

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Universitatea Tehnică “Gheorghe Asachi” din Iași

Specializarea: Electronică aplicată

# Lucrare de licență

Coordonator științific:

Ș.l. Dr. Ing. Florescu Roxana-Daniela

Student:

Moisă Paula

Iași, 2019

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Universitatea Tehnică “Gheorghe Asachi” din Iași

Specializarea: Electronică aplicată

# Primele etape spre dezvoltarea unui sistem autonom

Coordonator științific:

Ș.l.. Dr. Ing. Florescu Roxana Daniela

Student:

Moisă Paula

Iași, 2019

## Cuprins

1	Memoriu justificativ .....	5
2	Componentele hardware și instrumentele software utilizate .....	7
2.1	Componentele hardware .....	7
2.2	Instrumentele software .....	10
2.2.1	Arduino IDE .....	10
2.2.2	MIT App Inventor .....	11
2.2.3	Fritzing.....	12
3	Noțiuni teoretice.....	13
3.1	Servo-motorul .....	13
3.1.1	Generalități .....	13
3.1.2	Modul de control al servo-motoarelor .....	14
3.2	Motorul DC.....	15
3.2.1	Generalități .....	15
3.2.2	Modul de funcționare .....	16
3.2.3	Controlul motoarelor de curent continuu (DC) .....	17
3.3	Microcontrolerul .....	19
3.3.1	Generalități .....	19
3.4	Placa de dezvoltare Arduino UNO .....	20
3.5	ADC (analog-to-digital converter) .....	24
3.6	Senzori.....	26
3.6.1	Senzorul ultrasonic de distanță.....	26
3.6.2	Senzor fotoelectric reflectiv .....	28
3.6.3	Accelerometru .....	29
3.7	Interfața serială UART .....	30
3.7.1	Modulul Bluetooth HC-05 .....	36
3.8	Modul Buzzer .....	40
3.8.1	Generator de frecvență.....	40
3.9	Sistemul de alimentare pentru proiect.....	44
3.9.1	Bateria .....	44

3.9.2	Convertorul coborâtor(Buck) .....	45
4	Descrierea aplicației.....	50
4.1	Descrierea funcționalității.....	50
4.2	Control de la distanță.....	51
4.2.1	Implementare hardware .....	51
4.2.2	Interfața cu utilizatorul .....	53
4.2.3	Programarea microcontrolerului .....	57
4.3	Monitorizare alimentare.....	63
4.3.1	Implementare Hardware.....	63
4.3.2	Implementare software .....	65
4.3.3	Afișarea datelor pe interfața cu utilizatorul.....	66
4.4	Detecție de obstacole .....	67
4.4.1	Implementare hardware .....	68
4.4.2	Implementare software .....	68
4.5	Line follower .....	72
4.5.1	Implementare hardware .....	72
4.5.2	Implementare software .....	73
5	Concluzie .....	80

# 1 Memoriu justificativ

În ziua de azi asigurarea confortul și siguranței participanților în trafic a devenit una din temele principale de inovație în industria automotive. Lucrarea de față își propune să atingă unul din subiectele de interes în acest domeniu și anume dezvoltarea de aplicații și algoritmi software pentru mașini autonome.

Orientarea spre viitor impune dezvoltarea continuă de noi concepte și soluții care să satisfacă cerințele pieței și a clienților care se află în continuă schimbare. Pe măsură ce tehnologia se dezvoltă, cele mai mari companii din lume caută să inoveze și să implementeze tehnologii sofisticate, lucrând la dezvoltarea mașinilor autonome conduse de la distanță și chiar a mașinilor care se conduc singure. Astfel, ținta finală este reprezentată de realizarea mașinilor autonome care să fie capabile să transporte oamenii în condiții depline de siguranță și fără intervenție umană. Pentru a atinge acest țel se impune creșterea investițiilor în electronică, și deci și în soluții și aplicații software.

În cele ce urmează se vor prezenta etapele principale ce trebuie parcurse pentru a realiza o mașină în miniatură ce poate fi controlată de la distanță prin intermediul unui telefon mobil fie se poate mișca singură pe baza senzorilor cu care aceasta a fost echipată.

Principalii factori care m-au determinat să aleg și să dezvolt acest proiect au fost:

- importanța pe care o are domeniul automotive în economie;
- dobândirea cunoștințelor cu privire la: motoare și tehnici de control a acestora, senzori, microcontrolere;
- Îmbunătățirea cunoștințelor în domeniul dezvoltării software-ului, în special în limbajul C;

Aplicația implementată este o mașină autonomă controlată de la distanță prin intermediul unei telecomenzi. Aceasta are și alte funcții: urmărire linie și detecție de obstacole.

Când este în modul de urmărire linie, microcontrolerul primește date prin intermediul unui senzor fotoelectric cu 5 canale IR care ajută robotul să se mențină pe traseul de culoare

neagră. În cazul în care se întâlnește un obstacol, acesta este detectat prin intermediul unui sensor ultrasonic și motoarele se opresc automat.

## 2 Componentele hardware și instrumentele software utilizate

### 2.1 Componentele hardware

În cadrul proiectului a fost folosit un șasiu cu 4 roți cu tracțiune pe spate și mecanismul de direcție pe față.



Figura 2.1.1

Kit-ul care a fost utilizat în cadrul proiectului conține:

- Un motor DC cu perii
- Un servomotor MG996R
- Șasiu metalic
- 4 roți
- Mecanismul de direcție al servomotorului
- Diferite elemente de legătură pentru motoare și roți
- Fire de legătură

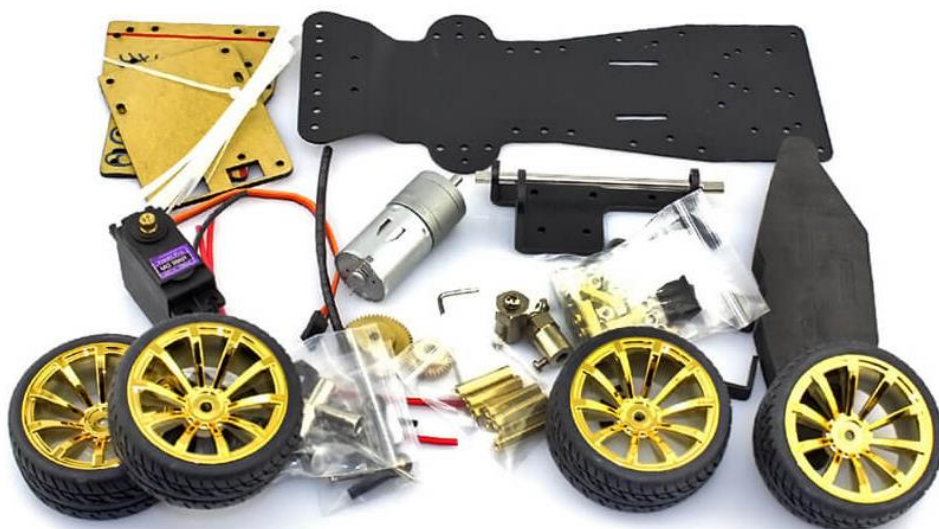


Figura 2.1.2

Alte circuite adiționale folosite în cadrul proiectului sunt:

- Placa de dezvoltare Arduino UNO cu microcontrolerul ATmega328P
- Senzori ultrasonici
- Modul Bluetooth
- Convertor nivel logic bi-direcțional
- Driver pentru motor DC
- Modul senzori urmărire linie
- Modul Buzzer
- Generator de frecvență (Timer 555)
- Întrerupător cu tranzistor

Motorul DC (de curent continuu) asigură deplasarea vehiculului înainte și înapoi. Direcția de rotație a motorului va fi dată de polaritatea tensiunii aplicată la borne. Dacă tensiunea aplicată motorului este pozitivă, vehiculul se va deplasa înainte, iar dacă tensiunea aplicată este negativă, motorul se va roti în direcția opusă deplasând vehiculul înapoi.

Servo-motorul MG995 asigură controlul cu precizie a direcției vehiculului. Inițial, acesta este fixat la o poziție de echilibru pentru ca deplasarea înainte să fie cât mai exactă. Când se



dorește schimbarea direcției, motorul servo va devia de la poziția de echilibru cu un unghi controlat de tensiunea aplicată pinului de comandă.

Placa de dezvoltare Arduino UNO conține tot ceea ce este necesar pentru ca microcontrolerul Atmega328P să funcționeze. Microcontrolerul are rolul de unitate principală de logică și control. Acesta preia datele primite de la senzori și comenzile date de utilizator prin Bluetooth și acționează corespunzător modului de funcționare dorit: fie acționare motoare, fie interacțiune cu utilizatorul, etc. De asemenea, microcontrolerul trimite date către interfața cu utilizatorul.

Pentru a putea controla motorul, vom avea nevoie de o placă specială, denumită în continuare driver/shield. Deoarece, puterea necesară controlului motorului este mai mare decât ceea ce ne oferă placa Arduino, vom folosi driverul DRV8833 conectat la o sursă de curent mai mare.

Pentru transmiterea și primirea datelor în timp real se folosește un modul Bluetooth ce permite comunicarea la distanță.

Se utilizează un convertor de nivel logic pentru a transforma tensiunea de 5V de la placa de dezvoltare în 3.3V necesari pentru a alimenta modulul Bluetooth, pentru ca utilizarea acestuia să fie sigură și pentru a se evita deteriorarea lui în timp.

Pentru oprirea automată a motoarelor în cazul în care apare un obstacol se folosesc senzori ultrasonici pentru măsurarea distanței. În completarea acestei funcții a “mașinii semi-automate” vine modulul buzzer, care generează sunete atunci când aceasta se apropie prea mult de un obstacol. Pentru a genera frecvența necesară funcționării buzzer-ului se utilizează un generator de semnal dreptunghiular cu Timer NE555 și frecvență ajustabilă. Acesta va transmite semnal către buzzer doar când primește comandă de la microcontroler prin intermediul unui întrerupător cu tranzistor.

Senzorii pentru urmărire linie au rolul de a ghida mașina pentru deplasarea pe o linie neagră atunci când acesta este în modul “Line Follower”.

Pentru alimentare s-au folosit următoarele elemente:

- 3 acumulatori Li-Ion de 3.7 V
- Modul de tip Buck (cu rol de a coborî tensiunea)

## 2.2 Instrumentele software

Programele utilizate în cadrul proiectului sunt:

- Arduino IDE
- MIT App Inventor
- Fritzing

În cele ce urmează se prezintă o serie de caracteristici pentru fiecare în parte.

### 2.2.1 *Arduino IDE*

Pentru programarea microcontrolerului se folosește programul Arduino IDE (Fig. 3.2.1), un mediu integrat de dezvoltare, care este o aplicație cross-platform, scrisă în Java. Acesta își are originile în mediul de dezvoltare pentru limbajul de programare Processing și în proiectul Wiring. Acesta a fost proiectat pentru a introduce programarea în lumea artiștilor și a celor nefamiliarizați cu dezvoltarea software.



Figura 2.2.1.1

Arduino IDE suportă limbajele de programare C și C++ folosind reguli speciale de organizare a codului. Un proiect tipic Arduino scris în C/C++ este compus din două funcții care sunt compilate și legate într-un program executabil cu o execuție ciclică:

- `setup()`: o funcție care este rulată o singură dată la începutul programului, când se inițializează setările.
- `loop()`: o funcție apelată în mod repetat până la oprirea alimentării plăcuței.

După compilare, codul poate fi încărcat pe plăcuța Arduino prin intermediul unui cablu USB A-B (Fig. 2.2.1.2).



Figura 2.2.1.2

## 2.2.2 MIT App Inventor

Programul MIT App Inventor este folosit pentru crearea aplicației pentru telefonul mobil cu ajutorul căreia va fi controlat robotul.



Figura 2.2.2.1

MIT App Inventor este un mediu de programare intuitiv, vizual, care permite realizarea de aplicații complet funcționale pentru smartphone-uri și tablete. Acesta utilizează o interfață grafică care oferă utilizatorilor posibilitatea să creeze o aplicație care poate rula pe dispozitive Android, prin introducerea (“drag-and-drop”) de obiecte vizuale. Programarea se realizează prin intermediul unor blocuri (programare vizuală) care au la bază limbajul C.

Aplicațiile pot fi construite în App Inventor astfel:

- în App Inventor Designer, sunt selectate componentele care vor alcătui aplicația
- în App Inventor Blocks Editor, blocurile din program sunt asamblate pentru a specifica modul în care componentele trebuie să se comporte. Se pot asambla programele vizual,

montând piesele împreună, ca piesele unui puzzle. Aplicația apare pe telefon pas-cu-pas, pe măsură ce piesele sunt adăugate în ea, așa că poate fi testată în timp ce este construită.

### 2.2.3 *Fritzing*

**Fritzing** este o aplicație software de automatizare a proiectării electronice, care permite crearea de scheme electrice, layout-uri pentru circuite imprimate (PCB), precum și efectuarea unor simulări înainte de a trece la hardware-ul real.



Figura 2.2.3.1

Acesta conține patru ferestre principale în care se pot realiza:

- simulări cu ajutorul unei plăci de testare
- schema electrică
- layout-uri pentru PCB
- generarea/ scrierea de cod



Figura 2.2.3.2

În cadrul proiectului, acest program a fost folosit pentru crearea schemelor electrice.

## 3 Noțiuni teoretice

### 3.1 Servo-motorul

#### 3.1.1 Generalități

Servomotoarele sunt motoare electrice speciale, de curent continuu sau curent alternativ cu viteză de rotație reglabilă într-o gamă largă în ambele sensuri. Acest tip de motor se utilizează pentru a deplasa un sistem mecanic (sarcina) într-un interval de timp prescris de-a lungul unei traiectorii date, realizând totodată și poziționarea acestuia la sfârșitul cursei cu o anumită precizie. Servomotoarele sunt proiectate special pentru a fi utilizate în sistemele automate de poziționare și care în general sunt de puteri reduse (până la puteri de ordinul câtorva kW).

Într-un sistem automat, servomotorul electric are rolul de a transforma un semnal electric de comandă într-un cuplu electromagnetic, respectiv într-o mișcare de rotație a arborelui său prin intermediul căruia este antrenat mecanismul care realizează operația dorită.

Spre deosebire de motoarele DC, care produc rotație continuă cât timp sunt conectate la o sursă de tensiune, motoarele servo sunt folosite pentru a obține rotații parțiale, stabile și controlate, pentru efectuarea unor operații cu amplitudine mică dar cu precizie ridicată.

Sistemele de reglare automată moderne impun servomotoarelor următoarele performanțe:

- gamă largă de modificare a vitezei de rotație în ambele sensuri
- funcționare stabilă la viteză foarte mică
- constante de timp cât mai reduse
- fiabilitate și robustețe ridicate
- raport cuplu/moment de inerție cât mai mare
- supra-sarcină dinamică admisibilă mare
- caracteristici de reglare liniare

În cadrul proiectului a fost folosit servo-motorul MG995 care are următoarele caracteristici tehnice:

- Unghiul de direcție: 0-360 grade
- Alimentare: 4.8V - 7.2V
- Lățimea benzii: 5  $\mu$ s
- Rezistent la temperaturi cuprinse între: 0°C - 55°C

- Forța în blocare: 8.5 kgf·cm (4.8 V ), 10 kgf·cm (6 V)
- Viteza de operare: 0.2 s/60° (4.8 V), 0.16 s/60° (6 V)
- Stabilizare la șocuri

### 3.1.2 Modul de control al servo-motoarelor

Interfațarea motoarelor servo cu restul aplicației se realizează prin intermediul a trei fire de diferite culori în funcție de producător. Culoarea roșie desemnează de obicei Vcc (5V), în timp ce firul de masă (GND) este de obicei negru sau maro. Cel de-al treilea fir este utilizat pentru transmiterea comenzii, care este de obicei galben, portocaliu sau alb.

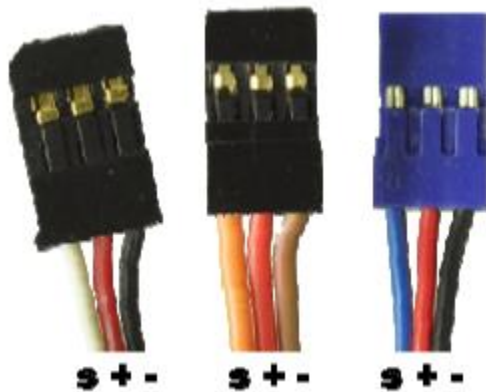


Figura 3.1.2.1

Motorul servo va devia de la poziția de echilibru cu un unghi controlat de tensiunea aplicată pinului de comandă. O modalitate de a modifica cu precizie valoarea unghiului de rotație a motorului este dată de folosirea unui semnal PWM (Pulse Width Modulation). Un semnal de tip PWM este caracterizat de următoarele aspecte:

- Este un semnal dreptunghiular
- Frecvența semnalului rămâne constantă
- Durata pulsurilor se modifică

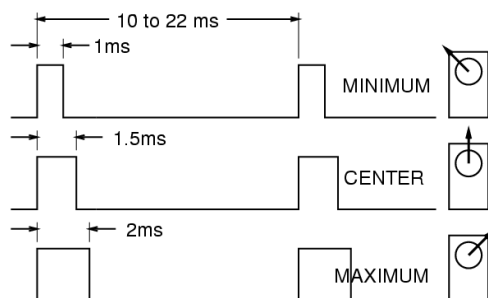


Figura 3.1.2.2

În cadrul aplicației curente, semnalul PWM va fi generat de microcontroler. Astfel pinul de ieșire a microcontroler-ului se va conecta la firul de comandă al servo-motorului MG995 așa cum se prezintă în figura de mai jos.

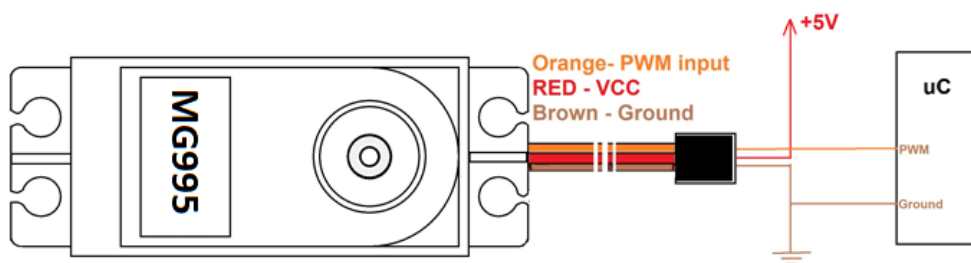


Figura 3.1.2.3

## 3.2 Motorul DC

### 3.2.1 Generalități

Motorul electric de curent continuu (cu perii) este un motor electric cu comutație internă care este alimentat de la o sursă de curent continuu prin intermediul **periilor** (care pot fi din cupru, carbon, etc). Construcția de principiu a unui motor de current continuu bipolar este prezentată în figura de mai jos:

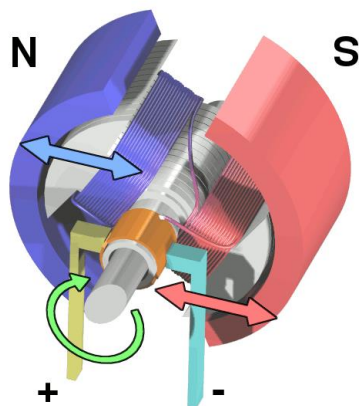


Figura 3.2.1.1

Așa cum se poate observa din Figura 3.2.1.1 motorul de curent continuu conține un magnet permanent (NS) denumit statorul motorului. La bornele motorului (reprezentate în figură cu +/-) se aplică o tensiune de curent continuu care determină apariția unui câmp electro-magnetic care va duce la punerea în mișcare a motorului. Direcția de rotație a motorului va fi dată de polaritatea tensiunii aplicată la borne (dacă tensiunea este pozitivă motorul se va roti într-o direcție, în timp ce dacă tensiunea aplicată este negativă, motorul se va roti în direcția opusă). De asemenea reglajul vitezei de rotație (turație) se va realiza tot prin intermediul tensiunii aplicate. Cu cât tensiunea aplicată este mai mare cu atât câmpul electro-magnetic indus va fi mai puternic ceea ce va duce la o mișcare de rotație mai rapidă.

### 3.2.2 Modul de funcționare

Când înfășurarea rotorului este alimentată, în jurul lui se generează un câmp magnetic (poziție relativă a polilor magnetici, de la stânga spre dreapta: N-NS-S). Polul nord al rotorului e respins de polul nord al statorului spre dreapta și e atras de polul sud al statorului (din dreapta), producând un cuplu mecanic motor care întreține mișcarea de rotație.

Rotorul continuă rotația până când ajunge în poziție orizontală (poziție relativă a polilor N-SN-S). Colectorul electric de comutare al sensului curentului continuu inversează sensul curentului prin înfășurarea rotorului, inversând polii câmpului magnetic produs de rotor, se ajunge astfel la poziția relativă a polilor magnetici "N-NS-S" și procesul se reia.



Colectoarele cu lamelele deformate, sparte sau cu perii de contact necorespunzătoare pot provoca vibrații excesive la motoarele de curent continuu cu comutație internă. Frecvența de defect va fi frecvența naturală a lamelei, care este egală cu numărul de lamele înmulțite cu turația de lucru, RPM.

Dacă vârful de la frecvența de 360 Hz din spectrul de vibrație se ridică în mod semnificativ, cauzele cele mai probabile sunt: un circuit deschis la înfășurări, precum și eventuale conexiuni electrice slăbite.

Motorul cu perii disipă destul de multă energie prin frecare și prin încălzire (apar scântei). Tot din cauza scânteilor de la perii se produce un factor de distorsiune a sistemului de alimentare prin reinjectarea de impulsuri parazite în sens invers. Din această cauză necesită întreținere (schimbare perii).

### 3.2.3 Controlul motoarelor de curent continuu (DC)

Pentru controlul motorului DC s-a folosit driver-ul DRV8838. Acesta poate controla motoare ce consumă până la 1.7A curent continuu, la o tensiune maximă de 11V.



Figura 3.2.3.1

Caracteristici tehnice:

- Tensiune de alimentare motor: 0V - 11V;
- Curent maxim motor: 1.7A;
- Curent de vârf motor: 1.8A;
- Frecvență maximă PWM: 250kHz;
- Tensiune de alimentare partea de logică: 1.8V - 7V;
- Protecție la subtensiune pe partea de logică, supra-curent, supra-temperatură și alimentare inversă pentru partea de putere.

Schema electrică minimă pentru conectarea unui microcontroler la un driver de motor DC DRV8838 este prezentată în figura de mai jos:

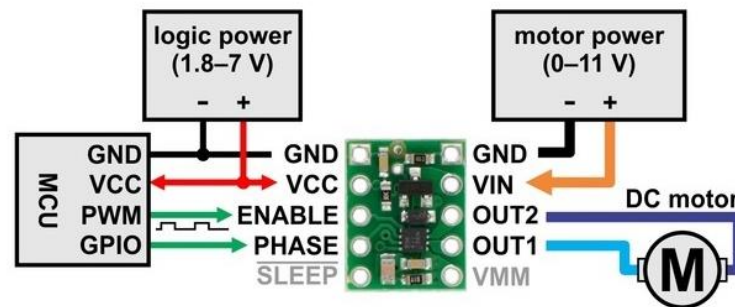


Figura 3.2.3.2

Semnificația pinilor din figura de mai sus este:

- VIN – intrare de alimentare cu protecție inversă
- VMM – intrare de alimentare pentru aplicațiile de foarte joasă tensiune (ocolește circuitul de protecție inversă)
- OUT1 și OUT2 – ieșirile din puntea H la care se conectează motorul DC
- VCC - conexiune logică de alimentare. Acest PIN poate fi alimentat dinamic de un microcontroler cu ieșire digitală.
- GND - Puncte de conectare la masă pentru alimentarea cu energie a motorului și a logicii.
- PHASE – Intrare pentru control direcție
- ENABLE - Intrare de control al vitezei
- SLEEP - Intrarea în modul Sleep

Următorul tabel arată modul în care funcționează driver-ul de motor:

PHASE	ENABLE	SLEEP	OUT1	OUT2	Modul de operare
0	PWM	1	PWM	L	înainte/frână la viteza PWM%
1	PWM	1	L	PWM	înapoi/frână la viteza PWM%

Figura 3.2.2.3

### 3.3 Microcontrolerul

#### 3.3.1 Generalități

Un microcontroler este un tip de circuit programabil care integrează o unitate de procesare și alte dispozitive periferice într-un singur chip, punându-se accent pe un cost redus de producție și un consum redus de energie electrică.

Componentele de bază ale unui microcontroler:

- unitatea centrală de procesare ( $\mu$ P core) cu o arhitectură care poate lucra pe 8, 16, 32 sau 64 de biți;
- memorie de date volatilă (RAM) sau nevolatilă pentru date sau program (Flash sau EEPROM);
- porturi digitale de intrare-ieșire;
- interfețe seriale (RS232, SPI, I2C, CAN, RS485).
- temporizatoare, generatoare de PWM și/sau WatchDog.
- convertoare analog-digitale.
- suport pentru programare și debugging (ISP).

În cadrul proiectului s-a utilizat microcontrolerul Atmega328, dezvoltat de compania Atmel, ulterior cumpărată de Microchip.



Figura 3.3.1.1

ATmega328 are următoarele caracteristici:

- Este un microcontroller de tip RISC
- Pune la dispoziție 28 de pini, din care 6 pini sunt analogici și 14 pini sunt digitali
- Permite comunicarea serială prin intermediul pinilor RXD(pin 2) și TXD(pin3)
- Are nevoie de un oscilator de cristal pentru generarea frecvenței. Se poate utiliza oscilator de cristal variind de la 4MHz la 40MHz

- Conține Watchdog Timer programabil cu oscilator intern, cinci moduri de economisire a puterii consumate, întreruperi interne și externe, programator de tip USART, port serial SPI.

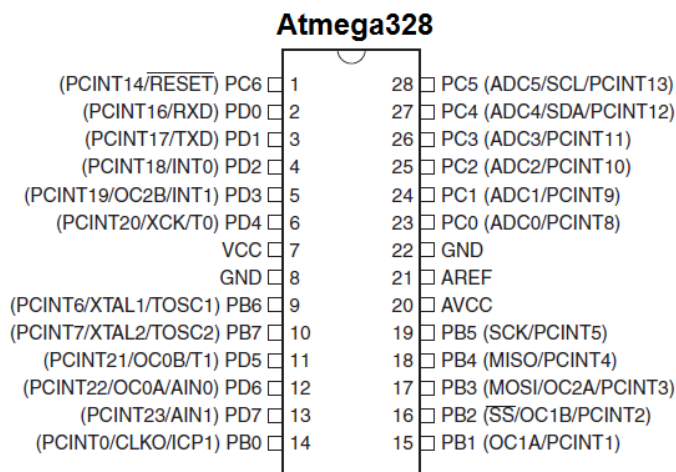


Figura 3.3.1.2

Microcontrolerul Atmega328 este inclus în placa de dezvoltare Arduino Uno.

### 3.4 Placa de dezvoltare Arduino UNO

Arduino UNO este o platformă de procesare open-source, bazată pe software și hardware flexibil și simplu de folosit. Constă într-o platformă de mici dimensiuni (6.8 cm / 5.3 cm – în cea mai des întâlnită variantă) construită în jurul unui procesor de semnal și este capabilă de a prelua date din mediul înconjurător printr-o serie de senzori și de a efectua acțiuni asupra mediului prin intermediul luminilor, motoarelor, servomotoare, și alte tipuri de dispozitive mecanice.

Procesorul este capabil să ruleze cod scris într-un limbaj de programare care este foarte similar cu limbajul C++.

Arduino UNO are 14 pini digitali (din care 6 pot fi utilizate ca ieșiri PWM), 6 intrări analogice, un oscilator cu quart de 16 MHz, o conexiune USB, o mufă de alimentare, o mufă ICSP și un buton de resetare.



Figura 3.4.1

#### Specificații Arduino UNO:

- Microcontroler: ATmega328
- Tensiune de lucru: 5V
- Tensiune de intrare (recomandat): 7-12V
- Tensiune de intrare (limita): 6-20V
- Pini digitali: 14 (6 PWM output)
- Pini analogici: 6
- Curent per pin I/O: 40 mA
- Curent 3.3V: 50 mA
- Memorie Flash: 32 KB (ATmega328) 0.5 KB pentru bootloader
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Clock Speed: 16 MHz

ARDUINO UNO conține tot ceea ce este necesar pentru ca microcontrolerul Atmega328P să funcționeze. Acesta are încorporat și microcontrolerul Atmega8U2 programat ca un convertor USB-la-serial.

Descrierea pinilor de pe placa Arduino UNO:

Există 14 pini digitali de intrare / ieșire (I/O sau input/output). Aceștia operează la o tensiune de 5V și pot fi controlați cu una din funcțiile pinMode(), digitalWrite() și digitalRead().

Fiecare pin poate primi sau trimite o intensitate de maxim 40 mA și au o rezistență internă între 20-50 kOhmi. În afară de semnalul standard I/O, unii dintre pini mai au și alte funcții specializate.

- 0 (serial) **RX** – pin serial, utilizat în special pentru recepția (intrare – **Rx**) datelor seriale asincrone (asynchronous serial communication). Protocolul serial asincron este o metodă foarte răspândită în electronică pentru a trimite și recepționa date între dispozitive. Acest protocol este implementat prin intermediul dispozitivului numit UART (Universal Asynchronous Receiver/Transmitter)
- 1 (serial) **TX** – pin serial, utilizat pentru trimiterea datelor asincrone (ieșire – **Tx**).
- 2 (External Interrupts) întrerupere externă. Acest pin poate fi configurat pentru a declanșa o întrerupere la o valoare mică, un front crescător sau descrescător, sau o schimbare în valoare.
- 3 (External Interrupts + PWM) întrerupere externă. Identic cu pinul 2. Suplimentar, toți pinii marcați cu semnul ~ pot fi folosiți și pentru **PWM** (pulse with modulation)
- 4 (**I/O**) pin standard intrare/ieșire
- 5 (**PWM**) poate furniza control de ieșire pe 8-bit pentru controlul PWM6 (**PWM**)
- 7 (**I/O**) pin standard intrare/ieșire
- 8 (**I/O**) pin standard intrare/ieșire
- 9 (**PWM**)
- 10 (**PWM** + SPI) – suportă comunicare prin interfața serială (Serial Peripheral Interface). SPI-ul are patru semnale logice specifice iar acest pin se folosește pentru **SS** – Slave Select (active low; output din master).
- 11 (**PWM** + SPI) – suportă SPI, iar acest pin se folosește pentru **MOSI/SIMO** – Master Output, Slave Input (output din master)
- 12 (SPI) – suportă SPI, iar acest pin se folosește pentru **MISO/SOMI** – Master Input, Slave Output (output din slave)

- 13 (LED + SPI) – suportă SPI, iar acest pin se folosește pentru **SCK/SCLK** – Ceas serial (output din master). De asemenea, pe placă este încorporat un LED care este conectat la acest pin. Când pinul este setat pe valoarea HIGH este pornit, când are valoarea LOW este oprit.
- 14 (**GND**) – conexiunea de masă. 15 (**AREF**) – Analog REFference pin – este utilizat pentru tensiunea de referință pentru intrările analogice.
- 16 (**SDA**) – comunicare I2S
- 17 (**SCL**) – comunicare I2S

Așa cum s-a menționat anterior, placa dispune de o serie de 6 pini pentru semnal analogic, numerotați de la A0 la A5.

Intern fiecare din aceste semnale analogice sunt convertite în semnale digitale cu o rezoluție de 10 biți (adică maxim 1024 de valori diferite). În mod implicit se măsoară de la 0 la 5V, deși este posibil să se schimbe limita superioară a intervalului folosind fie pinul 15 AREF fie funcția [analogReference\(\)](#). Funcțiile suplimentare pentru pinii analogici sunt:

- A0 standard analog pin
- A1 standard analog pin
- A2 standard analog pin
- A3 standard analog pin
- A4 (**SDA**) suportă comunicarea prin 2 fire (**I2C** (I-two-C) sau **TWI** (Two wire interface)). Acest pin este folosit pentru SDA (Serial Data) la TWI.
- A5 (**SCL**) identic cu pinul 4, doar că acest pin este folosit pentru **SCL** (Serial Clock) la TWI. Pentru controlul TWI se poate folosi [librăria Wire](#).

Pe lângă pinii analogici și digitali mai există un tip special de pini notați **POWER**. Aceștia sunt:

- **Vin** – intrarea pentru tensiunea de alimentare din sursă externă (input Voltage)
- **GND** –conexiunea de masă din sursă externă (ground Voltage)
- **GND** – negativ. Se folosește pentru piesele și componentele montate la arduino ca și masă/împământare/negativ.

- **5V** – ieșire pentru piesele și componentele montate la arduino. Furnizează o tensiune de referință de 5V dacă placa este alimentată cu o tensiune între 7 și 12 V
- **3,3V** – ieșire pentru piesele și senzorii care se alimentează la această tensiune. Tensiunea de referință furnizată este de 3.3 volți și maxim 50 mA.
- **RESET** – se poate seta acest pin pe LOW pentru a reseta controlerul de la Arduino. Este de obicei folosit de shield-urile care au un buton de reset și care anulează de obicei butonul de reset de pe placa Arduino.
- **5VREF** – este folosit de unele shield-uri ca referință pentru a se comuta automat la tensiunea furnizată de placa arduino (5 volți sau 3.3 volți) (Input/Output Reference Voltage)
- **pin** neconectat, este rezervat pentru utilizări ulterioare (la reviziile următoare ale plăcii).

Placa mai are o serie de pini marcați ICSP (In-Circuit Serial Programming). Acești pini pot fi folosiți pentru reprogramarea microcontrolerului, sau ca pini de expansiune cu alte microcontrolere compatibile. Sunt conectați standard și se poate folosi un cablu cu 6 fire (MOSI, MISO, SCK, VCC, GND, și pinul RESET).

Comunicarea cu calculatorul, altă placă arduino sau alte microcontrolere se poate realiza fie prin portul USB (și este văzut ca un port standard serial COMx), fie prin pinii 0 și 1 (RX și TX) care facilitează comunicarea serială UART TTL (5V).

### 3.5 ADC (analog-to-digital converter)

Convertorul analog-digital (ADC) este un circuit electronic integrat folosit pentru a converti semnalele analogice, cum ar fi tensiunile, în formă digitală sau binară.



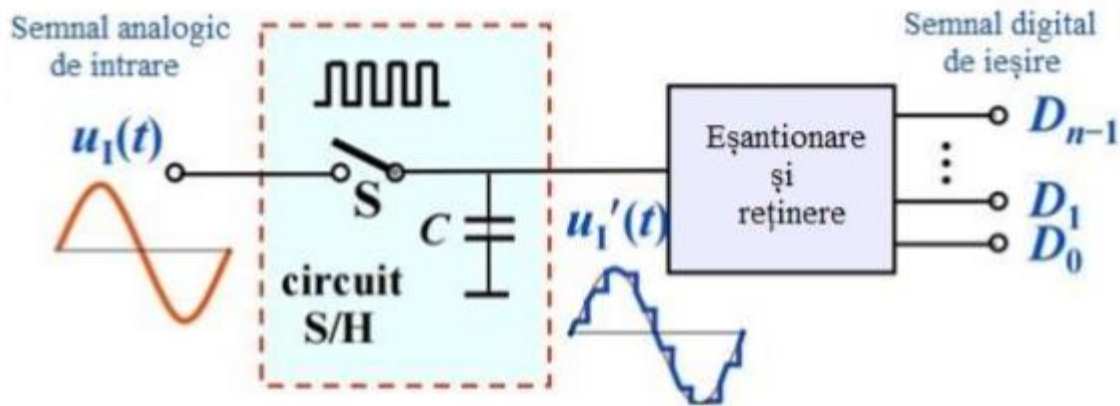


Figura 3.5.1

În principal, există doi pași pentru conversia analog-digital:

- ***S / H: Eșantionare și reținere***

Un semnal analogic se schimbă continuu în timp. Pentru a măsura semnalul, acesta trebuie păstrat constant pentru o scurtă durată, astfel încât să poată fi eșantionată. Aceasta se face printr-un circuit de eșantionare și menținere.

- ***Q / E: cuantificarea și codificarea***

Pe ieșirea (S/H), există un anumit nivel de tensiune. Acestuia i se atribuie o valoare numerică. Se caută cea mai apropiată valoare, în corespondență cu amplitudinea eșantionată și a semnalului de reținere. După identificarea celei mai apropiate valori, îi este atribuită o valoare numerică și este codificată sub forma unui număr binar.

În sistemul implementat ADC-ul măsoară tensiunea de la bornele bateriilor.

Microcontrolerul Atmega328P este echipat cu un circuit ADC cu referință de tensiune configurabilă. Acesta funcționează prin aproximări succesive și are o rezoluție configurabilă de până la 10 biți, adică are capacitatea de a detecta 1.024 ( $2^10$ ) niveluri discrete analogice.

Placa de dezvoltare Arduino Uno permite citirea a 6 canale diferite, fiecare conectate pe pinii A0, A1, ..., A5 .

## 3.6 Senzori

Senzorul este un dispozitiv care măsoară o mărime fizică (masă, distanță, presiune, temperatură, umiditate etc.) și o transformă într-un semnal care poate fi citit de către un observator printr-un instrument sau poate fi prelucrat.

În automatizare, informația calitativă/cantitativă măsurabilă livrată de senzori, după o eventuală amplificare și prelucrare servește la controlul și reglarea sistemelor tehnice automate.

În proiect au fost folosite două tipuri de senzori:

- Senzor ultrasonic de distanță
- Senzor urmărire linie

### 3.6.1 *Senzorul ultrasonic de distanță*

Senzorii ultrasonici emit pulsații acustice scurte, de înaltă frecvență, la intervale de timp regulate. Acestea se propagă prin aer cu viteza sunetului. Dacă lovesc un obiect, acestea sunt reflectate înapoi ca semnale ecou la senzor, care calculează distanța până la obiect pe baza intervalului de timp dintre emiterea semnalului și receptarea ecoului.

În proiect a fost folosit senzorul ultrasonic HC-SR04. Rolul său este de a detecta existența unui obstacol în calea robotului, caz în care motoarele se vor opri automat, evitând orice coliziune.

Acest senzor de distanță cu ultrasunete este capabil să măsoare distanțe între 2 cm și 400 cm. Este un dispozitiv de curent scăzut, astfel încât este potrivit pentru dispozitive alimentate cu baterii.



Figura 3.6.1.1

Caracteristicile tehnice principale ale senzorului ultrasonic sunt:

- Tensiune: 5V
- Tensiune HIGH: 5V
- Tensiune LOW: 0V
- Unghiul senzorului: 15 grade
- Distanța detectată: 2cm - 450cm
- Precizie: 0.3cm

Senzori de distanță cu ultrasunete folosesc impulsuri de sunet ultrasonic (sunet deasupra gamei de auz uman) pentru a detecta distanța dintre ele și obiectele solide din apropiere. Senzorii constau din două componente principale:

- Un transmițător care transmite impulsuri de sunet cu ultrasunete, funcționează la 40 KHz
- Un receptor care așteaptă impulsurile transmise. Dacă le primește, produce un puls de ieșire a cărui lățime poate fi utilizată pentru a determina distanța pe care a călătorit pulsul.

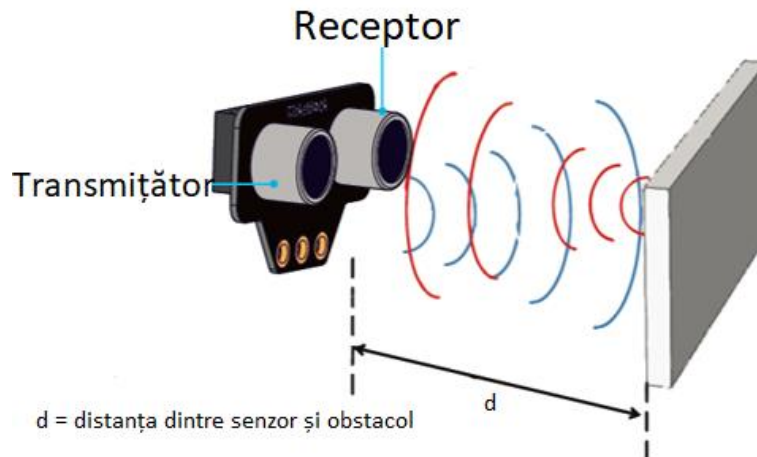


Figura 3.6.1.2

HC-SR04 are patru pini:

- **VCC** – alimentare pozitivă (5V).
- **Trig** – trimite impulsuri ultrasonice.

- **Echo** – acest pin produce un puls atunci când semnalul reflectat este primit.  
Lungimea pulsului este proporțională cu timpul care a trecut până când semnalul transmis a fost detectat.
- **GND** – pinul de masă(ground).

### 3.6.2 Senzor fotoelectric reflectiv

Pentru ca robotul să se mențină pe traseul de culoare neagră, microcontrolerul va primi date de la modulul de urmărire linie folosit, format din 3 senzori TCRT5000 ce funcționează pe principiul reflexiei luminii IR.



Figura 3.6.2.1

Senzorul fotoelectric reflectiv TCRT5000 este un senzor reflectorizant format dintr-un emițător în infraroșu și un fototranzistor care detectează culoarea pe care se află prin recepționarea fasciculului IR reflectat pe aceasta.

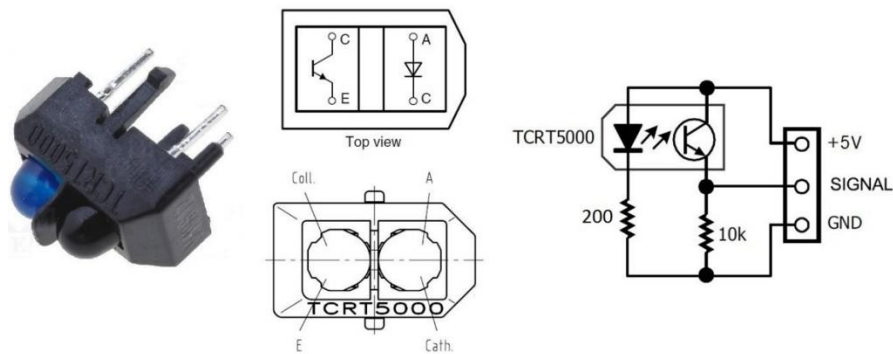


Figura 3.6.2.2

Principalele caracteristici tehnice ale acestui senzor sunt:

- Cădere tensiune LED: 1.25V;
- Curent led: 60mA (maxim);

- Tensiune colector - emitor: 70V;
- Curent colector: 100mA;
- Lungime de undă: 950nm;
- Distanță de operare optimă: 2.5mm;
- Distanță de operare: 0.2mm - 15mm.

### 3.6.3 Accelerometru

Accelerometrul este un senzor ce măsoară valorile accelerațiilor liniare sau unghiulare la care este supus un obiect.

Particularizat la smartphone-uri, accelerometrul sesizează schimbările de poziție ale telefonului prin măsurarea accelerațiilor unghiulare și liniare pe cele trei axe x, y, z. Funcție de valoarea și de axa pe care este măsurată accelerația, un accelerometru poate determina cu precizie orientarea telefonului în spațiu.

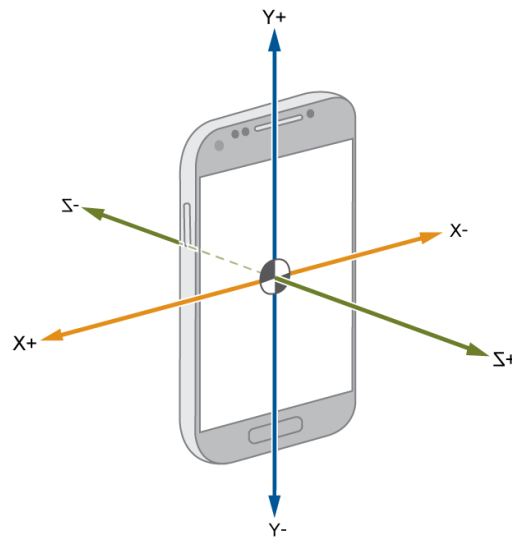


Figura 3.6.3.1

Un accelerometru poate fi dispus în orice poziție pe placa telefonului, însă, din motive ușor de înțeles, axele x și y sunt orientate paralel cu marginile smartphone-ului, în timp ce axa z este perpendiculară pe planul format de acestea, cu sens opus display-ului.

În cadrul proiectului, acest senzor se folosește pentru controlul servo-motorului. Astfel, se citesc valorile de pe axa Y și se adaptează la unghiurile necesare funcționării motorului servo.

De exemplu, în cazul poziționării telefonului ca în figura 3.6.3.1 valorile date de sensor vor fi:  $(x,y,z) = (0, 90, 0)$ .

Software-ul utilizat în dezvoltarea aplicației pentru telefonul mobil citește aceste valori divizate cu zece.

În cadrul aplicației create, poziția de echilibru (cea în care roțile mașinii vor fi centrate) este:  $(X,Y,Z) = (9, 0, 0)$ . Prin înclinare la stânga, mașina va vira la stânga și valorile citite de la senzor de pe axa Y vor fi în intervalul  $[-9, 0]$ , respectiv  $[0, +9]$  pentru înclinare la dreapta.

### 3.7 Interfața serială UART

Interfața serială USART (Universal synchronous/asynchronous receiver /transmitter) este un standard de comunicare serială între diverse dispozitive cum ar fi comunicarea între calculator (prin portul serial COM ) și alte dispozitive.

Comunicarea serială presupune folosirea unei singure legături între dispozitive dacă ea este unidirecțională, adică există un transmițător și un receptor sau invers. Dacă este bidirecțională avem nevoie de două legături, pe o legătură realizându-se transmisia iar pe cealaltă recepția. În cazul în care se folosește o transmisie sincronă, pe lângă legăturile pentru transmisia date se mai folosește o conexiune prin care se stabilește semnalul de tact între dispozitive.

În cazul nostru, vom folosi o transmisie asincronă (UART), bidirecțională.

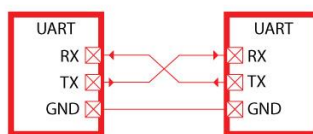


Figura 3.7.1

Cuvântul care este transmis este format din două părți, o parte care reprezintă datele iar altă parte care reprezintă codul de verificare. Orice cuvânt începe cu un bit de start care trebuie să fie 0. Următorii biți sunt cei de date care pot avea o lungime între 5 și 8 biți, iar la sfârșit avem un bit de stop care întotdeauna trebuie să fie 1. Pentru o mai bună securitate în transmisie se poate seta la sfârșitul cuvântului de date un bit de paritate care verifică paritatea biților de date.

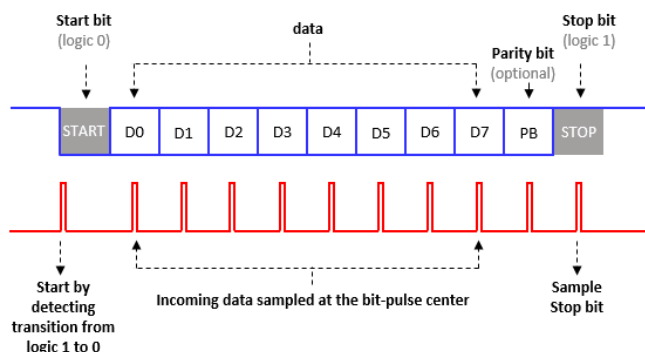


Figura 3.7.2

Principala sursă de eroare poate apărea atunci când configurarea între dispozitive nu este făcută corect, aceasta fiind o problemă de incompatibilitate în comunicare. Alte erori pot fi generate pe linile de transmisie, ele putând fi detectate prin bitul de paritate sau prin biții de stop.

La configurarea a două dispozitive care vor comunica serial asincron trebuie să avem grijă ca viteza de transfer să fie egală la transmisie și recepție și să avem aceeași lungime a cuvântului de date.

Viteza de transmisie a datelor se măsoară în BAUD, unitate care reprezintă numărul de biți transmiși într-o secundă. Ratele de BAUD suportate de Atmega328p sunt 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600 și 115200.

Pentru setarea vitezei de transfer a datelor se folosește registrul UBRR.

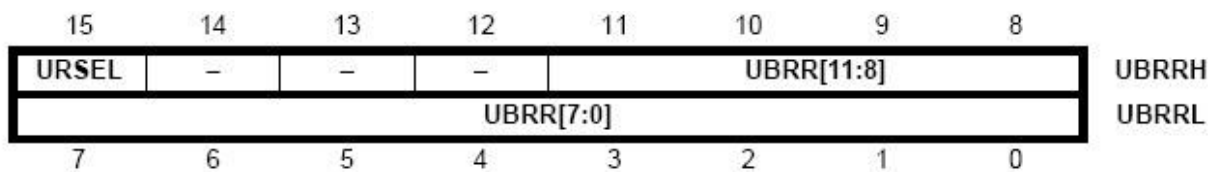


Figura 3.7.3

- Bitul 15 – URSEL – trebuie să fie 0 atunci când se efectuează operații cu acest registru.
- Biții 12,13,14 sunt biți rezervați nefolosiți în acest microcontroler.
- Biții 11:0 sunt folosiți pentru a seta viteza de transmisie a datelor. Totuși setarea se face doar cu biții cei mai semnificativi, 11:8. Viteza de transmisie a datelor se calculează după formula:

$$BAUD = \frac{f_{int}}{16 * (UBRR + 1)}$$

Rezultă că pentru o frecvență a ceasului și o rată de Baud date, trebuie scrisă în UBRR următoarea valoare:

$$UBRR = \frac{f_{int}}{16 * BAUD} - 1$$

Schimbul de date se face prin doi regiștrii, unul în care se înscriu datele și unul de shiftare prin care se transmite cuvântul. În ATmega328 datele se scriu sau se citesc din registrul UDR. De exemplu pentru transmiterea unui caracter (char) pe 8 biți trebuie să scriem caracterul în UDR. USART-ul va trece automat conținutul lui UDR în registrul serial apoi îl va transmite pe linia de legătură celui alt dispozitiv care va prelua datele tot în registrul serial al cărui conținut va fi transmis registrului UDR. Registrul serial, dacă dispozitivul este configurat ca transmițător, va pune pe lângă date și bitul de start, bitul de paritate și bitul/biții de final. Dacă dispozitivul este configurat ca receptor, registrul serial va verifica dacă transmisia este corectă și dacă da datele cuvântului vor fi copiate în UDR.

Registrul UDR are rolul de a stoca datele necesare pentru transmisie sau recepție.

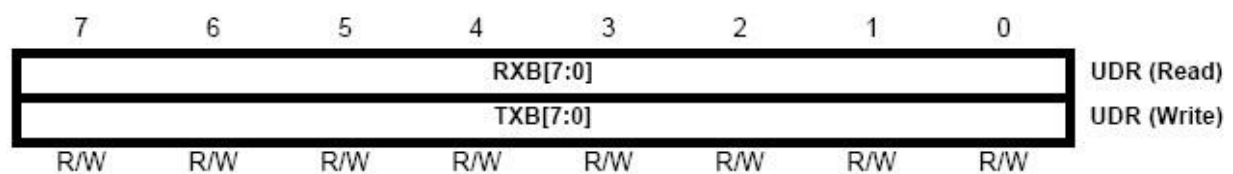


Figura 3.7.4



Dacă se realizează transmisia, datele sunt stocate în TXB iar dacă se realizează recepția datele se găsesc în RXB.

La transmiterea sau recepția mai multor cuvinte consecutive trebuie verificat dacă UART-ul a realizat operația completă de trimitere sau citire, altfel poate apărea o altă eroare de transmisie generată de data asta de partea software.

Registrul UCSRA:

7	6	5	4	3	2	1	0	
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
R	R/W	R	R	R	R	R/W	R/W	
0	0	1	0	0	0	0	0	

Figura 3.7.5

- Bitul 7 –RXC- indică starea registrului de recepției. Dacă încă se mai primesc date valoarea lui este 1 iar dacă recepția a fost efectuată complet valoarea lui este 0.
- Bitul 6 –TXC- indică starea registrului de transmisiei. Dacă încă se mai transmit date valoarea lui este 1 iar dacă transmisia a fost efectuată complet valoarea lui este 0.
- Bitul 5 – UDRE – indică dacă UDR poate fi citit sau scris. Dacă este 1 UDR poate fi scris, dacă este 0 nu se pot efectua operații cu acest registru. Acești 3 biți pot genera întreruperi care pot fi tratate în rutine speciale .
- Bitul 4 – FE – Detectează dacă există erori la transmiterea cuvântului verificând bitul/biții de final. Dacă acest/acești biți au o valoare diferită de 1 atunci bitul FE ia valoarea 1.Dacă totul este în regulă atunci valoarea lui FE va fi 0.
- Bitul 3 –DOR - Detectează dacă există erori de suprascrierea USAR-ului. De exemplu la recepție poate apărea o astfel de eroare dacă UDR și registrul serial este plin și alt caracter este pe cale de a fi primit. FE este 1 dacă s-a semnalat o astfel de eroare și 0 dacă nu avem o eroare de acest fel.
- Bitul 2 – PE – are rolul de a detecta eroarea de paritate. Dacă s-a detectat o astfel de eroare ia valoarea 1 iar dacă nu valoarea lui va fi 0.

- Bitul 1 – U2X – Acest bit este 1 dacă dorim să dublăm viteza de transmisie în modul asicron și 0 dacă se folosește viteza normală. Acest bit trebuie să aibă întotdeauna valoarea 0 dacă este folosit modul sincron.
- Bitul 0 – U2X- Se setează 1 dacă se folosește comunicarea multiprocesor și 0 dacă se folosește comunicarea uniprocessor. În exemplele ce vor urma îl vom seta ca 0.

Pentru generarea de întreruperi se folosește registrul UCSRB.

7	6	5	4	3	2	1	0	
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
0	0	0	0	0	0	0	0	

Figura 3.7.6

- Bitul 7-RXCIE – 1 setează generarea unei întreruperi la terminarea recepției, 0 întreruperea nu este folosită.
- Bitul 6-RXCIE – 1 setează generarea unei întreruperi la terminarea transmisiei, 0 întreruperea nu este folosită.
- Bitul 5-RXCIE – 1 setează generarea unei întreruperi atunci când registrul UDRIE poate fi folosit, 0 întreruperea nu este folosită.
- Bitul 4- RXEN – 1 setează USART-ul ca receptor.
- Bitul 3- TXEN – 1 setează USART-ul ca Transmițător.
- Bitul 2- UCSZ2 – se folosește împreună cu o parte din biții registrului UCSRC după cum vom vedea mai jos.
- Bitul 1 – RXB8 – este al 9-lea bit din schimbul de date pe 9 biți și se folosește la recepție.
- Bitul 0 – TXB8 – este al 9-lea bit din schimbul de date pe 9 biți și se folosește la transmisie.

Registrul UCSRC:

7	6	5	4	3	2	1	0	
<b>URSEL</b>	<b>UMSEL</b>	<b>UPM1</b>	<b>UPM0</b>	<b>USBS</b>	<b>UCSZ1</b>	<b>UCSZ0</b>	<b>UCPOL</b>	<b>UCSRC</b>
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
1	0	0	0	0	1	1	0	

Figura 3.7.7

- Bitul 7 –URSEL- se folosește pentru selecția regiștrilor astfel:1 dacă se lucrează cu registrul UCSRC și 0 dacă se lucrează cu registrul UBBRH.
- Bitul 6 –UMSEL-Prin acest bit se setează modul de funcționare al USART-ului:1 dacă funcționează în modul sincron și 0 dacă funcționează în modul asincron.
- Bitul 5 și 4 –UPM1 și UPM0 – Se folosesc pentru selecția parității:

Figura 3.7.8

UPM1	UPM0	
0	0	Nu se folosește bitul de paritate
0	1	Combinație rezervată
1	0	Se folosește bitul de paritate ca bit par
1	1	Se folosește bitul de paritate ca bit impar

- Bitul 3 –USBS- se folosește pentru a seta numărul biților care reprezintă sfârșitul cuvântului: 1- se setează doi biți de stop, 0-se setează un singur bit de stop.
- Biții 2 și 1– UCSZ1 și UCSZ0- se folosesc împreună cu UCSZ2 din UCSRB pentru a seta lungimea cuvântului de date:

UCSZ2	UCSZ1	UCSZ0	Lungimea Cuvântului
0	0	0	5 biți
0	0	1	6 biți
0	1	0	7 biți

0	1	1	8 biți
1	0	0	Combinație rezervată
1	0	1	Combinație rezervată
1	1	0	Combinație rezervată
1	1	1	9 biți

Figura 3.7.9

- Bitul 0 – UCPOL - Se folosește pentru modul sincron și setează ce front al semnalului de pe linia XCK va fi folosit în transmisie. În modul asincron acesta se setează la 0. Valorile pe care le poate lua sunt următoarele: 0 – se folosește frontul crescător, 1 se folosește frontul descrescător.

### 3.7.1 Modulul Bluetooth HC-05

Modulul de comunicare HC-05 permite controlul unui dispozitiv cu microcontroler de la distanță prin Bluetooth.

Acesta are 4 pini:

- doi pini de alimentare : GND și + 5V
- doi pini de interfață serială: RX(Primire date) și TX(Transmitere date).

Modulul Bluetooth poate avea încă doi pini: EN si STATE. Pinul de EN poate fi folosit pentru a comuta între modul de date (low) și modul de comandă (high). În mod implicit este în modul de date.

Pinul STATE este conectat la un LED. Acesta poate fi folosit ca un feedback pentru a verifica dacă modulul Bluetooth funcționează corect.

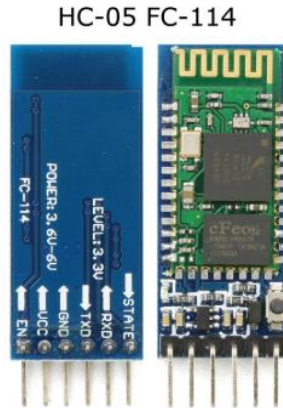


Figura 3.7.1.1

Specificațiile modulului Bluetooth sunt:

- Distanța de transmisie până la 10m
- Tensiune de alimentare: 3.6 - 6V
- Curent consumat: maxim 30mA
- Pinii de I/O sunt compatibili pentru 3.3V
- Comunica pe serial UART
- Baudrate: 9600 - 460800 bps
- Putere de transmisie: +4dBm
- Sensitivitate recepție: -80dBm

HC-05 are două moduri de operare, unul este modul de date în care poate trimite și primi date de la alte dispozitive Bluetooth, iar celălalt este modul AT Command în care se pot modifica setările implicite ale dispozitivului.

Modulul este conectat la microcontroler prin intermediul pinilor RXD și TXD. Pinul RXD de la modul va fi conectat la pinul Tx (pin 1) de la microcontroller, iar TXD la Rx (pin 0).

Doarece pinii de I/O ai modulului Bluetooth sunt compatibili cu 3.3V, se utilizează un convertor pentru a transforma nivelul logic de 5V de la Arduino în 3.3V necesari.

### 3.7.1.1 *Convertor nivel logic Bi-Direcțional*

Acest circuit se utilizează pentru a efectua comunicația între dispozitive ce funcționează la tensiuni diferite. Este un dispozitiv cu 4 canale care coboară în siguranță semnalele de 5V la 3,3V și ridică 3,3V la 5V dar funcționează și cu dispozitive de 2,8V și 1,8V.

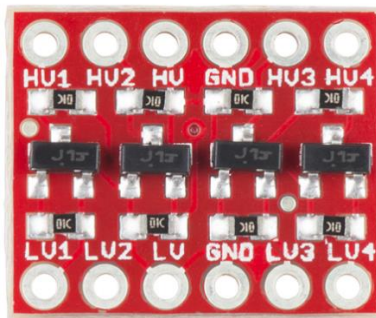


Figura 3.7.1.1.1

Convertorul de nivel logic poate fi folosit cu semnal serial obișnuit, I2C, SPI sau orice alt semnal digital. Nu funcționează cu semnal analogic.

Semnificația pinilor este:

- Intrări tensiune:
  - HV – intrare de înaltă tensiune
  - LV – intrare de joasă tensiune
  - GND – Ground
- Canale de date:
  - HV1, HV2, HV3, HV4 – canale de tensiune înaltă
  - LV1, LV2, LV3, LV4 – canale de tensiune joasă

Tensiunea furnizată la intrările HV și GND trebuie să fie mai mare decât cea furnizată pe partea LV. Este necesară furnizarea unei tensiuni constante, reglementate la ambele intrări.

Schema internă a unui canal de conversie:

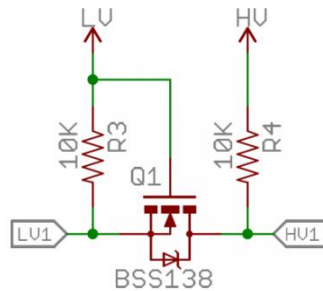


Figura 3.7.1.1.2

Circuitul folosește un MOSFET N-Channel și două rezistențe de pull-up pentru a realiza deplasarea la nivel bidirecțional. Există practic patru astfel de circuite pe placă pentru a crea patru canale de conversie.

Modul de conectare în cadrul proiectului este redat în figura de mai jos:

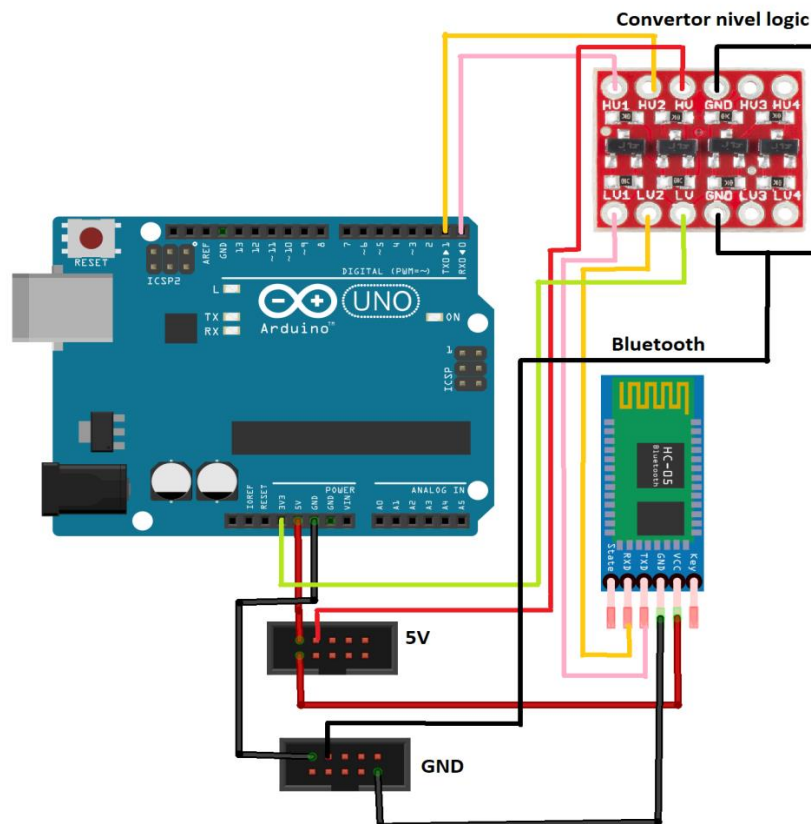


Figura 3.7.1.1.3

### 3.8 Modul Buzzer

Un buzzer este un dispozitiv de semnalizare audio care poate fi mecanic, electromecanic sau piezoelectric. În cadrul proiectului a fost utilizat un buzzer piezoelectric pasiv.



Figura 3.8.1

Caracteristicile tehnice ale acestui modul sunt:

- Tensiune de lucru: 3.3V-5V
- Curent maxim: 30mA
- Interfață I/O

Există 3 pini de conexiune:

- VCC
- GND
- I/O

Pinul I/O se conectează la microcontroller sau la sursa de semnal dreptunghiular pentru a genera sunetul.

În cazul nostru, sunetul a fost generat cu ajutorul unui generator de frecvență și pornit sau oprit de la microcontroler prin intermediul unui circuit întrerupător cu tranzistor.

Astfel, buzzer-ul pasiv devine activ.

#### 3.8.1 Generator de frecvență

Pentru generarea frecvenței necesară funcționării buzzer-ului s-a folosit un modul generator de semnal dreptunghiular cu Timer NE555, frecvență și factor de umplere reglabil.



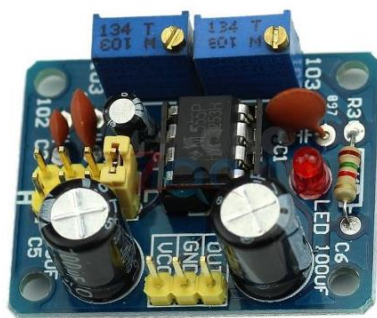


Figura 3.8.1.1

Specificații tehnice sunt:

- Tensiune de alimentare: 5V - 15V;
- Consum curent: maxim 100mA, în funcție de curentul consumat;
- Curent ieșire: 15mA,  $V_{in}=5V$ ; 35mA,  $V_{in}=15V$ ;
- Frecvență semnal: 1Hz - 50Hz, 50Hz - 1kHz, 1kHz - 10kHz, 10kHz - 200kHz, în funcție de conexiunea dată de jumper.
- Generatorul de semnal dreptunghiular are două potențiometre ce pot varia frecvența și factorul de umplere
- Modulul are 3 pini: VCC, GND și OUT.

Gama de frecvențe aleasă în cadrul proiectului este 50Hz – 1kHz, reprezentând gama a doua și a fost selectată prin poziționarea jumper-ului pe pinii corespunzători.

Schema internă a circuitului integrat NE555:

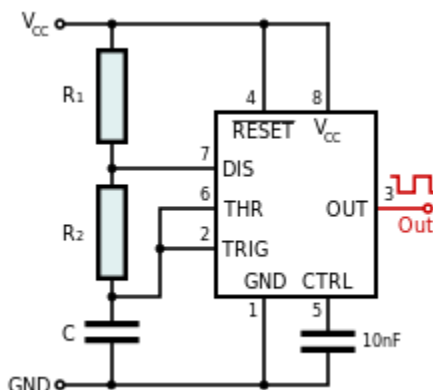


Figura 3.8.1.2

Acest circuit generează semnal în mod continuu și îl transmite către buzzer. Pentru a opri și porni redarea sunetelor s-a implementat un circuit cu rol de întrerupător, prezentat în figura 3.8.1.3.

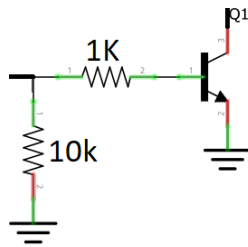


Figura 3.8.1.3

Deoarece curentul colectorului tranzistorului este limitat proporțional de curentul bazei, acesta poate fi folosit pe post de întrerupător controlat în curent. O cantitate relativ mică de electroni prin bază, poate exercita un control asupra unei cantități mult mai mari de electroni prin colector.

Interconectarea elementelor pentru comanda buzzer-ului este prezentată în figura 3.8.1.4.

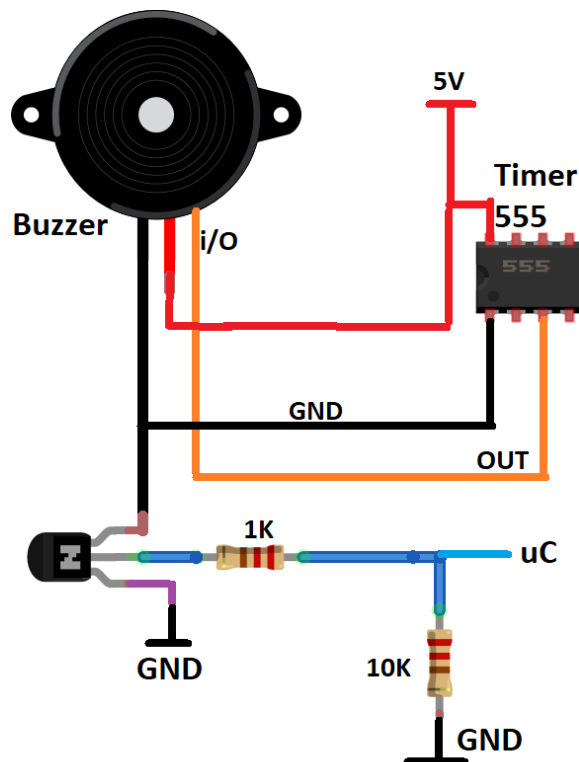


Figura 3.8.1.4

Modul de funcționare:

Atunci când se primește semnal de la microcontroler, tranzistorul intră în starea ON (va conduce) și semnalul generat de generatorul de frecvență (Timer-ul 555) va ajunge la buzzer, care va emite sunete cu frecvența selectată inițial.

Semnalul generat a fost vizualizat cu ajutorul unui osciloscop și este prezentat în figura 3.8.1.5. Frecvența aleasă este de 72.8Hz.

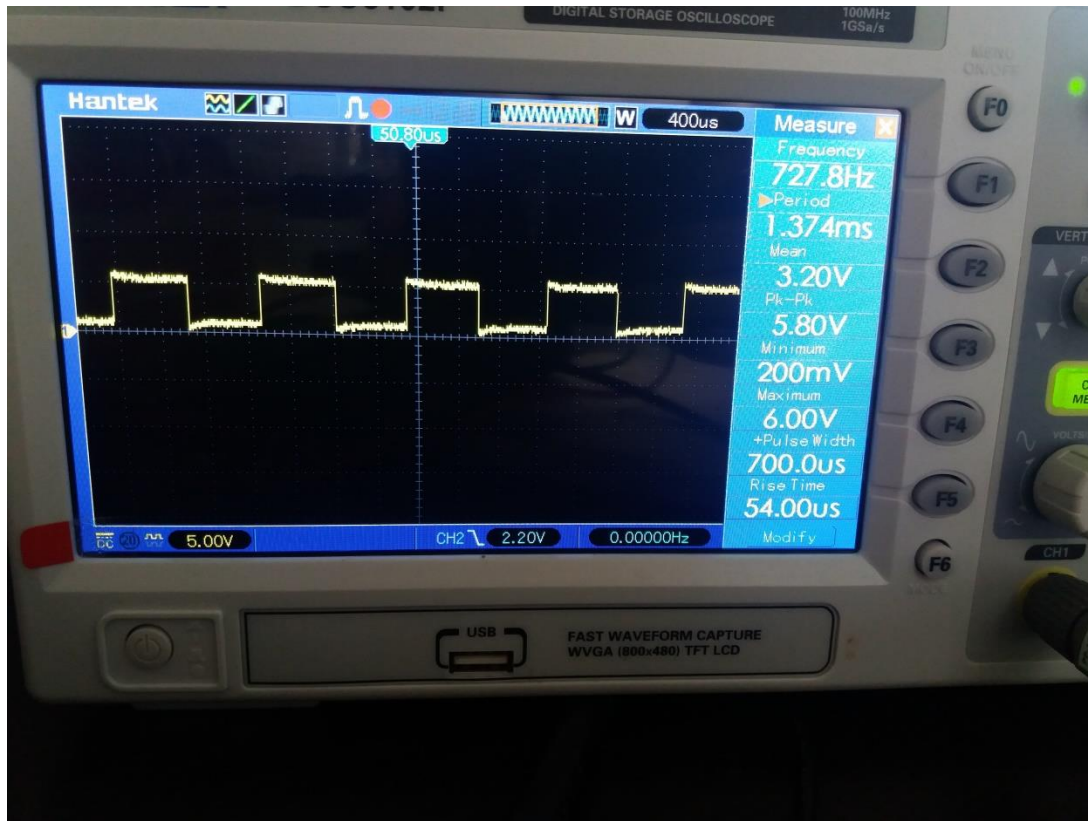


Figura 3.8.1.5

### 3.9 Sistemul de alimentare pentru proiect

#### 3.9.1 Bateria

Bateria electrică este un dispozitiv de stocare a energiei electrice sub formă de energie chimică. Procesul este reversibil, astfel că, la conectarea unui consumator la bornele bateriei, energia chimică se eliberează sub formă de energie electrică. Bateria electrică primară este bateria de unică folosință și nu se poate reîncărca, fiind folosită până la epuizare. Bateria electrică secundară numită și acumulator este reîncărcabilă. Bateria poate avea structura bazată pe una sau mai multe celule. Bateriile mari sunt compuse din pachete de baterii mai mici care sunt conectate în serie pentru a obține o tensiune ridicată sau sunt conectate în paralel pentru a debita un curent mai mare.

Principiul de funcționare al bateriei:

Substanțele chimice din interiorul bateriei produc electroni prin transformarea energiei chimice în energie electrică. Atunci când conectăm o baterie la un accesoriu (gadget), electronii se deplasează de la polul negativ al bateriei, prin dispozitivul pe care îl deservește, închizând circuitul prin polul pozitiv al bateriei. Diferențele dintre diferitele tipuri de baterii sunt date de substanțe chimice aflate în interiorul acestora. Fiecare tip de baterie este conceput astfel încât să deservească unei aplicații dependente de anumite criterii: preț, capacitate, curent maxim, greutate, volum etc.

În proiect au fost utilizate 3 baterii reîncărcabile Litium-Ion de 3.7V(maxim 4.2V) înseriate. Tensiunea totală disponibilă pentru alimentare este de 11.1V tensiune nominală, 12.6 tensiune maximă.

Specificații acumulator:

- Tip baterie: Li-Ion
- Dimensiune: MR18650
- Tensiune minimă: 2.5V
- Tensiune nominală: 3.7V
- Tensiune maximă: 4.2V
- Capacitate: 5000mAh
- Curent descărcare mare: 20A



Figura 3.9.1.1

Bateriile litiu-ion constituie noul standard în ceea ce privește bateriile pentru gadgeturi, reprezentând un progres considerabil sub aspectul energiei stocate.

### 3.9.2 Convertorul coborâtor(Buck)

Tensiunea generată de baterii (12.6V maxim) este mai mare decât tensiunea necesară funcționării servo-motorului (7.2V) și plăcii de dezvoltare Arduino (5V). Din acest motiv, în cadrul proiectului se utilizează un convertor de tip buck (coborâtor).

Convertorul Buck sau Step-Down SMPS este o sursă în comutație de tip coborâtor, unde tensiunea nestabilizată de la intrare este micșorată pentru a produce o tensiune continuă și stabilizată la ieșire.

Schema de bază a unui convertor buck este prezentată în figura de mai jos:

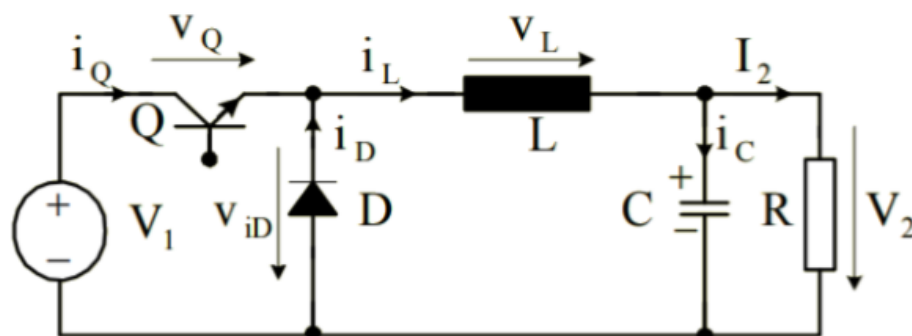


Figura 3.9.2.1

În general, o sursă de putere în comutație constă din cinci componente de bază: 1. Un circuit de control cu modulația impulsurilor în durată (PWM Controller); 2. Un tranzistor cu rol de comutator; 3. O inductanță; 4. O capacitate; 5. O dioda;

Circuitul de control cu modulația impulsurilor în durată este un integrat specializat și are rolul de a furniza semnale de comandă adecvate în scopul reglării și menținerii tensiunii de ieșire la o valoare constantă.

Tranzistorul, pe post de comutator, are rolul de a controla puterea transmisă sarcinii. Acestea pot fi de tip MOS sau bipolare, iar alegerea lor se face în funcție de putere și frecvență.

Inductorul este utilizat cu rol de filtru pentru a reduce riplul de curent. Reducerea are loc deoarece curentul prin inductor nu poate fi schimbat instantaneu. Când curentul prin inductor tinde să scadă, inductorul tinde să-l mențină la valoare constantă, având rolul de sursă de energie. Inductoarele utilizate în aceste convertoare sunt înfășurate de obicei pe miezuri toroidale, din ferită sau fier așchiat cu pierderi reduse la frecvențe înalte.

Capacitatea este utilizată cu rol de filtru pentru a reduce riplul de tensiune. Aceasta trebuie aleasă cu pierderi minime. Pierderile din capacitate sunt datorate rezistenței serie și inductanței proprii. Tipul capacității este ales după rezistența serie efectivă (ESR). Cele mai indicate capacități sunt cele din tantal. Uneori, pentru creșterea performanței regulatorului, se leagă în paralel câteva capacități de valoare mai mică pentru a micșora rezistența serie efectivă.

Dioda folosită este de circulație liberă (free-wheeling). Aceasta nu are rol de redresor, ci are funcția de a direcționa corect calea de curent prin inductanță. Este important ca dioda să comute în starea de blocat foarte rapid, de aceea se vor folosi diode rapide de recuperare sau diode Schottky, care sunt cele mai indicate.

### **Comanda convertorului**

Tranzistorul se comandă cu frecvența  $f=1/T$ , menținându-se saturat pe o durată  $dT$  și blocat pe o durată  $(1-d)T$ , unde  $d$  reprezintă factorul de umplere al semnalului de comandă.

Controlul convertorului buck poate fi făcut în două moduri:

- Funcționarea la frecvență constantă, sau controlul prin modularea impulsurilor în durată(PWM);
- Funcționarea la frecvențe variabile, sau controlul prin modularea în frecvență;

Funcționarea convertorului trebuie analizată în două intervale distincte de timp:

- intervalul I, în care tranzistorul Q conduce la saturație, iar dioda D este blocată, fiind polarizată invers. Considerând originea de timp în momentul comutației directe a lui Q, acest prim interval va fi:  $t \in [0, dT]$ .

Schemă electrică echivalentă este:

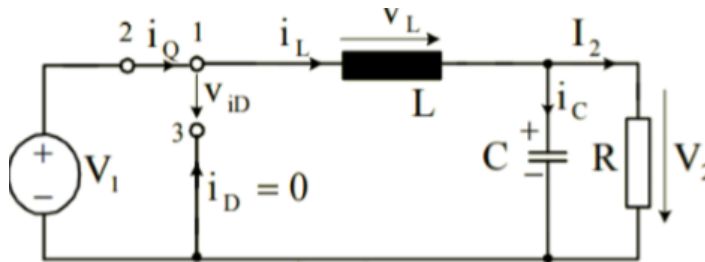


Figura 3.9.2.2

Valori calculate:

$$V_L = V_1 - V_2 = L \frac{di_L}{dt}, t \in [0, dT]$$

$$i_Q = i_L = I_{Lm} + \frac{V_1 - V_2}{L} t, t \in [0, dT]$$

- intervalul II, în care tranzistorul Q este blocat, iar dioda D conduce, asigurând închiderea curentului  $i_L$  menținut de la inductanța L.

Schemă electrică echivalentă:

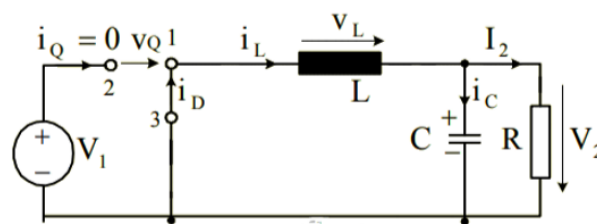


Figura 3.9.2.3

Valori calculate:

$$V_L = -V_2 = L \frac{di_L}{dt}, t \in [0, dT]$$

$$i_D = i_L = I_{LM} - \frac{V_2}{L}(t - dT), t \in [dT, t]$$

Pe baza relațiilor deduse au fost trasate formele de undă. Forma de undă a tensiunii  $v_L$  ne permite să deducem caracteristica de reglaj a convertorului. Deoarece valoarea medie a tensiunii pe inductanța  $L$  este nulă ( $V_{Lavr} = 0$ ).

În figura de mai jos sunt prezentate formele de undă specifice convertorului:

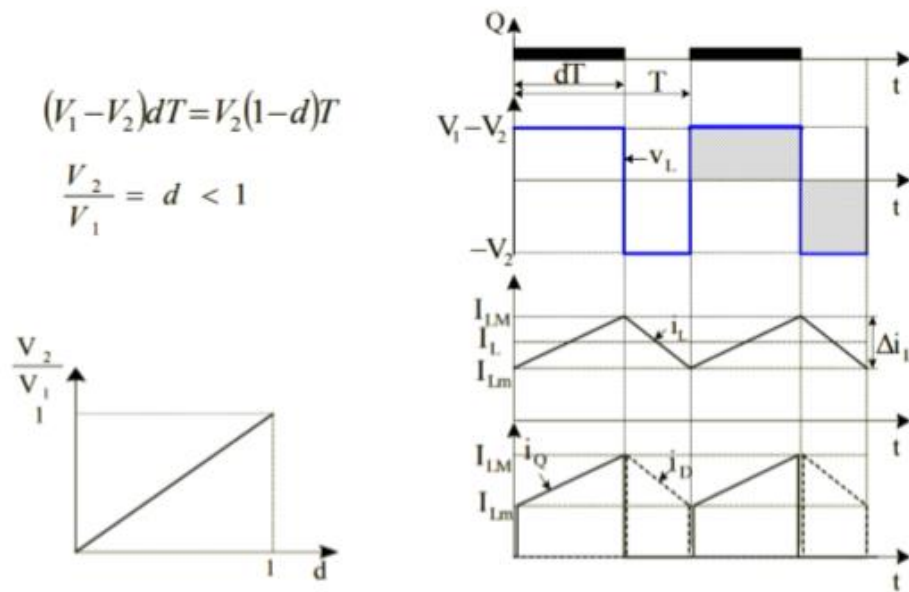


Figura 3.9.2.4

Convertorul buck utilizat în proiect este XL4005.

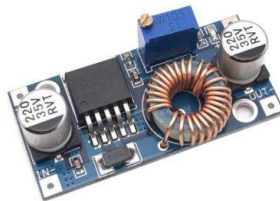


Figura 3.9.2.5



Specificații acestui convertor sunt:

- Tensiune de alimentare: 4V-38V
- Tensiune de ieșire: 1.25V-32V
- Curent de ieșire: ajustabil, maxim 5A
- Frecvența de funcționare: 180KHz
- Temperatura de funcționare: -40 °C până la +85 °C
- Dimensiune: 44x21x12mm

## 4 Descrierea aplicației

### 4.1 Descrierea funcționalității

Aplicația prezentată este un robot autonom care are mai multe funcții:

- Control de la distanță
- Monitorizare tensiune baterii
- Line follower
- Detecție de obstacole

Controlul de la distanță este realizat prin intermediul unui set de comenzi primite prin Bluetooth de la telefonul mobil. Robotul poate fi manevrat înainte și înapoi cu ajutorul unor butoane, și poate efectua viraje prin simpla înclinare a telefonului. Viteza de deplasare a robotului este reglabilă. Toate comenzile se transmit în timp real, făcând controlul ușor și precis.

Pentru ca verificarea stării bateriilor care alimentează mașina să fie mai ușoară și descărcarea completă a acestora să poată fi evitată a fost implementat un circuit de monitorizare a tensiunii și afișarea acesteia în procente pe interfața cu utilizatorul.

Când este în modul Line follower, microcontrolerul primește date prin intermediul unui senzor fotoelectric cu 3 canale IR (InfraRed – Infra rosu) care ajută robotul să se mențină pe traseul de culoare neagră.

Detecția de obstacole este realizată cu ajutorul a doi senzori ultrasonici care măsoară distanța până la acestea. Când distanța devine mai mică de 25 cm, microcontrolerul generează sunete de avertizare prin intermediul unui buzzer. Când distanța atinge 15 cm, motoarele se opresc automat.

## 4.2 Control de la distanță

### 4.2.1 Implementare hardware

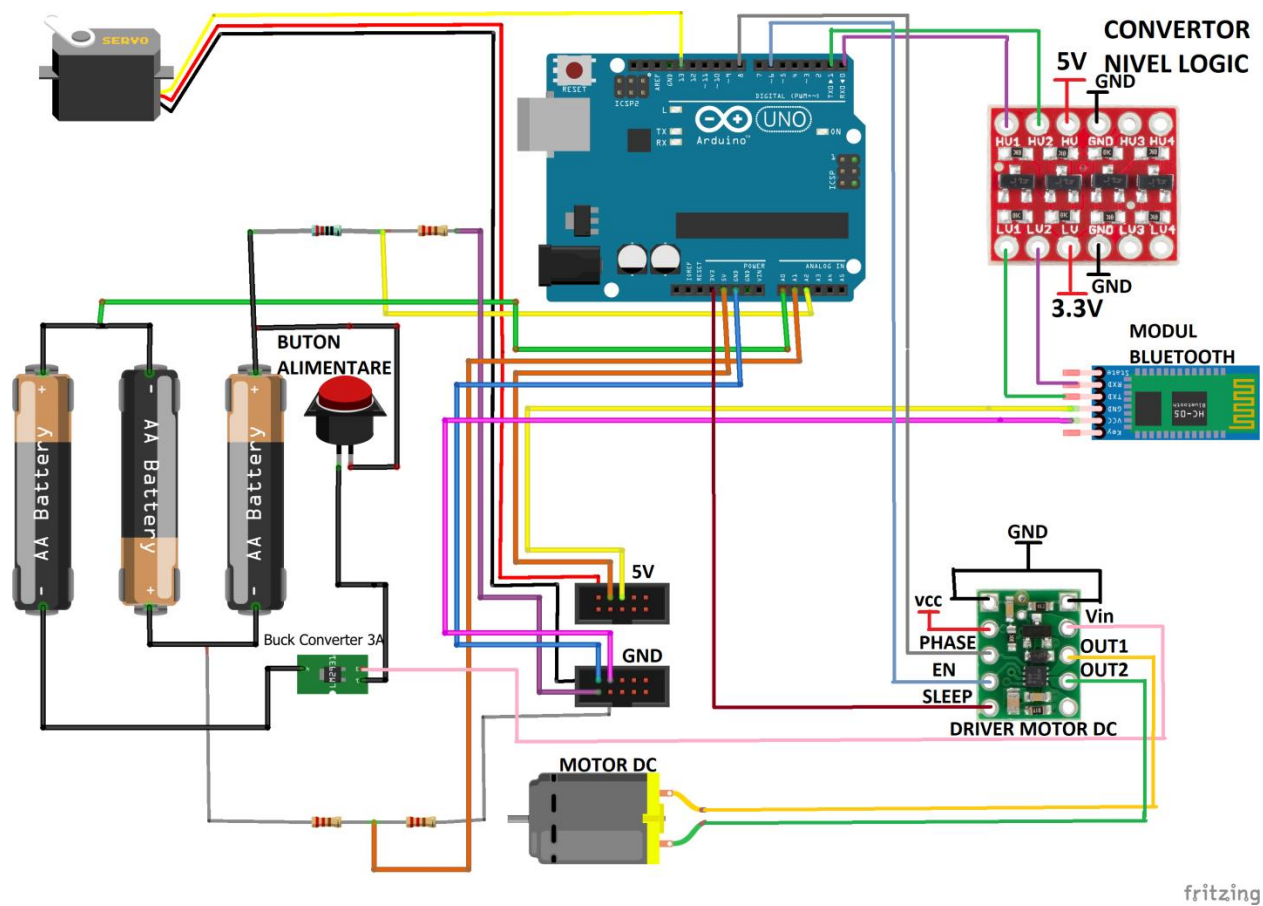


Figura 4.2.1.1

Pentru primirea și transmiterea datelor de la distanță este necesară conectarea modulului Bluetooth la microcontroler. Schimbul de date între modulul bluetooth și microcontroler se va realiza cu ajutorul unei interfețe seriale UART (Universal asynchronous receiver/transmitter).

În cazul nostru este nevoie de o transmisie bidirecțională.

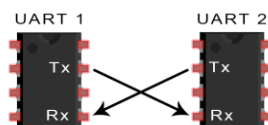


Figura 4.2.1.2

Pinul Rx de la placa Arduino va fi conectat la pinul Tx de la dispozitivul bluetooth, asigurând transmiterea datelor de la telefon la microcontroler, iar pinul Tx de la placa Arduino va fi conectat la pinul Rx de la dispozitivul Bluetooth, asigurând transmiterea datelor de la microcontroler spre telefon, așa cum se prezintă în figura 4.2.1.3.

Interconectarea dintre pinii Rx și Tx se face prin intermediul unui convertor de nivel logic, deoarece modulul bluetooth și placa Arduino Uno nu sunt compatibile (Arduino Uno funcționează cu 5V iar Bluetooth cu 3.3V).

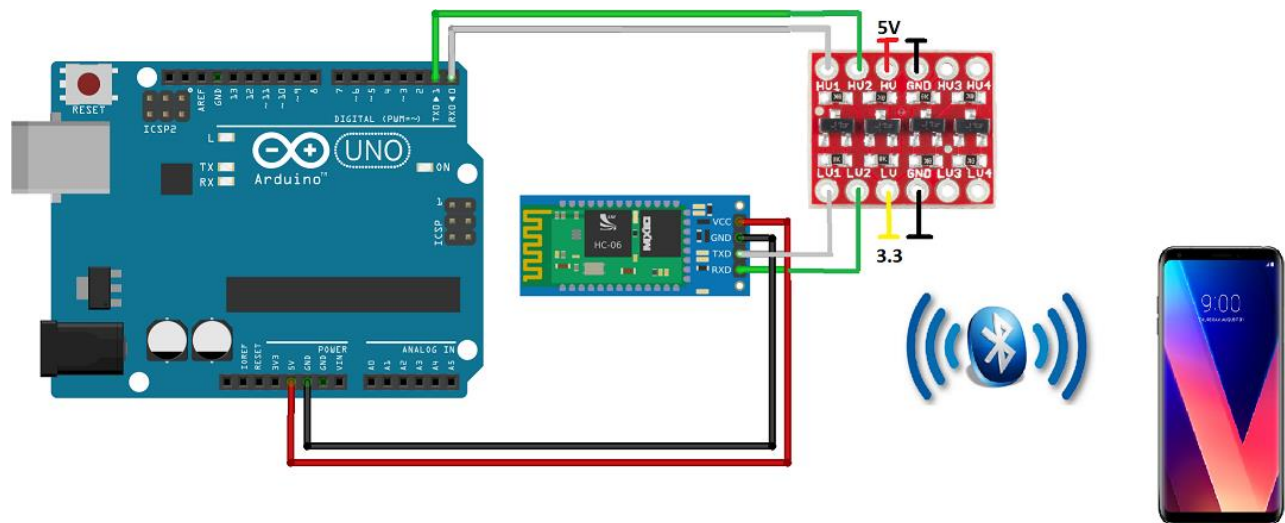


Figura 4.2.1.3

De asemenea, bluetooth-ul din cadrul telefonului mobil trebuie să fie activat și împerecheat cu respectivul modul bluetooth. Pentru ca transmisia să se realizeze eficient este necesar ca viteza de transfer să fie egală la transmisie și recepție și să avem aceeași lungime a cuvântului de date.

Viteza de transmisie a datelor se măsoară în BAUD, unitate care reprezintă numărul de biți transmiși într-o secundă. Ratele de BAUD suportate de Atmega328p sunt 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600 și 115200.

Rata de BAUD folosită în cadrul proiectului este 9600 bps și este definită software ca în figura 4.2.1.3.

```

void init_serial_interface(void)
{
    Serial.begin(9600); // opens serial port, sets baudrate to 9600 bps
}

```

Figura 4.2.1.4

## 4.2.2 Interfața cu utilizatorul

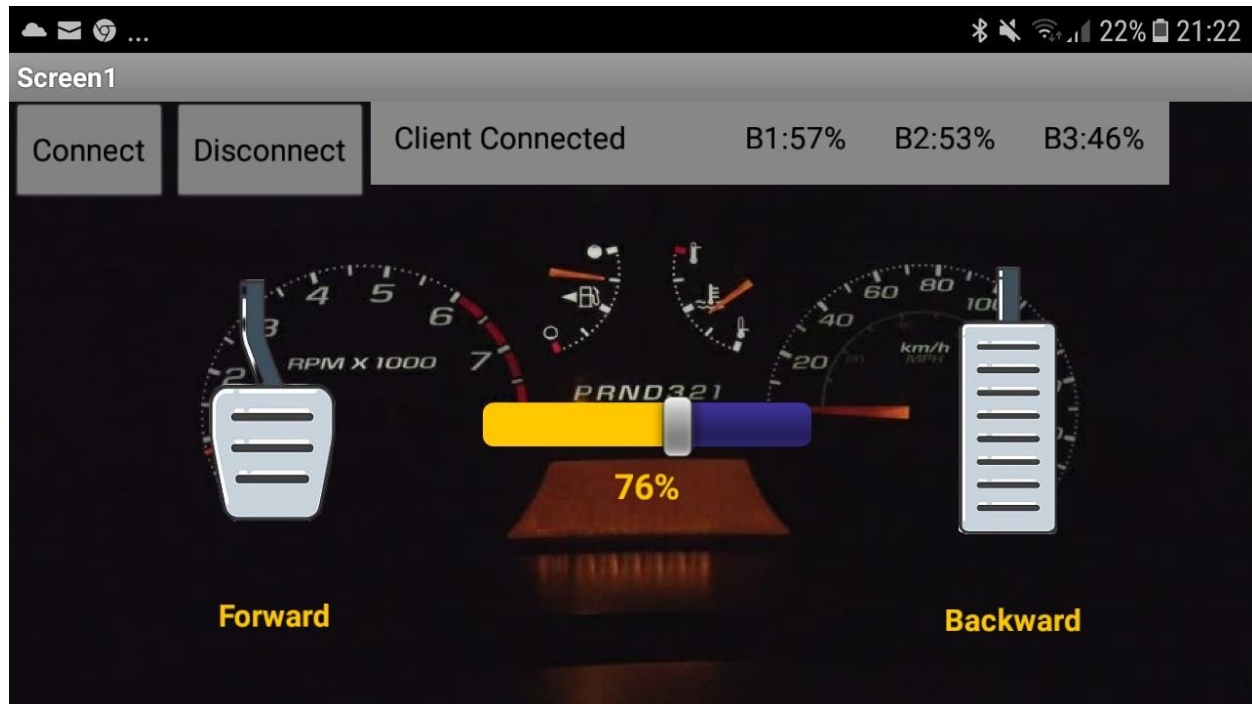


Figura 4.2.2.1

Figura 4.2.2.1 prezintă aplicația prin intermediul căreia se transmit comenzile către microcontroler. Aplicația a fost dezvoltată cu ajutorul software-ului Arduino IDE.

Elemente și funcționare:

- Butonul Connect - conectează telefonul mobil la modulul bluetooth atașat mașinii. În cazul în care telefonul nu are bluetooth activat, pe interfața cu utilizatorul va apărea o notificare în acest sens. Odată ce conectarea are loc, va fi afișat în textbox-ul numit “status” mesajul: Client Connected.
- Butonul Disconnect – deconectează bluetooth

În figura 4.2.2.2 se prezintă implementarea software care asigură funcționalitățile acestui buton:

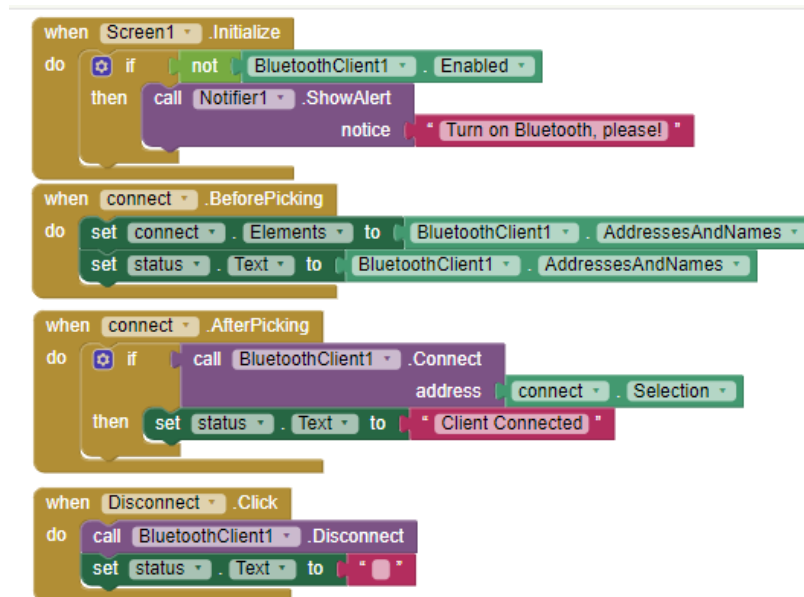


Figura 4.2.2.2

- Butonul Forward – deplasează robotul pe direcția înainte
- Butonul Backward – deplasează robotul pe direcția înapoi
- Slider – reglează viteza de deplasare între limitele 0 – 127.

Implementarea software pentru deplasarea înainte și înapoi este prezentată mai jos:

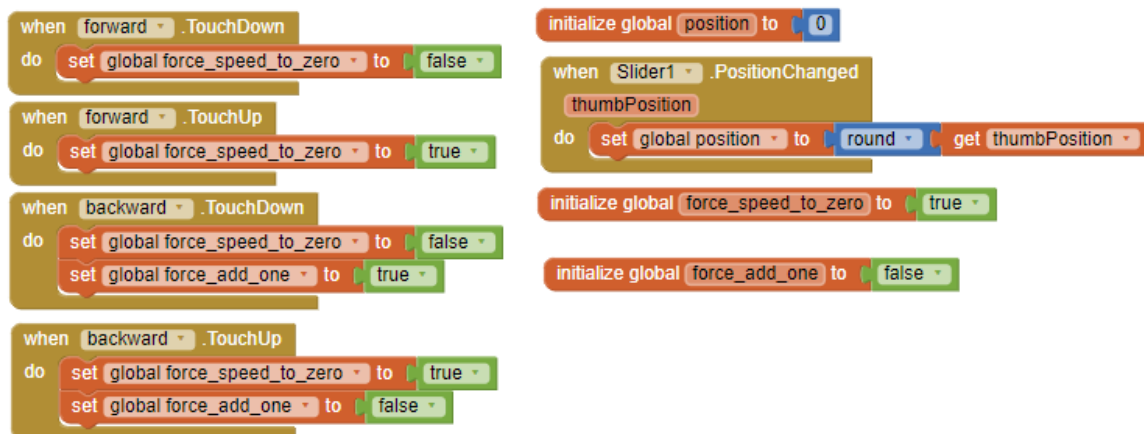


Figura 4.2.2.3

Butoanele Forward și Backward execută comenzile doar dacă sunt menținute apăsat.

Pentru ca deplasarea să poată avea loc trebuie ca viteza să fie mai mare ca zero.

Datele se trimit cu ajutorul unui Timer la un interval de 30ms. Se vor trimite trei bytes ce conțin datele necesare pentru ca robotul să funcționeze corespunzător. Aceste date sunt:

- Indicativul '11' care comandă deplasarea (pentru oprire se trimite indicativul '10')
- Unghiul de direcție
- Viteza de deplasare

Unghiul de direcție controlează servo-motorul. Pentru a controla acest unghi s-a folosit senzorul accelerometru din dotarea smartphone-ului. Astfel, se măsoară unghiul de înclinare al telefonului care va fi în intervalul [-9, 9] după care se adaptează această valoare la intervalul [0, 100] pentru a fi mai ușor de utilizat.

Formula utilizată este:

$$\text{Val transmisă} = (\text{Val citită} \times 5) + 50$$

Valoarea astfel calculată se trimite prin bluetooth și la rândul ei este utilizată pentru a comanda servo-motorul.

Implementarea software pentru controlul direcției de deplasare este redată în Fig. 4.2.2.4:

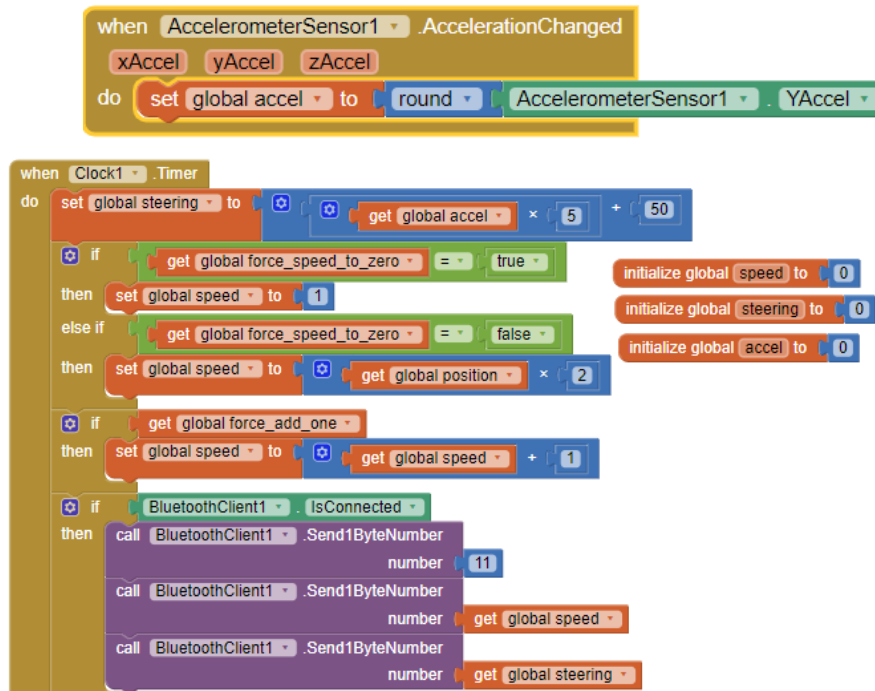


Figura 4.2.2.4

Byte-ul pentru viteza de deplasare conține și un bit folosit pentru selectarea mersului înainte sau înapoi. Astfel, bitul 0, cel mai puțin semnificativ va fi 0 logic dacă butonul Forward este apăsat și 1 logic dacă butonul Backward este apăsat.

Ceilalți 7 biți sunt folosiți pentru a coda valoarea vitezei. De exemplu, dacă slider-ul este la valoarea sa maximă (127), byte-ul trimis va fi:  $127 = 01111111$  (bin).

Se va face o shiftare a biților la stânga, de unde rezultă :

$$\text{Byte} = 11111110 = 254$$

Dacă se dorește ca deplasarea să se facă înainte, bitul 0 va rămâne 0 logic, în caz contrar va fi 1 logic, adică se va adăuga 1 la valoarea 254. Rezultă ca byte-ul final va fi:

$$\text{Byte} = 11111110 = 254 \text{ pentru mersul înainte}$$

$$\text{Byte} = 11111111 = 255 \text{ pentru mersul înapoi}$$

Descriere algoritm:



Se verifică dacă există date disponibile pentru primire. Dacă da, un contor începe să numere de la 0 până la 2 și se primesc și se afișează cei trei bytes de date. Când contorul ajunge la 2 se resetează și procesul se reia.

#### 4.2.3 Programarea microcontrolerului

- **Controlul motorului DC care asigură tracțiunea mașinii:**

Se utilizează numai doi pini digitali ai microcontrolerului: 6 și 8 care sunt configurați ca în figura 4.2.3.1.

```
// traction
#define TRACTION_PHASE_PIN 8
#define TRACTION_ENABLE_PIN 6
```

Figura 4.2.3.1

Pentru o bună funcționare, pinii Enable și Phase ai driverului de motor DRV8838 primesc niște setări inițiale, ca în figura 4.2.3.2.

```
void setup()
{
    pinMode(TRACTION_PHASE_PIN, OUTPUT);
    pinMode(TRACTION_ENABLE_PIN, OUTPUT);
    digitalWrite(TRACTION_PHASE_PIN, LOW);
    digitalWrite(TRACTION_ENABLE_PIN, LOW);
}
```

Figura 4.2.3.2

Funcția **setup()** este executată o singură dată, la inițializarea plăcii (de fiecare dată când este alimentată și de fiecare dată când este resetată placa).

Ulterior s-au dezvoltat un număr de funcții cu ajutorul cărora se controlează motorul DC (Figura 4.2.3.3).

```
void car_forward(unsigned char speed);
void car_backward(unsigned char speed);
void car_stop(void);
```

Figura 4.2.3.3

Fig. 4.2.3.4 prezintă codul asociat fiecărei funcții în parte:

```

void car_forward(unsigned char speed)
{
    digitalWrite(TRACTION_PHASE_PIN, LOW);
    analogWrite(TRACTION_ENABLE_PIN, speed);
}

void car_backward(unsigned char speed)
{
    digitalWrite(TRACTION_PHASE_PIN, HIGH);
    analogWrite(TRACTION_ENABLE_PIN, speed);
}

void car_stop(void)
{
    digitalWrite(TRACTION_PHASE_PIN, LOW);
    analogWrite(TRACTION_ENABLE_PIN, 0);
}

```

Figura 4.2.3.4

Cele trei funcții au rolul de a acționa motorul DC(**car\_forward()**; și **car\_backward()**; ) sau de a-l opri (**car\_stop()**).

- *Controlul servo-motorului care asigură virarea la stânga sau la dreapta.*

Pentru ca placa Arduino Uno să poată controla motoare cu servo-direcție este necesară includerea în proiect a bibliotecii Servo.h.

Configurarea pinilor:

Microcontrolerul folosește un singur pin pentru generarea semnalului PWM care va controla unghiul de rotație al servo-motorului (Figura 4.2.3.5).

```

// steering
#define STEERING_PIN 13

```

Figura 4.2.3.5

În urma unor teste au fost definite limitele între care se poate regla unghiul de direcție (Figura 4.2.3.6).

```

#define STEERING_MIN_POSITION 50 // turning maximum left
#define STEERING_MAX_POSITION 130 // turning maximum right
#define STEERING_CENTER 80

```

Figura 4.2.3.6

Definirea valorilor de steering ideale primite de la telefon(Figura 4.2.3.7):

```
#define STEERING_MIN_POSITION_CMD 0 // turning maximum left
#define STEERING_MAX_POSITION_CMD 100 // turning maximum right
#define STEERING_CENTER_CMD 50
```

Figura 4.2.3.7

Setări inițiale cu ajutorul funcției **setup()**; :

```
digitalWrite(STEERING_PIN, LOW);
pinMode(STEERING_PIN, OUTPUT);
steering.attach(STEERING_PIN);
center_steering();|
```

Figura 4.2.3.8

Funcțiile cu ajutorul cărora servo-motorul este controlat sunt:

```
center_steering();
void car_steer(unsigned char position);
int steering_cmd_to_real(unsigned char steering_cmd);|
```

Figura 4.2.3.9

Descriere funcții:

- **center\_steering(void)** – Centrează roțile mașinii  
Această funcție este apelată în setările inițiale dar și când comanda primită prin bluetooth necesită ca roțile mașinii să fie în poziția inițială.
- **car\_steer(unsigned char position)** și **steering\_cmd\_to\_real(unsigned char steering\_cmd)** – Aceste funcții preiau unghiul de direcție primit prin bluetooth și îl adaptează astfel încât modificarea direcției să se facă corespunzător comenzii date de către utilizator.

```

void center_steering(void)
{
    steering.write(STEERING_CENTER);
}

void car_steer(unsigned char position)
{
    if(position < STEERING_MIN_POSITION)
    {
        steering.write(STEERING_MIN_POSITION);
        return;
    }
    else if (position > STEERING_MAX_POSITION)
    {
        steering.write(STEERING_MAX_POSITION);
        return;
    }
    else
    {
        steering.write(position);
        return;
    }
}

int steering_cmd_to_real(unsigned char steering_cmd)
{
    // [A, B] --> [a, b]
    // se transforma intervalul [0, 100] primit de la telefon in intervalul [50, 130]
    int value = steering_cmd;
    int A, B, a, b;
    int real_value;
    if(value <= STEERING_CENTER_CMD)
    {
        A = STEERING_MIN_POSITION_CMD;
        B = STEERING_CENTER_CMD;
        a = STEERING_MIN_POSITION;
        b = STEERING_CENTER;
    }
    else
    {
        A = STEERING_CENTER_CMD;
        B = STEERING_MAX_POSITION_CMD;
        a = STEERING_CENTER;
        b = STEERING_MAX_POSITION;
    }
    real_value = (value - A) * (b - a) / (B - A) + a;
    return real_value;
}

```

Figura 4.2.3.10

Cu ajutorul funcției **steering\_cmd\_to\_real()** se transformă intervalul [0, 100] primit de la telefon în intervalul [50, 130] specific pentru servo-motor.

Primirea datelor pentru controlul de la distanță se face cu ajutorul funcției **Serial.read()**.

Se verifică în mod continuu ca setul de date primit să fie complet. În caz contrar datele se șterg și se citește din nou portul serial.

```
void listen_serial(void)
{
    if(Serial.available() == 0)
    {
        no_cmd_received_counter ++;
        if(no_cmd_received_counter > NO_CMD_RECEIVED_STOP_THRESHOLD)
        {
            no_cmd_received_counter = 0;
            car_stop();
        }
    }

    if(Serial.available() != 3)
    {
        clear_serial_buffer();
        no_cmd_received_counter = 0;
        return;
    }
    else
    {
        no_cmd_received_counter = 0;
        cmd = Serial.read();
        data_0 = Serial.read();
        data_1 = Serial.read();
        execute_command(cmd, data_0, data_1);
        return;
    }
}
```

Figura 4.2.3.11

Cei trei bytes primiți sunt stocați în trei variabile care apoi sunt folosite pentru execuția altor funcții. Primul byte va conține indicativul care permite sau nu deplasarea, acesta fiind codat prin următoarele valori:

```
#define CMD_STOP 10
#define CMD_DRIVE 11
```

Figura 4.2.3.12

Ceilalți doi bytes reprezintă viteza motorului DC (care include și selecția deplasării înainte și înapoi) și unghiul de direcție a servo-motorului.

După primirea setului de date se verifică primul byte și se apelează funcția specifică lui. Dacă acesta conține indicativul 10, ceilalți doi bytes nu mai contează.

```
void execute_command(unsigned char cmd, unsigned char data_0, unsigned char data_1)
{
    switch(cmd)
    {
        case CMD_STOP:
        {
            car_stop();
            break;
        }
        case CMD_DRIVE:
        {
            car_drive_cmd(data_0, data_1);
            break;
        }
    }
}
```

Figura 4.2.3.12

În cazul în care indicativul este '11', funcția **car\_drive\_cmd()** va fi apelată și mașina se va deplasa în funcție de viteza și unghiul de control a direcției primit prin Bluetooth.

Pentru ca citirea datelor să aibă loc în mod continuu, se apelează funcția pentru citire **listen\_serial()** în interiorul funcției **loop()** care se execută la infinit cu o întârziere de 25ms.

```
void loop()
{
    listen_serial();
    delay(25);
}
```

Figura 4.2.3.13

### 4.3 Monitorizare alimentare

Modul monitorizare alimentare permite utilizatorului să urmărească în timp real tensiunea de la bornele bateriei. Datele sunt preluate prin intermediul modului ADC (Analog to Digital Converter) al microcontrolerului, interpretate și apoi trimise prin Bluetooth și afișate pe telefonul mobil, în cadrul aplicației create pentru controlul mașinii.

#### 4.3.1 Implementare Hardware

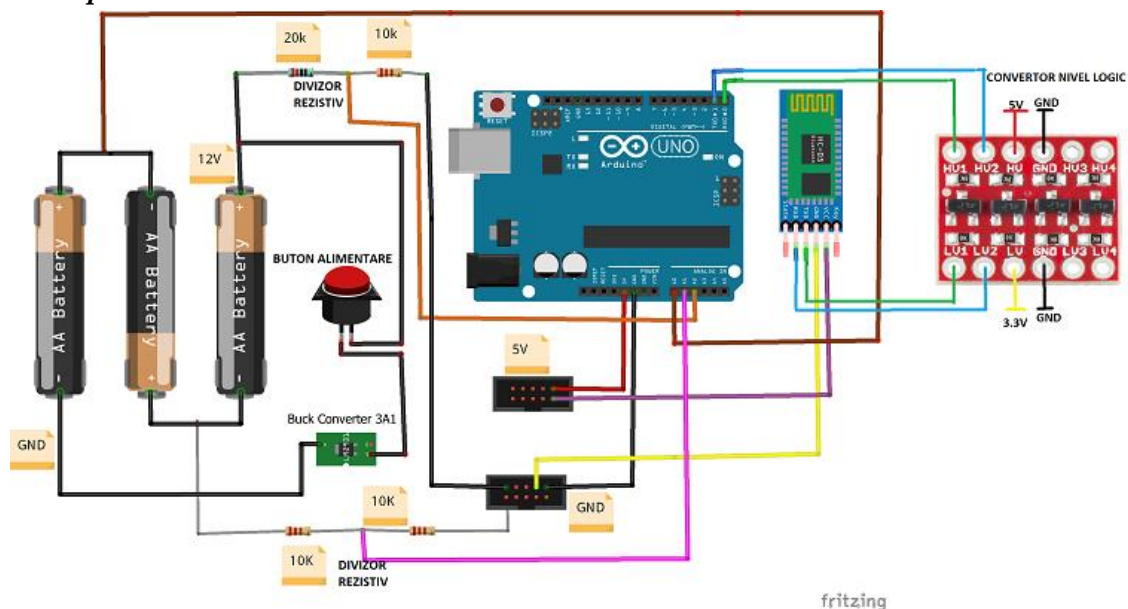


Figura 4.3.1.1

Arduino UNO are 6 canale ADC de la A0 la A5. În proiect s-au folosit canalele A0, A1 și A2 pentru cele trei baterii.

Tensiunea bateriilor este de maxim 12.6V. Având în vedere că acestea sunt în serie, pentru ca ADC să calculeze tensiunea pentru fiecare baterie în parte s-au folosit divizoare rezistive pentru bateriile 2 și 3, pentru prima nefiind necesar (Figura 4.3.1.2).

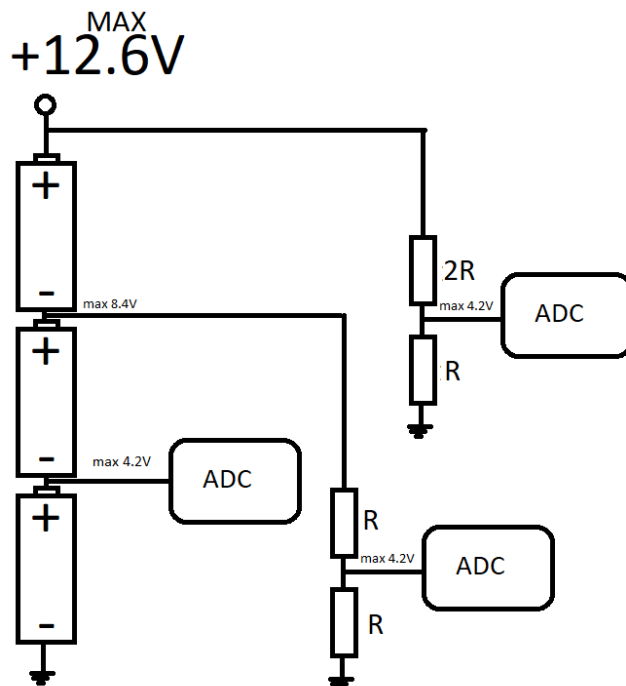


Figura 4.3.1.2

Având în vedere că valoarea tensiunii de referință a ADC-ului este de 5V, am ales valoarea de 4.2V ca valoare maxim acceptată pe intrarea fiecărui canal de ADC. Pe baza acestei informații s-au putut determina valorile necesare pentru rezistențele care alcătuiesc divizoarele rezistive.

Pentru bateria 2 s-au ales două rezistențe egale de valoare 10KΩ. În acest caz, tensiunea maximă la ieșirea divizorului rezistiv poate fi calculată astfel:

$$V_{in} = 8.4V \text{ maxim}$$

$$R1 = 10K\Omega$$

$$R2 = 10K\Omega$$

$$V_o = V_{in} \times \frac{R2}{R1+R2} = 8.4V \times \frac{1000}{1000+1000} = 8.4V \times \frac{1}{2} = 4.2V \text{ (maxim)}$$



Pentru bateria 3 s-au ales două rezistențe cu valorile de  $20K\Omega$ , respectiv  $10K\Omega$ . Tensiunea maximă la iesirea divizorului poate fi calculata astfel:

$$V_{in} = 12.6V \text{ maxim}$$

$$R1 = 20K\Omega$$

$$R2 = 10K\Omega$$

$$V_o = V_{in} \times \frac{R2}{R1+R2} = 12.6V \times \frac{1000}{2000+1000} = 12.6V \times \frac{1}{3} = 4.2V \text{ (maxim)}$$

#### 4.3.2 Implementare software

Pentru activarea caracteristicii ADC folosim funcția **analogRead()** care citește valoarea tensiunii aplicate pe pin-ul analogic specificat. Astfel ADC-ul va citi tensiuni de intrare între 0 și tensiunea de operare (4.2 maxim) și va returna valori întregi între 0 și 1023, întrucât rezoluția convertorului este de 12 biți.

Configurare pini:

```
int Battery_1 = A0;  
int Battery_2 = A1;  
int Battery_3 = A2;
```

Figura 4.3.2.1

Inițializare variabile de stocare a rezultatelor:

```
int digitalVal_b1 = 0;  
int digitalVal_b2 = 0;  
int digitalVal_b3 = 0;  
float Battery1_voltage = 0.00;  
float Battery2_voltage = 0.00;  
float Battery3_voltage = 0.00;  
int procent1 = 0;  
int procent2 = 0;  
int procent3 = 0;
```

Figura 4.3.2.2

Pentru calculul valorilor digitale, analogice si procente s-a implementat funcția **void Battery()**.

```

void Battery()
{
    digitalVal_b1 = analogRead(A0); // read the value from the analog channel
    //convert digital value to analog voltage
    Battery1_voltage = (digitalVal_b1 * 5.00)/1023.00;
    procent1 = (Battery1_voltage /4.2)*100;
    //battery 2
    digitalVal_b2 = analogRead(Battery_2); // read the value from the analog channel
    //convert digital value to analog voltage
    Battery2_voltage = (digitalVal_b2 * 5.00)/1023.00;
    procent2 = (Battery2_voltage /4.2)*100;
    //battery 3
    digitalVal_b3 = analogRead(Battery_3); // read the value from the analog channel
    //convert digital value to analog voltage
    Battery3_voltage = (digitalVal_b3 * 5.00)/1023.00;
    procent3 = (Battery3_voltage /4.2)*100;
}

```

Figura 4.3.2.3

Pentru trimiterea datelor către interfața cu utilizatorul se apelează funcția **sendData()** (Figura 4.3.2.4) în interiorul buclei infinite **loop()**.

```

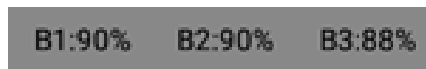
void sendData()
{
    Serial.write(procent1);
    Serial.write(procent2);
    Serial.write(procent3);
}

```

Figura 4.3.2.4

### 4.3.3 Afişarea datelor pe interfața cu utilizatorul

Datele sunt primite în mod continuu de la microcontroler și sunt afișate sub formă de procent pentru fiecare baterie în parte, ca în figura 4.3.3.1.



B1:90% B2:90% B3:88%

Figura 4.3.3.1

Aceste date sunt primite și afișate prin intermediul unui timer la un interval de timp de 500ms.

Codul sursă asociat monitorizării bateriilor este următorul:

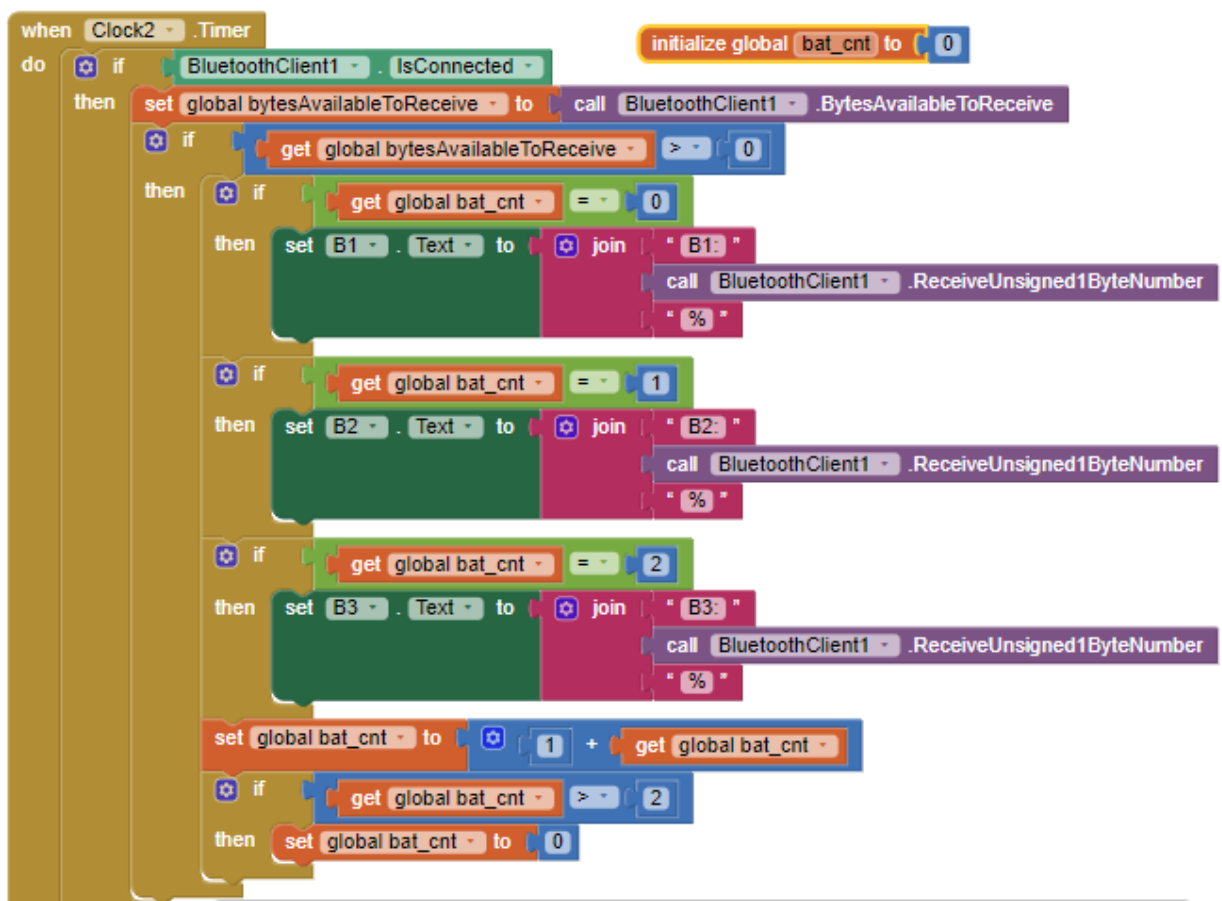


Figura 4.3.3.2

#### 4.4 Detecție de obstacole

Un pas important în dezvoltarea unui robot autonom este implementarea unui mecanism de detecție de obstacole respectiv de ocolire a obstacolelor. În cadrul aplicației de față am implementat cu ajutorul senzorilor ultrasonici un mecanism care să permită detecția de obstacole, iar în cazul în care distanța detectată până la obiect este mult prea mică, mașina este programată să se oprească. Astfel, dacă senzorii detectează obiecte la o distanță din intervalul 15-25cm atunci mașina este programată să emită sunete de avertizare. Dacă distanța față de obiect este sub 15cm mașina se oprește automat astfel evitându-se o posibilă coliziune cu obiectul detectat.

#### 4.4.1 Implementare hardware

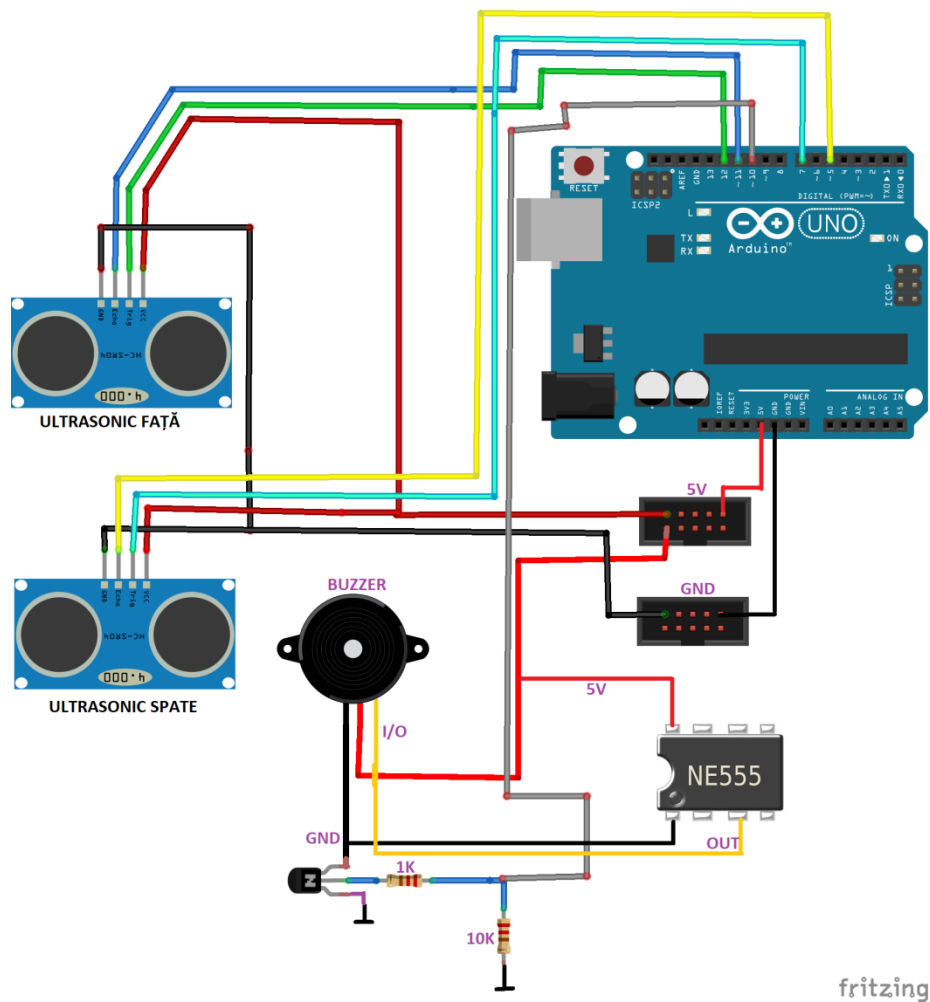


Figura 4.4.1.1

S-au folosit doi senzori ultrasonici HC-SR04, un buzzer, un circuit de temporizare NE555 și un circuit întrerupător cu tranzistor bipolar.

#### 4.4.2 Implementare software

Pentru ca senzorul ultrasonic SR04 să funcționeze corespunzător este necesară adăugarea în proiect a bibliotecii NewPing.

```
#include <NewPing.h>
```

Figura 4.4.2.1

Configurarea pinilor și definirea variabilelor necesare calculului distanței până la un posibil obstacol:

```
#define trigPin1 12
#define echoPin1 11
#define trigPin2 7
#define echoPin2 5
float duration1, distance1, duration2, distance2;
```

Figura 4.4.2.2

Setările inițiale necesare:

```
pinMode(trigPin1, OUTPUT);
pinMode(echoPin1, INPUT);
pinMode(trigPin2, OUTPUT);
pinMode(echoPin2, INPUT);
```

Figura 4.4.2.3

Pentru calculul distanței au fost dezvoltate funcțiile:

- void *Obstacle\_Detect\_Front()*

Această funcție calculează distanța până la obstacolul din fața mașinii și în cazul în care distanța devine mai mică de 25 cm, este setat un flag care activează buzzer-ul (Fig. 4.4.2.4)

Tot în această funcție se dă și comanda de oprire a motoarelor când se atinge distanța de 15cm, fiind permisă doar deplasarea înapoi.

- void *Obstacle\_Detect\_Back()*

Această funcție corespunde senzorului ultrasonic din spatele mașinii și funcționează identic cu funcția anterioară, cu excepția că după oprire va fi permisă doar deplasarea înainte.

```

void Obstacle_Detect_Front()
{
    digitalWrite(trigPin1, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin1, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin1, LOW);
    duration1 = pulseIn(echoPin1, HIGH);
    distancel = (duration1 / 2) * 0.0343;
    if(distancel <= 25 && distancel > 15)
    {
        ALARM_FWD_ON = 1;
        FIRST_TIME_ALARM_FWD_TRIGGERED = 0;
    }
    else if (distancel <= 15)
    {
        if(!FIRST_TIME_ALARM_FWD_TRIGGERED)
        {
            FIRST_TIME_ALARM_FWD_TRIGGERED = 1;
            car_stop();
        }
        ALARM_FWD_ON = 0;
        DRIVE_FWD_ALLOWED = 0;
    }
    else
    {
        ALARM_FWD_ON = 0;
        DRIVE_FWD_ALLOWED = 1;
        FIRST_TIME_ALARM_FWD_TRIGGERED = 0;
    }
}

```

Figura 4.4.2.4

Funcționarea senzorilor și calculul parametrilor:

HC-SR04 trimite un puls ultrasonic la pinul Trig, care se deplasează cu viteza sunetului prin aer și care va fi recepționat, după reflexia lui de un obstacol, la pinul Echo. Pinul Echo va arăta timpul (durata), în microsecunde, pe care l-au făcut undele sonore.

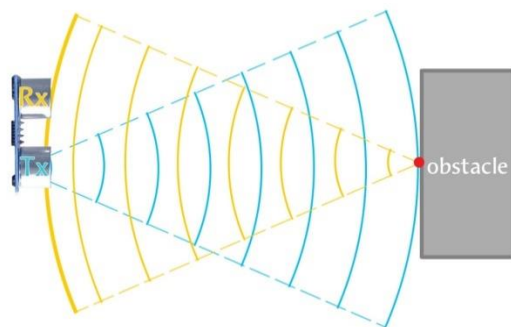


Figura 4.4.2.5

Distanța până la obstacol se determină pe baza intervalului de timp scurs de la momentul în care se transmite unda sonoră și până în momentul în care se recepționează eco-ul. Formula de calcul a distanței este:

Distanța = (durata / 2) \* 0.0343, unde 0.0343 reprezintă viteza sunetului în aer uscat, la ~ 14 °C , valoare obținută cu formula:

$$340 \frac{m}{s} * 100 \frac{cm}{m} * \frac{1 s}{10^6 \mu s} = 0,034 cm/\mu s$$

Activarea Buzzer-ului:

Definirea pinului conectat la microcontroler și a variabilelor necesare:

```
#define BUZZER_PIN 10
#define ALARM_THRESHOLD 5
unsigned char ALARM_FWD_ON = 0;
unsigned char ALARM_BKWD_ON = 0;
unsigned char FIRST_TIME_ALARM_FWD_TRIGGERED = 0;
unsigned char FIRST_TIME_ALARM_BKWD_TRIGGERED = 0;
unsigned char alarm_counter = 0;
```

Figura 4.4.2.6

Setări inițiale :

```
pinMode(BUZZER_PIN, OUTPUT);  
digitalWrite(BUZZER_PIN, LOW);
```

Figura 4.4.2.7

Atunci când flag-ul pentru activarea buzzer-ului este setat pe 1, se execută codul din figura 4.4.2.8, poziționat în bucla infinită loop().

```
if(ALARM_FWD_ON || ALARM_BKWD_ON)  
{  
    alarm_counter++;  
    if(alarm_counter >= ALARM_THRESHOLD)  
    {  
        alarm_counter = 0;  
        digitalWrite(BUZZER_PIN, HIGH);  
    }  
}  
delay(23);  
digitalWrite(BUZZER_PIN, LOW);
```

Figura 4.4.2.8

## 4.5 Line follower

### 4.5.1 Implementare hardware

S-a folosit un modul de urmărire linie cu 3 canale IR (InfraRed). Pentru ca mașina să intre în modul Line Follower s-a folosit un switch cu două poziții care atunci când este în poziția ON va bloca transmiterea datelor prin Bluetooth.



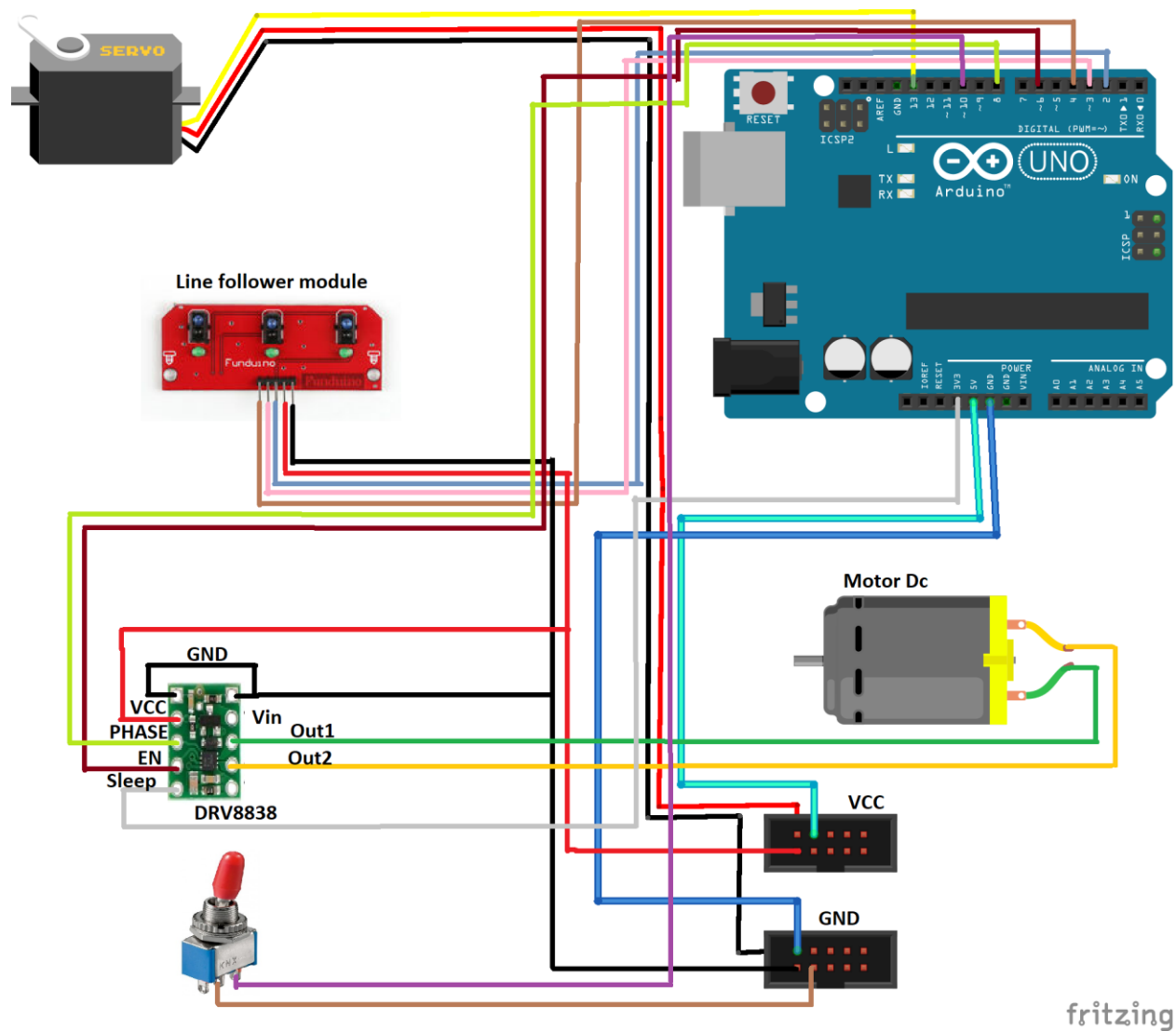


Figura 4.5.1.1

## 4.5.2 Implementare software

Configurare pini:

```
#define IR_PIN_LEFT 2
#define IR_PIN_CENTER 3
#define IR_PIN_RIGHT 4
```

Figura 4.5.2.1

Setări inițiale:

```

void setup()
{
    pinMode(IR_PIN_LEFT, INPUT);
    pinMode(IR_PIN_CENTER, INPUT);
    pinMode(IR_PIN_RIGHT, INPUT);
}

```

Figura 4.5.2.2

Inițializare variabile necesare:

```

unsigned char IR_LEFT, IR_CENTER, IR_RIGHT, IR_TOTAL, LF_STARTED = 0;
unsigned char LF_LAST_STEER = center, full_white_or_black_stop_counter = 0;

#define FULL_BLACK_STOP          0b000
#define STEER_RIGHT_SOFT        0b001
#define NOT_PLAUSIBLE_STOP      0b010
#define STEER_RIGHT_HARD        0b011
#define STEER_LEFT_SOFT         0b100
#define NO_STEERING              0b101
#define STEER_LEFT_HARD         0b110
#define FULL_WHITE_STOP         0b111

```

Figura 4.5.2.3

Pentru citirea senzorilor și controlul mașinii pe traseul de culoare neagră s-a dezvoltat funcția **void lineFollower()** în care s-au folosit unele funcții deja existente prezentate în capitolele anterioare:

- car\_stop();
- steering.write();

```

void lineFollower()
{
    if(!LF_STARTED)
    {
        LF_STARTED = 1;
        car_forward(140);
    }
    IR_LEFT = digitalRead(IR_PIN_LEFT);
    IR_CENTER = digitalRead(IR_PIN_CENTER);
    IR_RIGHT = digitalRead(IR_PIN_RIGHT);

    IR_TOTAL = 0;
    IR_TOTAL = IR_LEFT;
    IR_TOTAL <<= 1;
    IR_TOTAL |= IR_CENTER;
    IR_TOTAL <<= 1;
    IR_TOTAL |= IR_RIGHT;

    switch(IR_TOTAL)
    {
        case FULL_BLACK_STOP:

        {
            full_white_or_black_stop_counter ++;
            if(full_white_or_black_stop_counter > 100)
            {
                car_stop();
            }
            else
            {
                if(LF_LAST_STEER == left_hard)
                {
                    |
                }
                else if(LF_LAST_STEER == right_hard)
                {
                    |
                }
            }

            else
            {
                car_stop();
            }
        }
        break;
    }
    case STEER_RIGHT_SOFT:
    {

```

```

        steering.write(right_soft);
        full_white_or_black_stop_counter = 0;
        LF_LAST_STEER = right_soft;
        break;
    }
    case NOT_PLAUSIBLE_STOP:
    {
        car_stop();
        break;
    }
    case STEER_RIGHT_HARD:
    {
        steering.write(right_hard);
        full_white_or_black_stop_counter = 0;
        LF_LAST_STEER = right_hard;
        break;
    }
    case STEER_LEFT_SOFT:
    {
        steering.write(left_soft);

        full_white_or_black_stop_counter = 0;
        LF_LAST_STEER = left_soft;
        break;
    }
    case NO_STEERING:
    {
        steering.write(center);

        full_white_or_black_stop_counter = 0;
        LF_LAST_STEER = center;
        break;
    }
    case STEER_LEFT_HARD:
    {
        steering.write(left_hard);
        full_white_or_black_stop_counter = 0;
        LF_LAST_STEER = left_hard;
        break;
    }
    case FULL_WHITE_STOP:
    {
        full_white_or_black_stop_counter ++;
        if(full_white_or_black_stop_counter > 100)
        {
            car_stop();
        }
    }

```

---

```

else
{
    if(LF_LAST_STEER == left_hard)
    {

    }
    else if(LF_LAST_STEER == right_hard)
    {

    }
    else
    {
        car_stop();
    }
}
break;
}
default:
{
    car_stop();
    break;
}
}

```

Figura 4.5.2.4

Descriere algoritm:

Se citesc valorile digitale de la cei trei senzori și se stochează în variabilele IR\_LEFT , IR\_CENTER și IR\_RIGHT.

Valorile citite pot fi:

- 0 pentru culoarea neagră
- 1 pentru culoarea albă

Acestea se stochează la rândul lor într-o variabilă pe 8 biți (IR\_TOTAL) prin deplasare la stânga cu o poziție.

Cu ajutorul unei instrucțiuni switch case se verifică toate situațiile posibile în care se poate regăsi variabila IR\_TOTAL în urma citirii modulului IR.

Există 8 situații în care mașina va reacționa diferit.

În fiecare caz se va salva ultima acțiune, astfel încât dacă valorile citite sunt IR\_TOTAL = 000 , IR\_TOTAL = 111 sau IR\_TOTAL = 010, să se poată executa pentru un scurt timp această ultimă comandă evitându-se în acest mod o oprire eronată.

Caz 1: IR\_TOTAL = 000

Un contor va începe să numere până la 100, timp în care se va repeta ultima comandă după care motoarele se vor opri.

Caz 2: IR\_TOTAL = 001

Mașina va vira încet la dreapta cu o valoare prestabilită(Figura 4.5.2.5).

```
#define center 80
#define left_soft 60
#define left_hard 50
#define right_soft 100
#define right_hard 130
#define speed_1f
```

Figura 4.5.2.5

Caz 3: IR\_TOTAL = 010

Se va proceda ca în cazul 1.

Caz 4: IR\_TOTAL = 011

Mașina va vira la dreapta la valoarea maximă.

Caz 5: IR\_TOTAL = 100

Mașina va vira încet la stânga cu un unghi prestabilit de 60 de grade.

Caz 6: IR\_TOTAL = 101

Nu se va executa nici o modificare a direcției.

Caz 7: IR\_TOTAL = 110

Se va executa o virare maximă la stânga.

Caz 8:  $IR\_TOTAL = 111$

Mașina se va comporta ca în cazul 1.

## 5 Concluzie

Pe parcursul acestei lucrări de licență am prezentat etapele necesare construirii unui sistem ce poate fi fie controlat de la distanță, fie se poate deplasa ghidându-se după informațiile primite de la senzori (în modul LineFollower).

Lucrarea este structurată în 5 capitole. Primul capitol descrie succint ideea care a stat la baza dezvoltării acestui proiect. Atenția industriei automotivă se îndreaptă spre realizarea unei mașini autonome care să conducă în condiții de siguranță și fără intervenția omului. De aici și utilitatea aprofundării cunoștințelor dobândite pe parcursul dezvoltării acestui proiect.

Cel de-al doilea capitol descrie toate componentele hardware de care a fost nevoie pentru implementarea sistemului autonom. S-au folosit o varietate de dispozitive mecanice și electronice precum: șasiu, roți, motor de DC, servomotor, microcontroler, modul Bluetooth, senzori de distanță, senzori de lumină etc. După montarea mecanică a dispozitivelor a fost nevoie de utilizarea unui set larg de utilitare software precum: Arduino IDE, MIT Inventor, Fritzing. Acestea sunt descrise pe scurt tot în cadrul capitolului secund.

Al treilea capitol conține noțiunile teoretice ce trebuie asimilate înainte de implementarea propriu-zisă a proiectului. Sunt detaliate aspecte legate de: controlul motoarelor de curent continuu, controlul servomotoarelor, comunicație Bluetooth, monitorizare alimentare, utilizarea senzorilor ultrasonici precum și utilizarea senzorilor de lumină.

Capitolul 4 descrie în detaliu funcționalitățile proiectului precum și aspectele legate de implementarea hardware și software. Astfel pe lângă schemele electrice de conectare sunt prezentate bucăți de cod care puse împreună au dus la realizarea unui produs funcțional.

Pe viitor îmi propun îmbunătățirea acestui sistem prin: adăugarea mai multor senzori care să detecteze eventualele obstacole ce apar pe părțile laterale ale mașinii, deplasarea mașinii cu evitarea obstacolelor precum și analizarea posibilității de adăugare a unui sistem de camere care să completeze informațiile venite de la senzorii ultrasonici.



# Bibliografie

<https://www.scribd.com/doc/93204860/SERVOMOTOARE-ELECTRICE>

<http://www.ac.tuiasi.ro/~lmastacan/wp-content/uploads/C6.4-SMCC-Tipuri-constructive.pdf>

<http://users.utcluj.ro/~rdanescu/pmp-lab09.pdf>

<http://anycomponents.fnhost.org/product/senzor-reflectiv-ir-tcrt5000/>

<http://invataelectronica.blogspot.com/2011/08/usart.html>

<http://www.rasfoiesc.com/educatie/informatica/Interfata-seriala-UART-Univers79.php>

[http://www.mobilindustrial.ro/current\\_version/online\\_docs/COMPENDIU/motoare\\_de\\_curent\\_continuu.htm](http://www.mobilindustrial.ro/current_version/online_docs/COMPENDIU/motoare_de_curent_continuu.htm)

[http://cs.curs.pub.ro/wiki/pm/lab/lab3?fbclid=IwAR3DPTQFucINcXgf3TKzAl\\_5YGJVwA79sSYpR4IvdelsNULCMkLrllsNE9s](http://cs.curs.pub.ro/wiki/pm/lab/lab3?fbclid=IwAR3DPTQFucINcXgf3TKzAl_5YGJVwA79sSYpR4IvdelsNULCMkLrllsNE9s)

<https://www.scribd.com/document/232602912/Arduino-Uno>

<https://ro.wikipedia.org/wiki/Atmega328>

<http://roboromania.ro/2016/11/15/descrierea-pinilor-la-placa-arduino-uno-r3/>

<http://remotexy.com/en/help/bluetooth/>

<https://www.pololu.com/product/2990>

<https://www.optimusdigital.ro/ro/electronica-de-putere-driver-de-motoare/1084-driver-pentru-motor-dc-cu-perii-dr8838.html>

<https://learn.sparkfun.com/tutorials/analog-to-digital-conversion/all>

<https://dronebotworkshop.com/hc-sr04-ultrasonic-distance-sensor-arduino/>

<https://ardushop.ro/ro/home/343-senzor-urmarire-linie.html>

<https://contactelectric.ro/surse-miniatura/992-modul-converter-dc-dc-coborator-de-tensiune-step-down-xl4005.html>

<https://despretot.info/accelerometru-telefon-definitie-accelerometru-smartphone/>

<http://www.creeaza.com/tehnologie/electronica-electricitate/Tranzistorului-ca-si-intrerupa117.php>

<https://contactelectric.ro/placi-dezvoltare-si-module/978-modul-generator-semnal-dreptunghiular-cu-ne555-frecventa-si-factor-de-umplere-reglabil.html>

[https://www.electronics-tutorials.ws/waveforms/555\\_oscillator.html](https://www.electronics-tutorials.ws/waveforms/555_oscillator.html)

<https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide/all>

# Anexă

## Codul scris în Arduino IDE

```
#include <Servo.h>

#include <NewPing.h>

Servo steering;
Servo fancy_buzzer;

#define IR_PIN_LEFT 2
#define IR_PIN_CENTER 3
#define IR_PIN_RIGHT 4

unsigned char IR_LEFT, IR_CENTER, IR_RIGHT, IR_TOTAL, LF_STARTED = 0;
#define LFEP A5

// 0: BLACK, 1: WHITE
// L C R

#define FULL_BLACK_STOP      0b000
#define STEER_RIGHT_SOFT    0b001
#define NOT_PLAUSIBLE_STOP  0b010
#define STEER_RIGHT_HARD    0b011
#define STEER_LEFT_SOFT     0b100
#define NO_STEERING         0b101
#define STEER_LEFT_HARD     0b110
#define FULL_WHITE_STOP     0b111

#define center 80
#define left_soft 60
#define left_hard 52
#define right_soft 100
#define right_hard 130
#define speed_lf

unsigned char LF_LAST_STEER = center, full_white_or_black_stop_counter = 0;
#define NO_CMD_RECEIVED_STOP_THRESHOLD 4 // stop car after # loops with no cmd
from phone
#define trigPin1 12
```

```

#define echoPin1 11
#define trigPin2 7
#define echoPin2 5
float duration1, distance1, duration2, distance2;
#define SERIAL_BAUDRATE 9600 // bps
int Battery_1 = A0;
int Battery_2 = A1;
int Battery_3 = A2;
int digitalVal_b1 = 0; // variable to store the value
int digitalVal_b2 = 0; // variable to store the value
int digitalVal_b3 = 0; // variable to store the value
float Battery1_voltage = 0.00;
float Battery2_voltage = 0.00;
float Battery3_voltage = 0.00;
int procent1 = 0;
int procent2 = 0;
int procent3 = 0;
char CAR_STOPPED = 0;
char no_cmd_received_counter = 0;
// serial commands
#define CMD_STOP 10
#define CMD_DRIVE 11

// steering
#define STEERING_PIN 13

// valori de steering reale, specifice servo motorului
#define STEERING_MIN_POSITION 50 // turning maximum left
#define STEERING_MAX_POSITION 130 // turning maximum right
#define STEERING_CENTER 80
// valori de steering ideale primite de la telefon

```

```

#define STEERING_MIN_POSITION_CMD 0 // turning maximum left
#define STEERING_MAX_POSITION_CMD 100 // turning maximum right
#define STEERING_CENTER_CMD 50

#define BUZZER_PIN 10

// traction
#define TRACTION_PHASE_PIN 8
#define TRACTION_ENABLE_PIN 6
#define DIRECTION_FORWARD LOW
#define DIRECTION_BACKWARD HIGH
unsigned char cmd = 0 ;
unsigned char data_0 = 0;
unsigned char data_1 = 0;

unsigned char ALARM_FWD_ON = 0;
unsigned char ALARM_BKWD_ON = 0;
unsigned char FIRST_TIME_ALARM_FWD_TRIGGERED = 0;
unsigned char FIRST_TIME_ALARM_BKWD_TRIGGERED = 0;
unsigned char alarm_counter = 0;
#define ALARM_THRESHOLD 5
unsigned char battery_counter = 0;
unsigned char DRIVE_FWD_ALLOWED = 1;
unsigned char DRIVE_BKWD_ALLOWED = 1;

void setup()
{
    pinMode(TRACTION_PHASE_PIN, OUTPUT);
    pinMode(TRACTION_ENABLE_PIN, OUTPUT);
    pinMode(STEERING_PIN, OUTPUT);
    digitalWrite(TRACTION_PHASE_PIN, LOW);

```

```

digitalWrite(TRACTION_ENABLE_PIN, LOW);
digitalWrite(STEERING_PIN, LOW);
steering.attach(STEERING_PIN);
center_steering();

init_serial_interface();
clear_serial_buffer();
pinMode(IR_PIN_LEFT, INPUT);
pinMode(IR_PIN_CENTER, INPUT);
pinMode(IR_PIN_RIGHT, INPUT);

pinMode(LFEP, INPUT_PULLUP);

pinMode(trigPin1, OUTPUT);
pinMode(echoPin1, INPUT);
pinMode(trigPin2, OUTPUT);
pinMode(echoPin2, INPUT);
pinMode(BUZZER_PIN, OUTPUT);
digitalWrite(BUZZER_PIN, LOW);

}
void loop()
{
  if(digitalRead(LFEP))
  {
    Obstacle_Detect_Front();
    Obstacle_Detect_Back();
    listen_serial();
    CAR_STOPPED = 0;
    LF_STARTED = 0;
    battery_counter ++;

```

```

    if(battery_counter >= 40)
    {
        Battery();
        battery_counter = 0;
        sendData();
    }
}
else
{
    if(CAR_STOPPED == 0)
    {
        lineFollower();
    }
}
if(ALARM_FWD_ON || ALARM_BKWD_ON)
{
    alarm_counter ++;
    if(alarm_counter >= ALARM_THRESHOLD)
    {
        alarm_counter = 0;
        digitalWrite(BUZZER_PIN, HIGH);
    }
}
delay(23);
digitalWrite(BUZZER_PIN, LOW);
}
void Obstacle_Detect_Front()
{
    // Write a pulse to the HC-SR04 Trigger Pin

    digitalWrite(trigPin1, LOW);

```



```

delayMicroseconds(2);
digitalWrite(trigPin1, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin1, LOW);

// Measure the response from the HC-SR04 Echo Pin

duration1 = pulseIn(echoPin1, HIGH);
//duration2 = pulseIn(echoPin2, HIGH);

// Determine distance from duration
// Use 343 metres per second as speed of sound

distance1 = (duration1 / 2) * 0.0343;
if(distance1 <= 25 && distance1 > 15)
{
    ALARM_FWD_ON = 1;
    FIRST_TIME_ALARM_FWD_TRIGGERED = 0;
}
else if (distance1 <= 15)
{
    if(!FIRST_TIME_ALARM_FWD_TRIGGERED)
    {
        FIRST_TIME_ALARM_FWD_TRIGGERED = 1;
        car_stop();
    }
    ALARM_FWD_ON = 0;
    DRIVE_FWD_ALLOWED = 0;
}
else
{

```

```

    ALARM_FWD_ON = 0;
    DRIVE_FWD_ALLOWED = 1;
    FIRST_TIME_ALARM_FWD_TRIGGERED = 0;
}
}

void Obstacle_Detect_Back()
{
    // Write a pulse to the HC-SR04 Trigger Pin

    digitalWrite(trigPin2, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin2, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin2, LOW);

    // Measure the response from the HC-SR04 Echo Pin

    duration2 = pulseIn(echoPin2, HIGH);

    // Determine distance from duration
    // Use 343 metres per second as speed of sound

    distance2 = (duration2 / 2) * 0.0343;
    if(distance2 <= 25 && distance2 > 15)
    {
        FIRST_TIME_ALARM_BKWD_TRIGGERED = 0;
        ALARM_BKWD_ON = 1;
    }
    else if (distance2 <= 15)
    {

```

```

if(!FIRST_TIME_ALARM_BKWD_TRIGGERED)
{
    FIRST_TIME_ALARM_BKWD_TRIGGERED = 1;
    car_stop();
}
ALARM_BKWD_ON = 0;
DRIVE_BKWD_ALLOWED = 0;
}
else
{
    ALARM_BKWD_ON = 0;
    DRIVE_BKWD_ALLOWED = 1;
    FIRST_TIME_ALARM_BKWD_TRIGGERED = 0;
}
}

void Battery()
{
    digitalVal_b1 = analogRead(A0);// read the value from the analog channel
    //convert digital value to analog voltage
    Battery1_voltage = (digitalVal_b1 * 5.00)/1023.00;
    procent1 = (Battery1_voltage /4.2)*100;
    //battery 2
    digitalVal_b2 = analogRead(Battery_2);// read the value from the analog channel
    //convert digital value to analog voltage
    Battery2_voltage = (digitalVal_b2 * 5.00)/1023.00;
    procent2 = (Battery2_voltage /4.2)*100;
    //battery 3
    digitalVal_b3 = analogRead(Battery_3);// read the value from the analog channel
    //convert digital value to analog voltage
    Battery3_voltage = (digitalVal_b3 * 5.00)/1023.00;

```

```

    procent3 = (Battery3_voltage /4.2)*100;

}

void sendData()
{

    Serial.write(procent1);
    Serial.write(procent2);
    Serial.write(procent3);

}

void lineFollower()
{
    if(!LF_STARTED)
    {
        LF_STARTED = 1;
        car_forward(140);
    }
    IR_LEFT = digitalRead(IR_PIN_LEFT);
    IR_CENTER = digitalRead(IR_PIN_CENTER);
    IR_RIGHT = digitalRead(IR_PIN_RIGHT);
    IR_TOTAL = 0;
    IR_TOTAL = IR_LEFT;
    IR_TOTAL <<= 1;
    IR_TOTAL |= IR_CENTER;
    IR_TOTAL <<= 1;
    IR_TOTAL |= IR_RIGHT;
    switch(IR_TOTAL)
    {
        case FULL_BLACK_STOP:

```

```

{
    full_white_or_black_stop_counter++;
    if(full_white_or_black_stop_counter > 100)
    {
        car_stop();
        LF_STARTED = 0;
    }
    else
    {
        if(LF_LAST_STEER == left_hard)
        {

        }

        else if(LF_LAST_STEER == right_hard)
        {

        }

        else
        {
            car_stop();
            LF_STARTED = 0;
        }
    }
    break;
}

case STEER_RIGHT_SOFT:
{
    steering.write(right_soft);
    full_white_or_black_stop_counter = 0;
    LF_LAST_STEER = right_soft;

```

```

    break;
}
case NOT_PLAUSIBLE_STOP:
{
    car_stop();
    LF_STARTED = 0;
    break;
}
case STEER_RIGHT_HARD:
{
    steering.write(right_hard);
    full_white_or_black_stop_counter = 0;
    LF_LAST_STEER = right_hard;
    break;
}
case STEER_LEFT_SOFT:
{
    steering.write(left_soft);

    full_white_or_black_stop_counter = 0;
    LF_LAST_STEER = left_soft;
    break;
}
case NO_STEERING:
{
    steering.write(center);

    full_white_or_black_stop_counter = 0;
    LF_LAST_STEER = center;
    break;
}

```

```

case STEER_LEFT_HARD:
{
    steering.write(left_hard);
    full_white_or_black_stop_counter = 0;
    LF_LAST_STEER = left_hard;
    break;
}
case FULL_WHITE_STOP:
{
    full_white_or_black_stop_counter ++;
    if(full_white_or_black_stop_counter > 100)
    {
        car_stop();
        LF_STARTED = 0;
    }
    else
    {
        if(LF_LAST_STEER == left_hard)
        {

        }

        else if(LF_LAST_STEER == right_hard)
        {

        }
    }
    else
    {
        car_stop();
        LF_STARTED = 0;
    }
}
}

```

```

        break;
    }
    default:
    {
        car_stop();
        LF_STARTED = 0;
        break;
    }
}

}

void init_serial_interface(void)
{
    Serial.begin(SERIAL_BAUDRATE); // opens serial port, sets baudrate to 9600 bps
}

void clear_serial_buffer(void)
{
    while(Serial.available() > 0)
    {
        Serial.read();
    }
}

void listen_serial(void)
{
    if(Serial.available() == 0)
    {
        no_cmd_received_counter++;
        if(no_cmd_received_counter > NO_CMD_RECEIVED_STOP_THRESHOLD)
        {
            no_cmd_received_counter = 0;
            car_stop();

```



```

    }
}

if(Serial.available() != 3)
{
    clear_serial_buffer();
    no_cmd_received_counter = 0;
    return;
}
else
{
    no_cmd_received_counter = 0;
    cmd = Serial.read();
    data_0 = Serial.read();
    data_1 = Serial.read();
    execute_command(cmd, data_0, data_1);
    return;
}
}

void execute_command(unsigned char cmd, unsigned char data_0, unsigned char data_1)
{
    switch(cmd)
    {
        case CMD_STOP:
        {
            car_stop();
            break;
        }
        case CMD_DRIVE:
        {
            car_drive_cmd(data_0, data_1);

```

```

        break;
    }
}
}

void car_drive_cmd(unsigned char traction_speed, unsigned char steering_position)
{
    if(traction_speed == 0)
    {
        car_stop();
        return;
    }
    if(steering_position == STEERING_CENTER_CMD)
    {
        center_steering();
    }
    else
    {
        unsigned char steering_real_position = steering_cmd_to_real(steering_position);
        car_steer(steering_real_position);
    }

    unsigned char traction_direction = traction_speed & 0x01;
    if(traction_direction == DIRECTION_FORWARD)
    {
        if(DRIVE_FWD_ALLOWED)
        {
            car_forward(traction_speed);
        }
    }
    else
    {
        car_stop();
    }
}

```

```

    }
}
else
{
    if(DRIVE_BKWD_ALLOWED)
    {
        car_backward(traction_speed);
    }
    else
    {
        car_stop();
    }
}
}

void car_stop(void)
{
    digitalWrite(TRACTION_PHASE_PIN, LOW);
    analogWrite(TRACTION_ENABLE_PIN, 0);
    CAR_STOPPED = 1;
}

void car_forward(unsigned char speed)
{
    digitalWrite(TRACTION_PHASE_PIN, DIRECTION_FORWARD);
    analogWrite(TRACTION_ENABLE_PIN, speed);
}

void car_backward(unsigned char speed)
{
    digitalWrite(TRACTION_PHASE_PIN, DIRECTION_BACKWARD);
    analogWrite(TRACTION_ENABLE_PIN, speed);
}

void center_steering(void)

```

```

{
    steering.write(STEERING_CENTER);
}

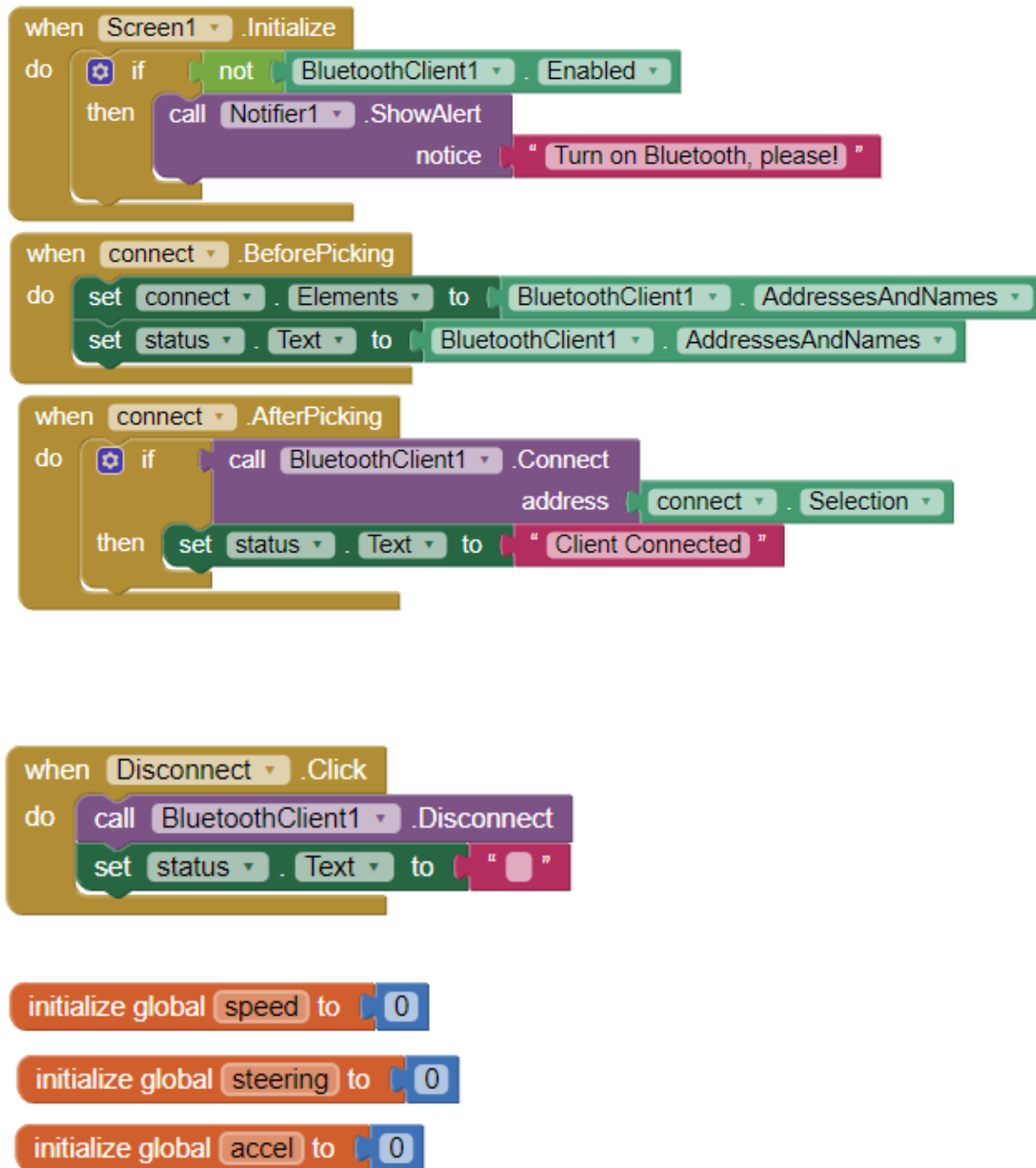
void car_steer(unsigned char position)
{
    if(position < STEERING_MIN_POSITION)
    {
        steering.write(STEERING_MIN_POSITION);
        return;
    }
    else if (position > STEERING_MAX_POSITION)
    {
        steering.write(STEERING_MAX_POSITION);
        return;
    }
    else
    {
        steering.write(position);
        return;
    }
}

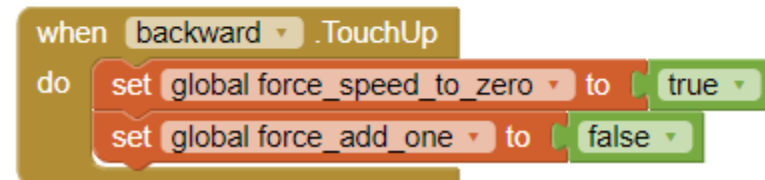
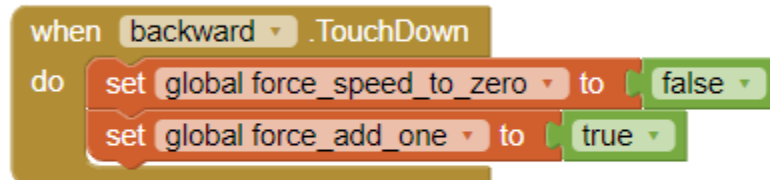
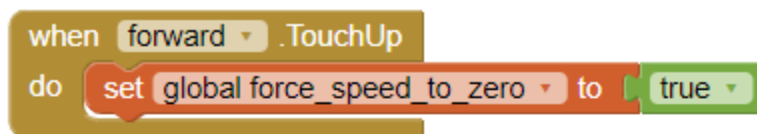
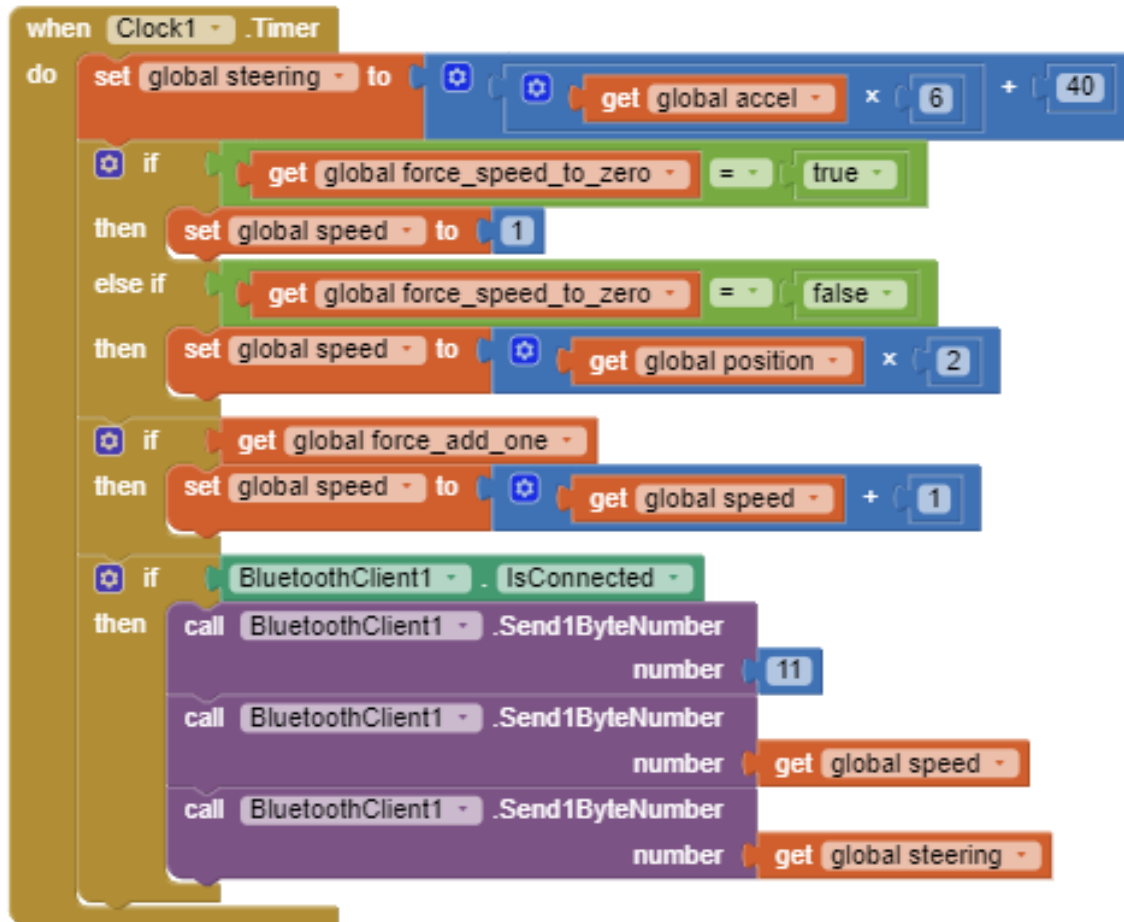
int steering_cmd_to_real(unsigned char steering_cmd)
{
    // [A, B] --> [a, b]
    // se transforma intervalul [0, 100] primit de la telefon in intervalul [50, 130], specific pentru serv
    int value = steering_cmd;
    int A, B, a, b;
    int real_value;
    if(value <= STEERING_CENTER_CMD)
    {
        A = STEERING_MIN_POSITION_CMD;

```

```
B = STEERING_CENTER_CMD;
a = STEERING_MIN_POSITION;
b = STEERING_CENTER;
}
else
{
    A = STEERING_CENTER_CMD;
    B = STEERING_MAX_POSITION_CMD;
    a = STEERING_CENTER;
    b = STEERING_MAX_POSITION;
}
real_value = (value - A) * (b - a) / (B - A) + a;
return real_value;
}
```

## Codul scris în MIT App inventor





```

when backward .TouchUp
do
  set global force_speed_to_zero to true
  set global force_add_one to false

when AccelerometerSensor1 .AccelerationChanged
  xAccel yAccel zAccel
do
  set global accel to round AccelerometerSensor1 . YAccel

initialize global force_speed_to_zero to true
initialize global force_add_one to false
initialize global position to 0

when Slider1 .PositionChanged
  thumbPosition
do
  set global position to round get thumbPosition

initialize global bytesAvailableToReceive to 0
initialize global bat_cnt to 0

```



