UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ŞI ELECTRONICĂ

DEPARTAMENTUL DE AUTOMATICĂ ŞI ELECTRONICĂ

PROIECT DE DIPLOMĂ

Voinea Andrei Sorin

COORDONATOR ŞTIINŢIFIC

Sef lucr. Dr. Ing. Florin Stîngă

Septembrie 2017

CRAIOVA

UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ŞI ELECTRONICĂ

DEPARTAMENTUL DE AUTOMATICĂ ŞI ELECTRONICĂ

ROBOT MOBIL ECHIPAT CU LiDAR

VOINEA ANDREI SORIN

COORDONATOR ŞTIINŢIFIC

Sef lucr. Dr. Ing. Florin Stîngă

Septembrie 2017

CRAIOVA

„In a world of talkers, be a thinker and a doer.”

DESTIN SANDLIN

DECLARAȚIE DE ORIGINALITATE

Subsemnatul Voinea Andrei Sorin, student la specializarea Automatică şi Informatică Aplicată din cadrul Facultății de Automatică, Calculatoare şi Electronică a Universității din Craiova, certific prin prezenta că am luat la cunoştinţă de cele prezentate mai jos şi că îmi asum, în acest context, originalitatea proiectului meu de licenţă:

- cu titlul ROBOT MOBIL ECHIPAT CU LiDAR,
- coordonată de Sef lucr. Dr. Ing. Florin Stîngă,
- prezentată în sesiunea SEPTEMBRIE 2017.

La elaborarea proiectului de licenţă, se consideră plagiat una dintre următoarele acţiuni:

- reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimele şi referinţa precisă,
- redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări, dacă nu se indică sursa bibliografică,
- prezentarea unor date experimentale obţinute sau a unor aplicaţii realizate de alţi autori fără menţionarea corectă a acestor surse,
- însuşirea totală sau parţială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Pentru evitarea acestor situaţii neplăcute se recomandă:

- plasarea între ghilimele a citatelor directe şi indicarea referinţei într-o listă corespunzătoare la sfărşitul lucrării,
- indicarea în text a reformulării unei idei, opinii sau teorii şi corespunzător în lista de referinţe a sursei originale de la care s-a făcut preluarea,
- precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele et caetera,
- precizarea referinţelor poate fi omisă dacă se folosesc informaţii sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data, Semnătura candidatului,

UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare şi Electronică

Departamentul de Automatică şi Electronică

Aprobat la data de
……
Şef de departament,
Prof. dr. ing.

Emil PETRE

PROIECTUL DE DIPLOMĂ

Numele şi prenumele studentului/-ei:	Voinea Andrei Sorin
Enunțul temei:	Robot mobil echipat cu LiDAR
Datele de pornire:	Proiectarea și dezvoltarea unui robot mobil echipat cu LiDAR
Conținutul proiectului:	1. Introducere 2. Microcontrollerul ESP8266 3. Senzorul VL53L0X 4. Motoare pas cu pas și Adafruit Motor Shield V2 5. Aplicație Arduino 6. Aplicație Java 7. Concluzii
Material grafic obligatoriu:	
Consultații:	Lunare
Conducătorul științific (titlul, nume și prenume, semnătura):	Sef lucr. Dr. Ing. Florin Stîngă
Data eliberării temei:	
Termenul estimat de predare a proiectului:	
Data predării proiectului de către student și semnătura acestuia:	

UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare şi Electronică

Departamentul de Automatică şi Electronică

REFERATUL CONDUCĂTORULUI ŞTIINŢIFIC

Numele şi prenumele candidatului/-ei: Voinea Andrei Sorin
Specializarea: *Automatică şi Informatică Aplicată*
Titlul proiectului: *Robot mobil echipat cu LiDAR*

Locaţia în care s-a realizat practica de documentare (se bifează una sau mai multe din opţiunile din dreapta):

- În facultate ☐
- În producţie ☐
- În cercetare ☐
- Altă locaţie: [*se detaliază*]

În urma analizei lucrării candidatului au fost constatate următoarele:

Nivelul documentării		Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine ☐
Tipul proiectului		Cercetare ☐	Proiectare ☐	Realizare practică ☐	Altul [*se detaliază*]
Aparatul matematic utilizat		Simplu ☐	Mediu ☐	Complex ☐	Absent ☐
Utilitate		Contract de cercetare ☐	Cercetare internă ☐	Utilare ☐	Altul [*se detaliază*]
Redactarea lucrării		Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine ☐
Partea grafică, desene		Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună ☐
Realizarea practică	Contribuţia autorului	Insuficientă ☐	Satisfăcătoare ☐	Mare ☐	Foarte mare ☐
	Complexitatea temei	Simplă ☐	Medie ☐	Mare ☐	Complexă ☐
	Analiza cerinţelor	Insuficient ☐	Satisfăcător ☐	Bine ☐	Foarte bine ☐
	Arhitectura	Simplă ☐	Medie ☐	Mare ☐	Complexă ☐
	Întocmirea specificaţiilor funcţionale	Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună ☐
	Implementarea	Insuficientă	Satisfăcătoar	Bună	Foarte bună

		☐	e ☐	☐	☐
	Testarea	Insuficientă ☐	Satisfăcătoare ☐	Bună ☐	Foarte bună ☐
	Funcționarea	Da ☐	Parţială ☐	Nu ☐	
Rezultate experimentale		Experiment propriu ☐		Preluare din bibliografie ☐	
Bibliografie		Cărţi	Reviste	Articole	Referințe web
Comentarii şi observaţii					

În concluzie, se propune:

ADMITEREA PROIECTULUI	RESPINGEREA PROIECTULUI
☐	☐

Data, Semnătura conducătorului ştiinţific,

CUPRINSUL

LISTA FIGURILOR

1 INTRODUCERE

1.1 Scopul

Această lucrare de licentă prezintă procesul de dezvoltare al unui robot mobil echipat cu tehnologie LiDAR. Ca obiect principal de cercetare s-a studiat funcționarea senzorului VL53L0X produs de ST Microelectronics cât și a microcontrollerului ESP8266 produs de Espressif Systems. Componentele software reprezintă poate cea mai importantă parte a proiectului și asta a necesitat un studiu amănunțit al librăriilor și al mediilor de dezvoltare folosite.

1.2 Motivaţia

Pasiunea pentru sistemele integrate și programarea atât la nivel de microprocesor cât și la nivel înalt au făcut ca proiectarea unui robot să fie o alegere evidentă pentru tema proiectului de licență.

Scopul acestui proiect este de a dezvolta un robot care se poate dovedi util în situații de urgență, oferind informații vitale din spații greu accesibile. Robotul oferă totodată posibilitatea măsurării perimetrelor neregulate care nu pot fi măsurate prin metode convenționale.

2 MICROCONTROLLERUL ESP8266

2.1 Informații generale

Microcontrollerul ESP8266 este un SOC(System on a chip) cu stivă TCP/IP integrată care îi permite accesul la rețele WiFi. Unitatea centrală de procesare de tip 16-bit RISC (Reduced Instruction Set Computer) pe care o conține, deși implicit rulează la 80 Mhz, poate fi setată să funcționeze la 160 Mhz, făcand ESP8266 un microcontroller foarte puternic în aplicațiile de tip Internet of Things. Chipul oferă 64 KiB memorie de program și 96 KiB memorie de date, mai mult decât suficient pentru dezvoltarea de aplicații embedded complexe. ESP8266 dispune de 16 pini de intrări și ieșiri cât și periferice precum SPI, I²C, UART și ADC cu rezoluție de 10 biți. Arhitectura 16-bit RISC și puterea mare de procesare îi permit microcontrollerului să folosească chiar sisteme de operare primitive precum ESP-Open-RTOS permițând tehnici de programare bazate pe evenimente și funcții de apel invers spre deosebire de tehnicile clasice, care folosesc bucle și întreruperi.[ESP17]

Inițial ESP8266 putea fi folosit doar ca o interfață WiFi pentru alte microcontrollere, el fiind comandat prin comenzi AT pe portul serial. În octombirie 2014, producătorul Espressif a lansat kit-ul de dezvoltare software care permite programarea microcontrollerului, transformând-ul practic dintr-o interfață WiFi pentru alte procesoare, într-un procesor de sine stătător cu interfață WiFi integrată.

La scurt timp după lansarea SDK-ului, alți dezvoltatori de software au lansat propriile kit-uri open-source pentru programarea microcontrollerului ESP8266, acesta câștigând popularitate foarte rapid datorită funcției WiFi cât și puterea de calcul mare. O metodă foarte populară de programare a microcontrollerului este folosind mediul de dezvoltare Arduino.[WIKIESP17]

2.2 Interfețe intrare/ieşire

Micrcontrollerul ESP8266 dispune de 17 pini de intrare/ieșire. Ei pot fi configurați să indeplinească diferite funcții. Fiecare pin poate fi configurat să declanşeze o întrerupere pe pantă crescătoare sau descrescătoare sau întrerupere de wake-up și dispune de rezistor pull-up intern programabil.

Pinii microcontrollerului sunt multiplexati cu funcții secundare precum I²C, I²S, UART, PWM, IR Remote Control, SPI sau ADC.

Tensiunea maximă de funcționare a interfețelor intrare/ieșire este de 3.3V și pot susține un curent de maxim 12mA.[ESP17]

2.3 Ceas de timp real

Ceasul de timp real al ESP8266 este folosit atât pentru transmisia și recepția de date cât și funcționarea unității centrale de procesare. Semnalul de clock este generat de un oscilator intern sau unul extern. Aceste oscilatoare pot varia între 24 Mhz și 52 Mhz.[ESP17]

Oscilatorul pe bază căruia se generează semnalul de clock este un factor foarte important în ceea ce privește performanță modulului de WiFi al microcontrollerului.

2.4 Radio și WiFi

Modulul de radio și WiFi al ESP8266 este compus din următoarele elemente:

- receptor 2.4 GHz;
- transmitător 2.4 GHz;
- generator de clock și oscilator de mare viteză și precizie;
- ceas de timp real;
- regulatoare;
- gestionare energetică;

Receptorul și transmițătorul de 2.4 GHz suportă 14 canale de frecvențe conform standardelor IEEE802.11b/g/n, ceea ce face ESP8266 compatibil cu orice alt echipament echipat cu WiFi, de la cele mai vechi până la cele de ultimă generație, standardul impunând compatibilitate cu versiunile anterioare.

Receptorul de 2.4 GHz convertește semnalele radio în semnal digital folosind două convertoare analogic-numerice de mare viteză. Având în vedere că semnalul recepționat poate varia în intensitate și zgomot, receptorul are integrat o serie de filtre și amplificatoare.

Transmițătorul de 2.4 GHz se ocupă de trecerea semnalului ce trebuie transmis în domeniul de 2.4 GHz și controlează antena folosind un amplificator de putere de tip CMOS. Transmițătorul se

calibrează dinamic pentru a oferi cea mai bună performanță, ajungând la o putere de +19.5 dBm în transmisie de tip 802.11b și +16 dBm în transmisie de tip 802.11n.

Generatorul de clock produce un semnal de 2.4 GHz pentru receptor și transmițător. Toate componentele generatorului de clock sunt integrate în chip și acesta se poate calibra și testa singur.

ESP8266 are implementată o stivă TCP/IP completă, protocolul 802.11 b/g/n/e/i WLAN MAC complet cât și WiFi Direct. Toate acestea permit microcontrollerului să fie folosit ca un client WiFi clasic, hotspot WiFi dar și conexiuni peer-to-peer. Procedurile de utilizare ale diferitelor tipuri de funcționare sunt relativ simple, microcontrollerul ocupându-se de scanare, conectare, menținerea conexiunii și gestionarea energetică complet autonom odată ce a primit comenzile de inițializare corespunzătoare.

Unul din marile avantaje ale ESP8266 este managementul energetic avansat, creat pentru a-l face un microcontroller potrivit aplicațiilor de tip Internet of Things. El poate opera în 3 moduri diferite: Active, Sleep sau Deep-Sleep. În modul Deep-Sleep microcontrollerul consumă aproximativ 20 µA și între 1.0 mă și 0.6 mă atunci când menține activă conexiunea la un punct de acces WiFi. [ESP17]

2.5 SPI

Serial Peripheral Interface (SPI) este o interfara serială compusă dintr-un semnal de ceas, unul de selecție a dispozitivului cu care se face comunicarea și un semnal de date. Recepția datelor începe pe frontul descrescătorul al semnalului de selecție (CS) iar pentru fiecare bit de date este general un tact pe semnalul de ceas, făcand seriala SPI o interfață sincronă. În timpul comunicației între două dispozitive, magistrala SPI nu este disponibilă pentru comunicația altor echipamente. [ESP17][SPKSPI17]

ESP8266 are 3 seriale SPI:

- o serială SPI ce poate funcționa în mod slave sau master;
- o serială SPI ce poate funcționa doar în mod slave;
- o serială HSPI ce poate funcționa în mod slave sau master;

Frecvența maximă a semnalului de ceas SPI pentru ESP8266 este de 80 Mhz.

2.6 I2C

Inter-integrated Circuit (I2C) este o interfață serială de comunicare care permite transferul de date între mai multe dispozitive slave și unul sau mai multe dispozitive master. Similar interfeței SPI, interfața I2C este proiectată pentru transferul de date pe distanțe scurte, de obicei în cadrul aceluiași dispozitiv între diferite componente ale acestuia. Fiecare magistrală I2C este alcătuită din două semnale, unul de ceas (ȘCL) și unul de date (SDA). Semnalul de ceas este mereu dat de dispozitivul master curent dar dispozitivele slave pot bloca acest semnal, forțând master-ul să ia o pauză pentru ca respectivele dispozitive să poată procesa datele. Avantajul interfeței I2C față de SPI este că aceasta nu necesită semnale de selecție a dispozitivelor deoarece negocierea master-ului curent de pe magistrala se face la nivel de protocol.[SPKI2C17]

ESP8266 dispune de o interfață I2C și aceasta poate funcționa atât în mod master cât și slave. Protocolul de comunicare se face software, deci este la latitudinea programatorului dar trebuie să se țină cont de protocoalele dispozitivelor cu care se va comunica. Frecvența maximă a semnalului de ceas este de 100 khz. Trebuie avut în vedere faptul ca frecvența semnalului de ceas ar trebui să fie mai mare decât cea mai mică frecvență de ceas a dispozitivelor de pe magistrală.[ESP17]

2.7 UART

Universal Asynchronous Receiver/Transmitter (UART) este o interfață de comunicare asincronă care permite configurarea formatului datelor transmise dar și viteza de transmisie. Interfață UART poate fi folosită atât pentru transmisia cât și recepția de date. Cel mai întâlnit și simplu mod de configurare a modulului UART al unui microcontroller este transimia de 8 biți de date, fără paritate, cu un singur bit de stop. Frecvența cu care se transmit biții de date se numește baud și reprezintă numărul de biți pe secundă. Conxiunea dintre dispozitivele de pe magistrală UART poate fi de tip simplex, half-duplex sau full-duplex.[WIKISPI17]

ESP8266 are două interfețe UART cu viteze până la 4.5 Mbps, adică 4608000 baud. Cu firmware-ul predefinit în frabrica, ESP8266 va transmite pe interfața UART date la pornire pentru a confirma funcționarea corectă.[ESP17]

2.8 PWM

Pulse Width Modulation (PWM) este un semnal în impulsuri a căror lățime este variabilă. Semnalul PWM este o metodă digitală de a simula un semnal analogic sau de a genera un semnal continuu dar cu amplitudine variabilă, între nivelele de 0 și 1 logic.

Semnalul PWM este caracaterizat de o perioadă și un factor de umplere. Factorul de umplere reprezintă un procent din perioadă în care semnalul va sta în starea 1 logic. ESP8266 dispune de 4 ieșiri capabile să genereze semnal PWM.[ESP17]

3 SENZORUL VL53L0X

3.1 Informații generale

Senzorul VL53L0X face parte dintr-o nouă generație de senzori Time-of-Flight (ToF) și a fost senzorul încorporat în cea mai mică capsulă la momentul lansării sale. El poate măsura distanțe până la 2 metri cu priecizie de 1 milimetru, ridicând standardele în aria senzorilor ToF.

VL53L0X folosește tehnologia SPAD (Single Photon Avalanche Diodes) iar emițătorul VCSEL (Vertical Cavity Surface-Emitting Laser) 940nm este complet înafara spectrului vizibil, împreună cu filtrele infraroșu încorporate permițând funcționarea la distanțe mai mari decât tehnologiile anterioare și oferă imunitate crescută la lumina ambientală.[VL16]

3.2 Funcţionare

Controlul senzorului VL53L0X se face folosind un API (Application Program Interface). API-ul oferă acces utilizatorului la un set de funcții avansate din firmware-ul senzorului care permit inițializarea și calibrarea acestuia, pornirea și oprirea achiziției de date, selectarea acuratetii, a vitezei de eșantionare sau a distanței maxime.[VL16]

Producătorul oferă un API compus dintr-un set de funcții C care permit controlul senzorului dar există și implementări ale altor dezvoltatori, controlul fiind redus la protocolul de comunicare I2C. Pentru a folosi senzorul VL53L0X putem folosi API-ul furnizat de către producător sau ne putem crea propriul API care să respecte protocolul de comunicare al senzorului.[STVL17]

3.3 Calibrarea

Producătorul recomandă o serie de calibrări ale senzorului în diferite situații pentru a garanta o precizie cât mai mare:

- La pornire trebuie efectuată o inițializare care implică o pauză de 40ms
- Calibrarea SPAD se face o singură dată, senzorul fiind capabil să rețină valorile rezultate. Calibrarea SPAD implică o pauză de 10ms.

- Calibrarea în funcție de temperatură se face în timpul funcționării iar producătorul recomandă o nouă calibrare atunci când diferență de temperatură depășește 8 grade Celsius în timpul funcționării. Calibrarea în funcție de temperatură durează 40ms.

- Compensarea erorii se face o singură data, senzorul fiind capabil să rețină valorile rezultate. Compensarea durează ~300ms.

Pentru a garanta o funcționare cât mai bună într-un mediu dinamic este necesară calibrarea SPAD. De obicei calibrarea făcută de către producător este suficientă pentru majoritatea aplicațiilor.

Calibrarea în funcție de temperatură reprezintă calibrarea a doi parametrii (VHV și phase cal) care sunt dependenți de schimbările de temperatură. Cei doi parametri sunt folosiți pentru a regla sensibilitatea senzorului și iar calibrarea făcută de către producător este suficientă excepție făcând cazurile extreme, unde temperatura are variații mari în timpul funcționării.[VLUSER16]

Compensarea erorii reprezintă un offset aplicat valorii măsurate, ea fiind liniară. Pentru performanțe cât mai mari este recomandată calibrarea făcută de către producător deoarece ca o nouă calibrare să fie corectă sunt necesare echipamente avansate de măsurare a distanței.[STVL17]

3.4 Moduri de operare

API-ul oferă 3 moduri diferite de eșantionare:

- Un singur eșantion (Single ranging). Măsurarea se face imediat după apelarea funcției API urmând ca senzorul să între în stand-by automat după ce valoarea măsurată a fost transmisă.

- Eșantionare continuă (Continous ranging). Măsurarea se face continuu după apelarea funcției API. Când s-a determinat valoarea unei măsurări, se începe o nouă măsurare fără nicio pauză.Utilizatorul este obligat să oprească eșantionarea continuă pentru a permite senzorului să intre în stand-by.

- Eșantionare periodică (Timed ranging). Măsurarea se face continuu după apelarea funcției API. Când s-a determinat valoarea unei măsurări, se începe o nouă măsurare după o pauză stabilită de către utilizator folosind API-ul.

Dacă o comandă de oprire vine de la utilizator în timpul unei măsurări, senzorul va termina măsurarea înainte să se oprească.[STVL17]

Pe lângă modurile de eșantionare, senzorul oferă și selectarea a 4 profile diferite:

- Implicit
- Viteză mare
- Acurateţe mare
- Distanţă mare

3.5 Eşantionarea

Eşantionarea este compusă din pregătirea pentru măsurare şi măsurarea efectivă.

În timpul măsurării, mai multe pulsuri infraroşu sunt emise de VCSEL, ele se reflectă pe suprafaţa obiectului faţă de care se face măsurarea şi sunt apoi detectatea de matricea receptoare. Detectorul folosit de VL53L0X foloseşte tehnologie SPAD ultra-rapidă pentru a oferi o acurateţe cât mai mare.

Timpul necesar unei măsurători este în general de 33ms, măsurătoarea efectivă durând aproximativ 23ms iar perioada de pregătire dinainte măsurării de aproximativ 8ms.

Procesarea digitală reprezintă ultima operaţie internă din cadrul unei măsurători, ea fiind responsabilă de calcularea, validarea şi eventual rejectarea rezultatului măsurătorii. Diferite etape ale procesării digitale sunt executate atât intern de către senzor, cât şi extern de către API.

Etape executate intern de către senzor:

- verificarea semnalului recepţionat
- compensarea erorii
- compensarea în cazul utilizarii unei sticle de protecţie
- calcularea valorii finale

Etape executate de către API:

- verificarea acurateţii

Dacă se doreşte dezvoltarea capabilităţilor senzorului, utilizatorul poate efectua diferite operaţii folosind valorile măsurate precum mediere, hysteresis sau filtrare numerică.

Eşantionarea se poate face prin polling sau prin întrerupere, la latitudinea utilizatorului. În modul polling utilizatorul trebuie să verifice continuu dacă s-au primit valori noi apelând o funcţie din

API. În modul prin întrerupere, senzorul generează un semnal pe pinul GPIO1 în momentul în care o nouă valoare este disponibilă.[STVL17]

3.6 Controlul senzorului

Interfața I2C este compusă din două semnale: linia seriala de date (SDA) și linia de ceas (ȘCL). Fiecare echipament conectat la magistrală are o adresă unică iar pe magistrală există reguli simple care impun care echipament este slave și care este master.

Ambele semnale (SDA și ȘCL) sunt conectate la sursă de tensiune continuă prin doi rezistori. Când pe magistrală nu se transmit date, ambele semnale sunt în starea 1 logic iar echipamentele le pot pune în 0 logic pentru a transmite date.

Semnalul de ceas (ȘCL) este transmis de către echipamentul master și acesta inițiază comunicația. VL53L0X poate funcționa la o viteză de maxim 400 kbits/s și are implicit adresă 0x52.

Informația este împachetată în serii de 8 biți și este mereu urmată de un bit de confirmare. Intern, achiziția de date se face eșantionând semnalul SDA pe pantă crescătoare a semnalului ȘCL. La transmisie, semnalul SDA trebuie să fie stabil pe frontul pozitiv al semnalului ȘCL, excepție făcând stările de început și sfârșit a transimisiei unde semnalul SDA își schimbă valoarea în timp ce semnalul ȘCL este 1 logic.

Fiecare mesaj conține o serie de octeți, precedați de o condiție de start și urmați fie de una de stop sau una repetată de start, urmată de alt mesaj. Primul octet conține adresa echipamentului (0x53) și specifică direcția în care se face transmisia. Dacă cel mai puțin semnificativ bit este 0 logic, transmisia e dinstre master spre slave. Dacă cel mai puțin semnificativ bit este setat, atunci master-ul citește valori de la slave.

Orice comunicare pe magistrală trebuie să înceapă cu o condiție de start. VL53L0X confirmă recepția unui mesaj trecând semnalul SDA în 0 logic. Datele recepționate de către slave sunt scrise bit cu bit în registrul serial/paralel. După recepționarea fiecărui octet, este generată o confirmare și octetul este stocat în regiștrii interni. În timpul unei citiri (de la slave de către master), conținutul unui registru este concatenat la octetul de adresă, rezultatul este trecut în registrul serial/paralel și este transmis bit cu bit pe frontul descrescător al semnalului de ceas. La sfârșitul fiecărei transmisii, fie ea făcută de către slave sau de către master, echipamentul care a primit datele transmite o confirmare. [STVL17]

4 MOTOARE PAS CU PAS ŞI ADAFRUIT MOTOR SHIELD V2

4.1 Informaţii generale despre Adafruit Motor Shield v2

Adafruit Motor Shield V2 este o placă destinată controlului motoarelor de curent continuu, servo-motoarelor şi motoarelor pas cu pas produsă de Adafruit, o companie înfiinţată în 2005 de către o absolventă MIT şi care are un catalog bogat de produse destinate pasionaţilor de electronică şi sisteme incorporate.[WIKIARD17]

4.2 Componente Adafruit Motor Shield v2

Placa foloseşte drivere MOSFET TB6612 pentru controlul motoarelor, ele garantând un curent de 1.2A pe fiecare canal şi sunt capabile să susţină vârfuri de 3A pentru aproximativ 20ms. Pentru controlul driverelor, producătorul a dispus placă cu PCA9685, un driver PWM dedicat care comunică cu microcontrollerul ales prin I2C. Astfel, pentru a controla 4 motoare de curent continuu sau 2 motoare pas cu pas, sunt ocupaţi doar 2 pini ai microcontrollerului, cei pentru I2C (ŞCL şi SDA) iar driverul PWM trebuie să aibă sursă comună cu microcontrollerul.[ADA17]

Adafrumos Motor Shield V2 este proiectată astfel încât mai multe plăci se pot suprapune. Pe placă există 5 pini de selectare a adresei I2C iar astfel este posibilă suprapunerea a 32 de plăci, deci controlul a 64 de motoare pas cu pas sau 128 de motoare de curent continuu, folosind doar doi pini ai microcontrollerului, cei pentru comunicaţia serială I2C.[GITADA17]

Specificatiile placii Adafruit Motor Control V2 sunt urmatoarele:

- 2 conectori pentru servo-motoare de 5V
- 4 punti H ce pot controla motoare de la 4.5V pana la 13.5V. Pot fi controlate 4 motoare de curent continuu cu rezolutia vitezei de 8 biti sau 2 motoare pas cu pas in configuratie single coil, dual coil, interleave sau microstep
- puntile H sunt prevazute cu diode de protectie
- motoarele sunt oprite implicit la alimentarea cu tensiune a placii
- nivele logice configurabile, de 3.3V sau 5V

Producatorul ofera un API opensource pentru controlul placii destinat mediului de dezvoltare Arduino.

4.3 Informații generale despre motoarele pas cu pas

Motorul pas cu pas este un convertor electromagnetic care transformă impulsurile electrice în mișcare discretă a axului sau. MPP-urile sunt motoare speciale foarte des întâlnite în aplicațiile care necesită poziționare exactă în raport cu un sistem de coordonate.[MPP14]

Axul motorului execută o rotație completă în pași discreți atunci când asupra înfășurărilor statorului este aplicată o anumită secvența de impulsuri. Atât direcția cât și viteză de rotație sunt strâns legate de secvența de impulsuri electrice aplicate.

Motoarele pas cu pas sunt motoare fără perii ce împart o revoluție completă într-un număr de pași. Motorul poate fi comandat să execute un pas într-o direcție sau să păstreze poziția la care se află fără niciun senzor de feed-back, controlul fiind în buclă deschisă însă este necesară dimensionarea corectă a motorului în funcție de cuplul necesar.

Spre deosebire de motoarele de curent continuu, motoarele pas cu pas, pe înfășurările motoarelor pas cu pas se aplică un tren de impulsuri.

4.4 Mod de funcţionare

Motoarele pas cu pas sunt alcătuite din mai mulți electromagneți poziționați în jurul unui ax cu roți dințate. Electromagneții sunt alimentați pe rând de către un circuit extern. Pentru a roți axul, un electromagnet este alimentat iar dinţii roţii dinţate de pe ax se aliniază cu acesta dar nu și cu următorul electromagnet. Astfel, la alimentarea următorului electromagnet și oprirea celui dintâi, axul se rotește cu un număr fix de grade pentru a alinia roata dințată cu următorul electromagnet. Acest proces este cunoscut sub numele de pas. Repetând procedura pentru fiecare electromagnet în parte, axul poate fi rotit continuu sau poziționat la un unghi fix.[WIKISTEP17]

4.5 Tipuri de motoare pas cu pas

Există 3 tipuri de motoare pas cu pas:

- motoare pas cu pas cu magneti permanenti
- motoare pas cu pas cu reluctanta variabila
- motoare pas cu pas hibride

4.5.1 Motoarele pas cu pas cu magneţi permanenţi

MPP-urile cu magneți permanenți au cel mai simplu mod de funcționare, el constând în reacție dintre magnetul permanent poziționat pe rotor și câmpul magnetic creat de către electromagnetul poziționat pe stator.

Aceste motoare pot fi împărțite la rândul lor în mai multe categorii:

- motoare pas cu pas unipolare
- motoare pas cu pas bipolare
- motoare pas cu pas multifaza

Motoarele unipolare au în general 6 borne de conectare, 2 dintre ele fiind legate la punctele mediane ale celor 2 bobine. Ele sunt comandate prin alimentarea secvențială dar cu aceeași polaritate a fiecărei din cele 4 secțiuni de bobine rezultante.[WIKISTEP17]

Motoarele bipolare au în general 4 borne de conectare și sunt comandate prin inversarea sensului curentului în una din borne, pe rând.

4.5.2 Motoarele pas cu pas cu reluctanţă variabilă

MPP-urile cu reluctanță variabilă sunt alcătuite dintr-un rotor și un stator, fiecare din ele având un număr diferit de dinți. Motoarele pas cu pas cu reluctanță variabilă pot fi ușor de diferențiat prin faptul că nu prezintă nicio rezistentă când se încearcă rotirea cu mână.

Statorul este compus dintr-un miez magnetic construit din lamele de oțel iar rotorul din fier nemagnetizat, dispus cu dinți și șanțuri.

4.5.3 Motoarele pas cu pas hibride

Motoarele pas cu pas hibride au în componența lor un rotor cu 2 poli separați de un magnet permanent.

MPP-urile hibride sunt cu 2 faze sau 5 faze, cele din urmă fiind folosite în aplicații speciale care necesită o rezoluție mare (pas cât mai mic) sau moment de inerție mic. Numărul mare de faze al motoarelor pas cu pas hibride implică un cost mai mare, deci este necesar să se țină cont de cerințele aplicației în care vor fi folosite.

4.6 Controlul motoarelor pas cu pas

Performanța motoarelor pas cu pas este direct strânsă de circuitul de control al acestora. Cuplul MPP-urilor la turații mari poate fi crescut dacă polaritatea înfăşurărilor statorului poate fi inversată mai rapid, limită fiind inductanță bobinelor. Pentru a combate inductanță și a inversa cât mai rapid polaritatea înfăşurărilor se folosește o tensiune mai mare. Această metodă duce la altă problemă, și anume limitarea curentului produs de tensiunea crescută.

Circuitele de control ale motoarelor pas cu pas sunt de doua feluri:

- circuite cu tensiune constantă
- circuite cu curent constant

4.6.1 Circuitele de tensiune constantă

Circuitele cu tensiune constantă se numesc astfel deoarece o tensiune fixă, cu valoare pozitivă sau negatica este aplicată fiecărei înfăşurări pentru a stabili poziția axului. Este cunoscut faptul că nu tesiunea, ci curentul care trece prin înfăşurare este cel care generează cuplul motorului. Curentul I care trece prin fiecare înfăşurare este strâns legat de tensiunea aplicată V, inductanță bobinei L dar și rezistivitatea R a acesteia. Rezistivitatea bobinei determină curentul maxim care o poate străbate conform legii lui Ohm. Inductanță L determină viteza maximă cu care se poate schimbă polaritatea. [WIKISTEP17]

Viteză cu care se poate schimba polaritatea bobinelor este data de raportul dintre inductanță și rezistentă lor (L / R), astfel încât dacă avem de exemplu o înfăşurare cu inductanță 10mH și rezistentă

2ohmi vor fi necesare aproximativ 5ms pentru a ajunge la două treimi din cuplul maxim iar după aproximativ 25ms se va ajunge la 99% din cuplul maxim. Astfel, pentru ca motoarele pas cu pas să aibă cuplu mare la turații mari este necesară o tensiune de alimentare mare dar rezistente și inductanțe mici ale bobinelor.

Folosind circuitele de tensiune constantă este posibilă comanda motoarelor cu rezistentă mică adăugând rezistențe externe pe înfășurările lor. Rezistențele externe vor risipi energie, transformând-o în căldură, dar asta reprezintă o soluție simplă și ieftină de control.

4.6.2 Circuitele de curent constant

Circuitele de curent constant se numesc astfel deoarece ele mențin un curent aproximativ constant prin fiecare înfășurare. La fiecare pas, o tensiune foarte mare este aplicată bobinei, crescând curentul bursc. Curentul este monitorizat, de obicei măsurând tensiunea pe o rezistență conectată în serie cu fiecare înfășurare. Atunci când curentul depășește o anumită valoare, tensiunea este decuplată folosind tranzistoare de putere. Asta va face curentul să scadă, astfel declanșând conectarea tensiunii din nou. Astfel curentul este menținut aproximativ constant pentru fiecare pas. Această metodă de control oferă cuplu și viteze mai mari motoarelor și nu este dificil de implementat datorită circuitelor integrate specializate.

4.7 Secvențe de control

Motoarele pas cu pas sunt motoare sincrone de curent alternativ cu mai multe faze și în mod ideal sunt controlate de un curent sinusoidal. Având în vedere că de obicei controlul lor se face cu semnale numerice, o aproximare a semnalului sinusoidal va fi un semnal dreptunghiular. Există mai multe metode de a genera secvența de pulsuri la terminalele motorului.

4.7.1 Secvența cu o singură fază activă (Wave drive)

Când este folosită această metodă, o singură fază este activă la orice moment de timp. Metoda cu o singură fază activă produce același număr de pași ca și metoda cu două faze active dar produce un cuplu mult mai mic. Controlul folosind secvența cu o singură fază activă este rar întâlnită deoarece nu oferă avantaje față de secvența cu două faze active.

4.7.2 Secvenţa cu două faze active (Full-step)

Full-step drive reprezintă cea mai întâlnită metodă de control a motoarelor pas cu pas. În orice moment de timp două din înfăşurările motorului vor fi alimentate, oferind maximul de cuplu de care motorul este capabil. Odată ce o fază este oprită, următoarea este alimentată, permiţând acelaşi număr de paşi ca şi Wave drive dar cuplu mult îmbunătăţit.

4.7.3 Secvenţa cu jumătate de pas (Half-stepping)

Folosind Half-stepping secvenţa de control a înfăşurărilor alternează între alimentarea a două faze şi a unei singure faze, crescând rezoluţia unghiulară a motorului. Dezavantajul acestei metode este cuplul scăzut dar poate fi compensat crescând curentul pe fiecare înfăşurare. Secvenţa de control cu jumătate de pas nu necesită modificări ale circuitului de control deoarece diferenţa constă doar în mărimea intervalului de timp în care fazele sunt active. Spre deosebire de metodele anterioare de control, Half-stepping permite un număr dublu de paşi.

4.7.4 Micropăşire (Microstepping)

Micropăsirea reprezintă controlul motoarelor pas cu pas prin semnale sinusoidale aproximate cu diferite perioade. Odată cu micşorarea perioadei, motorul permite o rezoluţie mai mică dar şi o funcţionare mai fluidă, spre deosebire de metodele anterioare care produc mişcări discrete la viteze mici. Totuşi, este posibil ca atunci când perioada semnalelor este prea mică motorul să aibă nevoie de mai multe pulsuri pentru a efectua un singur pas.

5 APLICAȚIE ARDUINO

5.1 Informaţii generale

Arduino este o companie ce dezvoltă soluții hardware și software specializate scopurilor educaționale și orientate către entuziaști, nu neapărat mediului industrial. Produsele sunt distribuite conform licențelor GNU Lesser general Public License și GNU General Public License, deci sunt complet opensource permițând oricui să producă sau să modifice proiectele.[ARD17]

Plăcile Arduino folosesc o gamă variată de microcontrollere, de la AVR 8-bit până la arhitecturi Intel Quark x86 sau ARM Cortex-M3. Ele sunt echipate cu interfețe intrare/ieșire conectate la pinii microcontrollerului ce fac posibilă conectarea cu diferite plăci secundare.[WIKIARD17]

5.2 Hardware-ul Arduino

Schemele de referință Arduino sunt distribuite sub licență opensource. Atât schemele cât și cablajele plăcilor sunt disponibile.

Majoritatea plăcilor de dezvoltare Arduino sunt echipate cu un microcontroller Atmel 8 bit, mai multe versiuni cu diferite configurații de memorie, pini sau periferice fiind disponibile. În 2012, Arduino a lansat placă Arduino Due care folosește microcontrollerul Atmel SAM3X8E 32-bit. Plăcile de dezolvatare Arudino sunt echipate cu un rând sau două de conectori ce permit conectarea cu plăci externe numite shield-uri. Cele mai multe plăci și shield-uri permit suprapunerea prin modul în care sunt proiectate și prin faptul ca, în general, folosesc protocolul de comunicare I2C.

Microcontrollerele Arduino vin programate din fabrică cu un bootloader care simplifică programarea lor, deoarece astfel nu mai este nevoie de un programator special. Ele pot fi programate prin Universal Serial Bus (USB) care este implementat folosind un adaptor USB la serial.

Fiind un proiect opensource, există mai multe tipuri de plăci de dezvoltare sau shield-uri compatibile cu plăcile Arduino. Plăcile de dezvoltare produse de terți care s-au bazat pe arhitectură oferită de Arduino sunt în mare parte identice din punct de vedere funcțional.[WIKIARD17]

Plăcile de dezvoltare Arduino oficiale:

- Arduino RS232 – Placă de dezvoltare simplă ce are implementat protocolul de comunicare RS232.

- Arduino Decimila – Placă de dezvoltare ce folosește microcontrollerul Atmega168. „Decimila” inseamnă 10.000 în italiană și placa a fost lansată pentru a marca faptul că 10.000 de plăci Arduino au fost produse.
- Arduino Duemilanove – Placă de dezvoltare ce folosește microcontrollerul Atmega168 lansată în anul 2009.
- Arduino Uno R2 – Cea mai cunoscută și vandută placă de dezvoltare Arduino. Ea folosește microcontrollerul Atmega328P și a fost lansată pentru a marca lansarea primei versiuni a mediului de dezvoltare Arduino Software.
- Arduino Uno SMD – Versiune a plăcii Arduino Uno R2 ce folosește varianta SMD a micrcontrollerului.
- Arduino Leonardo – Placă de dezvoltare ce folosește microcontrollerul Atmega32u4. Spre deosebire de plăcile anterioare, Arudino Leonardo este prima placă ce nu folosește un microcontroller secundar pentru programarea Atmega32u4 deoarece acesta are protocolul de comunicare USB deja implementat.
- Arduino Pro – Placă de dezvoltare echipată cu microcontrollerul Atmega328 care are un design minimalist, majoritatea pinilor micrcontrollerului fiind conectați direct la conectori externi.
- Arduino Mega – Placă de dezvoltare ce folosește microcontrollerul Atmega 2560 cu o conectivitate mărita datorită celor 54 de pini ai microcontrollerului.
- Arduino Nano – O placă de dezvoltare compactă, echipată cu Atmega 328. Ideală pentru proiectele mici datorită dimensiunilor reduse.
- Arduino LilyPad – Placă de dezvoltare destinată textilelor. Ideea care a stat la baza acestei plăci a fost ca ea să fie integrată în articolele vestimentare.
- Arduino Robot – Arduino Robot este de fapt un kit, nu doar o placă de dezvoltare. Robotul este echipat cu roți și două microcontrollere, unul pentru controlul motoarelor și unul pentru controlul senzorilor.
- Arduino Esplora – Placă de dezvoltare echipată cu lumini și difuzoare, joystick, sensor de temperatură, accelerometru și multe altele, foarte potrivită pentru proiectele entuziaștilor de electronică. Ea este bazată pe Arduino Leonardo și folosește Atmega32U4
- Arduino Ethernet – Placă de dezvoltare foarte similară cu Arduino Uno dar echipată cu port ethernet și microSD cârd reader.

- Arduino Yun – Placă de dezvoltare echipată cu Atmega32U4 dar și Atheros AR9331, cel din urmă suportant Linux. Placa este prevăzută cu WiFi, microSD card reader și multe alte periferice care o fac potrivită pentru proiectele de tip Internet of Things care necesită putere mare de calcul.

- Arduino Due – Prima placă echipată cu un microcontroller de 32 biți ARM. Microcontrollerul Atmel SAM3X8E îi oferă o putere foarte mare de procesare și o mulțime de periferice dar, din păcate, nu este compatibil cu majoritatea shield-urilor proiectate pentru restul plăcilor Arduino.

5.3 Software-ul Arduino

Compania din spatele platformei Arduino pune la dispoziție un mediu de dezvoltare software (IDE), capabil să ruleze pe mai multe platforme (Windows, Linux, MacOS) în special datorită faptului că a fost implementat în limbajul Java. IDE-ul are integrată o consolă pentru depanare, butoane pentru compilare și scrierea programului în memoria microcontrollerului, capabilități de editor text performanțe și evidențierea sintaxei.

Un program pentru microcontrollere scris folosind mediul de dezvoltare Arduino se numește schița. Schițele sunt salvate sub formă unor fișiere text dar folosesc extensia .ino. Arduino IDE suportă limbajele C și C++ iar pentru compilare folosește o variantă modificată a compilatorului GNU. Pentru scrierea programului în memoria microcontrollerului Arduino IDE folosește avrdude. [WIKIARD17]

5.4 Librării

Mediul de dezvoltare Arduino permite utilizarea librăriilor, la fel ca majoritatea mediilor de dezvoltare și limbajelor de programare. Librăriile Arduino extind funcționalitatea mediului de dezvoltare, în cele mai multe cazuri acționând ca API-uri pentru diverse componente hardware.

5.4.1 Librăria ESP8266WiFi

Librăria ESP8266WiFi este un proiect opensource cu scopul de a permite folosirea utilizării mediului de dezvoltare Arduino pentru programarea microcontrollerului ESP8266. Librăria vine integrată cu suport pentru controlul modulului WiFi folosind protocol TCP sau UDP, posibilitatea

implementării serverelor HTTP, mDNS, DNS sau SSDP, programare OTA (over the air), adică programarea microcontrollerului fără fir, scriere şi citire carduri SD, controlul servo motoarelor cât şi implementarea protocoalelor de comunicare SPI şi I2C.[GITARD17]

5.4.2 Librăria ArduinoOTA

Librăria ArduinoOTA este o componentă a librăriei ESP8266WiFi ce permite programarea microcontrollerului folosind conexiune WiFi, eliminând astfel necesitatea conexiunii USB. Ea este foarte uşor de folosind având în vedere avantajul substanţial pe care îl aduce. Pentru a putea programa microcontrollerul prin WiFi este necesar apelul doar a două funcţii, funcţia begin a librăriei la pornirea programului şi respectiv funcţia handle la fiecare iteraţie a buclei principale.

5.4.3 Librăria Wire

Librăria Wire este o componentă a pachetului standard Arduino IDE şi se ocupă cu comunicarea folosind protocolul I2C. Ea vine însoţită de mai multe exemple explicate pas cu pas cât şi documentaţia necesară accesibilă pe site-ul platformei.

Librăria Wire este necesară librăriei VL53L0X şi permite comunicarea atât cu senzorul VL53L0X cât şi cu driverele motoarelor pas cu pas.

5.4.4 Librăria Adafruit_MotorShield

Librăria Adafruit_MotorShield este o librărie opensource pusă la dispoziţie de către Adafruit cu scopul de a controla motoarele folosind placa Adafruit Motor Shield v2. Librăria permite controlul motoarelor de curent continuu şi a motoarelor pas cu pas în diferite moduri de funcţionare, inclusiv micro-păşire. Adafruit_MotorShield este făcută astfel încât să permită folosirea mai multor plăci de control a motoarelor, respectând principiul de suprapunere a plăcilor.[GITADA17]

Librăria se ocupă cu controlul impulsurilor pentru motoarele pas cu pas într-un mod cât mai uşor pentru utilizator, fiind necesare doar configurarea motoarelor, setarea vitezi sau a numărului de păşi.

5.4.5 Librăria Adafruit_MS_PWMServoDriver şi AccelStepper

Această librărie este o componentă a librăriei AccelStepper ce foloseşte Adafruit_MotorShield. Librăria Adafruit_MS_PWMServoDriver permite controlul avansat al motoarelor pas cu pas având următoarele capabilităţi:[ACCL17]

- controlul accelerarii si decelerarii motoarelor pas cu pas
- controlul mai multor motoare pas cu pas complet independent
- libraria nu blocheaza bucla principala a programului
- controlul motoarelor pas cu pas cu 2,3 sau 4 terminale
- controlul la viteze foarte mici

5.4.6 Librăria VL53L0X

Librăria VL53L0X este un proiect opensource bazat pe API-ul pus la dispoziţie de către ST Microelectronics pentru controlul senzorului VL53L0X. Scopul librăriei este de a oferi utilizatorului o metodă rapidă şi simplă de utilizare a senzorului în cadrul mediului de dezvoltare Arduino. Librăria VL53L0X foloseşte librăria Wire din pachetul standard Arduino pentru comunicarea microcontrollerului cu senzorul folosind protocolul I2C.[GITVL17]

Librăria VL53L0X are următoarele capabilităţi:

- returnarea stării tranzacţiei curente între microcontroller şi senzor pe magistrala I2C prin funcţia *last_status*
- modificarea adresei I2C a senzorului VL53L0X prin funcţia *setAddress(uint8_t new_addr)*
- iniţializarea şi configurarea senzorului la pornire şi returnarea unei variabile ce semnifică dacă intilizarea şi configurarea s-au efectuat cu succes prin funcţia *init(bool io_2v8 = true)*
- scrierea regiştrilor 8 bit ai senzorului prin funcţia *writeReg(uint8_t reg, uint8_t value)*
- scrierea regiştrilor 16 bit ai senzorului prin funcţia *writeReg16Bit(uint8_t reg, uint16_t value)*
- scrierea registrilor 32 bit ai senzorului prin functia *writeReg32Bit(uint8_t reg, uint32_t value)*
- citirea regiştrilor 8 bit ai senzorului prin funcţia *readReg(uint8_t reg)*

- citirea regiştrilor 16 bit ai senzorului prin funcţia *readReg16Bit(uint8_t reg)*
- citirea regiştrilor 32 bit ai senzorului prin funcţia *readReg32Bit(uint8_t reg)*
- scrierea unui şir de octeţi în memoria senzorului, începând cu un anumit registru, prin funcţia *writeMulti(uint8_t reg, uint8_t const * src, uint8_t count)*
- citirea unui şir de octeţi din memoria senzorului, începând cu un anumit registru, prin funcţia *readMulti(uint8_t reg, uint8_t * dst, uint8_t count)*
- setarea numărului de eşantioane pe secundă, setare ce este direct legată de acurateţea senzorului, prin functia *setSignalRateLimit(float limit_Mcps)*
- returnarea numărului de eşantioane folosind funcţia *getSignalRateLimit(void)*
- setarea timpului minim acordat fiecărei eşantionări, setare ce este direct legată de acurateţea senzorului, prin funcţia *setMeasurementTimingBudget(uint32_t budget_us)*
- returnarea timpului minim acordat fiecărei eşantionări folosind funcţia *getMeasurementTimingBudget(void)*
- pornirea modului de eşantionare continuă folosind funcţia *startContinuous(uint32_t period_ms = 0)*

5.4.7 Librăria ArduinoJson

ArduinoJson este o librărie a mediului de dezvoltare Arduino ce permite codarea şi decodarea de şiruri JSON. JavaScript Object Notation (JSON) reprezintă un format de reprezentare a datelor uşor de interpretat atât pentru microcontrollere cât şi de către utilizator. JSON este un format text compatibil în totalitate cu orice limbaj de programare şi foloseşte convenţii din limbajele populare de programare pentru o interpretare cât mai uşoară.[GITJSON17]

ArduinoJson este una din implementările JSON concepute special pentru mediul de dezvoltare Arduino, având în scop principal utilizarea cât mai eficientă a memoriei. Librăria are următoarele capabilităţi:

- codare JSON
- decodare JSON
- API uşor de utilizat

- alocare de memorie fixă (nu foloseşte alocare dinamică)
- nu depinde de alte librării
- licenta opensource

5.4.8 Librăria TaskScheduler

TaskScheduler este o librărie pentru mediul de dezvoltare Arduino ce permite executarea sarcinilor de lucru pe mai multe fire independente cu diferite priorităţi şi configurări.[GITTASK17]

TaskScheduler permite:

- execuţia periodică a unei sarcini cu perioada dinamică configurabilă în milisecunde sau microsecunde
- configurarea numărului de interatii a unei sarcini
- modificarea parametrilor unei sarcini în mod dinamic, în timpul execuţiei
- moduri de gestionare a energiei atunci când nicio sarcină nu este activă
- executarea sarcinilor cu declanşare la evenimente
- prioritizarea sarcinilor

O sarcină reprezintă o parte a programului care necesită să fie executată în anumite condiţii.

Sarcină este caracaterizata de următoarele elemente:

1. Porţiunea de cod specifica sarcinii
2. Intervalul la care sarcina este executata
3. Numărul de iteratii ale sarcinii
4. Evenimentul care declanseaza executarea sarcinii (optional)

Sarcinile sunt legate între ele printr-un lanţ de execuţie care coordonează execuţia fiecărei sarcini. Fiecare sarcină îşi execută porţiunea de cod printr-o funcţie de tip callback. Lanţul de execuţie apelează funcţia callback a fiecărei sarcini periodic până când sarcina a fost dezactivată sau numărul de iteraţii a ajuns la 0.[TASK17]

5.5 Software

În continuare va fi prezentat și explicat programul implementat pe microcontrollerul ESP8266 care se ocupă de controlul robotului.

```
void setup() {
  Serial.begin(115200);

  Serial.println("boot...");

  esp_init();
  comm_init();
  motor_init();
  sensor_init();
  tasks_init();

  pinMode(BUILTIN_LED, OUTPUT);
}

void loop() {
  runner.execute();
  stepperright.run();
  stepperleft.run();
  stepperrotate.runSpeed();
}
```

Orice program scris în mediul de dezvoltare Arduino trebuie să conțină obligatoriu cele două funcții setup și loop. Funcția setup este apelată imediat după alimentarea microcontrollerului și este executată o singură data. Funcția loop este apelată continuu după ce funcția setup și-a terminat execuția.

În funcția setup vom inițializa comunicația serială pe USB ce va fi folosită pe viitor la depanare. Tot aici vom apela funcțiile de inițializare pentru diferite module și vom seta LED-ul încorporat ca ieșire digitală, LED ce va funcționa pe post de martor.

În funcția loop vom apela funcțiile obiectelor din librăriile TaskScheduler și Adafruit_MotorShield care se vor ocupă cu gestionarea sarcinilor, respectiv controlul motoarelor. Apelul acestor funcții trebuie făcut cât mai des pentru o funcționare bună a celor două librării. Apelul lor nu inseamnă neapărat că ele și execută o sarcina anume, dar apelul cât mai des le permite să execute sarcinile cu o întârziere cât mai mică atunci când ele apar.

```
void motor_init()
{
  AFMSTop.begin();
  AFMSBot.begin();

  stepperright.setMaxSpeed(100.0);
  stepperright.setAcceleration(50.0);

  stepperleft.setMaxSpeed(100.0);
  stepperleft.setAcceleration(50.0);

  stepperrotate.setMaxSpeed(25.0);
}
```

Funcția motor_init, apelată în cadrul funcției setup, se ocupă de inițializarea obiectelor AFMTop și AFMBot care reprezintă cele două plăci de control a motoarelor pas cu pas. Tot această funcție se ocupă de inițializarea motoarelor și configurarea lor.

Motoarele pas cu pas ale roților sunt configurate cu viteză maximă de 100.0 pași pe secundă și accelerația 50 pași pe secundă pe secundă. Viteza maximă a motorului pas cu pas care rotește senzorul VL53L0X este de 25 de pași pe secundă iar acesta nu va fi controlat luând în considerare accelerație, el va primi un semnal treaptă pentru a păstra eșantioanele senzorului cât mai precise.

```
void esp_init()
{
  WiFi.softAP("ESPap");
  server.begin();

  ArduinoOTA.begin();
}
```

Funcția esp_init se ocupă de configurarea modulului WiFi al microcontrollerului ESP8266, acesta fiind setat în mod Punct de Acces. Tot aici este pornit și server-ul la care se va conecta clientul și este inițializată librăria ArduinoOTA pentru programarea prin WiFi a microcontrollerului.

```
void comm_init()
{
  Wire.begin();
}
```

Funcția comm_init este apelată în cadrul funcției setup și este responsabilă de inițializarea librăriei Wire care se ocupă cu comunicarea I2C dintre microcontroller, senzor și driverele motoarelor pas cu pas. Este necesar că aceasta inițializare să se facă înaintea inițializării obiectelor responsabile de senzorul VL53L0X sau driverele de motoare.

```
void sensor_init()
{
  sensor.init();
  sensor.setTimeout(20);

  sensor.startContinuous();
}
```

Funcția sensor_init, apelată în cadrul funcției setup, se ocupă cu inițializarea și configurarea senzorului VL53L0X. Obiectul sensor este o componentă a librăriei VL53L0X iar funcția init trebuie apelată după ce este inițializată librăria Wire. Funcția setTimeout(20) forțează senzorul să returneze o valoare măsurată la fiecare 20 de milisecunde cel mult iar startCountinous() declanșează eșantionarea valorilor senzorului în mod continuu.

```
void tasks_init()
{
  runner.init();
  runner.addTask(task_led_blink);
  runner.addTask(task_send_data);
  runner.addTask(task_sensor_read);
  runner.addTask(task_rotate);
  runner.addTask(task_motor_control);
  runner.addTask(task_loop);

  task_led_blink.enable();
  task_send_data.enable();
  task_sensor_read.enable();
  task_motor_control.enable();
  task_rotate.enable();
  task_loop.enable();
}
```

Funcția tasks_init este apelată în cadrul funcției setup și este responsabilă de inițializarea obiectului runner, obiect al librăriei TaskScheduler. Obiectul este inițializat și sarcinile sunt adăugate în lanțul de sarcini. După inițializare și adăugare în lanțul de sarcini, toate sarcinile sunt pornite iar obiectul runner se va ocupa de executarea lor conform configurărilor.

```
#include <ESP8266WiFi.h>
#include <ArduinoOTA.h>
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWMServoDriver.h"
#include <AccelStepper.h>
#include <VL53L0X.h>
#include <ArduinoJson.h>
#include <TaskScheduler.h>

WiFiServer server(1212);
WiFiClient client;

VL53L0X sensor;

Adafruit_MotorShield AFMSTop = Adafruit_MotorShield(0x60);
Adafruit_MotorShield AFMSBot = Adafruit_MotorShield(0x61);
Adafruit_StepperMotor *rightMotor = AFMSTop.getStepper(48, 2);
Adafruit_StepperMotor *leftMotor = AFMSTop.getStepper(48, 1);
Adafruit_StepperMotor *rotateMotor = AFMSBot.getStepper(48, 2);
```

În această secvența de program sunt incluse toate librăriile folosite în programul microcontrollerului și sunt definite obiectele pe care le vom folosi.

Obiectul server de tip WiFiServer reprezintă un server TCP creat de către ESP8266 care ascultă portul 1212 și este deschis oricărei conexiuni a unui client.

Obiectul client de tip WiFiClient reprezintă un client TCP și va fi definit în momentul în care un client este conectat.

Obiectele AFMTop și AFMBot reprezintă obiectele responsabile de plăcile ce controlează motoarele pas cu pas, puse la dispoziție de către librăria Adafruit_MotorShield. Obiectul AFMTop reprezintă driverul de motoare pas cu pas cu adresa 0x60 de pe magistrală I2C. Obiectul AFMBot reprezintă driverul de motoare pas cu pas cu adresa 0x61 de pe magistrală I2C.

Obiectele rightMotor, leftMotor și rotateMotor fac parte din librăria Adafruit_MotorShield și reprezintă ele 3 motoare pas cu pas. Fiecare din ele este configurat că având 48 de pași, rightMotor și rotateMotor fiind pe terminalul 2 al plăcii iar leftMotor pe terminalul 1 al plăcii.

```
#define MOVE_TYPE                INTERLEAVE

void forwardstepright() {
  rightMotor->onestep(FORWARD, MOVE_TYPE);
}
void backwardstepright() {
  rightMotor->onestep(BACKWARD, MOVE_TYPE);
}

void forwardstepleft() {
  leftMotor->onestep(FORWARD, MOVE_TYPE);
}
void backwardstepleft() {
  leftMotor->onestep(BACKWARD, MOVE_TYPE);
}
void forwardsteprotate() {
  rotateMotor->onestep(FORWARD, DOUBLE);
}
void backwardsteprotate() {
  rotateMotor->onestep(BACKWARD, DOUBLE);
}

AccelStepper stepperright(forwardstepright, backwardstepright);
AccelStepper stepperleft(forwardstepleft, backwardstepleft);
AccelStepper stepperrotate(forwardsteprotate, backwardsteprotate);
```

Librăria AccelStepper necesită definirea unor funcţii de păşire pentru fiecare direcţie a fiecărui motor, permiţând astfel configurarea motoarelor în diferite secvenţe de control. Pentru controlul motoarelor roţilor vom folosi secvenţe de control INTERLEAVE în funcţiile forwardstepright, backwardstepright, forwardstepleft şi backwardstepleft. Acest tip de control permite atingerea unor viteze suficient de mari pentru controlul robotului. Pentru controlul motorului pe care este montat senzorul vom folosi secvenţa de control DOUBLE care asigură un control mai fiabil, înjumătăţind şansele că motorul să piardă un pas comparativ cu secvenţa de control SINGLE.

Obiectele stepperright, stepperleft şi stepperrotate sunt componente ale librăriei AccelStepper şi reprezintă obiectele ce controlează cele 3 motoare. Este foarte important să nu confundăm obiectele rightMotor, leftMotor şi rotateMotor cu stepperright, stepperleft şi stepperrotate, primele fiind obiectele ce reprezintă motoarele iar cele din urmă fiind obiectele ce controlează cele 3 motoare.

```
char rx_buffer[100];
int rx_buffer_index = 0;
int sensor_data[35];
int sensor_data_index = 0;
int right_data;
int left_data;


unsigned char isRotatingReading = false;

int last_sensor_read = 0;

unsigned char has_data = 0;
unsigned char send_data = 0;
```

Secvența de program de mai sus definieste o serie de variabile care vor fi folosite în continuare pentru transimisia și recepția de date, stocarea eșantioanelor primite de la senzor cât și diferite flag-uri pentru anumite stări în care se află programul.

```
// Callback methods prototypes
void task_led_blink_callback();
void task_send_data_callback();
void task_motor_control_callback();
void task_sensor_read_callback();
void task_rotate_callback();
void task_loop_callback();

//Tasks
Task task_led_blink(250, TASK_FOREVER, &task_led_blink_callback);
Task task_send_data(10, TASK_FOREVER, &task_send_data_callback);
Task task_motor_control(250, TASK_FOREVER, &task_motor_control_callback);
Task task_sensor_read(35, TASK_FOREVER, &task_sensor_read_callback);
Task task_rotate(50, TASK_FOREVER, &task_rotate_callback);
Task task_loop(100, TASK_FOREVER, &task_loop_callback);
```

Librăria TaskScheduler necesită definirea prototipurilor funcțiilor ce urmează să devină sarcini înainte ca ele să fie implementate. În prima porțiune a secvenței de program sunt definite prototipurile funcțiilor de tip callback ce vor fi apelate de sarcinile definite în a două porțiune de program. Sarcinile sunt definite cu o serie de parametrii precum perioadă, tipul task-ului și funcția de tip callback pe care urmează să o apeleze. După cum se poate observă, am ales diferite perioade de declanșare pentru sarcini iar ele au fost impărtite astfel încât fiecare din ele să controleze o parte a robotului. Au fost definite 6 sarcini care se ocupă de aprinderea led-ului martor, transmisia de date, controlul motoarelor, citirea eșantioanelor primite de la senzor, controlul motorului pe axul căruia este montat senzorul și o buclă principală pentru executarea altor funcții auxiliare.

```
void task_led_blink_callback() {
  digitalWrite(BUILTIN_LED, !digitalRead(BUILTIN_LED));
}
```

Funcția de tip callback de mai sus se ocupă cu aprinderea led-ului martor și este executată o data la 250ms. Pentru aprinderea efectivă a led-ului ea folosește funcțiile digitalWrite și digitalRead din librăria standard Arduino.

```
void task_send_data_callback() {

  if (!send_data)
    return;

  const size_t bufferSizetx = 2 * JSON_ARRAY_SIZE(1) + JSON_ARRAY_SIZE(6) + JSON_OBJECT_SIZE(3)
                                                                                + 50;
  DynamicJsonBuffer jsonBuffertx(bufferSizetx);

  JsonObject& roottx = jsonBuffertx.createObject();
  JsonArray& sensor = roottx.createNestedArray("s");

  while (sensor_data_index > 0) {
    sensor_data_index--;
    sensor.add(sensor_data[sensor_data_index]);
  }

  roottx["r"] = right_data;
  roottx["l"] = left_data;


  char buffer[250];
  roottx.printTo(buffer, sizeof(buffer));
  client.println(buffer);

  left_data = 0;
  right_data = 0;
  send_data = 0;
}
```

Funcția de tip callback de mai sus se ocupă cu transmisia datelor odată la 10ms. Ea verifică mai întâi dacă flag-ul send_data a fost setat iar apoi, dacă acesta a fost setată, crează un șir de caractere în format JSON ce conține datele primite de la senzor, pașii făcuti de motorul din dreaptă și pașii făcuti de motorul din stânga. Șirul de caractere în format JSON este salvat într-un buffer intermediar iar apoi este transmis către obiectul client care se va ocupă de transmisia TCP.

```
void task_motor_control_callback() {
  if (!has_data)
    return;

  const size_t bufferSizerx = JSON_OBJECT_SIZE(2) + 20;
  DynamicJsonBuffer jsonBufferrx(bufferSizerx);
  JsonObject& rootrx = jsonBufferrx.parseObject(rx_buffer);

  int command = rootrx["c"];
  int value = rootrx["v"];
  if (rootrx.success()) {
    switch (command) {
```

Funcția de tip callback task_motor_control_callback se ocupă cu controlul motoarelor prin folosirea obiectelor stepperright, stepperleft și stepperrotate. Funcția verifică flag-ul has_data iar apoi, dacă acesta este activ, va decoda șirul de caractere JSON din variabila rx_buffer. În șirul JSON variabilele c și v reprezintă tipul comenzii și valoarea acesteia. În funcție de tipul comenzii funcția task_motor_control_callback va acționa unul sau mai multe motoare.

```
    case 3:
      stepperright.move(10);
      stepperleft.move(10);
      right_data = 10;
      left_data = 10;
      break;
```

Mai sus avem un exemplu de comandă, și anumite comanda 3 care va muta robotul în față 10 pași. Porțiunea de cod folosește obiectele stepperright și stepperleft din librăria AccelStepper, apelând funcția move a lor apoi pune valorile cu care s-au mutat motoarele în variabilele right_data și left_data. Aceste două variabile vor fi transmise de către sarcina task_send_data.

```
void task_sensor_read_callback() {
  last_sensor_read = sensor.readRangeContinuousMillimeters();
  sensor_data[sensor_data_index] = last_sensor_read;
  sensor_data_index++;
  if (sensor_data_index == 28)
    send_data = 1;
}
```

Funcția de tip callback task_sensor_read_callback este apelată la fiecare 35ms și se ocupă cu achiziția datelor de la senzor. Perioada de 35ms este aleasă conform limitărilor senzorului, fiind minimul stabil pentru funcționarea robotului. Funcția folosește obiectul sensor din librăria VL53L0X, salvează valoarea acestuia în variabila last_sensor_read pe care apoi o adaugă la sfârșitul unui șir de numere întregi, și anume sensor_data. Atunci când șirul ajunge la elementul 28, flag-ul send_data este activat iar datele vor fi transmise de către sarcina task_send_data.

```
void task_rotate_callback() {
  if (isRotatingReading)
  {
    stepperrotate.setSpeed(25);
  }
  else {
    stepperrotate.setSpeed(0);
  }
}
```

Funcția de tip callback task_rotate_callback se ocupă de controlul motorului pas cu pas pe axul căruia este montat senzorul. Ea folosește obiectul stepperrotate care semnifică controlul motorului din librăria AccelStepper. Controlul motorului se face în funcție de flag-ul isRotatingReading. Spre deosebire de funcția task_motor_control_callback această funcție nu folosește move pentru controlul motorului, deoarece în librăria AccelStepper, funcția move ia în considerare accelerația motorului. Pentru că valorile senzorului să fie corect afișate se ignoră accelerația motorului, iar acestuia i se aplică un semnal treaptă folosind funcția setSpeed.

```
void task_loop_callback() {
  ArduinoOTA.handle();

  if (server.hasClient()) {
    if (!client.connected()) {
      client = server.available();
      client.setNoDelay(false);
      server.setNoDelay(false);
      Serial.println("New client!");
    }
  }

  while (client.available()) {
    rx_buffer[rx_buffer_index] = client.read();
    if (rx_buffer[rx_buffer_index] == '#')
      has_data = 1;
    else
      rx_buffer_index++;
  }
}
```

Funcția task_loop_callback este funcția de tip callback a sarcinii task_loop care se ocupă cu executarea proceselor auxiliare şi este executată la fiecare 100ms. Prima linie de cod a funcției apelează funcția handle a obiectului ArduinoOTA din librăria ArduinoOTA pentru programarea prin WiFi a microcontrollerului. În continuare funcția task_loop_callback verifică dacă server-ul WiFi al ESP8266 are o cerere de conectare de la un client, dacă da, acel client va fi salvat în variabila client pentru a permite comunicarea cu acesta în celelalte funcții callback. Tot aici atât clientul cât şi serverul sunt configurați astfel încât să păstreze conexiunea chiar dacă nu există trafic. Având în vedere că recepționarea de date este un proces care nu poate fi controlat de către microcontroller, ea se face blocând execuția principală de sarcini, printr-o buclă while. Fiecare octet de date primit de la client este copiat într-un buffer rx_buffer iar atunci când se detectează caracterul #, care semnifică sfârşitul transmisiei, flag-ul has_data este activat. În caz că nu s-a primit caracterul care semnifică sfârşitul transmisiei, un contor al buffer-ului de recepție este incrementat.

6 APLICAȚIE JAVA

6.1 Informații generale

Aplicația Java din cadrul proiectului reprezintă interfața utilizatorului cu robotul. Ea este folosită atât pentru controlul robotului cât și pentru vizualizarea datelor transmise de acesta.

Java este un limbaj avansat de programare orientat pe obiecte dezvoltat special pentru a depinde de cât mai puține elemente externe. Limbajul Java a fost dezvoltat cu scopul de a respecta principiul “write once, run everywhere” care permite programatorilor să scrie un program o singură dată și să îl ruleze pe diferite platforme și sisteme de operare. Limbajul Java este unul din cele mai populare limbaje de programare și a fost dezvoltat de către Sun Microsystems și lansat în anul 1995 mai târziu urmând ca Sun Microsystems să fie cumpărată de Oracle Corporation. Implementarea originală a limbajului, compilatorului și a librăriilor standard a fost făcută sub licență proprietară dar în 2007 Sun a decis să modifice licență tuturor componentelor la GNU General Public License, devenind opensource. Ultima versiune, și anume Java 8, este momentan întreținută de către Oracle și utilizarea sa nu implică niciun cost.[WIKIJAVA17]

Când James Gosling a început să dezvolte limbajul de programare Java, el a enunțat 5 principii care încă stau la baza limbajului:

- trebuie sa fie sigur, orientat pe obiecte si familiar
- trebuie sa fie robust si securizat
- trebuie sa fie portabil si neutru din punct de vedere al arhitecturii pe care ruleaza
- trebuie sa fie executat cu performanta sporita
- trebuie sa fie un limbaj interpretat, cu fire de executie si dinamic

Unul din obiectivele limbajului este ca acesta sa fie portabil, ceea ce inseamna ca programele scrise in Java sa ruleze la performante similare indiferent de arhitectura hardware sau sistemul de operare. Acest obiect a fost indeplinit compiland codul Java intr-un cod intermediar, numit Java bytecode, spre deosebire de alte limbaje care compileaza codul direct in cod masina (ASM) specific arhitecturii procesorului pe care ruleaza. Instructiunile Java bytecode sunt similare instructiunilor codurilor masina dar ele urmeaza sa fie executate de catre o masina virtuala, care la randul sau a fost scrisa pentru platforma hardware specifica pe care ruleaza. Utilizatorii care ruleaza aplicatii Java

folosesc Java Runtime Enviorment instalat pe platformele lor iar pentru aplicatii web, se folosesc Java applets.

Folosirea unor instructiuni universale (bytecode) fac codul portabil, singura conditie fiind ca platforma pe care urmeaza sa fie executat sa aiba o masina virtuala Java (Java VM) instalata. Java VM face parte din Java Runtime Enviorment. Totusi, deoarece codul nu este executat direct de catre procesor, programele Java vor rula mereu mai lent decat cele scrise in limbaje de programare native.

6.2 Mediul de dezvoltare IntelliJ IDEA

IntelliJ IDEA este un mediu de dezvoltare (IDE) pentru limbajul de programare Java. IDEA a fost creat de către compania JetBrains și este disponibil în două versiuni, community edition sub licență Apache 2 și commercial sub licență proprietară.

Prima versiune IntelliJ IDEA a fost lansată în ianuarie 2001 și a fost unul din primele medii de dezvoltare avansate pentru limbajul Java. În 2010 IDEA a primit cel mai bun scor din 4 cele mai populare medii de dezvoltare pentru Java. În decembrie 2014, Google a lansat Android Studio, un mediu de dezvoltare opensource pentru dezvoltarea de aplicații Android, care a fost bazat pe versiunea liberă a IntelliJ IDEA.[WIKIIDEA17]

Ultima versiune a mediului de dezvoltare IntelliJ IDEA are capabilități precum:

- completare automata avansata in functie de context
- navigare prin diferite clase in functie de context
- sugestii la detectarea eventualelor erori sau nerespectarea standardelor de programare
- unelte precum Git, SVN, Maven, Microsoft SQL Server, MySQL
- ii pot fi adaugate peste 1500 de plugin-uri care pot extinde functionalitatea

6.3 Librării

6.3.1 Swing

Swing este o librărie GUI (Graphic User Interface) pentru Java, componentă a Java Foundation Classes, adică librăriile standard ale pachetului de dezvoltare Java.

Swing a fost dezvoltat pentru a înlocui AWT (Abstract Window Toolkit), reprezentând o implementare mai avansată a setului de componente de interfață. Librăria oferă implicit interfață nativă pentru diferite platforme şi poate fi configurată astfel încât aplicaţiile să arate ca şi cum ar rula pe alte platforme sau le pot fi aplicate diferite teme din surse externe.

Librăria Swing este independentă de platformă, fiind scirsa în totalitate în limbajul Java. Ea a fost dezvoltată astfel încât să fie cât mai modulară, permițând utlizatorilor să își creeze propriile componente de interfață (propriul tip de buton spre exemplu). Având în vedere că Java este un limbaj orientat pe obiecte, componentele librăriei Swing pot fi extinse pentru a crea variante diferite dar care au totuși la bază obiectele de Swing.[WIKISWING17]

Componentele librăriei Swing:

- Eticheta – JLabel
- Butoane – JButton, JCheckBox, JRadioButton
- Componente pentru afisarea progresului – JSlider, JProgressBar, JScrollBar
- Separator – JSeparator
- Containere – JFrame, JDialog, JPanel, JScrollPane, JTabbedPane
- Text – JTextField, JPasswordField, JFormattedTextField, JTextArea
- Tabele şi liste – JTable, JList
- Casete de selectare – JComboBox

Fiecare componentă a interfeței Swing permite utilizarea unor evenimente. Fiecărui eveniment i se poate atribui o acțiune sau metodă care să fie executată la declanșarea sa (precum funcțiile de tip callback ale librăriei TaskScheduler din aplicația Arduino). Aceste porțiuni de program sunt numite handler-e.

6.3.2 AWT Event

AWT este o librărie standard a limbajului Java din care a derivat librăria Swing. AWT Event este un pachet de componente ale librăriei care se ocupă de diferite evenimente. Evenimentul reprezintă o modificare a stării unui obiect și descrie această schimbare. Exemple de evenimente ale limbajului Java pot fi apăsarea unui buton, mișcarea cursorului, tastarea unui caracter sau selectarea unui element dintr-o lista. Evenimentele pot fi catalogate astfel:

- Evenimente în prim plan (Foreground events) – Aceste evenimente necesită o interacțiune directă a utilizatorului cu aplicația. Ele sunt generate ca o consecință a interacțiunii utilizatorului cu aplicația. Apăsarea butoanelor, tastarea de caractere, selectarea și mișcarea cursorului sunt exemple de evenimente în prim plan.

- Evenimente în fundal (Background events) – Evenimentele care necesită acțiunea utilizatorului pentru a lua decizii în continuare. Evenimentele în fundal sunt evenimente care nu au fost declanșate de utilizator. Ele pot fi declanșate de întreruperi externe, timer-e care au expirat sau completarea unei sarcini.

Tratarea evenimentelor (Event Handling) reprezintă mecanismul care controlează și decide ce urmează să se întâmple la declanșarea unui eveniment. Aceste mecanisme conțin porțiunea de cod care va fi executată în cazul declansării evenimentelor. Avantajul acestei aboradari este acela că logica interfeței este complet separată de logica care declanșează evenimentul. Tratarea evenimentelor are în federe doi factori:

- Sursa – Obiectul care a declanșat evenimentul. Sursa este responsabilă de a pune la dispoziție toate informațiile necesare care au declanșat evenimentul.
- Auditorul – Cunoscut și ca event handler. Auditorul este responsabil de generarea unui răspuns pentru evenimentul declanșat.

Principiile tratării evenimentelor vor fi folosite pentru implementarea interfeței cu utilizatorului atât grafic prin butoane și casete de text cât și prin tastarea unor caractere.[TUTAWT17]

6.3.3 Java.io și BufferedImage

Cele două librării vor fi folosite pentru exportarea unei imagini generate în urma achiziției datelor de la robot. Pachetul Java.io este inclus în pachetul standard al limbajului Java și conține toate elementele necesare pentru efectuarea de operații intrări/ieșiri. BufferedImage este o subclasă a clasei Image din librăria AWT care permite generarea unei imagini. Această imagine va fi apoi salvată pe disc folosind librăria java.io.

6.3.4 Timer şi TimerTask

Clasele Timer şi TimerTask fac parte din pachetul standard al limbajului Java şi sunt folosite pentru a programa executarea unei sarcini la un anumit moment de timp. Java Timer poate fi folosit pentru a programa executarea sarcinilor o singură data sau de mai multe ori la intervale regulate. [JOUAWT17]

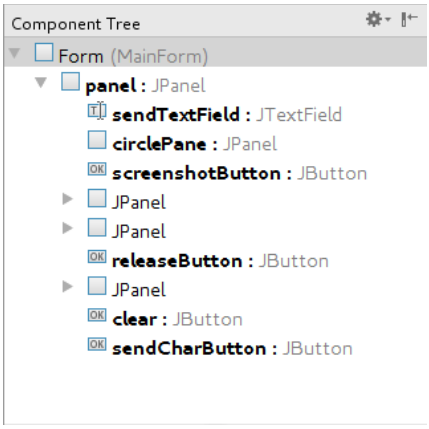
Cele două clase vor fi folosite pentru executarea unor instrucţiuni sau contorul timpului, necesare spre exemplu la calculul vitezei de transmisie a datelor între robot şi aplicaţie.

6.4 Interfaţa grafica

Interfeţele grafice ale aplicaţiilor Java sunt în mod clasic implementate prin cod. Această metodă este anevoioasă şi necesită testare intensivă până se ajunge la rezultatele dorite. Implementarea interfeţei grafice prin codul sursă devine din ce în ce mai complicată cu cât aplicaţiile devin mai complexe. Dezvoltatorii de IDE-uri au sesizat că acest lucru scade mult productivitatea programatorilor şi mai nou le pun la dispoziţie metode grafice de a implementa interfaţa grafica a programului.

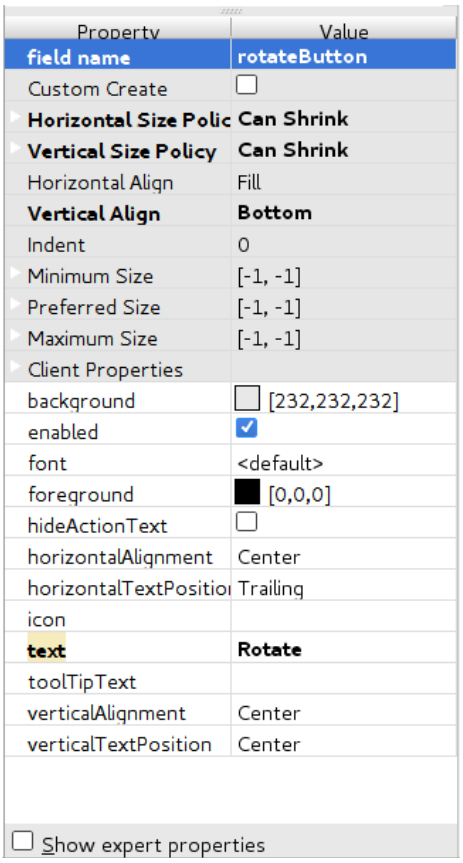
IntelliJ IDEA are implementată o astfel de funcţionalitate, fiind considerată cea mai stabilă şi performantă comparativ cu celelalte medii de dezvoltare în momentul de faţă. Fereastra dedicată implementării interfeţei grafice din cadrul mediului de dezvoltare IDEA este împărţită în 4 zone:

- Arborele de componente
- Proprietatile componentei
- Paleta de componente
- Vizualizarea interfetei



1. Figura: Arbore

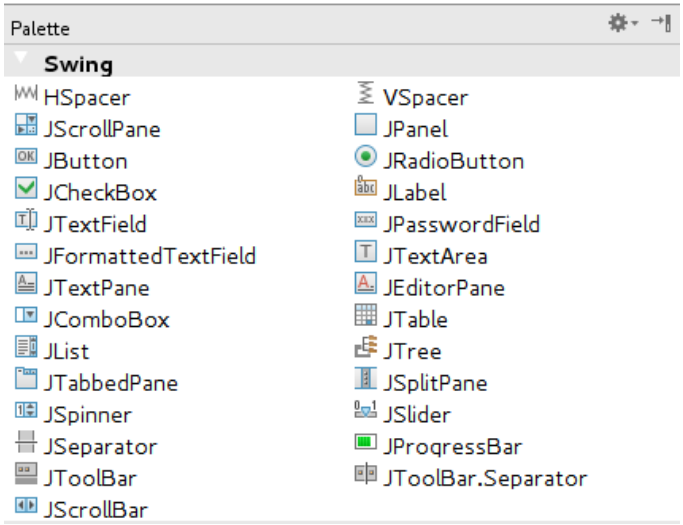
Arborele de componente afișează componentele care fac parte din interfața grafica. Componentele sunt aranjate sub formă de arbore respectând legăturile pe care le au între ele. Fiecărei componentă i se atribuie un nume și un tip.



2. Figura: Proprietati

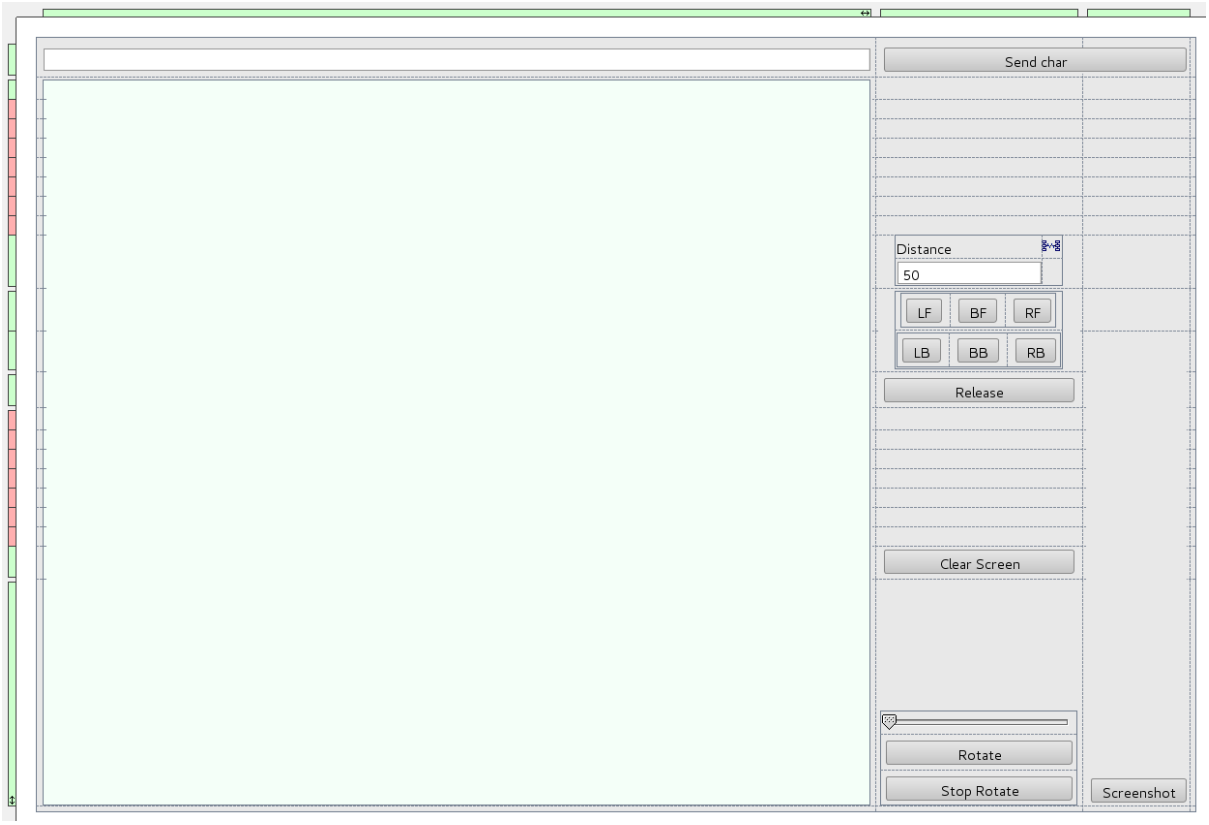
Fereastra de proprietăti este implicit goală iar la selectarea unei componente ea este populată cu proprietățile pe care componenta respectivă la are. Proprietățile componentelor Swing pot fi

comune mai multor tipuri de componente (de exemplu culoarea fundalului) sau specifice tipului respectiv de componentă (de exemplu valoarea maximă pe care o poate avea un slider).



3. Figura: Paleta

Paleta de componente conține diferite elemente ale librăriei Swing care pot fi incluse în interfața grafica. Ele vor fi selectate și incluse în interfață prin drag-and-drop. Paleta de componente poate include și elemente care nu fac parte din librăria standard Swing, de exemplu elemente ce fac parte din librării terțe, dar acestea trebuie mai întâi incluse în lista de librării a proiectului.



4. Figura: Interfata grafica

Fereastra centrală de dezvoltare și vizualizare a interfeței grafice este locul unde programator alege cum să aranjeze componentele interfeței. Interfața grafica trebuie să aibă ca rădăcina a arborelui o componentă de tip JPanel care va conține restul de componente grafice. Una din proprietățile JPanel este Layout Manager, adică modul în care acesta va fi impărtită și modul în care elementele pe care le conține vor fi aliniate între ele.

6.5 Software

Datorită complexității și respectării principiilor programării orientate pe obiecte a limbajului Java, aplicația este formată din mai multe componente separate:

- MainForm – Clasa responsabilă de interfața grafica și legătura cu utilizatorul unde sunt implementate evenimentele butoanelor și ale tastaturii
- CircleScreen – Clasa care extinde clasa Jpanel din librăria Swing, responsabilă cu afișarea datelor primite de la robot într-un spațiu bidimensional
- PacketReceiveHandler – Clasa responsabilă cu decodarea și interpretarea datelor primite prin TCP

- Pt – Clasa ce definieste obiectul care reprezintă un punct pe graficul CircleScreen
- TCPClient – Clasa ce extinde clasa Thread unde este implementată conexiunea TCP cu microcontrollerul ESP8266 într-un fir de execuție separat, pentru a nu bloca interfața grafică

6.5.1 MainForm

```
public class MainForm extends JFrame implements KeyListener {

    private static MainForm instance;
    TCPClient client;
    CircleScreen circleScreen;

    private JTextField sendTextField;
    private JButton sendCharButton;
    private JPanel panel;
    private JPanel circlePane;
    private JButton screenshotButton;
    private JButton releaseButton;
    private JButton rotateButton;
    private JButton stopRotateButton;
    private JButton LFButton;
    private JButton RFButton;
    private JButton BFButton;
    private JButton LBButton;
    private JButton RBButton;
    private JButton BBButton;
    private JTextField distanceTextField;
    private JButton clear;
    public JSlider calibrateSlider;

    private int screenshotCnt = 0;
```

Porțiunea de cod reprezintă definirea clasei MainForm, care este o extensie a clasei JFrame și implementeza interfața KeyListener, astfel clasa MainForm devine clasă care reprezintă interfața grafică dar și cea care primește evenimente prin funcții de tip callback la apăsarea tastelor. Tot aici sunt definite variabilele locale și globale ale clasei precum butoane, casete de text, contoare și obiectele care se ocupă de conexiunea cu microcontrollerul și afișarea datelor.

```
public static MainForm getInstance() {
    if (instance == null) {
        instance = new MainForm();
    }
    return instance;
}

public static void main(String[] args) throws IOException {
    MainForm.getInstance();
}
```

Fiecare aplicație Java trebuie să conțină metoda main. Pentru a putea avea acces la variabilele clasei MainForm am ales implementarea sa prin singleton. Singleton reprezintă o tehnică de instanțiere a claselor care asigură unicitatea lor, aici implementată în variabila de tip MainForm instance.

```
private MainForm() {
    setContentPane(panel);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize( width: 1200,  height: 800);
    setVisible(true);

    circlePane.setLayout(new BoxLayout(circlePane, BoxLayout.PAGE_AXIS));
    circleScreen = new CircleScreen();
    circlePane.add(circleScreen);

    client = new TCPClient( address: "192.168.4.1",  port: 1212);
    client.startClient();

    this.addWindowListener((WindowAdapter) windowClosing(e) → {
            client.sendCommand(TCPClient.RELEASE_MOTORS);
            System.exit( status: 0);
    });
```

Constructorul clasei MainForm este locul unde este instantiata interfața grafica prin funcția setContentPane, moştenită de la JFrame. Tot aici este instantiata clasa CircleScreen şi este direcționată către panoul circlePane, este instantiata şi executată secvența de pornire a clasei TCPClient care controlează traficul TCP cu microcontrollerul şi este implementată o comandă de oprire a motoarelor la declanşarea evenimentului windowClosing, adică la oprirea programului.

```
sendTextField.addKeyListener( k: this);

RFButton.addActionListener(
        e -> client.sendTravelDistance(TCPClient.WALK_FORWARD_RIGHT, distanceTextField.getText()));
LFButton.addActionListener(
        e -> client.sendTravelDistance(TCPClient.WALK_FORWARD_LEFT, distanceTextField.getText()));
BFButton.addActionListener(
        e -> client.sendTravelDistance(TCPClient.WALK_FORWARD_BOTH, distanceTextField.getText()));

RBButton.addActionListener(
        e -> client.sendTravelDistance(TCPClient.WALK_BACKWARD_RIGHT, distanceTextField.getText()));
LBButton.addActionListener(
        e -> client.sendTravelDistance(TCPClient.WALK_BACKWARD_LEFT, distanceTextField.getText()));
BBButton.addActionListener(
        e -> client.sendTravelDistance(TCPClient.WALK_BACKWARD_BOTH, distanceTextField.getText()));

sendCharButton.addActionListener(e -> client.sendString(sendTextField.getText()));
releaseButton.addActionListener(e -> client.sendCommand(TCPClient.RELEASE_MOTORS));
rotateButton.addActionListener(e -> {client.sendCommand(TCPClient.ROTATE);
                                     circleScreen.points.clear();});
clearButton.addActionListener(e -> circleScreen.points.clear());

calibrateSlider.addChangeListener(
        e -> circleScreen.calibrateValue = calibrateSlider.getValue()/10000.00f);

stopRotateButton.addActionListener(
        e -> client.sendCommand(TCPClient.STOP_ROTATE));
screenshotButton.addActionListener(
        e -> {
    BufferedImage bi = new BufferedImage(circlePane.getSize().width,
            circlePane.getSize().height, BufferedImage.TYPE_INT_ARGB);
    Graphics g = bi.createGraphics();
    circlePane.paint(g);  //this == JComponent
    g.dispose();
    try {
        ImageIO.write(bi,  formatName: "png", new File( pathname: "screenshot" + screenshotCnt + ".png"));
        screenshotCnt++;
    } catch (Exception excep) {
    }
});
```

În constructorul MainForm sunt implementate şi evenimentele butoanelor şi acțiunile acestora. Pentru implementarea acțiunii la eveniment am ales expresii lamda pentru a produce un cod mai scurt şi mai eficient. Expresiile lamda au fost adăugate în limbajul Java în versiunea 8 şi au reprezentat cea mai importantă schimbare a respectivei versiuni. Ele facilitateaza tehnicile de programare funcțională, uşurează dezvoltarea de aplicații şi scad timpul necesar programării.

```
@Override
public void keyPressed(KeyEvent e) {
    switch(e.getKeyCode())
    {
        case KeyEvent.VK_UP:
            client.sendCommand(TCPClient.STEP_FORWARD);
            break;
        case KeyEvent.VK_DOWN:
            client.sendCommand(TCPClient.STEP_BACKWARD);
            break;
        case KeyEvent.VK_RIGHT:
            client.sendCommand(TCPClient.STEP_RIGHT);
            break;
        case KeyEvent.VK_LEFT:
            client.sendCommand(TCPClient.STEP_LEFT);
            break;
    }
}

@Override
public void keyReleased(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_LEFT || e.getKeyCode() == KeyEvent.VK_RIGHT ||
            e.getKeyCode() == KeyEvent.VK_UP || e.getKeyCode() == KeyEvent.VK_DOWN)
        client.sendCommand(TCPClient.RELEASE_MOTORS);
}


@Override
public void keyTyped(KeyEvent e) {

}
```

Această porțiune de cod reprezintă implementarea metodelor primite de la interfața KeyListener, de unde și anotarile @Override. Implementarea interfeței KeyListener ne obligă să definim metodele keyPressed, keyReleased și keyTyped, metode de tip callback care vor fi apelate la apăsarea și eliberarea unei taste sau doar la apăsarea acesteia. În metoda keyPressed am implementat controlul robotului prin tastele de direcție, pentru fiecare din ele transmițându-se o comandă diferită robotului. La eliberarea tastelor, adică în metoda keyReleased, în caz că a fost eliberată una din tastele folosite la controlul robotului, se transmite comanda de oprire a motoarelor.

6.5.2 CircleScreen

```
public class CircleScreen extends JPanel {
    private static final float RPS = .5f;
    private static final float STEP_VALUE = 1/RPS * 1000.0f/35.0f;
    private static final int POINT_SIZE = 3;

    List<Pt> points = new ArrayList<>();

    int forwarddelta;

    public float calibrateValue = 0;

    public CircleScreen() {
        super( isDoubleBuffered: true);
        this.setPreferredSize(new Dimension(getWidth(), getWidth()));
    }
```

Clasa CircleScreen este definită ca moștenitor al clasei JPanel din librăria Swing și va reprezenta zona de vizualizare a datelor primite de la senzor. Ea conține mai multe variabile private ce vor fi folosite la calculul poziției punctelor cât și o listă de elemente Pt ce reprezintă mulțimea de puncte ce vor fi afișate.

```
@Override
protected void paintComponent(Graphics g) {
    List<Pt> pointsclone = new ArrayList<>(points);
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;
    g2d.setRenderingHint(
            RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);

    int a = getWidth() / 2;
    int b = getHeight() / 2;

        Iterator<Pt> pointsIterator = pointsclone.iterator();
        int currentPoint = 0;
        while (pointsIterator.hasNext()) {
            Pt p = pointsIterator.next();
                double t = 2 * Math.PI * currentPoint / STEP_VALUE + calibrateValue;

                if(p.x == 0 && p.y == 0) {
                    p.x = (int) Math.round(a + p.distance * Math.cos(t));
                    p.y = (int) Math.round(b + p.distance * Math.sin(t)) - forwarddelta;
                }
                Color tmpcolor = new Color(p.color.getRed(), p.color.getGreen(),
                        p.color.getBlue(), p.alpha < 0? 0:p.alpha);
                p.alpha -= 50;
                g2d.setColor(tmpcolor);
                g2d.fillOval(p.x, p.y, POINT_SIZE, POINT_SIZE);
                currentPoint++;
        }
    }
}
```

Metoda paintComponent reprezintă o suprascriere a metodei moștenită de la clasa JPanel. Codul ei va fi executat de fiecare dată când se execută metoda paintComponent a clasei moștenite și devine astfel o „extensie” a funcționalității acesteia. Metoda paintComponent va fi executată de fiecare dată când JVM (mașina virtuală Java) decide că interfața trebuie actualizată sau atunci când este apelată metoda updateUI, metodă care forțează actualizarea interfeței grafice.

În cadrul metodei vor fi desenate punctele ce reprezintă valorile primite de la senzorul robotului. Pentru a asigura rularea pe mai multe fire de execuție fără probleme, deoarece în limbajul Java nu este recomandat ca actualizarea interfeței grafice să folosească variabile care sunt folosite și pe alte fire de execuție, mai întâi se crează o copie a listei de puncte. Copia listei de puncte, numită pointsclone, va fi parcursă iar pentru fiecare element al sau se va calcula poziția corespunzătoare în cadrul panoului CircleScreen. Unul din parametrii obiectelor Pt este alpha și reprezintă transparența punctului respectiv. Transparența punctelor este scăzută la fiecare interatie, astfel punctele vechi dispar odată cu apariția celor noi. Tot aici se poate roti întreaga reprezentare a punctelor deoarece motorul pas cu pas nu ne oferă o poziție inițială fixă, deci nu putem garanta că orientarea punctelor este aceeași cu orientarea robotului. Orientarea se va face folosind cursorul calibrateSlider care este reprezentat în program prin variabila calibrateValue.

6.5.3 Pt

```
public class Pt {
    public int x = 0;
    public int y = 0;
    public int distance = 0;
    public Color color = Color.red;
    public int alpha = 255;

    public Pt(int _d) { distance = _d; }
}
```

Clasa Pt este instantiata în obiectele din cadrul CircleScreen şi reprezintă un punct de pe panoul de vizualizare. Ea nu conţine metode proprii înafară de constructor şi are 5 parametrii:

- x – coordonata în cadrul panoului CircleScreen
- y – coordonata în cadrul panoului CircleScreen
- distance – distanţa faţă de centrul cercului care reprezintă poziţia robotului. Variabila distance este mărimea măsurată primită de la robot
- color – culoarea punctului, predefinita la culoarea roşie
- alpha – transparenta culorii punctului, predefinita la valoarea 255, adică 0% transparenta

Constructorul clasei *Pt* primeşte parametrul *_d* care este copiat apoi în variabila *distance.*

6.5.4 TCPClient

```
public class TCPClient extends Thread {

    public static final int STEP_FORWARD = 3;
    public static final int STEP_BACKWARD = 4;
    public static final int STEP_LEFT = 5;
    public static final int STEP_RIGHT = 6;
    public static final int ROTATE = 1;
    public static final int STOP_ROTATE = 2;
    public static final int RELEASE_MOTORS = 7;

    public static final int WALK_FORWARD_RIGHT = 8;
    public static final int WALK_BACKWARD_RIGHT = 9;
    public static final int WALK_FORWARD_LEFT = 10;
    public static final int WALK_BACKWARD_LEFT = 11;
    public static final int WALK_FORWARD_BOTH = 12;
    public static final int WALK_BACKWARD_BOTH = 13;
```

Clasa TCPClient este cea care se ocupă de transferul de date între aplicaţie şi robot iar pentru a asigura acest transfer fără a bloca interfaţa, TCPClient este un moştenitor al clasei Thread, adică va rula pe un fir separat de execuţie.

Secvenţa de program de mai sus definieste constantele ce semnifica diferitele comenzi pe care le poate primi robotul precum păşire a fiecărui motor în parte, rulare a unei anumite distanţe pentru fiecare motor în parte, rotirea motorului pe care este montat senzorul sau oprirea motoarelor.

```
Socket server;
String line = "";
DataOutputStream dOut;
private int port;
private String address;
private boolean running;
private int bps;

public TCPClient(String address, int port) {
    this.port = port;
    this.address = address;
}
```

Clasa TCPClient are o serie de variabile private definite precum obiectul server, care reprezintă dispozitivul la care se conectează, în cazul nostru serverul TCP creat de către ESP8266, portul și adresa IP a acestui server și alte contoare și flag-uri.

Clasa este instantiata de către clasa MainForm la pornirea aplicației iar constructorul sau primește ca parametrii portul și adresa pe care le copiază în variabilele locale.

```
public void startClient() {

    try {
        server = new Socket(address, port);
        server.setSoTimeout(10000);
        dOut = new DataOutputStream(server.getOutputStream());
        this.start();

        Timer t = new Timer();
        t.schedule(new TimerTask() {
            @Override
            public void run() {
                //System.out.println(bps + "packs per sec");
                bps = 0;
            }
        }, delay: 1000, period: 1000);

        System.out.println("startClient");
    } catch (UnknownHostException e) {
        System.out.println(e);
    } catch (IOException e) {
        System.out.println(e);
    }
}
```

Metoda startClient este apelată în MainForm imediat după instantierea clasei TCPClient. În cadrul acestei metode este instantiat obiectele server care reprezintă conexiunea cu robotul, dOut care reprezintă fluxul de date transmise către robot dar este apelată și metoda start moștenită de la clasă Thread. Tot aici este definit, instantiat și executat un timer care va rula o dată pe secundă și va reseta contorul de biți pe secundă.

```
@Override
public void run() {

    running = true;
    while (running) {
        try {
            BufferedReader inFromServer = new BufferedReader(
                    new InputStreamReader(server.getInputStream()));
            while ((line = inFromServer.readLine()) != null) {
                PacketReceiveHandler.handle(line);
                bps++;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Metoda run a clasei TCPClient este moştenită din clasa Thread şi reprezintă porţiunea de cod care urmează să fie executată pe un fir separat de execuţie. În cadrul acestei metode se va executa o buclă infinită unde se va încerca mereu citirea de noi octeţi de date de la robot. Această metoda de abordare a transmisiei de date se numeşte pooling şi este recomandată doar în cazul în care poate fi implementată pe un fir separat de execuţie. Dacă această implementare nu ar fi făcută pe un fir separat de execuţie moştenind clasa Thread, în momentul în care execuţia ar ajunge la bucla infinită while(running) unde running este mereu adevărat, aplicaţia s-ar bloca complet. Când bucla infinită a primit un şir de caractere de la robot, îl trimite clasei PacketReceiveHandler pentru interpretare prin metoda handle a acesteia.

```
public void sendCommand(Object c) {
    JSONObject obj = new JSONObject();
    obj.put("c", c);
    try {
        dOut.write(obj.toString().getBytes());
        dOut.write( b: '#');
    } catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println("sent " + obj.toString());
}
```

Metoda sendCommand din candrul clasei TCPClient se ocupă de codarea şi transmisia unei comenzi fără parametru către robot. Metoda primeşte ca parametru un obiect (fie el număr întreg, octet sau şir de caractere) pe care îl codează în format JSON iar apoi îl transmite către robot folosind obiectul dOut care reprezintă fluxul de date de transmis. La sfârşitul transmisiei este adăugat caracterul # care în implementarea software a robotului semnalizează sfârşitul recepţiei.

```
public void sendTravelDistance(Object c, Object d){
    JSONObject obj = new JSONObject();
    obj.put("c", c);
    obj.put("v",d);
    try {
        dOut.write(obj.toString().getBytes());
        dOut.write( b: '#');
    } catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println("sent " + obj.toString());
}
```

Metoda sendTravelDistance este metoda din cadrul clasei TCPClient care se ocupă cu transmisia unei comenzi cu parametru. Ea primește doi parametrii de orice tip, unul semnificând comanda iar al doilea parametrul comenzii. Metoda codează în format JSON datele ce urmează să fie transmise iar apoi le trimite către obiectul dOut pentru transmisie. La sfârșit se transmite caracterul # pentru a marca sfârșitul transmisiei.

```
public void sendString(String s) {
    try {
        byte[] data = s.getBytes( charsetName: "UTF-8");
        dOut.write(data);
        dOut.write( b: '#');
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Metoda sendString primește ca parametru un șir de date și îl transmite către robot fără a modifica conținutul acestuia. Această metodă a fost folosită la depanarea transmisiei de date, codarea JSON fiind făcută manual, de către utilizator.

6.5.5 PacketReceiveHandler

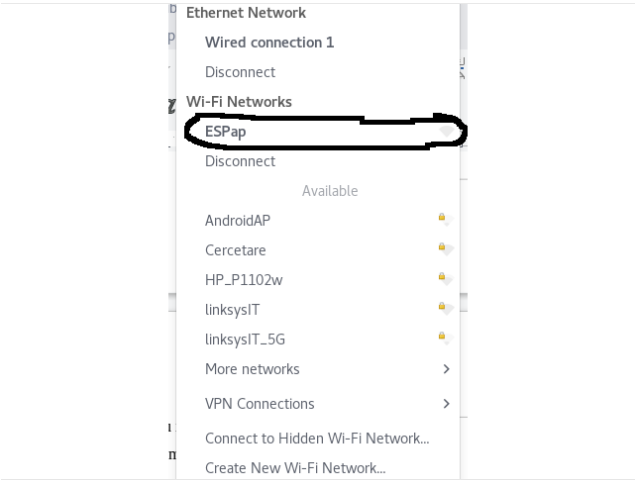
```
public class PacketReceiveHandler {

    static int lastRight;
    static int lastLeft;
    public static void handle(String line) {
        System.out.println("Received line: " + line);
        JSONObject jsonObject = new JSONObject(line);

        JSONArray sensorJson = jsonObject.getJSONArray( key: "s");
        int rightValue =  jsonObject.getInt( key: "r");
        int leftValue = jsonObject.getInt( key: "l");


        int right = rightValue * 22 / 96;
        int left = leftValue * 22 / 96;

        System.out.println(right  +  "    " + left);

        for (int i = 0; i < sensorJson.length(); ++i) {
            int sensor = sensorJson.getInt(i);
            sensor = sensor/5;
            Pt point = new Pt(sensor);
            MainForm.getInstance().circleScreen.points.add(point);
        }

        MainForm.getInstance().circleScreen.updateUI();

        System.out.println("Right : " + rightValue + "   Left : " + leftValue);
    }
}
```

Clasa PacketReceiveHandler este responsabilă cu interpretarea datelor primite de la clasa TCPClient. Ea nu este instantiata, deoarece are o singură metodă statică, şi anume metoda handle.

Handle primeşte un parametru şir de caractere pe care îl afişează pentru depanare apoi decodează şirul în format JSON. La sfârşitul decodarii rezultă 3 variabile: un şir de valori ale senzorului, numărul de paşi făcut de motorul drept şi numărul de paşi făcut de motorul stâng. Valorile senzorului sunt folosite ca parametru la generarea de obiecte de tip Pt care sunt apoi transmise către clasa CircleScreen pentru afişare printr-o buclă for. Variabilele care reprezintă numărul de paşi ale motoarelor sunt mai întâi convertite din paşi în centimetri iar apoi sunt afişate pentru depanare. La sfârşitul fiecărei interpretări a datelor este apelată metoda updateUI pentru a afişa datele respective în interfaţa grafica.

6.6 Mod de funcţionare

La alimentarea robotului acesta va crea un punct de acces WiFi cu numele ESPap la care ne putem conecta. Este necesară conectarea la punctul de acces înaintea pornirii aplicaţiei.



5. Figura: Conectare

Odată ce conectarea a fost efectuată şi este pornită aplicaţia, această va crea automat conexiunea cu serverul robotului şi va începe să preia date, să le afişeze pe consolă de depanare şi în panoul de vizualizare.

```
/usr/lib/jvm/java-8-jdk/bin/java ...
startClient
onStart
Received line: {"s":[436,439,438,441,438,442,437,439,442,433,439,439,438,440,439,439,438,439,437,437,437,441,437,437,444,436,435,440],"r":0,"l":0}
0    0
Right : 0   Left : 0
```

6. Figura: Consola

La pornire robotul are toare motoarele oprite şi este necesară acţionarea butonului Rotate pentru a ca motorul pe axul căruia este montat senzorul să se rotească. În acelaşi timp pe panoul de vizualizare se va genera harta bidimensională a punctelor.

7. Figura: Butoane

8. Figura: Harta

Orientarea hărții generate este posibil să nu fie aceeaşi cu orientarea robotului, acest lucru este datorat faptului că nu putem cunoaşte poziția motorului pas cu pas la pornire. Pentru alinierea manuală a orientării hărții se poate folosi slider-ul poziționat deasupra butonului Rotate.

În orice moment al generării hărții se poate apăsa butonul Screenshot care va salva o imagine cu extensia png şi numele în formatul screenshot[număr] în folderul curent al aplicației.

Pentru mişcarea robotului se pot folosi tastele de poziționare care vor declanşa evenimentele din cadrul clasei MainForm. Aceste evenimente vor muta robotul în următorul mod:

- la apăsarea tastei înainte, ambele motoare vor executa 10 paşi înainte
- la apăsarea tastei înapoi, ambele motoare vor executa 10 paşi înapoi
- la apăsarea tasei stânga, motorul stâng va executa un pas înapoi iar motorul drept va executa un pas înainte
- la apăsarea tastei dreapta, motorul drept va executa un pas înapoi iar motorul stâng va executa un pas înainte

Tot pentru mişcarea robotului putem folosi butoanele LF, BF, RF, LB, BB, RB. Aceste butoane apelează metoda de comandă cu parametru a robotului. Parametrul comenzii reprezintă numărul de paşi şi se va introduce în caseta de text Distance. Butoanele acționează în următorul mod:

- LF – left forward – acţionează motorul stâng înainte
- BF – both forward – acţionează ambele motoare înainte
- RF – right forward – acţionează motorul drept înainte
- LB – left backward – acţionează motorul stâng înapoi
- BB – both backward – acţionează ambele motoare înapoi
- RB – right backward – acţionează motorul drept înapoi

Pentru că atunci când motoarele ajung la destinație ele continuă să fie alimentate în poziția respectivă de către driver, butonul Release opreşte alimentarea lor complet pentru a evita supraîncălzirea.

Deşi punctele afişate pe hartă îşi scad transparența, butonul Clear screen goleşte lista de puncte din cadrul clasei CircleScreen şi implicit şterge punctele din panoul de vizualizare.

7 CONCLUZII

Am dorit prin proiectul prezentat sa pun accentul pe importanţa componentelor software ale unei aplicații folosind hardware simplu și accesibil. Consider ca am reușit deși mereu rămâne loc pentru imbunatatiri.

Pe parcursul proiectului am intampinat diferite probleme, fie ele de proiectare, electrice sau de programare, mare parte din ele datorate lipsei de experienţa. Consider ca a fost de mare ajutor suportul primit din partea facultatii și al coodronatorului stiintific.

8 BIBLIOGRAFIE

[ESP17] – ESP8266EX Datasheet
[VL16] – VL53L0X Datasheet
[VLUSER16] – VL53L0X User Manual
[MPP14] – Laurean Bogdan, MPP Construcție Funcționare

9 REFERINȚE WEB

[WIKIESP17] - https://en.wikipedia.org/wiki/ESP8266
[SPKI2C17] - https://learn.sparkfun.com/tutorials/i2c
[WIKISPI17] - https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
[SPKSPI17] - https://learn.sparkfun.com/tutorials/serial-communication/uarts
[STVL17] - http://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html
[ADA17] - https://cdn-learn.adafruit.com/downloads/pdf/adafruit-motor-shield-v2-for-arduino.pdf
[WIKISTEP17] - https://en.wikipedia.org/wiki/Stepper_motor
[WIKIARD17] - https://en.wikipedia.org/wiki/Arduino
[GITARD17] - https://github.com/esp8266/Arduino
[GITADA17] - https://github.com/adafruit/Adafruit_Motor_Shield_V2_Library
[ARD17] - https://www.arduino.cc/en/Reference/HomePage
[ACCL17] - http://www.airspayce.com/mikem/arduino/AccelStepper/
[GITVL17] - https://github.com/pololu/vl53l0x-arduino
[GITJSON17] - https://github.com/bblanchon/ArduinoJson
[GITTASK17] - https://github.com/arkhipenko/TaskScheduler
[TASK17] - http://www.smart4smart.com/TaskScheduler.pdf
[WIKIJAVA17] - https://en.wikipedia.org/wiki/Java_(programming_language)
[WIKIIDEA17] - https://en.wikipedia.org/wiki/IntelliJ_IDEA
[WIKISWING17] - https://en.wikipedia.org/wiki/Swing_(Java)
[TUTAWT17] - https://www.tutorialspoint.com/awt/awt_event_handling.htm
[JOUAWT17] - http://www.journaldev.com/1050/java-timer-timertask-example

10 CODUL SURSĂ

Codul sursa al ambelor aplicații este prezent pe CD-ul atașat proiectului.

11 CD / DVD

Autorul atașează în această anexă obligatorie, versiunea electronică a aplicației, a acestei lucrări, precum și prezentarea finală a tezei.