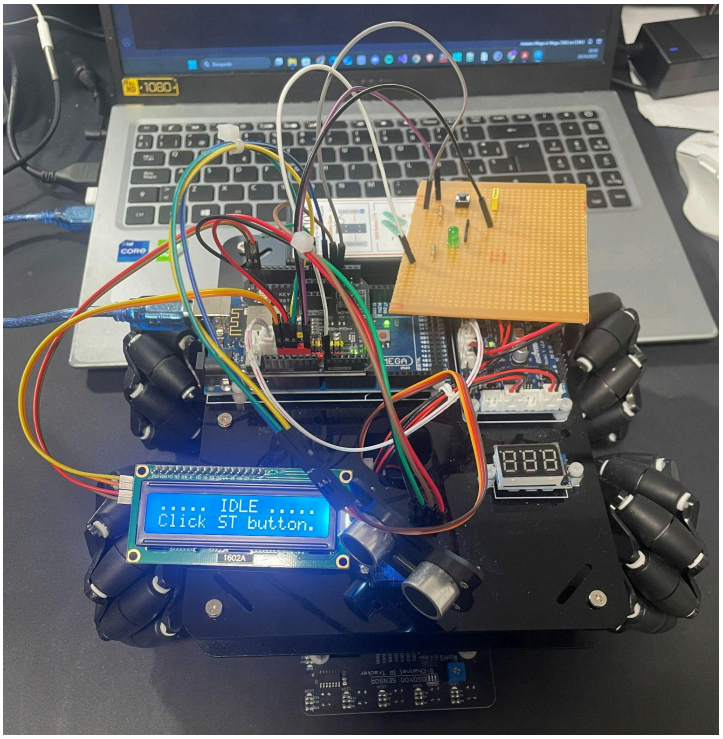

Figura 2.34. Robot amb sensor panoràmic muntat i funcionant a la realitat

Seguidament podem veure una imatge del robot fent les mesures, s'aprecia com al LCD display s'hi mostren tres "X" i dos "-", cosa que indica que l'obstacle es troba al davant i a la dreta del robot.

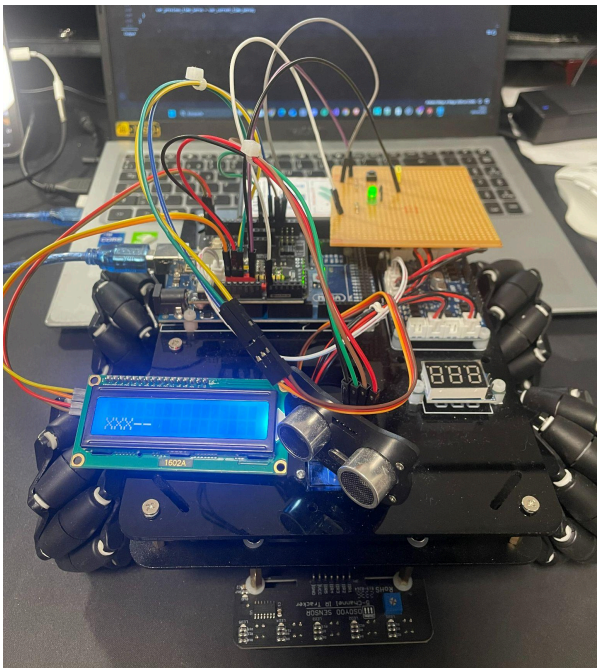

Figura 2.35. Robot amb sensor panoràmic mostrant el LCD display en funcionament

2.3. Moviment i evasió d'obstacles

En aquest subapartat afegirem als avenços existents la lògica per implementar un sistema de rodes que portin al cotxe cap a una direcció determinada en funció dels resultats obtinguts pel sensor ultrasònic adjunt.

2.3.1. Especificacions

Per a dur a terme aquesta funcionalitat no s'afegirà cap altre flag ni variable, aquesta acció s'executarà de manera seqüencial amb els altres estats, ja que ho implementarem en un punt del funcionament pel qual sempre que hi passi s'executi, així la implementació serà més senzilla que en els altres casos.

En aquest cas d'ús és necessari implementar un nou component per a controlar les rodes, es tracta del controlador dels 4 drivers que utilitzarem, en aquest cas el fabricant ens proporciona un controlador perfectament adient per les nostres necessitats, aquest component ja l'hem introduït abans, es pot veure una il·lustració a la figura 1.12.

Connectarem cada parell de cables del driver a un slot de cada canal.
Per a simular aquest dispositiu amb Proteus haurem de realitzar una adaptació que especificarem més endavant, ja que aquest no es troba al simulador. Per a tenir una visió general del sistema podem consultar la següent figura.

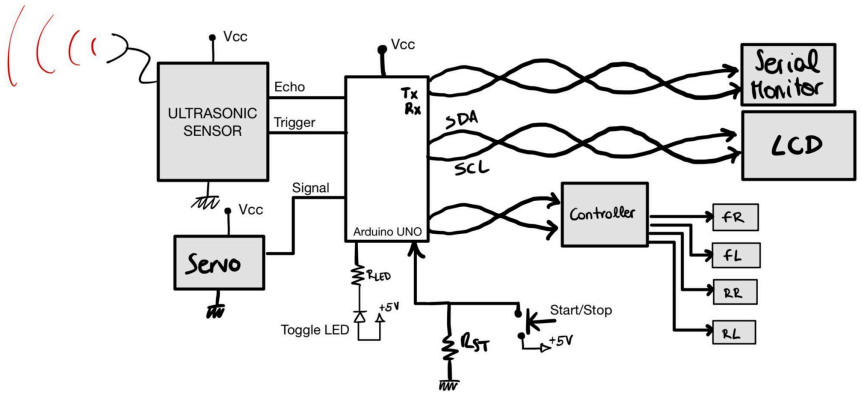

Figura 2.36. Esquema general del robot amb evasió d'obstacles

Si volem implementar aquesta funcionalitat és necessari modificar el diagrama d'estats, cal ressaltar que els estats queden igual que anteriorment, únicament se li ha afegit l'estat *Move*, la proposta per a aquesta secció és la següent:

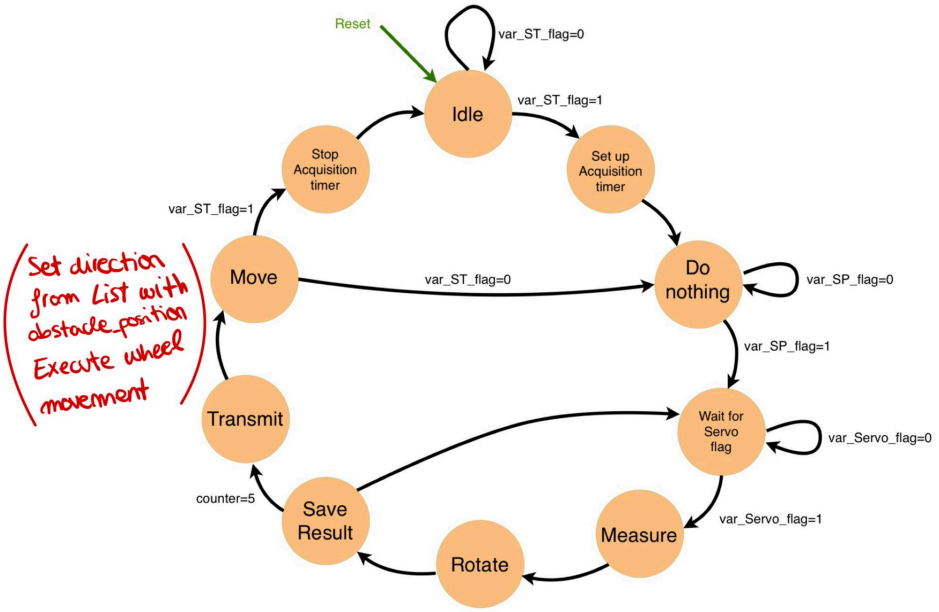

Figura 2.37. Diagrama d'estats del robot amb evasió d'obstacles

És notable que s'ha afegit un estat respecte al diagrama d'estats anterior, seguint la dinàmica del diagrama d'estats, s'executarà després de transmetre els resultats de les mesures al LCD display, el qual ens permetrà veure de manera gràfica els obstacles detectats pel cotxe i la seva conseqüent decisió d'encaminament i moviment.

2.3.2. Planificació del projecte

Les taules de la veritat seràn molt similars a les sintetitzades pel pas anterior, amb els únics canvis que implica afegir aquest nou estat.

Per afegir l'estat al codi es modificarà la funció *state_logic()*. Pel que fa a la funció *output_logic()* implementarem la lògica per a la decisió de la direcció a prendre, es pot veure a la taula 2.8. En aquesta representació s'aprecien el total de casos possibles, ja que al tenir 5 posicions de mesura d'obstacles i dos casos possibles per posició (hi ha obstacle o no n'hi ha) tenim 32 possibilitats a valorar, per així evitar casos on el robot es pugui quedar en blanc, per tant no moure's, i que això porti al fet que s'hagi de moure el robot de manera manual.

Seguidament es poden veure les noves taules de la veritat.

counter	var_Servo_flag	var_SP_flag	var_ST_flag	var_current state		var current state +
X	X	X	0	Idle		Idle 🔄
X	X	X	1	Idle		Set up acquisition timer ➡️
X	X	X	X	Set up acquisition timer		Do nothing ➡️
X	X	0	X	Do nothing		Do nothing 🔄
X	X	1	X	Do nothing		Wait for Servo flag ➡️
X	0	X	X	Wait for Servo flag		Wait for Servo flag 🔄
X	1	X	X	Wait for Servo flag		Measure➡️
X	X	X	X	Measure		Rotate➡️
X	X	X	X	Rotate		Save result ➡️
0, 1, 2, 3 o 4	X	X	X	Save result		Wait for Servo flag ➡️
5	X	X	X	Save result		Transmit ➡️
X	X	X	X	Transmit		Move ➡️
X	X	X	0	Move		Do nothing ➡️
X	X	X	1	Move		Stop Acquisition Timer ➡️
X	X	X	X	Stop Acquisition Timer		Idle ➡️

Taula 2.5. Taula de la veritat de la funció state_logic() de robot amb evasió d'obstacles

var_current state	var_S	var_SLED	Sensor	Servo position	Wheel movement
Idle	0	0	-	-90°	None
Set up acquisition timer	1	0	-	-90°	None
Do nothing	2	blinking	-	previous position	None
Wait for Servo flag	3	blinking	-	previous position	None
Measure	4	blinking	Trigger ON	previous position	None
Rotate	5	blinking	-	previous position + 45°	None
Save Result	6	blinking	-	previous position	None
Transmit	7	blinking	-	previous position	None
Move	8	blinking	-	previous position	Decision and movement
Stop Acquisition Timer	9	0	-	-90°	None

Taula 2.6. Taula de la veritat de la funció output_logic() del robot amb evasió d'obstacles

Seguint l'esquema cal aclarir que les funcions per encaminar el cotxe, seran combinacions de les funcions llistades a continuació. Aquestes funcions envien el senyal directament a les rodes que es desitja que es moguin, per així resultar en el moviment buscat.

- turnRight(): S'activen les rodes FL i RL cap endavant.
- turnLeft(): S'activen les rodes FR i RR cap endavant.
- moveForward(): S'activen les quatre rodes cap endavant.
- stopMotors(): Es posa velocitat 0 a totes les rodes.

Més enllà de les activacions de les rodes, també hem de considerar la velocitat que se li donarà a cadascuna des dels pins de modulació per ample de pols (PWM). Aquesta variable té una propietat important, serà la mateixa velocitat per a totes les rodes, per tant, només haurem de declarar una variable per aquesta finalitat. Si es desitgés executar frenades o accelerades més graduals només hauriem de posar més variables de manera esglaonada. Tot i això, s'ha de tenir molt en compte el valor que se li atribueix a aquesta variable, ja que pot provocar que el robot xoqui contra els obstacles en funció del llindar que definim per al sensor ultrasònic.

Llavors, sigui d_{TH} la distància llindar pels obstacles en metres del sensor ultrasònic, i v la velocitat a la qual ha d'anar el cotxe en metres per segon, tenim la següent expressió:

$$d_{TH}(m) > v(\frac{m}{s}) \cdot t(s) \quad (2.3.2.)$$

Si definim que el temps en el qual el cotxe es mou és d'un segon, i que la distància llindar en la que es considera l'obstacle és de 50 cm, ens resulta la següent condició:

$$v < 0.5 \frac{m}{s}$$

Per tant, la velocitat límit a la que ha d'anar per a no xocar-se amb obstacles no detectats és

$$v_{TH} = 0.5 \frac{m}{s}$$

Segons les especificacions del fabricant del model de motor utilitzat ens indica el següent:

Motor JGA25-370-35.5KG	
Rated Voltage	DC 6V
Rated Voltage	170 ±10% RPM
No-load Current	0.1 A
Rated Current	0.5 A
Rated Speed	131 ±10% RPM
Rated Torque	0.6 Kg·cm
Stall Torque	3.3 Kg·cm
Stall Current	2.6 A

Taula 2.7. Especificacions tècniques del motor JGA25-370-35.5KG

Veiem que el motor ens proporciona una velocitat de 131±10% revolucions per minut, per tant si agafem el pitjor cas per a la nostra aplicació estaríem parlant de 144 revolucions per minut. Si tenim en compte que el radi de les Mecanum Wheels és de 4 centímetres, podem estimar:

$$v_{max} = \frac{144\,rpm \cdot 2\pi r}{60\,s} \simeq 0.6 \frac{m}{s}$$

Com que $v_{max} > v_{TH}$ no podem posar la potència màxima dels motors, per saber quin valor hem de posar com a màxim en els pins de pols d'amplitud modulada ho podem fer mitjançant una regla de tres. Finalment, veiem que el valor màxim que podem posar en els respectius pins és de 212, per tenir més marge posarem un valor més baix.

Per motius pràctics i de llicències, estem fent les simulacions al software Proteus amb l'Arduino UNO, tot i que després, quan ho passem a la realitat ho connectem a l'Arduino Mega 2560. Això ens porta, en aquest punt del projecte, a que ens estem quedant sense pins disponibles a l'Arduino UNO, mentre que a l'altre microcontrolador sí que n'hi ha. A causa d'aquest inconvenient realitzarem una simulació més senzilla, que ens permeti veure el comportament dels motors amb el driver, d'aquesta manera quan passem a l'Arduino Mega 2560 només caldrà descomentar certes línies del codi, això farà que s'activin els altres pins i funcioni degudament.
Més enllà de la limitació dels pins, aquesta mesura també és necessària de fer amb motiu de les característiques dels motors i limitacions del driver amb el que treballem, ja que del contrari podem fer malbé el driver si connectem dos motors pel mateix canal de sortida arran de la *Stall Current*.
Per facilitar aquesta tasca farem que els motors de l'eix dret es moguin exactament de la mateixa manera, igualment succeirà amb les rodes de l'eix esquerre.

He definit 6 direccions, endavant (FORWARD), endarrere (BACKWARD), esquerra (LEFT), esquerra diagonal (LEFTD), dreta (RIGHT) i dreta diagonal (RIGHTD), fent referència a les direccions introduïdes en la figura 2.25, analitzades pel sensor de proximitat panoràmic.

La taula de decisió és la següent, en funció de la variable *obstacle_position*:

Direcció	*obstacle_position*	Direcció	*obstacle_position*
FORWARD	0b00000000	RIGHTD	0b00001000
LEFTD	0b00000001	LEFT	0b00001001
LEFTD	0b00000010	LEFT	0b00001010
LEFTD	0b00000011	LEFT	0b00001011
LEFT	0b00000100	RIGHT	0b00001100
LEFT	0b00000101	LEFT	0b00001101
LEFT	0b00000110	LEFT	0b00001110
LEFT	0b00000111	LEFT	0b00001111

Taula 2.8. Taula de decisió de la direcció a prendre en funció de la variable obstacle_position part 1

Direcció	*obstacle_position*	Direcció	*obstacle_position*
RIGHT	0b00010000	RIGHTD	0b00011000
FORWARD	0b00010001	RIGHTD	0b00011001
RIGHT	0b00010010	LEFT	0b00011010
LEFTD	0b00010011	BACKWARD	0b00011011
RIGHT	0b00010100	RIGHT	0b00011100
BACKWARD	0b00010101	BACKWARD	0b00011101
RIGHT	0b00010110	RIGHT	0b00011110
BACKWARD	0b00010111	BACKWARD	0b00011111

Taula 2.8. Taula de decisió de la direcció a prendre en funció de la variable obstacle_position part 2

Per tant, la lògica de decisió del robot la podem concebre amb el diagrama de flux que adjunto a la figura 2.38.

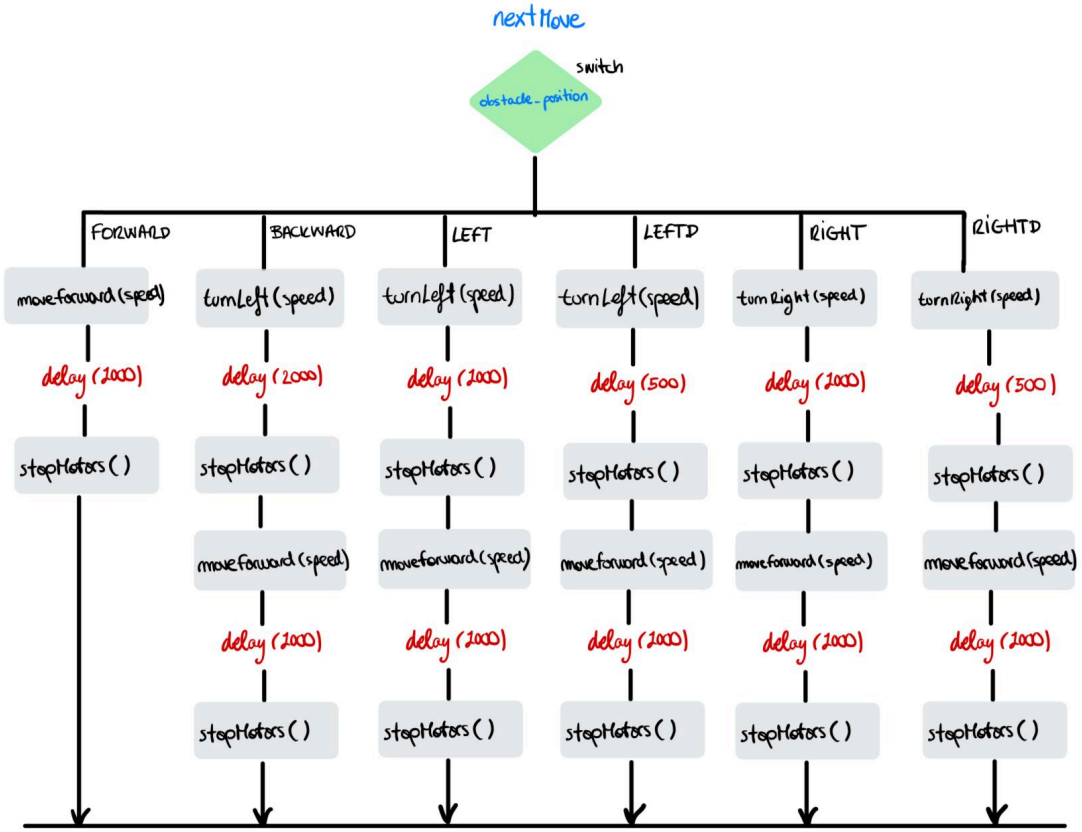

Figura 2.38. Diagrama de flux de les funcions cridades en funció de la direcció presa

S'han definit 3 funcions amb les quals s'assoleixen tots els moviments contemplats, *moveForward()*, *turnLeft()*, *tunRight()* i *stopMotors()*, com bé s'ha comentat abans. Cada funció específica crida a les funcions que mouen els motors, cada motor té 2 funcions assignades, en el cas del motor que mou la roda frontal dreta són FR_bck() i FR_fwd(), el mateix pels altres tres motors. Per tant, com a exemplificació, la funció *moveForward()*, crida a les funcions FR_fwd(), FL_fwd(), RR_fwd(), RL_fwd().

Una vegada clarificada la programació de l'algorisme podem passar a la simulació amb el Proteus.

2.3.3 Desenvolupament i test

2.3.3.1. Hardware

Seguidament es pot apreciar la simulació realitzada amb el Proteus, aplicant l'incís que s'ha fet anteriorment sobre les limitacions de pins existents amb l'Arduino UNO, per tant, utilitzarem el mateix pin de velocitat per tots els motors, i posarem en paral·lel els motors del costat dret i els del costat esquerre. Per qüestions de format no és possible documentar el moviment dels motors a temps real tot apreciant els resultats de les mesures.

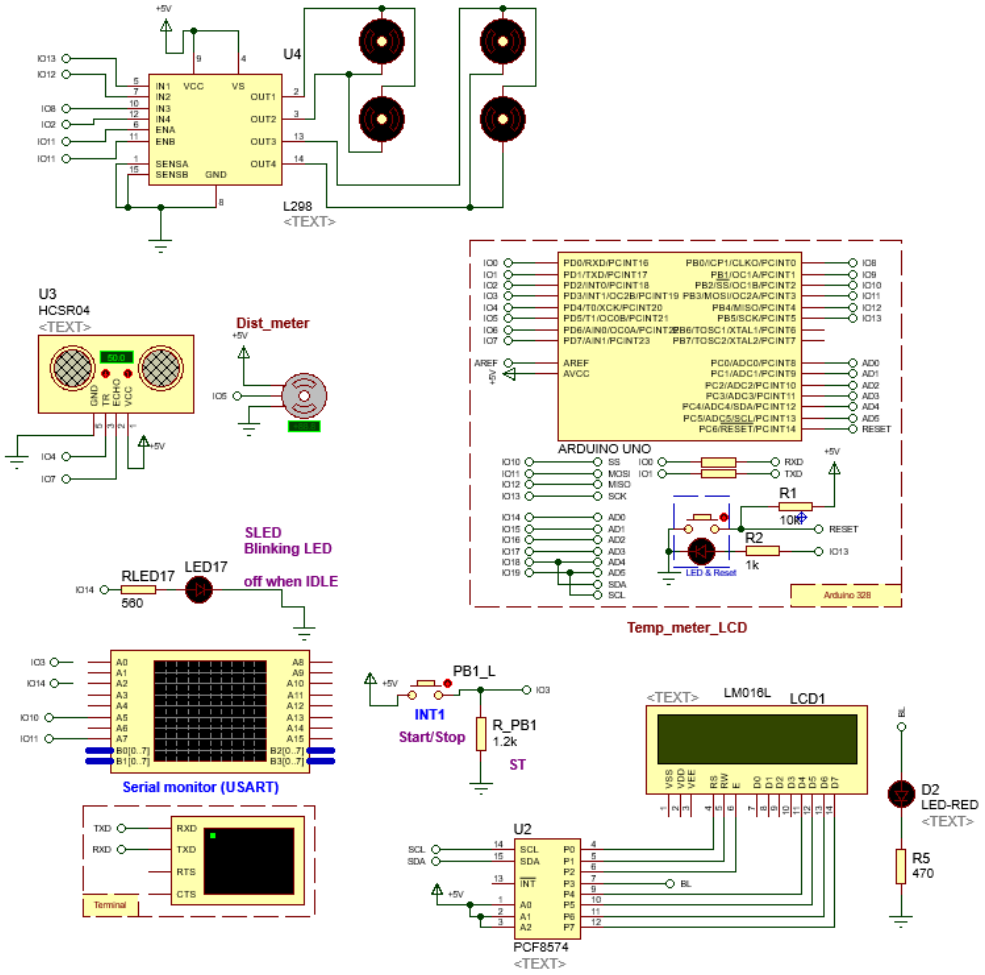

Figura 2.39. Hardware del robot amb evasió d'obstacles simulat a Proteus

2.3.3.2. Programari

A continuació s'adjunten captures d'una part del codi desenvolupat per a l'estat *Move* que acabem de concebre en aquests últims apartats, cal destacar que la presa de decisions s'ha fet mitjançant una llista creada, donant una direcció a totes les possibilitats de la variable *obstacle_avoidance.*

```
case MOVE_CAR:
        var_S = 8;

        nextMove(obstacle_position, 125);

        break;
```

Figura 2.40. Crida a la funció de decisió i execució de moviment a l'estat Move

```
//WHEEL MOVEMENT DECISION============================
//===================================================
enum MOVEMENTS {
  FORWARD, BACKWARD, LEFT, LEFTD, RIGHT, RIGHTD, STOP
};

const MOVEMENTS decision_table[32] = {

  FORWARD,    // 0b00000000
  LEFTD,      // 0b00000001
  LEFTD,      // 0b00000010
  LEFTD,      // 0b00000011
  LEFT,       // 0b00000100
  LEFT,       // 0b00000101
  LEFT,       // 0b00000110
  LEFT,       // 0b00000111
  RIGHTD,     // 0b00001000
  LEFT,       // 0b00001001
  LEFT,       // 0b00001010
  LEFT,       // 0b00001011
  RIGHT,      // 0b00001100
  LEFT,       // 0b00001101
  LEFT,       // 0b00001110
  LEFT,       // 0b00001111
  RIGHTD,     // 0b00010000
  FORWARD,    // 0b00010001
  RIGHT,      // 0b00010010
  LEFTD,      // 0b00010011
  RIGHT,      // 0b00010100
  BACKWARD,   // 0b00010101
  RIGHT,      // 0b00010110
  BACKWARD,   // 0b00010111
  RIGHTD,     // 0b00011000
  RIGHTD,     // 0b00011001
  LEFT,       // 0b00011010
  BACKWARD,   // 0b00011011
  RIGHT,      // 0b00011100
  BACKWARD,   // 0b00011101
  RIGHT,      // 0b00011110
  BACKWARD    // 0b00011111

};
```

```
void nextMove(uint8_t obstacle_position, int speed) {
  MOVEMENTS m = decision_table[obstacle_position];
  switch(m) {
    case FORWARD:
    moveForward(speed);
    delay(1000);
    stopMotors();
    break;

    case BACKWARD:
    turnLeft(speed);
    delay(2000);
    stopMotors();
    moveForward(speed);
    delay(1000);
    stopMotors();
    break;

    case LEFT:
    turnLeft(speed);
    delay(1000);
    stopMotors();
    moveForward(speed);
    delay(1000);
    stopMotors();
    break;
```

Figura 2.41. Llista creada i fragment de switch per a cridar les funcions en funció de la direcció

```
void moveForward(int speed){
   RL_fwd(speed);
   RR_fwd(speed);
   FR_fwd(speed);
   FL_fwd(speed);
}
```

```
void FR_fwd(int speed)  //front-right wheel forward turn
{
  digitalWrite(directionPinFR1,LOW);
  digitalWrite(directionPinFR2,HIGH);
  analogWrite(speedPinFR,speed);
}
```

Figura 2.42. Funcions de moviment de les rodes

En la següent captura es pot apreciar el *LCD display,* com decideix la direcció on anar, en funció dels resultats representats al *LCD* que es veuen a les captures, per ordre cronològic.

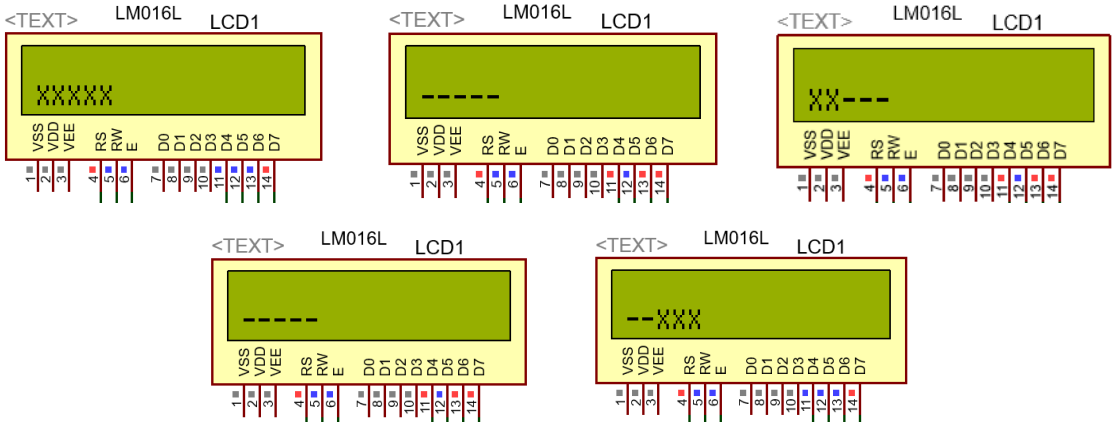

Figura 2.43. LCD display mostrant la posició dels obstacles existents en diferents casos

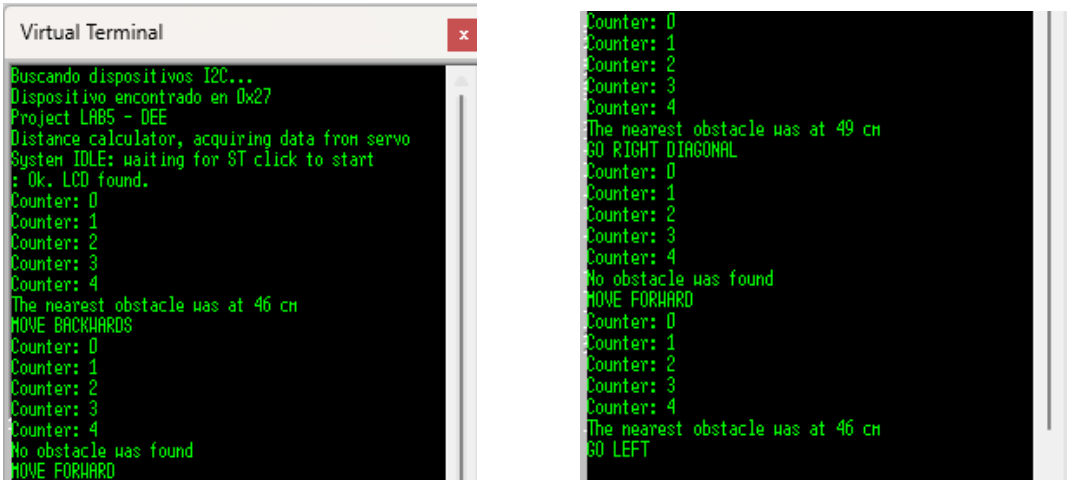

Figura 2.44. Monitor Serial mostrant la variable counter, la distància de l'obstacle més proper i la direcció presa

Podem veure el sistema complet en funcionament a la figura 2.45.

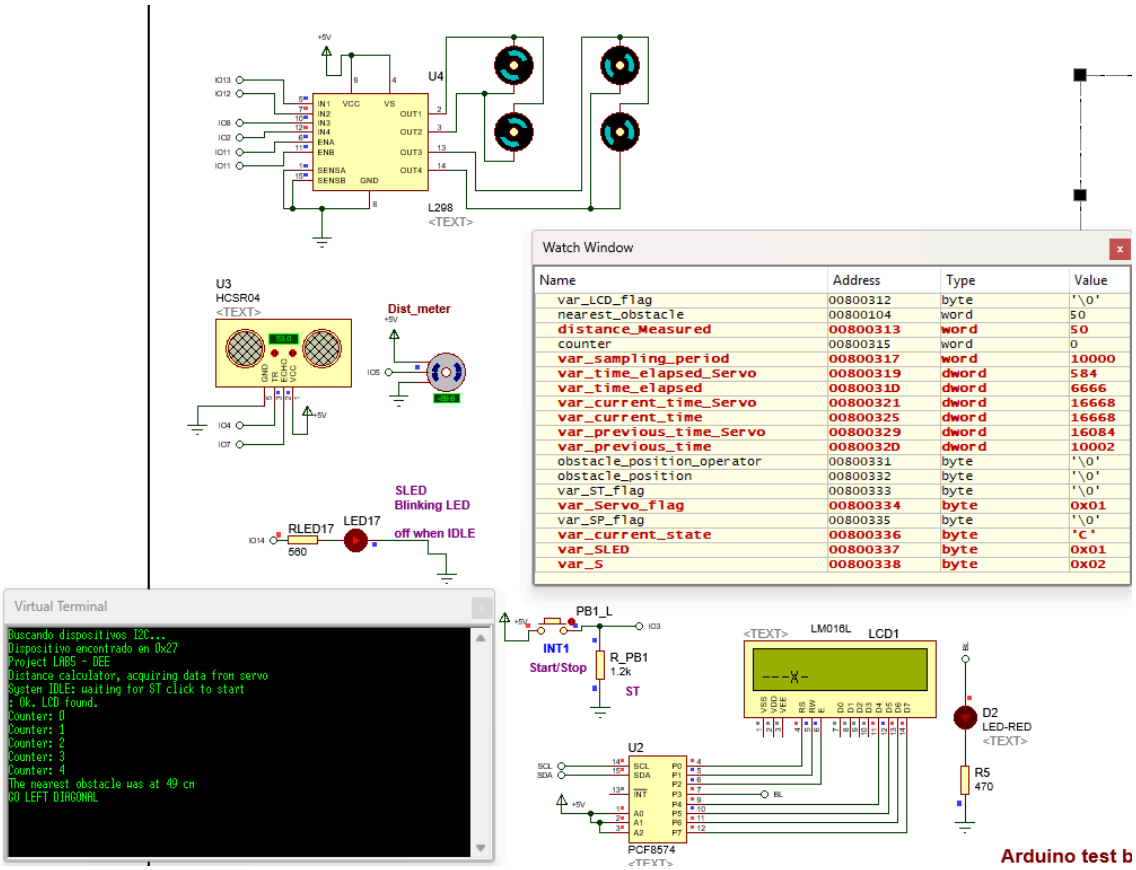

Figura 2.45. Sistema de robot amb evasió d'obstacles complet en funcionament a Proteus amb Watch window

2.3.4. Prototipatge

Primerament hem de clarificar els pins als quals posarem aquells dels que ens hem després en l'Arduino UNO. Per a mostrar-ho de manera concisa es poden apreciar les diferències en les connexions en la taula adjuntada.

Per a entendre amb claredat aquesta taula és recomanable tornar a revisar la figura 1.8 i la figura 1.12, ja que es pot veure clarament els ports que es mencionen en aquesta. Com hem mencionat abans, aquesta adaptació és necessari fer-la per culpa de la falta de pins que tenim a l'Arduino UNO, microcontrolador amb el qual hem simulat al Proteus.

Aquest pas d'adaptació de connexions també necessita una adaptació del codi en consonància, una vegada realitzada ja funcionarà correctament.

Connexions amb Arduino Mega 2560	
FR wheel M+/M-	AK3 Controller channel
FL wheel M+/M-	AK1 Controller channel
RR wheel M+/M-	BK3 Controller channel
RL wheel M+/M-	BK1 Controller channel
A channel ENA, ENB	Digital PWM pin 12, 13
B channel ENA, ENB	Digital PWM pin 10, 11
Channel A IN1, IN2, IN3, IN4	Digital pin 49, 48, 51, 50
Channel b IN1, IN2, IN3, IN4	Digital pin 41, 40, 43, 42

Taula 2.9. Taula de connexions de les rodes amb el driver i el driver amb l'Arduino Mega 2560

Una vegada realitzat l'aclaraciment podem connectar els cables corresponents i adaptar el codi anterior als pins actuals. Seguidament adjunto una imatge on es pot veure el robot amb les connexions actualitzades a aquest pas.

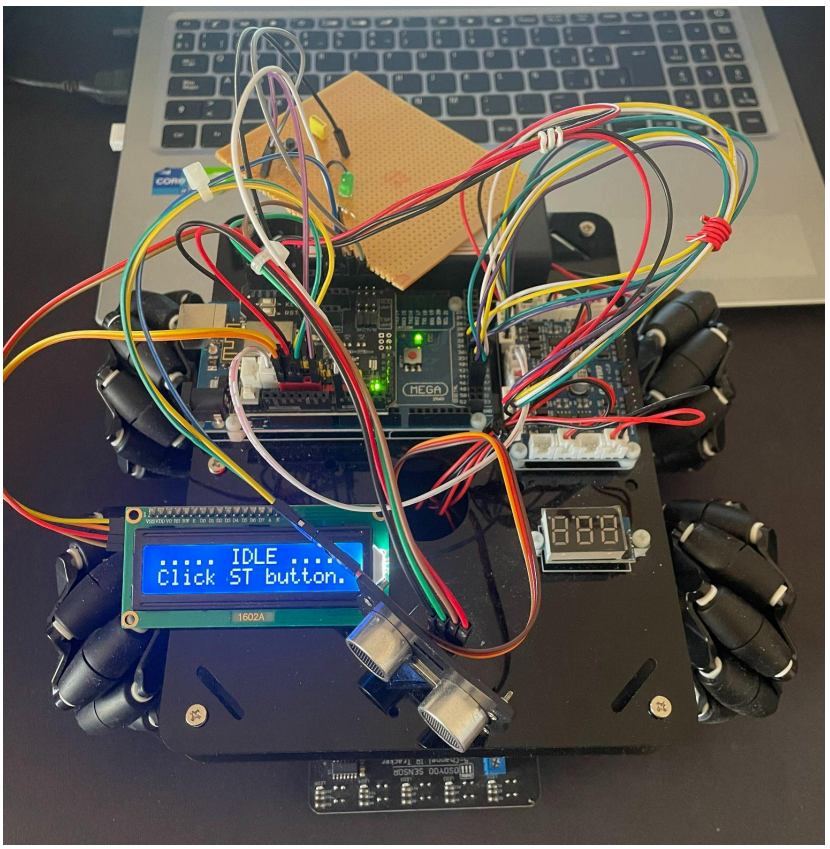

Figura 2.46. Robot amb evasió d'obstacles en funcionament real

Per a demostrar que el robot funciona correctament, es pot apreciar en la següent seqüència d'imatges la trajectòria que traça el robot en funcionament real. Cal remarcar que el robot és capaç de detectar obstacles com parets o armaris, però obstacles com les potes d'una taula, per exemple, no té perquè detectar-les, ja que no té la suficient resolució com per a detectar obstacles d'una amplada inferior a uns 40 cm aproximadament.

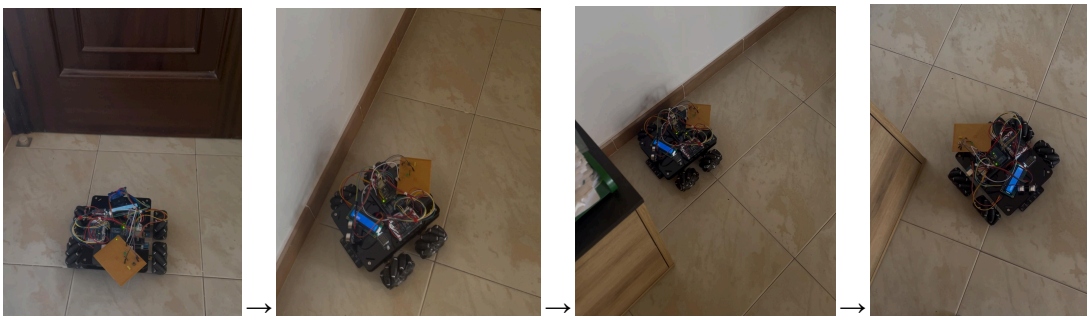

Figura 2.47. Trajectòria traçada pel robot amb evasió d'obstacles en un entorn rea

CAPÍTOL 3. MODE DE RÀDIO CONTROL PER BLUETOOTH

En aquest apartat modificarem el robot per tal que pugui treballar en un nou mode independent. El robot tindrà dos modes de funcionament, el primer es tracta del que ja hem realitzat, l'evasió d'obstacles de manera automàtica. El segon mode de funcionament serà el control a distància mitjançant Bluetooth. Ho duem a terme a partir d'una variable que s'afegirà, permetent així el desdoblament cap a un altre cicle d'estats, que permeti que aquest mode sigui independent del que ja s'ha aconseguit.

En aquesta secció és necessària la instal·lació del software *Virtual Serial Port Driver* per a simular la connexió Bluetooth entre els terminals transmissor i receptor a Proteus.

3.1. Especificacions

Per a realitzar aquesta funcionalitat del robot és necessari incorporar un mòdul Bluetooth que aporta el fabricant, es tracta de l'*HC01*, que connectarem directament al *shield* que ens aporta també el fabricant (i que hem utilitzat fins ara). Aquest mòdul es comunica per UART amb l'Arduino, així que si el mòdul rep una dada, en el cas d'estar en mode esclau, la comunica a l'Arduino mitjançant el protocol mencionat, seguidament l'Arduino s'encarrega de processar el missatge i executar l'acció indicada en el codi. Podem veure una il·lustració del mòdul que utilitzarem a la figura 1.14.

La funcionalitat dels pins ja s'ha especificat al punt 1.2.3. Per fer possible la comunicació haurem de fer servir dos pins extres de l'Arduino MEGA 2560, tot aprofitant la més que suficient quantitat de parells de pins de comunicació que té aquest model d'Arduino, que recordem que és en el que estem basant el nostre projecte, tot i fer les simulacions de Proteus amb l'Arduino UNO per qüestió de llicències.
Seguidament afegirem aquest dispositiu a l'esquema general del sistema, que quedaria de la següent manera:

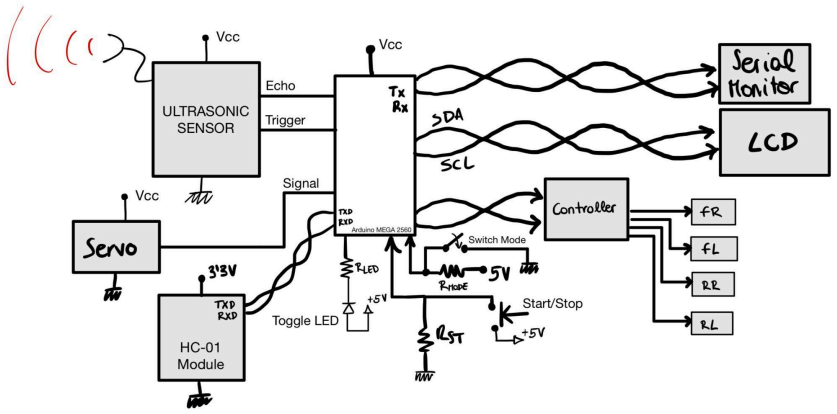

Figura 3.1. Esquema general del sistema complet

Com he comentat s'ha d'afegir un switch extern que controli una variable nova que s'implementarà, aquesta és la que controla el mode en el qual funcionarà el robot, depenent de la posició d'aquest. Es tracta de la variable *var_Mode_flag,* funcionarà de la següent manera:

var_Mode_flag = 0 → Mode d'evasió d'obstacles automàtic.

var_Mode_flag = 1 → Mode de radiocontrol mitjançant Bluetooth.

Per marcar de manera estable la freqüència amb la qual el mòdul Bluetooth llegeix i executa comandes, definim un nou flag relacionat directament amb la variable *var_SP_flag*, inicialitzada abans de la decisió dels modes. Aquesta nova variable l'anomenarem *var_BTSample_flag*, i s'activarà cada 0,5 segons. Seguirà la mateixa lògica que la *var_Servo_flag* de la figura 2.22.

Perquè aquestes modificacions tinguin efecte sobre el disseny existent hem d'actualitzar el diagrama d'estats, marcant així la decisió del robot segons el valor d'aquesta última variable.

En el diagrama d'estats actualitzat hi podem notar que hi ha una fase sencera nova, totalment independent de l'anterior, que anava des de l'estat *Wait for Servo flag*, fins a *Move*, els estats nous, que conformen aquest nou mode de funcionament són els següents:

- *Initialize Module:* Aquest estat serveix per inicialitzar el nou Serial pel qual rebrem les comandes del mòdul bluetooth de recepció.

- *Wait for Sample period:* Aquest nou estat serveix per esperar el període de mostreig, que ve indicat per la variable *var_BTSample_flag*. Fa la mateixa funció que l'estat *Wait for Servo flag* al mode de funcionament d'evasió d'obstacles.

- *Read Command:* La comanda del Serial es guarda en una variable, anomenada receivedCommand, del tipus char, ja que hem definit que les comandes a rebre seran Forward (F), Backward (B), Right (R), Left (L) i Stop (S).

- *Execute Command:* Aquest estat s'encarrega d'assignar una acció per l'Arduino depenent de la variable *receivedCommand* que ha rebut, si rep 'F', cridarà a la funció *moveForward()*, amb la mateixa velocitat en totes les funcions de moviment.

Ara podem apreciar el diagrama d'estats actualitzat, tal com s'acaba de descriure.

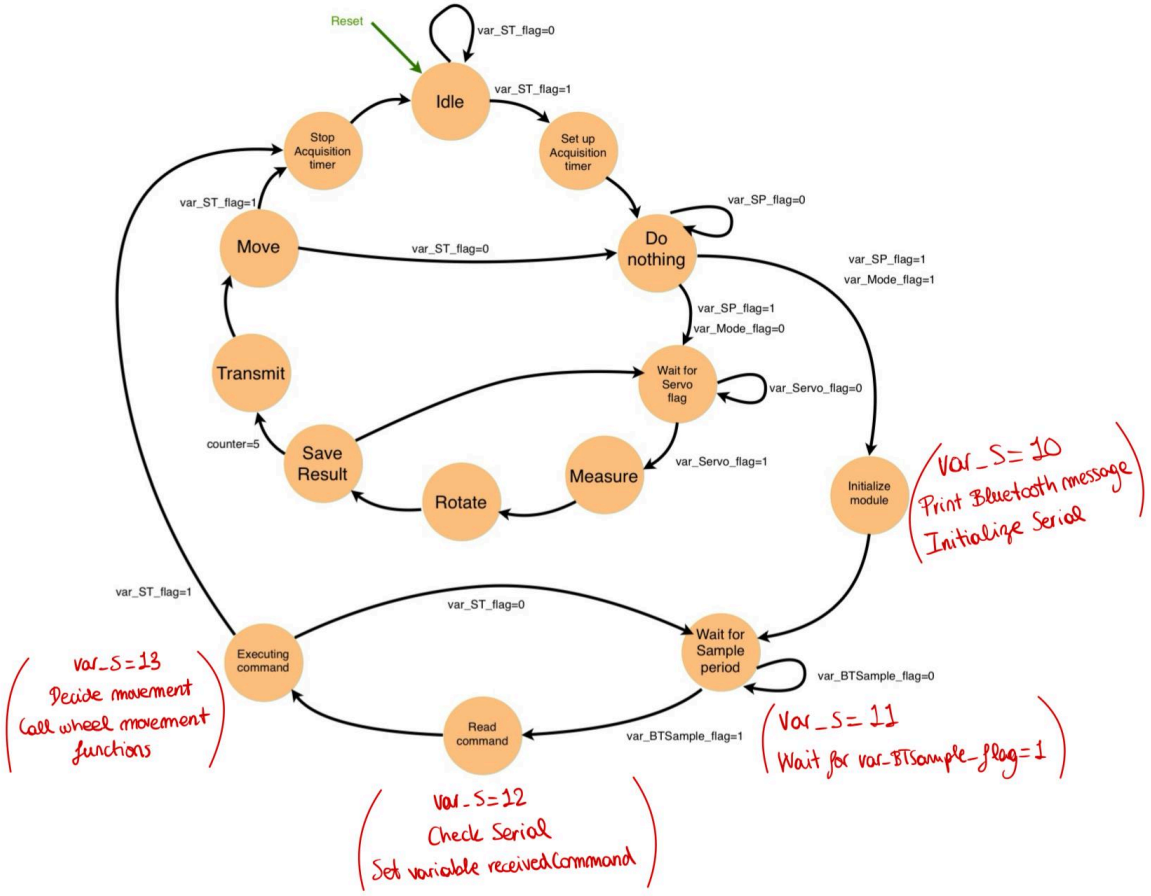

Figura 3.2. Diagrama d'estats complet

Primerament, per a veure de manera clara la nova variable *var_BTSample_flag* podem afegir-la al diagrama temporal de la figura 2.21.

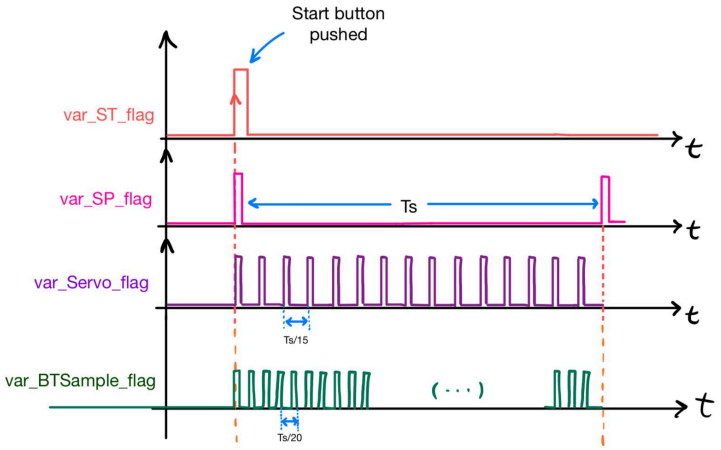

Figura 3.3. Diagrama de temps representant la variable var_BTSample_flag

3.2. Planificació del projecte

A causa del redisseny del diagrama d'estats també hem d'actualitzar les taules de la veritat, que ara sí que seran les definitives. Es poden apreciar les taules de *state_logic()* i d'*output_logic()* a continuació, respectivament.

counter	var_Servo_flag	var_SP_flag	var_Mode_flag	var_BTSample_flag	var_ST_flag	var_current state		var current state +
X	X	X	X	X	0	Idle		Idle 🔄
X	X	X	X	X	1	Idle		Set up acquisition timer ➡️
X	X	X	X	X	X	Set up acquisition timer		Do nothing ➡️
X	X	0	X	X	X	Do nothing		Do nothing 🔄
X	X	1	0	X	X	Do nothing		Wait for Servo flag ➡️
X	X	1	1	X	X	Do nothing		Initialize module ➡️
X	0	X	X	X	X	Wait for Servo flag		Wait for Servo flag 🔄
X	1	X	X	X	X	Wait for Servo flag		Measure➡️
X	X	X	X	X	X	Measure		Rotate➡️
X	X	X	X	X	X	Rotate		Save result ➡️
0, 1, 2, 3 o 4	X	X	X	X	X	Save result		Wait for Servo flag ➡️
5	X	X	X	X	X	Save result		Transmit ➡️
X	X	X	X	X	0	Transmit		Do nothing ➡️
X	X	X	X	X	1	Transmit		Stop Acquisition Timer ➡️

X	X	X	X	X	X	Stop Acquisition Timer		Idle ➡️
X	X	X	X	X	X	Initialize Module		Wait for Sample period ➡️
X	X	X	X	0	X	Wait for Sample period		Wait for Sample period 🔄
X	X	X	X	1	X	Wait for Sample period		Read Command ➡️
X	X	X	X	X	X	Read Command		Executing Command ➡️
X	X	X	X	X	1	Executing Command		Stop Acquisition Timer ➡️
X	X	X	X	X	0	Executing Command		Wait for Sample Period ➡️

Taula 3.1. Taula de la veritat final de la funció state_logic()

var_current_state		var_S	var_SLED	LCD display	Sensor	Servo	Wheel movement	Bluetooth Module
Idle		0	0	Idle message	-	-90°	None	-
Set up acquisition timer		1	0	Idle message	-	-90°	None	-
Do nothing		2	blinking	Previous obstacle results	-	previous position	None	-
Wait for Servo flag		3	blinking	Previous obstacle results	-	previous position	None	-

Measure		4	blinking	Previous obstacle results	Trigger ON	previous position	None	-
Rotate		5	blinking	Previous obstacle results	-	previous position + 45°	None	-
Save Result		6	blinking	Previous obstacle results	-	previous position	None	-
Transmit		7	blinking	Updated obstacle results	-	previous position	None	-
Move		8	0	Previous obstacle results	-	previous position	Decision and movement	-
Stop Acquisition Timer		9	blinking	Previous message	-	-90°	None	-
Initialize Module		10	blinking	Bluetooth Mode message	-	-90°	None	Initializing
Wait for Sample period		11	blinking	Bluetooth Mode message	-	-90°	None	Linking
Read Command		12	blinking	Bluetooth Mode message	-	-90°	None	Linked
Executing Command		13	blinking	Bluetooth Mode message	-	-90°	Decision and movement	Linked

Taula 3.2. Taula de la veritat final de la funció output_logic()

A continuació afegim al diagrama de flux de la funció *state_logic()* amb els estats concebuts en aquest últim apartat.

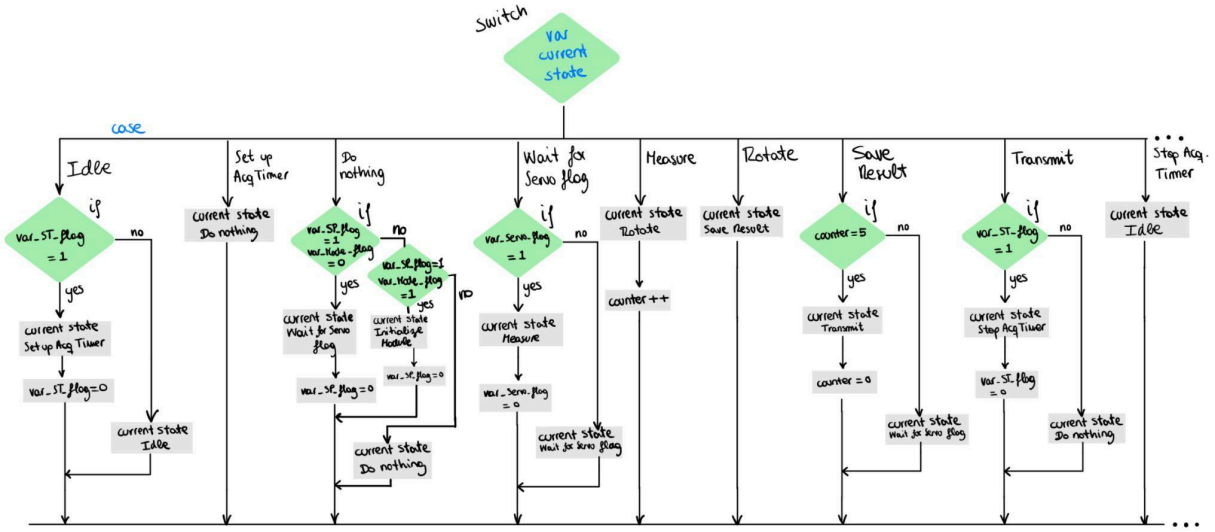

Figura 3.4. Diagrama de flux de la funció state_logic() final part 1

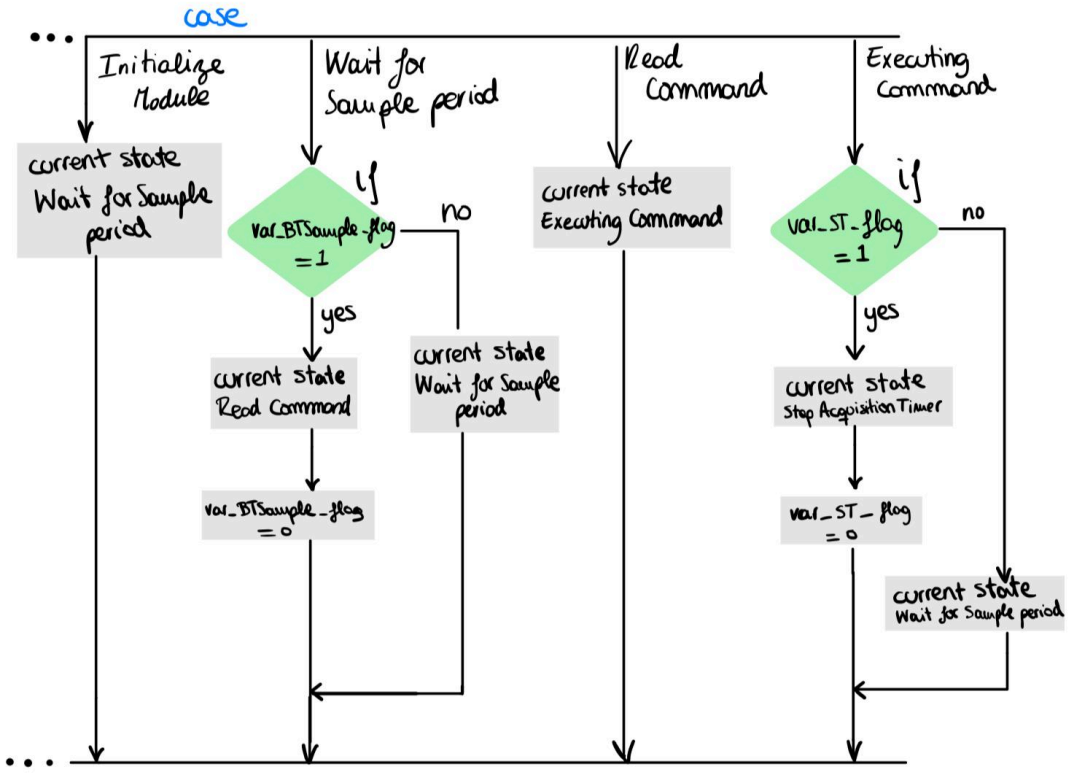

Figura 3.4. Diagrama de flux de la funció state_logic() final part 2

Ara concebem el diagrama de flux de la funció *output_logic()*, que també ha sofert alguns canvis apreciables en les figures.

Cal destacar que els asteriscs 1, 2 i 3 es troben explicats a les figures 41, 42 i 43.

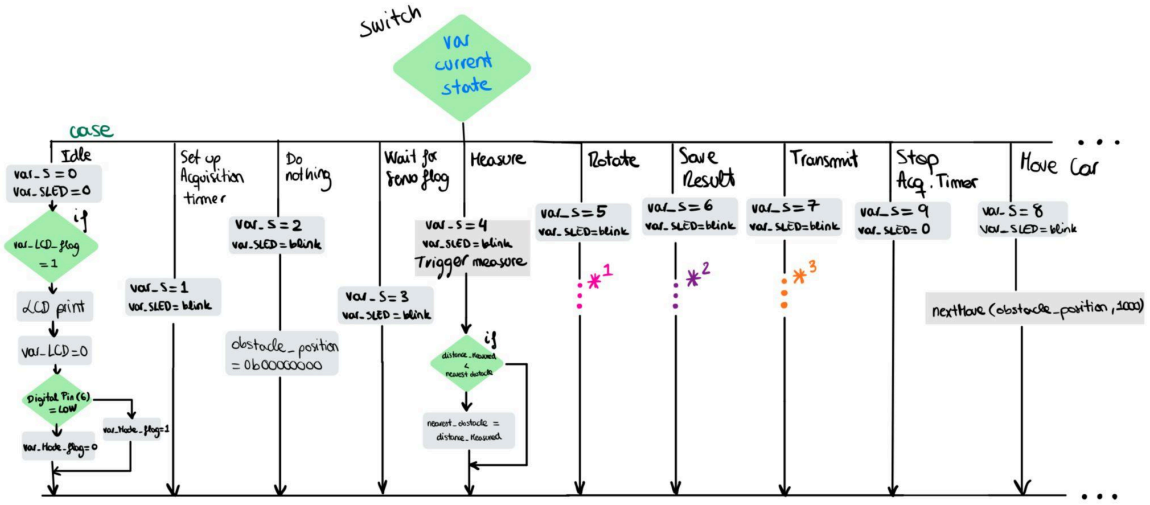

Figura 3.5. Diagrama de flux de la funció output_logic() final part 1

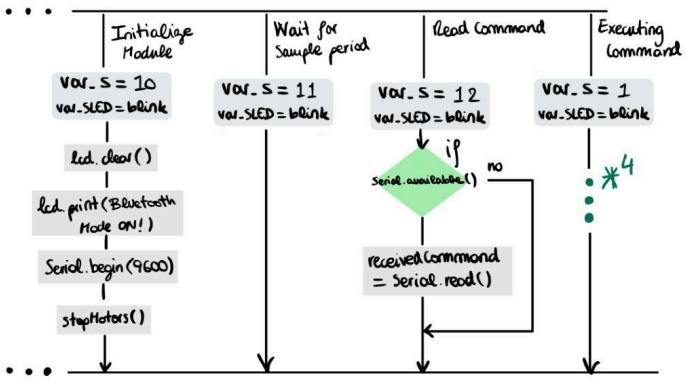

Figura 3.5. Diagrama de flux de la funció output_logic() final part 2

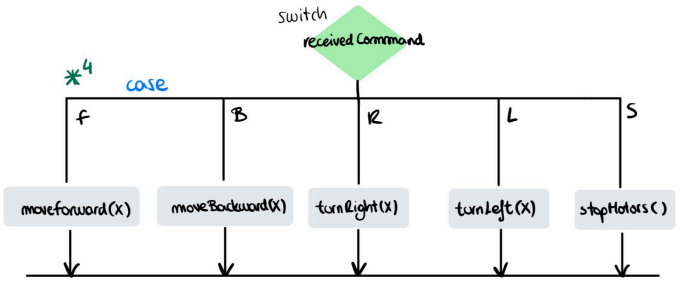

Figura 3.5. Diagrama de flux de la funció output_logic() final part 3

Finalment podem fer una representació *hardware/software* del sistema global, similar al de la figura 2.9. Aquest diagrama ens ajudarà molt a tenir una vista general de tots els mòduls, components externs, variables i flags que s'utilitzen per fer funcionar tot el sistema de manera adequada.

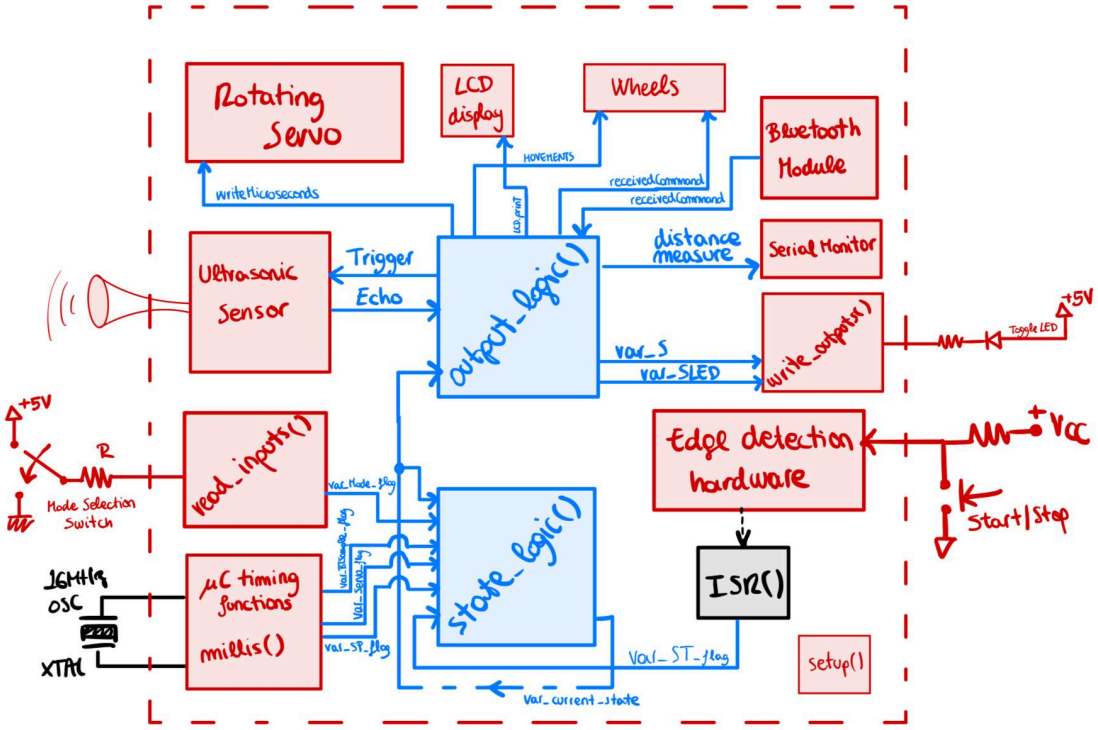

Figura 3.6. Diagrama hardware/software del sistema final

3.3 Desenvolupament i test

Una vegada hem concebut els diagrames de flux de manera adequada, seguint el mateix ordre que el que hem seguit amb els altres apartats d'implementació, podem procedir a la simulació amb el simulador Proteus.

3.3.2. Hardware

Tot i que sabem que el mòdul HC-05 no és el que ens ve per defecte en el kit d'aprenentatge, és el més adequat que podem utilitzar, ja que els pins disponibles són els mateixos que en el HC-01, l'obtingut amb el kit d'aprenentatge, en aquest cas serà una bona aproximació al mòdul real.

El hardware a Proteus es veu de la següent manera:

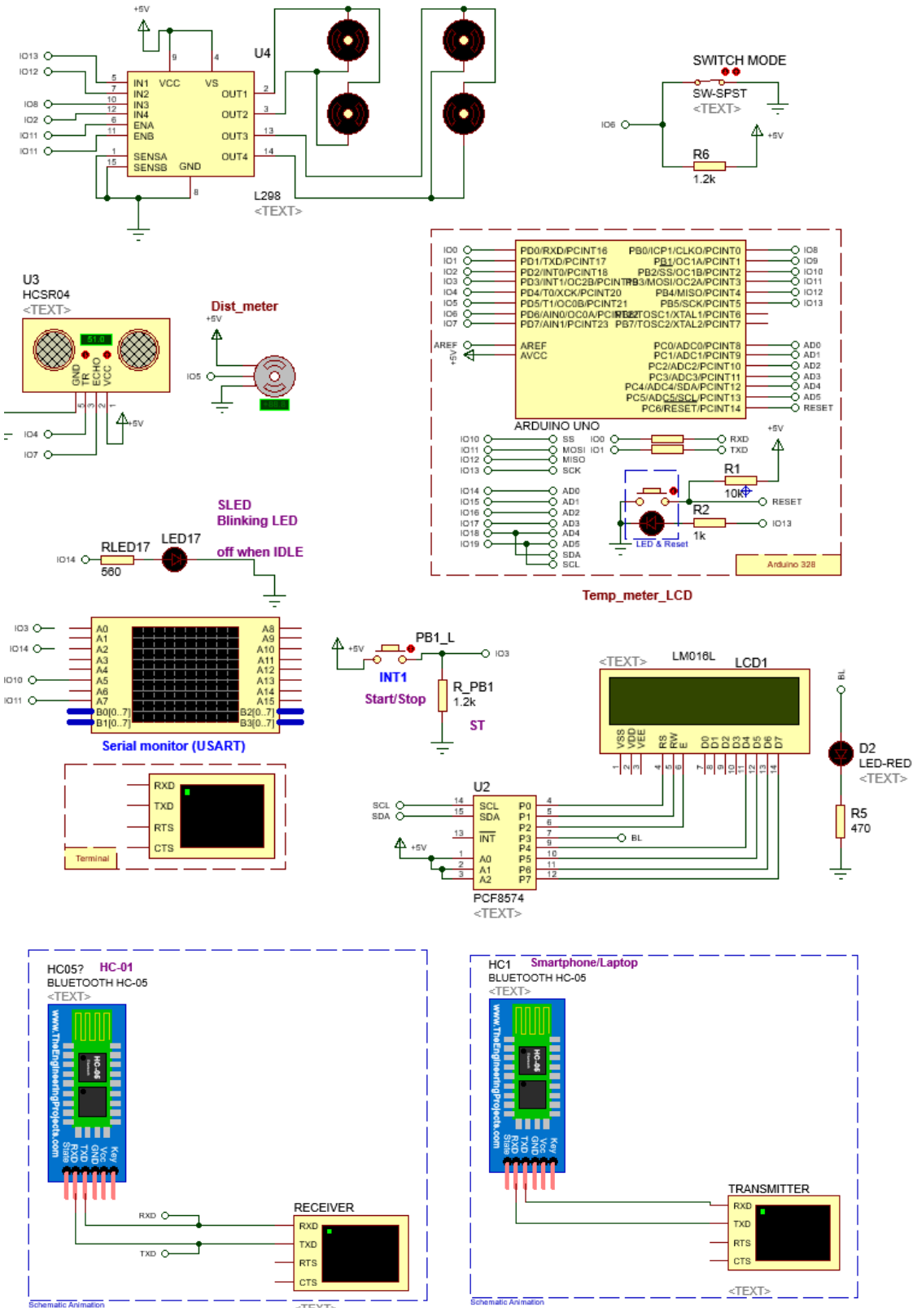

Figura 3.7. Hardware de la versió final en simulació a Proteus

Es pot veure els dos nous mòduls implementats. Un d'ells està simulant el mòdul Bluetooth d'un *smartphone* o ordinador portàtil, mentre que l'altre fa referència al mòdul connectat

directament a l'Arduino, el transmissor i receptor respectivament, fent així possible la comunicació entre els dos terminals.

3.3.3. Programari

Seguidament mostraré com codifico els estats, en la figura 3.8 es pot veure com s'inicialitza el mòdul Bluetooth, espera al flag i llegeix si hi ha alguna comanda disponible per a llegir en l'estat *Read Command*.

```
case INIT_MODULE:
    var_S = 10;
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Bluetooth Mode");

    lcd.setCursor(0,1);
    lcd.print("ON");
    //SoftwareSerial bluetooth(9, 10);
    Serial.begin(9600);
    //bluetooth.begin(9600);
    stopMotors();

break;

case WAIT_SAMPLE_PERIOD:
    var_S = 11;

break;

case READ_COMMAND:
    var_S = 12;
    if(Serial.available()){ //Si hi ha dades
     receivedCommand=Serial.read();
     //Serial.print("He rebut: ");
     //Serial.println(receivedCommand);
```

Figura 3.8. Codificació estats Initialize Module, Wait sample period i Read Command

També podem veure la selecció de direcció en funció de la variable *receivedCommand* obtinguda en l'estat *Read Command.*

```
case EXECUTING_COMMAND:
var_S = 13;

switch(receivedCommand){
case 'F':
moveForward(125);
break;

case 'B':
moveBackward(125);
break;

case 'R':
turnRight(125);
break;

case 'L':
turnLeft(125);
break;

case 'S':
stopMotors();
break;
}

break;
```

Figura 3.9. Codificació de selecció de direcció en funció de la comanda Bluetooth obtinguda

Perquè la comunicació Bluetooth en la simulació sigui possible serà necessari que iniciem el programa *Configure Serial Port Driver* per a configurar els ports COM1 i COM2 i assignar-los a cada *Virtual Terminal*, cal especificar que sense aquest pas la simulació no serà efectiva.

Quan comencem la simulació ens apareixen els *Virtual Terminals* del transmissor i receptor, en aquest punt és quan escriurem una de les comandes en el terminal transmissor (F, B, R, L o S), simulant així les comandes que podria rebre des d'un terminal mòbil. La comanda es replicarà al terminal receptor, serà detectada per l'Arduino i executarà les funcions que mouen les rodes per a realitzar el moviment desitjat. A la següent figura podem veure un exemple dels terminals.

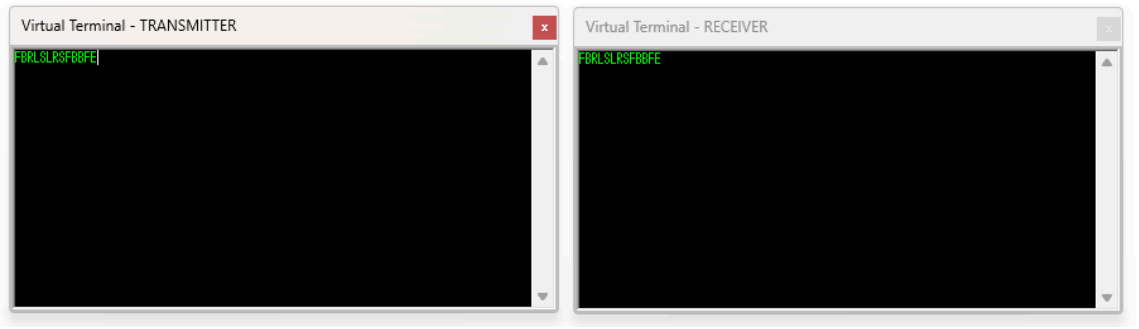

Figura 3.10. Terminal transmissor (smartphone) i terminal receptor (mòdul HC-01)

També podem fixar-nos en el LCD display per a veure que el programa entra en els estats que ha de recórrer, ja que indica quan estem en el mode de control per Bluetooth.

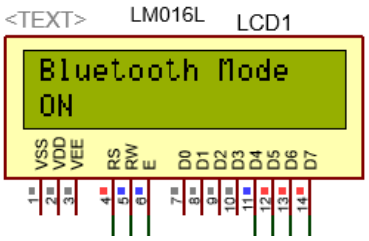

Figura 3.11. LCD display mostrant el missatge conforme entra correctament al mode Bluetooth

Finalment, podem veure la Watch window amb les variables utilitzades en el projecte i el sistema al complet en funcionament.

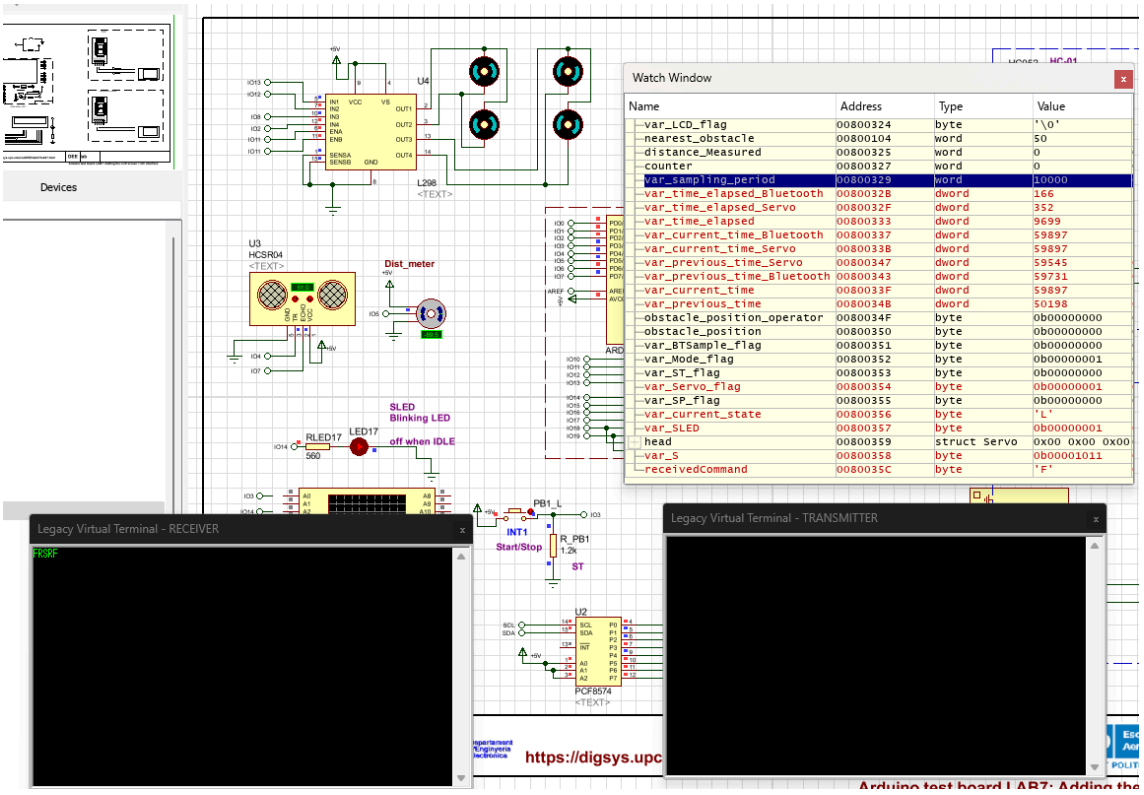

Figura 3.12. Sistema complet funcionant a Proteus

Una vegada ja funciona degudament al simulador podem seguir amb la implementació real del sistema.

3.4. Prototipatge

El següent pas és intentar portar el disseny a la realitat, primer de tot és necessari soldar el nou component per elegir el mode de funcionament (switch) a la placa universal que es pot apreciar a la figura 2.33. Seguidament es pot apreciar la versió final de la placa universal, amb tots els components externs soldats.

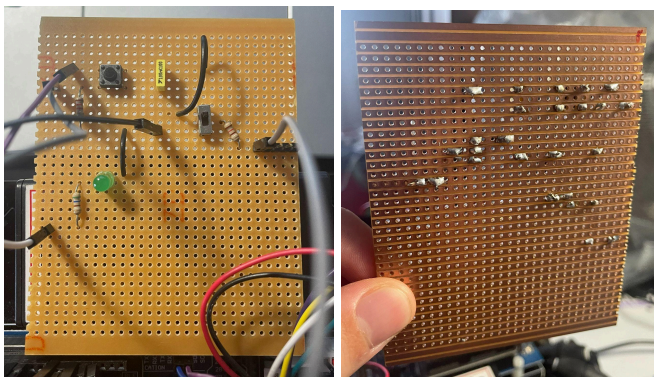

Figura 3.13. Tots els components soldats a la placa universal

El següent pas consisteix en la integració de l'última actualització del robot a tot el sistema. Una vegada hem adaptat el codi perquè funcioni amb l'Arduino MEGA 2560 (es poden apreciar les diferències principals en el codi adjunt a l'annex), encenem el robot per a comprovar que tot funcioni correctament.
Una vegada hem compilat i pujat el codi a l'Arduino ens trobem que, quan el switch el tenim en la posició "HIGH" accedeix correctament a l'Estat que nosaltres desitgem, per tant, romandrà a l'espera fins que rebi una comanda.

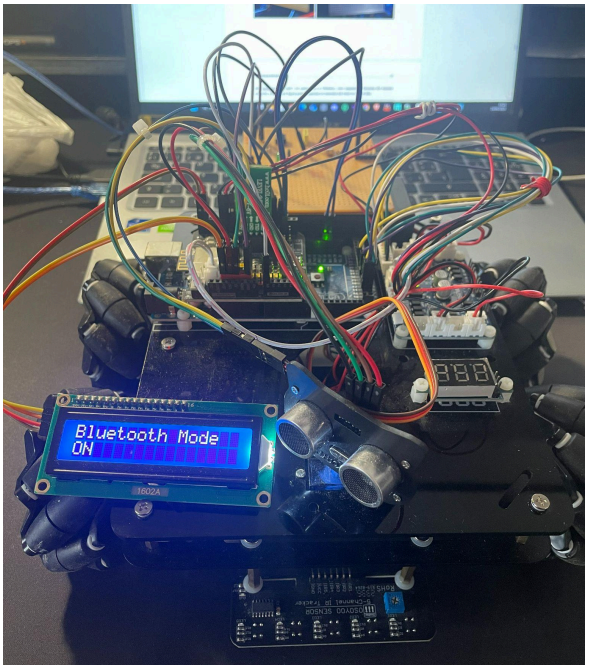

Figura 3.14. Robot final en el mode Bluetooth en funcionament real

Com bé indica el *LCD display* ha accedit al mode Bluetooth amb èxit. Ara simplement haurem de programar l'aplicació. Disponible un diagrama de connexions simplificat de l'Arduino amb els dispositius i components auxiliars a l'annex 3. També podem trobar el codi sencer compilat i pujat a l'Arduino Mega 2560 del sistema complet a l'annex 5.

3.5. Programació de l'aplicació

Per a fer aquest punt el més fàcil possible he utilitzat una plataforma gratuïta i visual creada pel *Massachussetts Institute of Technology (MIT)*, que permet a qualsevol persona crea una aplicació de manera fàcil i intuïtiva. Com que la nostra aplicació no ha de ser gens enrevessada, aquesta eina funciona molt bé, ja que es basa en la programació per blocs que representen funcions i estructures lògiques. Aquesta eina ens permet també descarregar l'aplicació final en format *.apk*, que posteriorment instal·larem al nostre *smartphone*.

El codi està disponible a l'annex 4, en format blocs i en format codi convencional. Podem veure la interfície gràfica de l'aplicació en la següent figura, tenint en compte que es tracta d'una aplicació que simplement ha de servir per a controlar el robot.

Figura 3.15. Front-End de l'aplicació

Una vegada ja tenim llesta l'aplicació per enviar peticions amb mòdul Bluetooth que hem instal·lat a l'Arduino (existent en la caixa original del kit d'electrònica), podem procedir a comprovar el funcionament real del robot en els dos modes, seguidament es troba un link amb un vídeo on s'ensenya breument el funcionament del robot (Nebot Parra, juliol 2025).

3.6. Integració del sistema en una PCB

Com a part final del projecte s'ha dissenyat un *shield* per l'Arduino Mega 2560 mitjançant el software KiCad. Aquest *shield* integra totes les connexions necessàries per als components electrònics externs que utilitzem, en el nostre cas, el LED que ens serveix per veure que el robot està funcionant correctament, el switch per a seleccionar el mode de funcionament, el botó Start/Stop perquè el robot comenci o pari de funcionar, el suport i connexió del *LCD display* mitjançant el protocol *I2C* i també les connexions necessàries amb el mòdul Bluetooth. Finalment, es pot apreciar el disseny realitzat específicament per a aquest projecte.

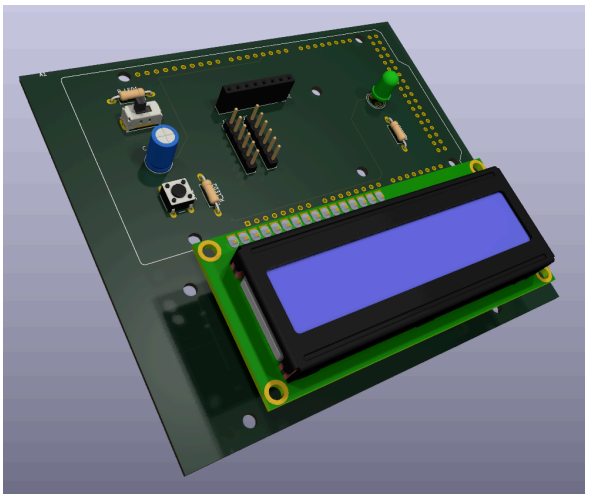

Figura 3.16. Disseny de shield amb KiCad part 1

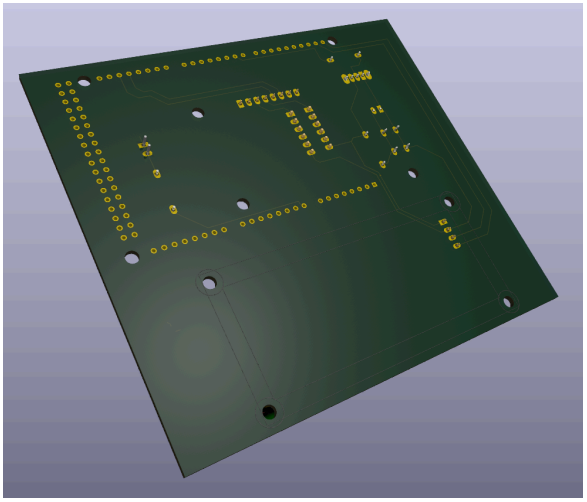

Figura 3.16. Disseny de shield amb KiCad part 2

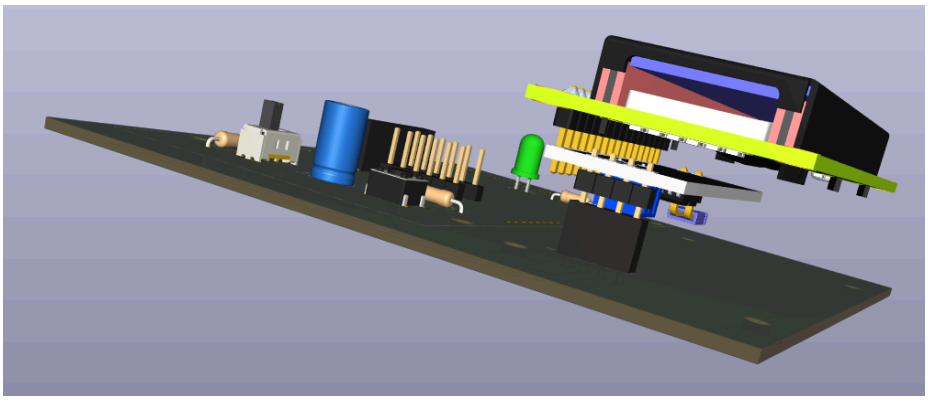

Figura 3.16. Disseny de shield amb KiCad part 3

Estudi de sostenibilitat

Amb aquest projecte tenim tres tipus d'impactes a valorar, es tracta de l'impacte econòmic, social i mediambiental.

Pel que fa a l'estudi econòmic, aquest projecte aprofita un kit comercial assequible com a base per al disseny i programació, això fa que es redueixin considerablement els costos del projecte, ja que no ho hem de fabricar des de zero. També és important destacar que el pas referent al disseny de la PCB amb el KiCad implica també una fabricació de la mateixa i, per tant, un possible impacte econòmic major.

Considerant l'impacte social, clarament promou una activitat acadèmica, ja que aquest projecte ha estat pensat i dissenyat per a implementar-se en el pla docent de la pàgina *Design of Electronic Equipment* (Departament d'Enginyeria Electrònica - UPC, n.d.). Es tracta d'un projecte pensat per ampliar els coneixements dels estudiants i que puguin anar un pèl més lluny en quant al disseny, programació i fabricació de sistemes electrònics, sempre seguint la mateixa estructura aplicada a l'assignatura de Circuits i Sistemes Digitals impartida a l'Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels.

Finalment tenint en compte l'impacte mediambiental és un projecte que no té gaire impacte en negatiu en aquest aspecte, ja que aquestes són tres característiques que el defineixen:

- Reutilització de materials: Torno a reiterar que aquest projecte es duu a terme a partir d'una base, un kit d'aprenentatge d'electrònica que en cap cas ha estat fabricat amb l'objectiu de fer aquest treball de fi de grau, sinó que l'empresa fabricant ven al seu catàleg de manera general.
- Baix consum energètic: Els components electrònics que s'han utilitzat tenen un consum molt baix d'energia, de fet, aquest kit està alimentat per un parell de piles recarregables, cosa que redueix de manera notable els residus.
- Durabilitat, escalabilitat i fàcil manteniment: Aquest projecte ha estat dissenyat de manera modular, per parts i a partir de fer un projecte petit, cada vegada més gran afegint-li components, això fa que si hi hagués algun problema de funcionament es pugui trobar de manera analítica els possibles errors. Això també ens porta a un projecte el qual podria ser la base de projectes futurs, col·laborant així en la reducció de residus i reutilització.

Per acabar aquest breu anàlisi de sostenibilitat puc dir que aquest projecte és molt bo en quant a l'impacte econòmic, social i mediambiental, ja que permet la seva reutilització, està alimentat per una font d'energia recarregable, i el més important, té un fi clarament educatiu.

Conclusions i possibles millores

Com bé s'indica al llarg del treball s'ha seguit en tot moment l'esquelet de programació d'una Finite State Machine, mentre en cada cas s'ha respectat el format per fases (Especificacions, Planificació, Desenvolupament i test i Prototipatge) que s'utilitza a l'assignatura de Circuits i Sistemes Digitals. Aquests aspectes han fet possible que el projecte no només sigui molt amigable pels estudiants als qui va orientat, per aprendre i poder implementar el mateix sistema o similars, sinó que també l'ha fet un projecte escalable i fàcil d'entendre per a qualsevol estudiant que no hagi cursat Circuits i Sistemes Digitals.

En aquest document s'han assolit diverses fites considerables i molt profitoses per als alumnes, com per exemple:

- Sistema de mesura de distància: Es tracta del primer projecte dut a terme.
- Sistema de control d'un servo per a analitzar una certa superfície: Citem el segon projecte elaborat.
- Sistema de processament de les dades obtingudes de diferents direccions per a la decisió de la direcció a prendre mitjançant les Mecanum Wheels: En el tercer projecte ho hem vist.
- Sistema complet anterior amb un mode de control Bluetooth afegit: Ho podem trobar al quart projecte.

Tots aquests punts han anat complimentats de l'enteniment dels senyals que s'han d'enviar als diferents components adjunts a l'Arduino, i l'adaptació a un microcontrolador amb més pins per a permetre el correcte funcionament del robot al complet. També és important ressaltar que s'ha anat més enllà, no només s'ha simulat a l'Arduino, sinó que s'han soldat els components a la placa universal i hem pogut veure el seu funcionament en un escenari real, acabant amb la simulació de *shield* per a fer de suport del LCD i connectar els components com LEDS i switches externs de la manera més còmoda.

Com a millores future shi ha un punt que m'agradaria especificar, es tracta de la resolució en el mode d'*Obstacle Avoidance*, ja que s'ha realitzat amb únicament cinc direccions analitzades, per tal de fer-ho menys feixuc pels alumnes. Això fa que obstacles estrets com podrien ser les potes d'una taula o d'una cadira, no les detecti correctament, tot i que es podria solucionar fent un escombrat amb més direccions, cosa que permet un anàlisi més complet i detallat dels voltants del robot. També vull destacar que és possible també afegir un altre mode de funcionament que permeti al robot seguir una línia negra a terra, ja que comptem amb el sensor pertinent al kit d'aprenentatge.

Finalment, es pot concloure que aquest projecte està perfectament adequat perquè estudiants d'enginyeria puguin realitzar-lo i de manera analítica i per passos, siguin capaços de realitzar-lo amb èxit, i per tant poder aprendre d'un kit d'electrònica per a aficionats.

Bibliografia

[1] Departament d'Enginyeria Electrònica - UPC. (n.d.). *Computer Systems Design (CSD)*. Universitat Politècnica de Catalunya. https://digsys.upc.edu/csd/index.html

[2] Departament d'Enginyeria Electrònica - UPC. (n.d.). *Digital Electronic Design (DEE)*. Universitat Politècnica de Catalunya. https://digsys.upc.edu/csd/DEE/DEE.html

[3] Osoyoo. (2022, juliol). *V2 Metal Chassis Mecanum Wheel Robotic for Arduino Mega2560 – Introduction*. https://osoyoo.com/2022/07/05/v2-metal-chassis-mecanum-wheel-robotic-for-arduino-mega2560-introduction-model-2021006600/

[4] Gudino, M. (2021, febrer 28). *Arduino Uno vs. Mega vs. Micro*. Arrow Electronics. https://www.arrow.com/es-mx/research-and-events/articles/arduino-uno-vs-mega-vs-micro

[5] Generation Robots. (n.d.). *Mecanum wheel application*. https://www.generationrobots.com/media/Mecanum-wheel-application.pdf

[6] Taheri, H., & Taghirad, H. D. (2015). *Kinematic model of a four mecanum wheeled mobile robot*. ResearchGate. https://www.researchgate.net/publication/276344731_Kinematic_Model_of_a_Four_Mecanum_Wheeled_Mobile_Robot

[7] Zaragoza Maker Space. (n.d.). *Curso de Arduino y robótica – Motores paso a paso*. https://zaragozamakerspace.com/courses/arduino-basico/lessons/curso-de-arduino-y-robotica-motores-paso-a-paso/

[8] ETC2. (n.d.). *HC-SR04 datasheet*. AllDatasheet. https://www.alldatasheet.com/datasheet-pdf/view/1132204/ETC2/HCSR04.html

[9] Jameco Electronics. (n.d.). *How Servo Motors Work*. https://www.jameco.com/Jameco/workshop/Howitworks/how-servo-motors-work.html

[10] Colvin, J. (n.d.). *What is an H-Bridge?* Digilent. Recuperado de https://digilent.com/blog/what-is-an-h-bridge/?srsltid=AfmBOorZqn7ZoyXgaSJf_o6JsQh4aivLZ7JorfVTpbcipvltltBx7x_S

[11] Grabowiecki, A. (n.d.). *Drawing of probably the first omni-directional wheel as described in Grabowiecki's US Patent* [Figure]. ResearchGate. https://www.researchgate.net/figure/Drawing-of-probably-the-first-omni-directional-wheel-as-described-in-Grabowieckis-US_fig8_224692506

[12] STMicroelectronics. (n.d.). *L298 datasheet*. AllDatasheet. https://www.alldatasheet.com/datasheet-pdf/view/22437/STMICROELECTRONICS/L298.html

[13] Wu, J. (2022, novembre). *A Basic Guide to I2C* (Application Note SBAA565). Texas Instruments. https://www.ti.com/lit/an/sbaa565/sbaa565.pdf

[14] Nebot Parra, V. (2025, juliol 2). *Adaptació de kits d'aprenentatge de circuits electrònics a l'ensenyament d'enginyeria* [Vídeo]. YouTube. https://www.youtube.com/watch?v=Ya-BwpK-UWY

Annex 1. Diagrama de pins Arduino UNO

A continuació podem apreciar el diagrama de pins de l'Arduino UNO.

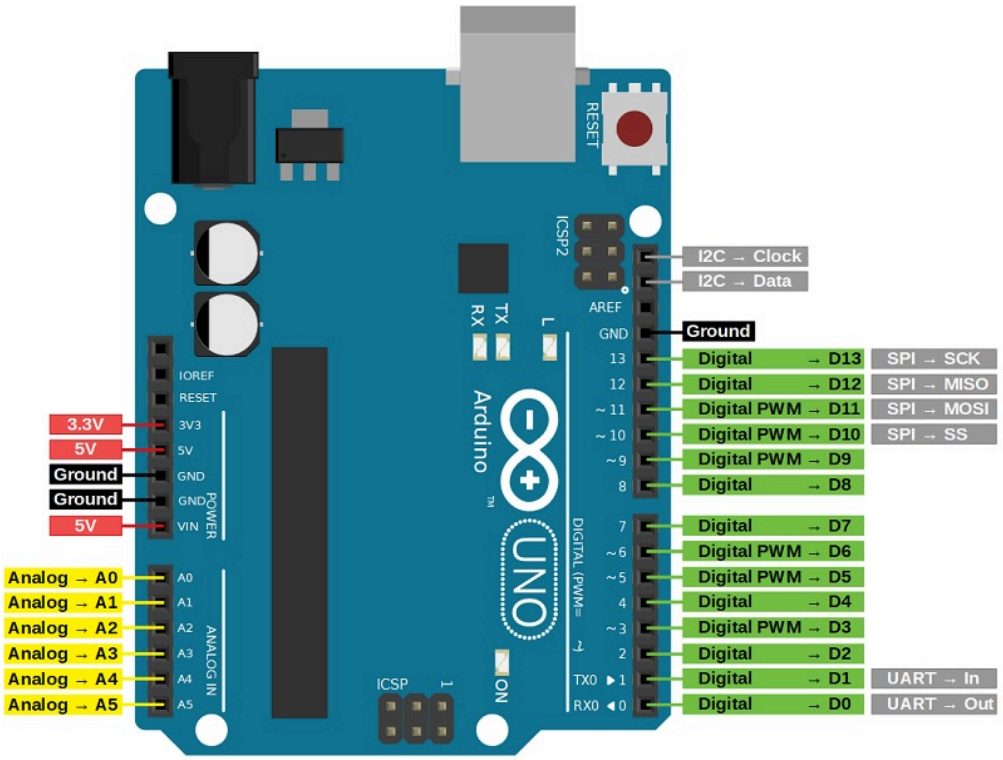

Figura A.1. Diagrama de pins Arduino UNO

Annex 2. Diagrama de pins de l'Arduino MEGA 2560

A continuació trobem el diagrama de pins de Arduino MEGA 2560.

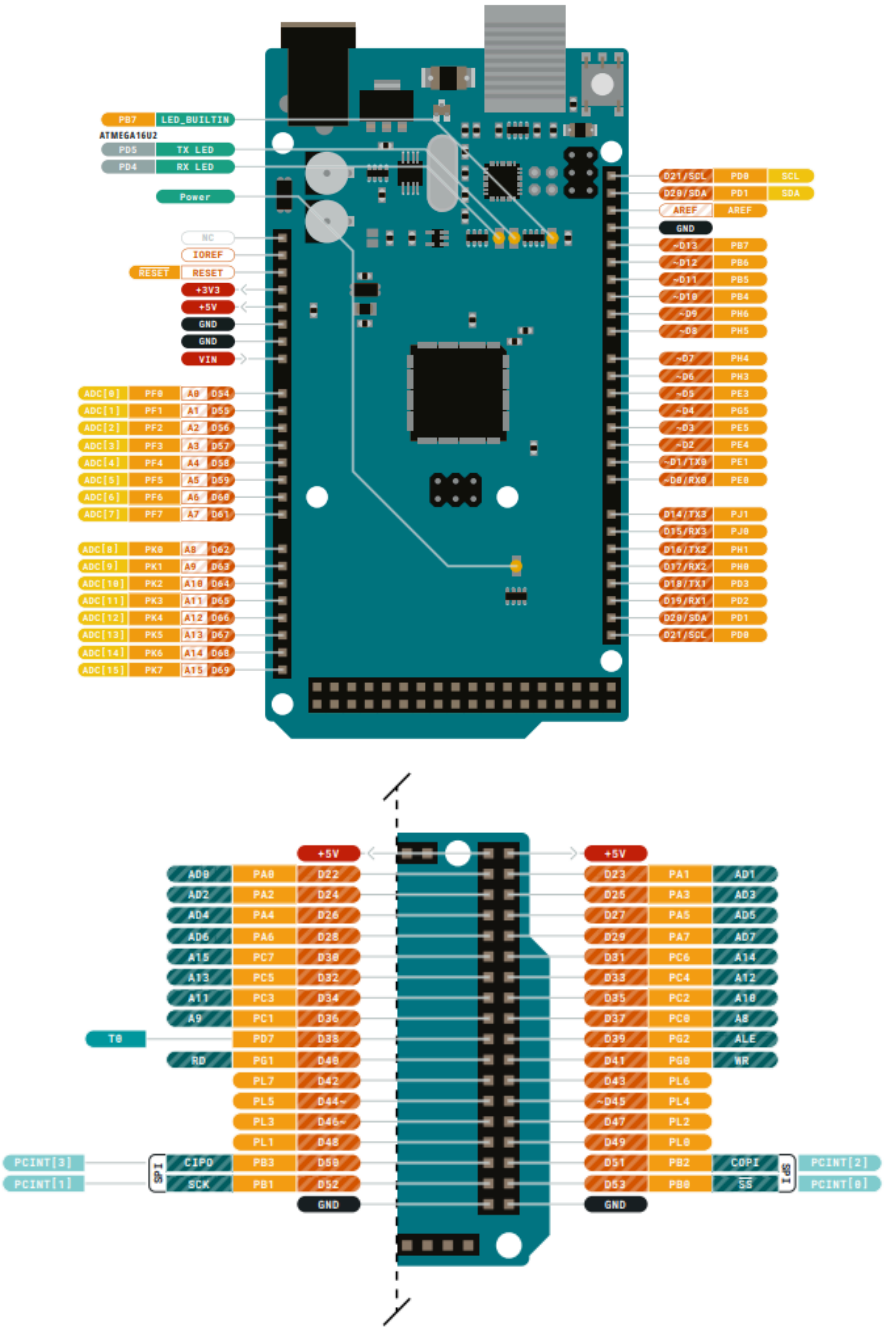

Figura A.2. Diagrama de pins de Arduino MEGA 2560

Annex 3. Diagrama de connexions de l'Arduino MEGA 2560

A continuació es pot apreciar el diagrama de connexions de l'Arduino de manera simplificada.

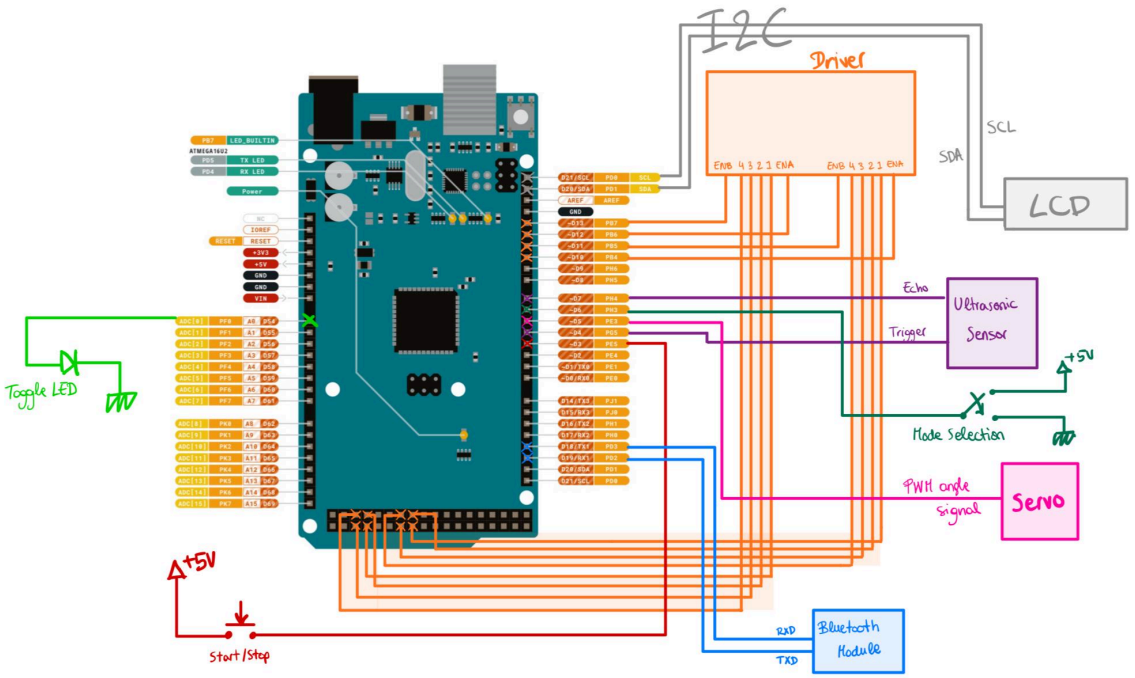

Figura A.3. Diagrama de connexions l'Arduino MEGA 2560

Annex 4. Codi de l'aplicació

En aquest annex podem veure el codi de l'aplicació en format de blocs i en format estàndard.

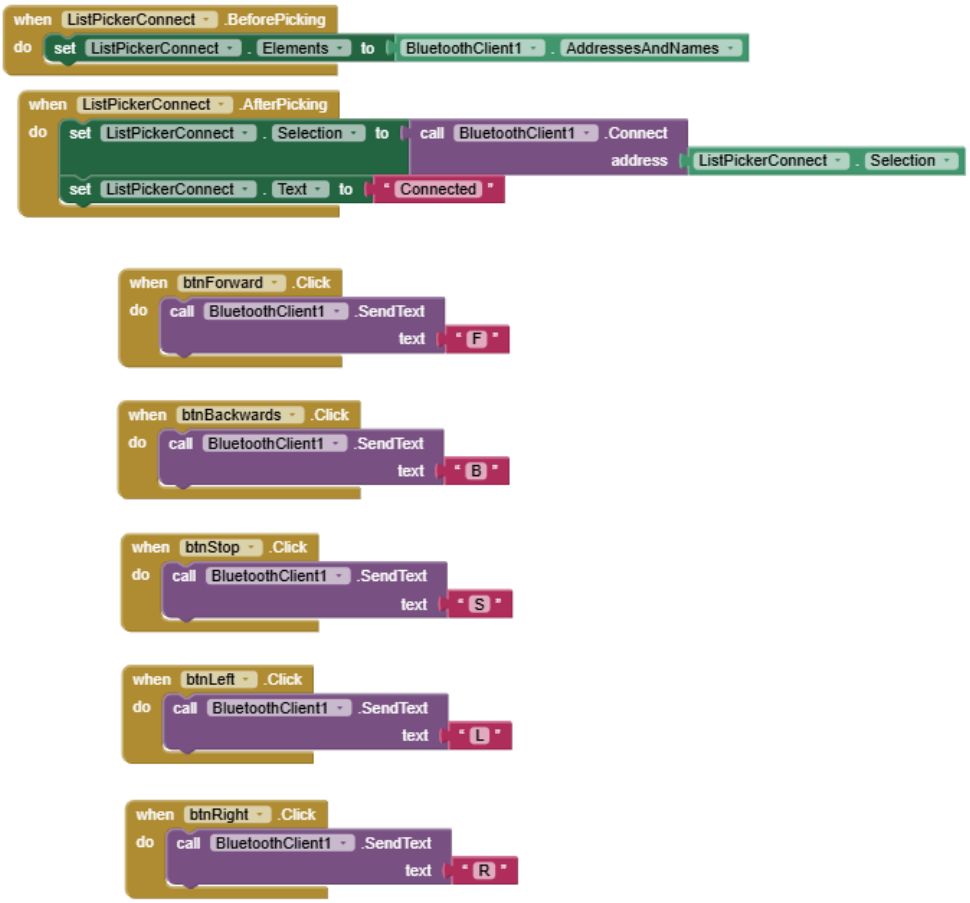

Figura A.4. Codi de l'aplicació en format blocs

```
# Before showing the list of devices
when ListPickerConnect.BeforePicking:
    ListPickerConnect.Elements = BluetoothClient1.AddressesAndNames

# After selecting a device from the list
when ListPickerConnect.AfterPicking:
    BluetoothClient1.Connect(address=ListPickerConnect.Selection)
    ListPickerConnect.Text = "Connected"

# Move Forward
when btnForward.Click:
    BluetoothClient1.SendText("F")

# Move Backward
when btnBackwards.Click:
    BluetoothClient1.SendText("B")

# Stop Movement
when btnStop.Click:
    BluetoothClient1.SendText("S")

# Turn Left
when btnLeft.Click:
```

```
    BluetoothClient1.SendText("L")

# Turn Right
when btnRight.Click:
    BluetoothClient1.SendText("R")
```

Annex 5. Codi del robot

Per últim, s'adjunta el codi del robot en funcionament real, es tracta del robot amb Bluetooth, Sensor panoràmic, moviment i decisió de direccions amb rodes independents. És el codi adaptat per a funcionar amb la plataforma que ens ve inclosa en el kit, amb l'Arduino MEGA 2560.

```
/*
================================================================================
 PC - EETAC - CSD - DEE - http://digsys.upc.edu

 * LAB6: Analogue inputs, analogue sensors
 * Documentation location: https://digsys.upc.edu/csd/DEE/lab06/lab06.html
 * Hardware for prototyping: Arduino Uno board. Chip ATmega328P
 * Similar circuit:        https://digsys.upc.edu/csd/P12/Temp/Temp.html

Planning:

- Copy and adapt the previous project LAB5: FSM with interrupts

- Prepare the hardware, so that adapt the Arduino board to this
particular application. For instance:

    - A/D function. Measure temperature

- Prepare the software. Hardware/software diagram. State diagram

    - Real-time measurement using millis()
       - Sampling period flag.
       - Process and transmit temperatures
================================================================================*
/

#include <Servo.h>
#include <SoftwareSerial.h>
#include <Wire.h>
#include <LiquidCrystal_PCF8574.h>

 // set the LCD address to 0x27 for a 16 chars and 2 line display
LiquidCrystal_PCF8574 lcd(0x27);


//WHEEL MOVEMENT
PINS=============================================================================
=================
//=============================================================================
==================================

const char speedPinFR = 13; //velocitat eix dret
const char speedPinFL = 12; // velocitat eix esquerra
const char speedPinRR = 11;
const char speedPinRL = 10;

const char directionPinFR1 = 50;
const char directionPinFR2 = 51;
const char directionPinFL1 = 48;
const char directionPinFL2 = 49;
const char directionPinRR1 = 42;
const char directionPinRR2 = 43;
const char directionPinRL1 = 40;
const char directionPinRL2 = 41;

char receivedCommand = 0;
```

```
const char pin_Echo = 7; ////////pins conectats al sensor ultrasònic (pins
digitals)
const char pin_Trig = 4;

const char pin_US_Servo = 5; //////pin PWM connectat al servo motor que mou el
sensor ultrasònic

// Start-stop push-button. Pin PD3 is also the microcontroller's external INT1
const char INT1_pin_ST_BUTTON = 3;  //No canvio

// LED for generating the 1 Hz squared waveform when acquiring
const char pin_SLED = 54; ////No canvio
const char switchMode = 6;

// FSM list of states
// FSM list of states
const char IDLE            = 'A';
const char SET_ACQ_TIMER   = 'B';
const char Do_NOTHING      = 'C';
const char Wait_Servo_Flag      = 'D';
const char DIST_MEAS       = 'E';
const char Rotate          = 'F';
const char Save_Result           = 'G';
const char TRANSMIT        = 'H';
const char MOVE_CAR        = 'I';
const char STOP_ACQ_TIMER  = 'J';
const char INIT_MODULE           = 'K';
const char WAIT_SAMPLE_PERIOD    = 'L';
const char READ_COMMAND          = 'M';
//const char PROCESSING_COMMAND  = 'N';
const char EXECUTING_COMMAND     = 'N';

// Default timer period to sample analogue inputs
const unsigned long SAMPLING_PERIOD = 10000; // 10000 ms   (0.1 Hz sampling
frequency)//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////
const int threshold = 50; //limit de distancia en cm per considerar que hi ha
un obstacle al
davant//////////////////////////////////////////////////////////////////////////
/////////////////////////////////////
/*
=============================================================================
      Global RAM variables
      =============================================================================
*/
Servo head;

char         var_S;                    // For representing the state in
binary
char         var_SLED;           // Squared waveform when running

char   var_current_state ;    // This is the key variable: FSM state memory

char   var_SP_flag; // Sample period flag
char   var_Servo_flag;
char   var_ST_flag;    // It will be set when one click detected on interrupt
char   var_Mode_flag; //Bluetooth flag
char   var_BTSample_flag; //Bluetooth flag

char   var_USART_flag; // For writing or sending ASCII messages only when

char   obstacle_position;
char   obstacle_position_operator;                                    //
new information


// For timing the square waveform:
static unsigned long var_previous_time;
```

```
static unsigned long var_previous_time_Servo;
static unsigned long var_previous_time_Bluetooth;
static unsigned long var_current_time;    // current time in ms
static unsigned long var_current_time_Servo;
static unsigned long var_current_time_Bluetooth;
static unsigned long var_time_elapsed;   //  time elapsed since last flag
static unsigned long var_time_elapsed_Servo;
static unsigned long var_time_elapsed_Bluetooth;

static unsigned int var_sampling_period=0;

static unsigned int counter;
static unsigned int distance_Measured;
static unsigned int nearest_obstacle = threshold;

byte         var_LCD_flag;



//WHEEL MOVEMENT
DECISION=============================================================================
======================
//=============================================================================
==================================
enum MOVEMENTS {
  FORWARD, BACKWARD, LEFT, LEFTD, RIGHT, RIGHTD, STOP
};

const MOVEMENTS decision_table[32] = {

  FORWARD,     // 0b00000000
  LEFTD,       // 0b00000001
  LEFTD,       // 0b00000010
  LEFTD,       // 0b00000011
  LEFT,        // 0b00000100
  LEFT,        // 0b00000101
  LEFT,        // 0b00000110
  LEFT,        // 0b00000111
  RIGHTD,      // 0b00001000
  LEFT,        // 0b00001001
  LEFT,        // 0b00001010
  LEFT,        // 0b00001011
  RIGHT,       // 0b00001100
  LEFT,        // 0b00001101
  LEFT,        // 0b00001110
  LEFT,        // 0b00001111
  RIGHT,       // 0b00010000
  FORWARD,     // 0b00010001
  RIGHT,       // 0b00010010
  LEFTD,       // 0b00010011
  RIGHT,       // 0b00010100
  BACKWARD,    // 0b00010101
  RIGHT,       // 0b00010110
  BACKWARD,    // 0b00010111
  RIGHTD,      // 0b00011000
  RIGHTD,      // 0b00011001
  LEFT,        // 0b00011010
  BACKWARD,    // 0b00011011
  RIGHT,       // 0b00011100
  BACKWARD,    // 0b00011101
  RIGHT,       // 0b00011110
  BACKWARD     // 0b00011111

};

void nextMove(uint8_t obstacle_position, int speed) {
  MOVEMENTS m = decision_table[obstacle_position];
  switch(m) {
```

```
    case FORWARD:
    Serial.println("MOVE FORWARD");
    moveForward(speed);
    delay(1000);
    stopMotors();
    break;

    case BACKWARD:
    Serial.println("MOVE BACKWARDS");
    turnLeft(speed);
    delay(1000);
    stopMotors();
    moveForward(speed);
    delay(1000);
    stopMotors();
    break;

    case LEFT:
    Serial.println("GO LEFT");
    turnRight(speed);
    delay(500);
    stopMotors();
    moveForward(speed);
    delay(1000);
    stopMotors();
    break;

    case LEFTD:
    Serial.println("GO LEFT DIAGONAL");
    turnRight(speed);
    delay(250);
    stopMotors();
    moveForward(speed);
    delay(1000);
    stopMotors();
    break;

    case RIGHT:
    Serial.println("GO RIGHT");
    turnLeft(speed);
    delay(500);
    stopMotors();
    moveForward(speed);
    delay(1000);
    stopMotors();
    break;

    case RIGHTD:
    Serial.println("GO RIGHT DIAGONAL");
    turnLeft(speed);
    delay(325);
    stopMotors();
    moveForward(speed);
    delay(1000);
    stopMotors();
    break;

              case STOP:
              stopMotors();
  }
}
//=================================================================================
==================================
//=================================================================================
==================================
```

```
//WHEEL ACTIVATION
FUNCTIONS=============================================================================
===============
//=============================================================================
==================================

void moveForward(int speed){
   RL_fwd(speed);
   RR_fwd(speed);
   FR_fwd(speed);
   FL_fwd(speed);
}

void moveBackward(int speed){
   RL_bck(speed);
   RR_bck(speed);
   FR_bck(speed);
   FL_bck(speed);
}

void turnLeft(int speed){
   RL_bck(speed);
   RR_fwd(speed);
   FR_fwd(speed);
   FL_bck(speed);
}

void turnRight(int speed){
   RL_fwd(speed);
   RR_bck(speed);
   FR_bck(speed);
   FL_fwd(speed);
}

void stopMotors(){
  analogWrite(speedPinFR,0);
  analogWrite(speedPinFL,0);
  analogWrite(speedPinRR,0);
  analogWrite(speedPinRL,0);
}
      void FR_fwd(int speed)  //front-right wheel forward turn
      {
       digitalWrite(directionPinFR1,LOW);
       digitalWrite(directionPinFR2,HIGH);
       analogWrite(speedPinFR,speed);
      }

      void FR_bck(int speed)  //front-right wheel forward turn
      {
       digitalWrite(directionPinFR1,HIGH);
       digitalWrite(directionPinFR2,LOW);
       analogWrite(speedPinFR,speed);
      }

      void FL_fwd(int speed)  //front-right wheel forward turn
      {
       digitalWrite(directionPinFL1,LOW);
       digitalWrite(directionPinFL2,HIGH);
       analogWrite(speedPinFL,speed);
      }

      void FL_bck(int speed)  //front-right wheel forward turn
      {
       digitalWrite(directionPinFL1,HIGH);
       digitalWrite(directionPinFL2,LOW);
       analogWrite(speedPinFL,speed);
      }
```

```
      void RR_fwd(int speed)  //front-right wheel forward turn
      {
       digitalWrite(directionPinRR1,LOW);
       digitalWrite(directionPinRR2,HIGH);
       analogWrite(speedPinRR,speed);
      }

      void RR_bck(int speed)  //front-right wheel forward turn
      {
       digitalWrite(directionPinRR1,HIGH);
       digitalWrite(directionPinRR2,LOW);
       analogWrite(speedPinRR,speed);
      }

      void RL_fwd(int speed)  //front-right wheel forward turn
      {
       digitalWrite(directionPinRL1,LOW);
       digitalWrite(directionPinRL2,HIGH);
       analogWrite(speedPinRL,speed);
      }

      void RL_bck(int speed)  //front-right wheel forward turn
      {
       digitalWrite(directionPinRL1,HIGH);
       digitalWrite(directionPinRL2,LOW);
       analogWrite(speedPinRL,speed);
      }

//================================================================================
==================================
//================================================================================
==================================


//-----------------------------------------------------------------//
//-----------------------------------------------------------------//
int watch(){
  long echo_distance=0;
  digitalWrite(pin_Trig,LOW);
  delayMicroseconds(5);
  digitalWrite(pin_Trig,HIGH);
  delayMicroseconds(15);
  digitalWrite(pin_Trig,LOW);
  echo_distance=pulseIn(pin_Echo,HIGH);
  echo_distance=echo_distance*0.01657; //how far away is the object in cm
 //Serial.println((int)echo_distance);
  return round(echo_distance);
}
//-----------------------------------------------------------------//
//-----------------------------------------------------------------//


void interrupt_service_routine_INT1 (void){

       var_ST_flag = 1;

       // LAB experimentation on debouncing techniques:
       //delay(5); // wait some time

}

void read_inputs(void){


  // Do nothing now in design step #1


}
```

```
/*
================================================================================
       Establishing the sampling frequency using internal timer
       ================================================================================
*/
void update_time_flag(void){

  // From internal CLK:
    var_current_time = millis();
    var_current_time_Servo = millis();
             var_current_time_Bluetooth = millis();
    var_time_elapsed =  var_current_time - var_previous_time;
    var_time_elapsed_Servo =  var_current_time_Servo -
var_previous_time_Servo;
             var_time_elapsed_Bluetooth =  var_current_time_Bluetooth -
var_previous_time_Bluetooth;

       if((var_time_elapsed  > var_sampling_period) && var_sampling_period!=0)
{
                    // is the flag is set is because time elapsed is sampling
period
             var_SP_flag = 1;


             //var_Servo_flag = 1;
                    // to start another sampling period cycle
             var_previous_time = var_current_time;
       }
       if((var_time_elapsed_Servo  > var_sampling_period/15) &&
var_sampling_period!=0){
             // Serial.print("current time");
            // Serial.print((long)var_current_time_Servo, 1);
             var_SLED = var_SLED^0b00000001;
              var_Servo_flag = 1;

              var_previous_time_Servo = var_current_time_Servo;
       }
       if((var_time_elapsed_Bluetooth  > var_sampling_period/20) &&
var_sampling_period!=0){
             // Serial.print("current time");
            // Serial.print((long)var_current_time_Servo, 1);
             //var_SLED = var_SLED^0b00000001;
              var_BTSample_flag = 1;

              var_previous_time_Bluetooth = var_current_time_Bluetooth;
       }
}


/*
================================================================================
       System initialisation

================================================================================*/
void setup() {

  int error;

//DRIVER CONNECTED
PINS============================================================================
===============
//================================================================================
==================================
  pinMode(speedPinFR,OUTPUT);
  pinMode(directionPinFR1,OUTPUT);
  pinMode(directionPinFR2,OUTPUT);
```

```
  pinMode(speedPinFL,OUTPUT);
  pinMode(directionPinFL1,OUTPUT);
  pinMode(directionPinFL2,OUTPUT);

  pinMode(speedPinRR,OUTPUT);
  pinMode(directionPinRR1,OUTPUT);
  pinMode(directionPinRR2,OUTPUT);

  pinMode(speedPinRL,OUTPUT);
  pinMode(directionPinRL1,OUTPUT);
  pinMode(directionPinRL2,OUTPUT);

       pinMode(switchMode,INPUT);
  if(digitalRead(6)==LOW){
  var_Mode_flag = 0;
  }
  else{
  var_Mode_flag = 1;
  }
//================================================================================
==================================
//================================================================================
==================================


      head.attach(5);

//pins del sensor de distÃ ncia
       pinMode(pin_Echo,INPUT);
       pinMode(pin_Trig,OUTPUT);

//pin PWM del servo que mou el sensor ultrasònic
       pinMode(pin_US_Servo,OUTPUT);


// Output where to connect the sampling LED that will toggle every Ts
      pinMode(pin_SLED, OUTPUT);

// Configuring digital input pin for reading VM (only in design step #2)
      // pinMode(pin_VM, INPUT);

// This is the special pin to be an external interrupt source INT1:
      pinMode(INT1_pin_ST_BUTTON, INPUT_PULLUP);

// Serial monitor initialisation
      Serial.begin(9600);

     Serial.println("Buscando dispositivos I2C...");
     Wire.begin();
     for (byte i = 8; i < 120; i++) {
       Wire.beginTransmission(i);
       if (Wire.endTransmission() == 0) {
        Serial.print("Dispositivo encontrado en 0x");
        Serial.println(i, HEX);
    }
  }


 //Initialise the FSM and the timer
      var_current_state = IDLE;
      var_previous_time = 0;

            // Serial monitor ASCII message:
      Serial.println("Project LAB5 - DEE");
      Serial.println("Distance calculator, acquiring data from servo");
      Serial.println("System IDLE: waiting for ST click to start");
```

```
//
------------------------------------------------------------------------------
// This is the interrupt handler. Read the Arduino documentation at:
//
https://www.arduino.cc/reference/en/language/functions/external-interrupts/att
achinterrupt/
// attachInterrupt(digitalPinToInterrupt(pin), ISR, mode) (recommended)

       attachInterrupt(digitalPinToInterrupt(INT1_pin_ST_BUTTON),
interrupt_service_routine_INT1, RISING);
       // Interrupt when detecting a falling edge to start and stop operations

       Wire.begin();
  Wire.beginTransmission(0x27);
  error = Wire.endTransmission();
 // Serial.print("Error: ");
  //Serial.print(error);

  if (error == 0) {
    Serial.println(": Ok. LCD found.");
   // Init LCD using convenient libraries
   // initialize the lcd for 16 chars 2 lines, turn on backlight
    lcd.begin(16, 2);  // initialize the lcd

  } else {
       // in this way we can verity the wiring SDA and SCL
    Serial.println(": LCD not found.");
  }

  lcd.setBacklight(255);
    lcd.home();
    lcd.clear();
    lcd.print("Distance meter:)");
    delay(1000);

    lcd.setBacklight(0);
    delay(400);
    lcd.setBacklight(255);

    var_LCD_flag = 1;

    head.writeMicroseconds(1000);

}

/*
================================================================================
       Infinite loop to run the FSM
       ================================================================================
*/
void loop() {
       state_logic();    // calculating the next state to go
       output_logic();            // calculating outputs
}

/*
================================================================================
       State logic to determine where to go next
       ================================================================================
*/
void state_logic(void){
       read_inputs();             // capture the switch voltage values into RAM
variables
       update_time_flag();    //  to keep track of real time and be able to
                                                  // acquire signals at sampling
frequency

       switch (var_current_state) {
```

```
       case IDLE:   // Do nothing, wait for start signal
              if (var_ST_flag == 1) {      // start the process
                       var_current_state = SET_ACQ_TIMER;
                       var_ST_flag = 0;    //Clear flag when one used
              }
                else
                     var_current_state = IDLE;

              break;

       case SET_ACQ_TIMER: // Do nothing, wait for start signal
              var_current_state = Do_NOTHING;
              break;

       case Do_NOTHING:    // Wait for sampling period
              if (var_SP_flag == 1 && var_Mode_flag==0) {      // start
acquiring
                     var_current_state = Wait_Servo_Flag;
                     var_SP_flag = 0;

                     var_Servo_flag = 0;


                     //Clear flag when one used
              }
              else if(var_SP_flag==1 && var_Mode_flag==1){
              var_current_state = INIT_MODULE;
              var_SP_flag = 0;

              }
                else
                     var_current_state = Do_NOTHING;
              break;

       case Wait_Servo_Flag:
              if (var_Servo_flag == 1) {   // start acquiring
                     var_current_state = DIST_MEAS;
                     var_Servo_flag = 0;

                     //Clear flag when one used
              }
                else
                     var_current_state = Wait_Servo_Flag;
              break;

       case DIST_MEAS:
              var_current_state = Rotate;
              break;

        case Rotate:
              var_current_state = Save_Result;
              break;

       case Save_Result:
              if (counter == 5) {   // stop acquiring
                     var_current_state = TRANSMIT;
                     counter = 0;   //Clear flag when one used
              }
                else
                     var_current_state = Wait_Servo_Flag;

              break;

       case TRANSMIT:
              var_current_state = MOVE_CAR;
              break;

       case MOVE_CAR:
```

```
            if (var_ST_flag == 1) {      // stop acquiring
                  var_current_state = STOP_ACQ_TIMER;
                  var_ST_flag = 0;      //Clear flag when one used
            }
              else
                  var_current_state = Do_NOTHING;
            break;

      case STOP_ACQ_TIMER:       // go to Idle to wait for another Start pulse
            var_current_state = IDLE;
            break;



      case INIT_MODULE:
            var_current_state = WAIT_SAMPLE_PERIOD;
            break;

      case WAIT_SAMPLE_PERIOD:
            if (var_BTSample_flag == 1) {       // stop acquiring
                  var_current_state = READ_COMMAND;
                  var_BTSample_flag = 0;       //Clear flag when one used
            }
              else
                  var_current_state = WAIT_SAMPLE_PERIOD;

            break;

      case READ_COMMAND:
            var_current_state = EXECUTING_COMMAND;
            break;


      case EXECUTING_COMMAND:
            if (var_ST_flag == 1) {      // stop acquiring
                  var_current_state = STOP_ACQ_TIMER;
                  var_ST_flag = 0;      //Clear flag when one used
            }
              else
                  var_current_state = WAIT_SAMPLE_PERIOD;

            break;
      }
}

/*
=============================================================================
      Output logic
      =============================================================================
*/
void output_logic(void){
int var_buff;

      switch (var_current_state) {
      case IDLE:
            var_SLED = 0;
            var_S = 0;

            if (var_LCD_flag == 1){
                  lcd.setCursor(0,0); //Start at character 4 on line 0
                  lcd.print("..... IDLE .....");
                  lcd.setCursor(0,1);
                  lcd.print("Click ST button.");
                  var_LCD_flag = 0; // Allows new information in main loop

            }

            if(digitalRead(6)==LOW){
```

```
              var_Mode_flag = 0;
              }
              else{
              var_Mode_flag = 1;
             }

             break;

      case SET_ACQ_TIMER:
             var_SLED = 0;
             var_S = 1;


             var_sampling_period = SAMPLING_PERIOD;
             break;

      case Do_NOTHING:
             var_S = 2;

   obstacle_position = 0b00000000; //torno a reestablir la variable dels
obstacles, ja que s'ha complert un cicle, o és el primer cicle

             break;

      case Wait_Servo_Flag:
             var_S = 3;


             break;

      case DIST_MEAS:
             // Toggle operation with a XOR simply to indicate running
acquisition

             var_S = 4;

              distance_Measured = watch();

             if(distance_Measured < nearest_obstacle){

               nearest_obstacle = distance_Measured;

             }


             break;

      case Rotate:

      Serial.print("Counter: ");
   Serial.println(counter);

             var_S = 5;
             if (counter == 0) {
                    head.writeMicroseconds(1220);
             }
             else if (counter == 1){
                    head.writeMicroseconds(1500);
             }
             else if (counter == 2){
                    head.writeMicroseconds(1780);
             }
             else if (counter == 3){
                    head.writeMicroseconds(2000);
             }
             else if (counter == 4){
                    head.writeMicroseconds(1000);
             }
```

```
            else{

            }

            break;

      case Save_Result:
            var_S = 6;

            if((distance_Measured < threshold) && counter == 0){
                obstacle_position = obstacle_position | 0b00010000;
            }
            if((distance_Measured < threshold) && counter == 1){
                obstacle_position = obstacle_position | 0b00001000;
            }
            if((distance_Measured < threshold) && counter == 2){
                obstacle_position = obstacle_position | 0b00000100;
            }
            if((distance_Measured < threshold) && counter == 3){
                obstacle_position = obstacle_position | 0b00000010;
            }
            if((distance_Measured < threshold) && counter == 4){
                obstacle_position = obstacle_position | 0b00000001;
            }

            counter++;

            break;

      case TRANSMIT:      // measurement equation
            var_S = 7;
            lcd.clear();
////////////////////////////////////////////////////////////////////////////
/////////////

            obstacle_position_operator = obstacle_position & 0b00010000;
            if(obstacle_position_operator == 0b00010000){

                lcd.setCursor(0,1);
                lcd.print("X");
            }
            else{
                lcd.setCursor(0,1);
                lcd.print("-");
            }
////////////////////////////////////////////////////////////////////////////
/////////////
            obstacle_position_operator = obstacle_position & 0b00001000;
            if(obstacle_position_operator == 0b00001000){

                lcd.setCursor(1,1);
                lcd.print("X");
            }
            else{
                lcd.setCursor(1,1);
                lcd.print("-");
            }
////////////////////////////////////////////////////////////////////////////
/////////////

            obstacle_position_operator = obstacle_position & 0b00000100;
            if(obstacle_position_operator == 0b00000100){

                lcd.setCursor(2,1);
                lcd.print("X");
            }
            else{
                lcd.setCursor(2,1);
```

```
                lcd.print("-");
            }
////////////////////////////////////////////////////////////////////////////
/////////////

            obstacle_position_operator = obstacle_position & 0b00000010;
            if(obstacle_position_operator == 0b00000010){

                lcd.setCursor(3,1);
                lcd.print("X");
            }
            else{
                lcd.setCursor(3,1);
                lcd.print("-");
            }
////////////////////////////////////////////////////////////////////////////
/////////////

            obstacle_position_operator = obstacle_position & 0b00000001;
            if(obstacle_position_operator == 0b00000001){

                lcd.setCursor(4,1);
                lcd.print("X");
            }
            else{
                lcd.setCursor(4,1);
                lcd.print("-");
            }
////////////////////////////////////////////////////////////////////////////
/////////////

    if(nearest_obstacle == threshold){

      Serial.print("No obstacle was found\r\n");

    }

    else{

      Serial.print("The nearest obstacle was at ");
      Serial.print((int)nearest_obstacle, 1);
      Serial.println(" cm");
    }
      nearest_obstacle = threshold;
            break;

       case MOVE_CAR:
            var_S = 8;

            nextMove(obstacle_position, 100);

            break;



       case STOP_ACQ_TIMER:
            var_S = 9;
            var_sampling_period=0;
            var_LCD_flag = 1;

            break;

            case INIT_MODULE:
          var_S = 10;
          lcd.clear();
          lcd.setCursor(0,0);
          lcd.print("Bluetooth Mode");
```

```
            lcd.setCursor(0,1);
            lcd.print("ON");
            //SoftwareSerial bluetooth(9, 10);
            Serial1.begin(9600);
            //bluetooth.begin(9600);
            stopMotors();

        break;

        case WAIT_SAMPLE_PERIOD:
            var_S = 11;

        break;

        case READ_COMMAND:
            var_S = 12;
            if(Serial1.available()){ //Si hi ha dades
             receivedCommand=Serial1.read();
            //Serial1.print("He rebut: ");
             //Serial1.println(receivedCommand);




            }

        break;

        case EXECUTING_COMMAND:
        var_S = 13;

        switch(receivedCommand){
        case 'F':
        moveForward(75);
        break;

        case 'B':
        moveBackward(75);
        break;

        case 'R':
        turnRight(75);
        break;

        case 'L':
        turnLeft(75);
        break;

        case 'S':
        stopMotors();
        break;
        }

        if (var_ST_flag == 1) {
               stopMotors();
               receivedCommand = 'A';
        }
        break;

        }
        write_outputs(); // Convert RAM variable values into pin output voltage

}

void write_outputs(void){
char var_buf;  // local variable for saving partial results
```

```
    if (var_SLED == 1)
        digitalWrite(pin_SLED, HIGH); // Active-high connected LED
    else
        digitalWrite(pin_SLED, LOW);

}
```