

UNIVERSITATEA POLITEHNICA BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL INGINERIA SISTEMELOR



PROIECT DE DIPLOMĂ

Modelarea și automatizarea unei platforme mobile alimentată pe baza
unei sistem DAST și controlată printr-o aplicație IoT

Coordonator științific:

Sl. dr. ing. Radu Pietraru

Absolvent:

Radu – Cristian Pelin

BUCUREŞTI

2019

Rezumat

Automatizarea industrială este un proces evolutiv de mare intensitate, ce a cunoscut o dezvoltare uluitoare în ultimul deceniu, odată cu apariția serviciilor Internet of Things și celor de Cloud. Dacă adăugăm și conceptul de Machine Learning, împreună cu cele anterior menționate, putem observa că există un interes și o dezvoltare tot mai mare a sistemelor autonome, atât din punct de vedere energetic cât și al executării unor activități sau luarea unor decizii.

Sistemele cyber fizice, serviciile IoT și de Cloud, algoritmii complecși implementați pe mașinile de calcul și integrarea numeroaselor tipuri de senzori, sporesc autonomia proceselor și mașinilor industriale. Pe măsură ce performanța tehnologiei se îmbunătățește datele și rezultatele obținute de la senzori precum și pe baza algoritmilor, se stochează pe Cloud de unde putem analiza rezultate specifice în timp real. De asemenea, o astfel de abordare ne permite monitorizarea eficienței și robusteții sistemelor create.

Roboții industriali joacă și ei un rol foarte important în industrie și există un interes deosebit în realizarea de aplicații inteligente integrate în aceste sisteme, cu scopul efectuării de sarcini multiple și cooperative după caz, la un nivel de complexitate deosebit de ridicat.

Proiectul își propune dezvoltarea unui prototip pentru un sistem mobil autonom din punct de vedere energetic, ce poate susține deplasarea unor astfel de roboți industriali ce execută sarcinii în câmp deschis și în continuă mișcare. O astfel de platformă atașabilă își găsește cu preponderență utilitatea în activități agricole.

Sistemul vine împreună cu un mecanism de urmărire solară pe două axe, proiectul urmărind și o analiză a eficienței unei astfel de metode de captare a energiei solare în timp real. La toate acestea se adaugă și existența unei aplicații Android de control manual și automatizat al platformei, bazat pe servicii de Cloud și IoT. Atât platforma cât și aplicația de control poartă numele de SmartPlatform și ne vom referi deseoară la acestea folosind acest nume.

Platforma prezintă și senzori de proximitate utili în evitarea de obstacole. Aplicația SmartPlatform permite controlul de la distanță de pe propriul dispozitiv Android, smartphone sau tableta, în zone cu conexiune la internet. SmartPlatform oferă utilizatorului atât posibilitatea de control manual cât și automatizat, existând opțiunea de creare a unor cicluri de deplasare repetitivă pe trasee stabilite de către utilizator.

Cuprins

Capitolul 1	4
Introducere în urmărirea solară	
1.1 Sisteme de urmărire solară	6
1.2 Soluția de urmărire solară aleasă.....	8
Capitolul 2	15
Proiectarea și implementarea sistemului de urmărire solară ales	
2.1 Modelarea fizică a sistemului.....	15
2.2 Alegerea mecanismului de urmărire.....	26
2.2.1 Alegerea componentelor hardware.....	26
2.2.2 Integrare	28
2.3 Rezultat.....	32
Capitolul 3	33
Proiectarea și implementarea platformei controlată la distanță	
3.1 Alegerea componentelor hardware.....	35
3.2 Integrare	37
3.3 Alegerea serviciului BaaS	41
3.4 Dezvoltarea mecanismelor de control tip IoT compatibil cu serviciul BaaS	42
Capitolul 4	63
Dezvoltarea aplicației de control	
4.1 Procesul de autentificare	64
4.2 Procesul de acces și conectare la platformă	69
4.3 Procesul de control	73
4.3.1 Control manual	73
4.3.2 Control automatizat	81
4.4 Alte funcționalități.....	90
Capitolul 5	92
Obținerea sistemului fizic final	
5.1 Alegerea circuitului de încărcare și a panoului solar	92
5.2 Sincronizarea cu sistemul DAST	95
5.3 Rezultat final	98

Capitolul 1

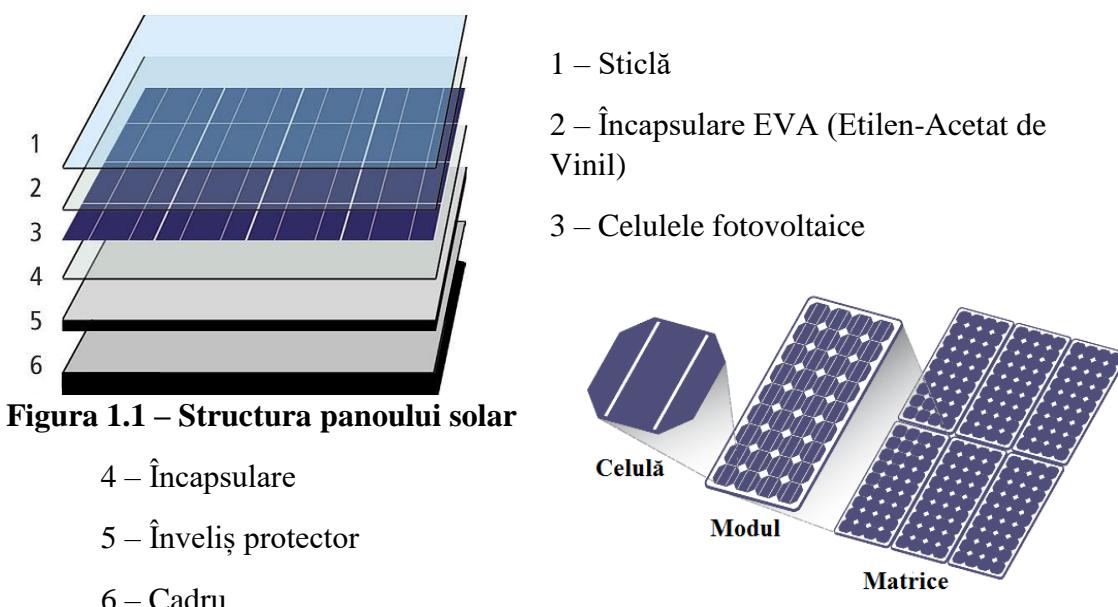
Introducere în urmărirea solară

Soarele funcționează ca un reactor nuclear natural, eliberând în spațiu energie sub formă unor mici particule numite fotoni. Modalitatea de conversie a acestor particule în energie electrică este cunoscută ca energia solară.

Energia solară este una din cele mai răspândite metode de utilizare a unei surse de energie regenerabilă. La început, această sursă nu avea randament pentru generarea de electricitate, fiind nevoie de panouri fotovoltaice scumpe, suprafețe mari de teren și o sursă constantă de lumină solară. “Ulterior, cercetătorii au dezvoltat sisteme eficiente și accesibile din punct de vedere al costurilor, iar în prezent, eficiența unei singure celule fotovoltaice este de 14%.” (Chowdhury et al., pg. 7, 2017)

Celulele fotovoltaice din care sunt compuse sistemele de captare a energiei solare nu poluează, nu produc zgomote, sunt ușor de fabricat și întreținut, însă există o piedică deoarece randamentul lor este încă scăzut comparativ cu costurile de producție.

Sistemele de captare a energiei solare sau așa numitele panouri solare, sunt componenta de baza a oricărui sistem fotovoltaic. Un panou solar este compus din mai multe celule fotovoltaice, aranjate într-un model asemănător unei grile, interconectate într-o structură plată. El este de asemenea descris ca fiind un set de module fotovoltaice. Un modul fotovoltaic este compus dintr-o matrice de celule interconectate de dimensiunea 6x10 sau 4x9. Panoul solar este construit cu o tehnologie ce permite captarea luminii de la soare și transformarea ei în energie electrică.



Instalarea panourilor solare previne emisia de noxe și ajută la reducerea încălzirii globale prin scăderea dependenței de combustibili fosili și surse tradiționale de energie.

Singura problemă a panourilor solare este că sunt scumpe în comparație cu randamentul lor, având în vedere că avem nevoie de acces la energie solară în permanență pentru o funcționare continuă și eficientă.

Celulele din care sunt realizate panourile sunt fabricate din silicon, care este un material semiconductor. Ele sunt construite dintr-un strat pozitiv și unul negativ ce permite crearea unui câmp electric.

ACESTE straturi au o deosebită importanță în ceea ce privește funcționarea și randamentul celulei. Cristalele din silicon nu sunt foarte bune din punct de vedere al conductivitatii electrice, dar la contactul cu alte materiale sau impurități se crează un curent electric între straturile N și P.

Primul strat este dopat cu bor și în contact cu siliconul formează un semiconductor de tip P ce permite încărcarea pozitivă, iar cel de al doilea strat este dopat cu fosfor și generează împreună cu siliconul un semiconductor de tip N ce permite încarcarea negativă. Spațiul dintre cele două straturi poartă numele de joncțiune P-N și prin aceasta circulă fluxul de curent electric. Acest spațiu este unul mic și de aceea o singură celulă nu poate genera suficientă energie electrică..

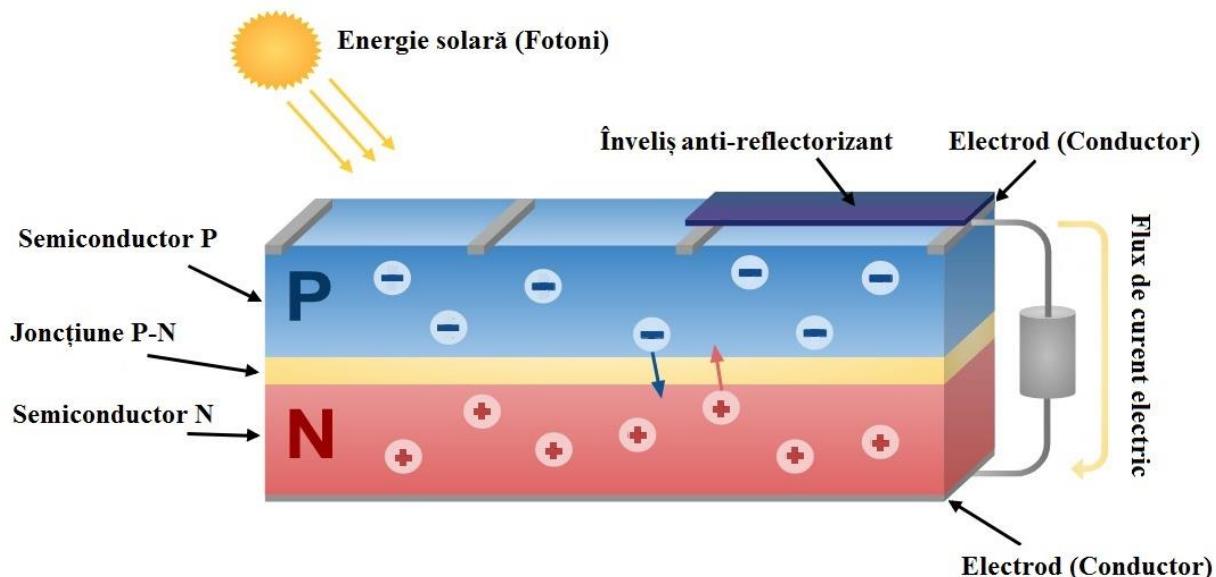


Figura 1.3 – Structura și comportamentul unei celule fotovoltaice

Deplasarea electronilor prin joncțiune generează un câmp electric ce permite ulterior electronilor să circule doar între stratul P și stratul N. Când lumina soarelui ajunge la suprafața celulei, fotonii lovesc electronii liberi din ambele straturi. Datorită încărcării diferite a celor două straturi, electronii tend să circule de la semiconductorul N la P. Deși câmpul electric generat de joncțiunea P-N previne acest lucru să se întâpte, prezența unui circuit extern asigură calea necesară pentru ca electronii de tip N să poată călători spre stratul de tip P și să rămână la suprafața joncțiunii.

Partea din spate încărcată pozitiv și partea din față încărcată negativ, pot fi conectate printr-un circuit pentru a putea extrage energie electrică și tensiune.

1.1 Sisteme de urmărire solară

Sistemele de urmărire solară sunt folosite pentru a crește cantitatea de energie produsă de panourile solare. Pe lângă panoul solar atașat, aceste sisteme conțin mecanisme sau dispozitive ce permit urmărirea poziției razelor solare sau a luminii față de panou și menținerea unei unghiuri drept între razele de lumină și acesta. Abaterea tot mai mică de la acest unghi ne oferă detalii despre calitatea sistemului de urmărire.

Cu alte cuvinte, urmărirea solară are ca scop modificarea orientării față de soare a panoului solar de-a lungul zilei cu scopul creșterii randamentului și pentru a genera mai multă putere. Modificarea se face pe baza unor motoare sau servomotoare. Aceste sisteme pot crește cu până la 40% producerea de energie în anumite regiuni, comparativ cu sistemele fixe, conform lucrării *Performance Comparison Between Fixed Panel, Single-axis and Dual-axis Sun Tracking Solar Panel System*.

Astfel de mecanisme se pot realiza ținând cont de “deplasarea” zilnică a soarelui de la est la vest, utilizând un sistem de urmărire pe o axă (orizontal). Dacă pentru acest sistem ținem cont de coordonatele geografice și de poziția Pământului față de Soare, putem obține un sistem de urmărire pe două axe (vertical și orizontal). La un anumit interval calculat cu precizie, modificăm axa verticală, iar axa orizontală rămâne să se modifice și ea conform unei ecuații ce indică poziția soarelui pe cer în funcție de ziua din an și locație.

De asemenea, există sisteme pe două axe active (Azimuth - Altitude DAST sau SmartDAST), ce își poartă acest nume datorită funcționalității diferite față de restul mecanismelor de urmărire. Azimut reprezintă unghiul pe care îl face direcția de deplasare a unui obiect în mișcare cu polul nord geografic. Un sistem SmartDAST permite urmărirea solară în timp real, fiind capabil să repoziționeze panoul solar când acesta se află în mișcare continuă sau periodică. În articolul *A simple and low-cost active dual-axis solar tracker*, pe baza cercetării s-a obținut o creștere de 36.26% a totalui de energie produsă. Datele au fost preluate timp de 6 minute, în condițiile în care panoul nu se afla în mișcare.

Dacă considerăm o mișcare a sistemului pe orizontală (modificarea poziției la fiecare X secunde), utilizând același servomotor ca în lucrarea menționată, putem vedea cât de eficient este sistemul în cazul în care îl folosim la capacitate maximă. În cele 6 minute, s-a obținut conform lucrării mai sus menționate, o putere totală de 314,27 W pentru sistemul SmartDAST și 230,17 W pentru sistemul fix. Ținând cont că la sarcină maximă, motorul MG996R consumă 5.4 W, pentru modificarea unghiului la X secunde obținem următorul rezultat:

X (s)	Putere Consumată (W)	Câștig %
60	32,4	19,69
55	35,35	18,36
50	38,88	16,81
45	43,2	14,96
40	48,6	12,73
35	55,54	10,00
30	64,8	6,54
25	77,76	2,06
20	97,2	-4,00

Tabel 1.1.1 – Eficiența sistemului DAST

Se observă că sistemele SmartDAST rămân eficiente în cazul unei modificări a poziției de deplasare, în medie, odată la aproximativ jumătate de minut. Dacă se dorește o urmărire continuă, consumul este mult prea mare pentru ca mecanismul de urmărire să poată fi autonom.

Sistemul de urmărire solară dezvoltat ulterior în această lucrare este de tip SmartDAST însă se vor folosi motoare cu consum mult mai scăzut, pentru panouri solare flexibile și ușoare.

A. Sistem cu axe fixe

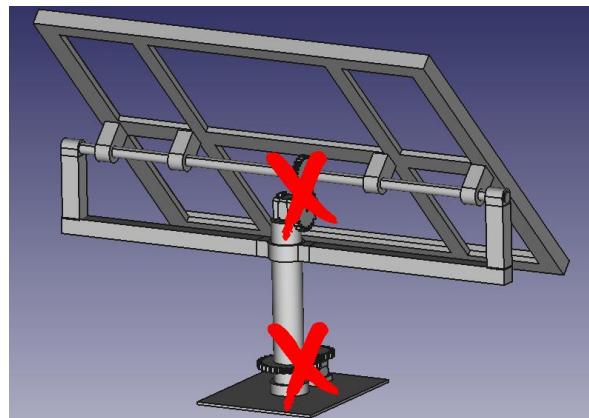


Figura 1.1.1 – Sistem fix

La aceste sisteme, panourile solare sunt montate pe cadre metalice înclinate la un unghi fix numit și unghiul optim de înclinare. Pentru a obține o eficiență maximă de la panourile solare, acestea trebuie să fie îndreptate în direcția care captează cel mai mult soare. Sistemele de înclinare fixe, fiind imobile, sunt simplu de proiectat, construit și întreținut. Deoarece nu au părți în mișcare ele sunt foarte rezistente, dar din păcate nu sunt optime și permit producerea unei cantități mai mici de energie comparativ cu alte sisteme.

B. Sistem de urmărire solară pe o axă

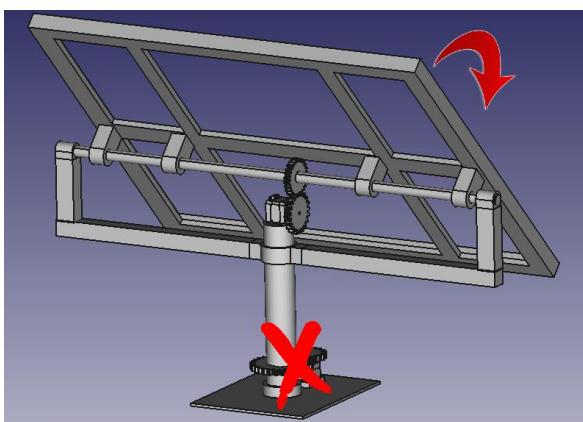


Figura 1.1.2 – Urmărire pe orizontală



Figura 1.1.3 – Urmărire pe verticală

Sistemele de urmărire solară pe o axă au un singur grad de libertate care reprezintă axa de rotație a mecanismului. Conform Ing. Kamrul Islam Chowdhury, axa de rotație pentru un astfel de sistem se stabilește de regulă după meridianul nordic, însă se pot stabili și alte direcții cardinale folosind algoritmi de urmărire.

Sunt numeroase tipuri de sisteme de urmărire solară pe o axă: urmărire pe orizontală (Figura 1.1.2), urmărire pe orizontală cu plan vertical înclinat, urmărire pe verticală (Figura 1.1.3) și urmărire pe verticală cu plan orizontal înclinat.

Este foarte important să stabilim care este axa de rotație deoarece în funcție de locație, acest aspect ne oferă și detalii legate de performanță și ne permite obținerea unui randament mai bun. De exemplu, “tipul de sistem orizontal este utilizat în zona tropicală, în regiuni unde soarele devine foarte mare la prânz, dar zilele sunt scurte. Pe de altă parte tipul vertical este folosit la latitudini mari unde soarele nu este foarte mare, dar zilele de vară pot fi foarte lungi.” (Chowdhury et al., pg. 13, 2017).

C. Sistem de urmărire solară pe două axe (DAST)

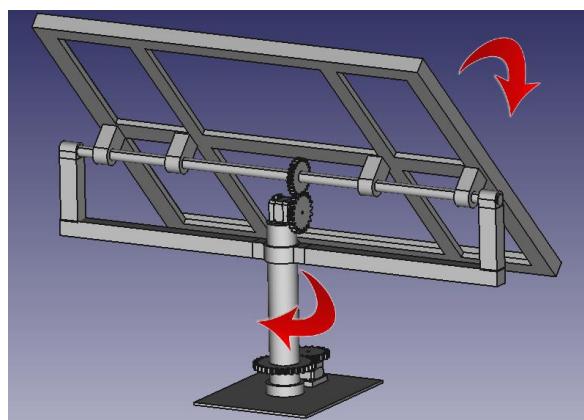


Figura 1.1.3 – Urmărire pe două axe

Sistemele de urmărire pe două axe au implicit două grade de libertate ce servesc drept axe de rotație pentru mecanismul de urmărire. Cele două axe sunt perpendiculare una pe cealaltă, iar axa care este perpendiculară cu solul este considerată axa primară (verticală). Panoul solar este montat în paralel cu axa secundară (orizontală).

Există două tipuri de sisteme de urmărire pe două axe: Azimuth-Altitude DAST și Tip-Tilt DAST. Ambele sisteme funcționează pe același principiu: indiferent unde se află soarele pe cer, sistemul de urmărire pe două axe este în stare să își regleze unghiurile astfel încât panoul solar atașat sistemului să fie perpendicular cu razele soarelui.

1.2 Soluția de urmărire solară aleasă

Pentru implementarea sistemului de urmărire solară atașat platformei, am ales să folosesc modelul Azimuth-Altitude DAST sau SmartDast. Acest sistem este capabil să realizeze urmărirea folosindu-se de cele două axe de rotație și un sistem format din patru senzori analogici ce sunt capabili să identifice intensitatea radiațiilor luminoase ce intră în contact cu aceștia.

Am ales ca senzori patru fotorezistențe însă există mai multe tipuri de senzori ce permit identificarea unor astfel de valori, precum:

A. Fotorezistență (LDR – Light Dependent Resistor)

“Este un resistor, fabricat din materiale semiconductoare omogene, a cărui

rezistență se modifică sub incidența unui flux luminos. Se bazează pe fenomenul de fotoconductivitate prin care sub influența radiației luminoase sunt eliberați electroni liberi care cresc conductivitatea electrică a semiconductorului și implicit, rezistența electrică scade odată cu creșterea intensității fluxului luminos aplicat pe suprafața sensibilă.”^[3]

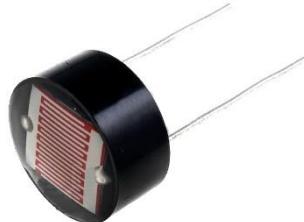


Figura 1.2.1 – Foterezistență

B. Fotodiodă

“Fotodioda este o diodă compusă dintr-o joncție P-N fotosensibilă sau un contact metalic semiconductor fotosensibil, utilizate întotdeauna în regim de polarizare inversă, deoarece în acest regim se poate fructifica în condiții optime influența fluxului luminos asupra curentului prin dispozitiv. Curentul prin diodă crește proporțional cu intensitatea luminii. Când joncționa nu este luminată, curentul este aproape neglijabil și se numește curent de întuneric.”^[4]



Figura 1.2.2 – Fotodiodă

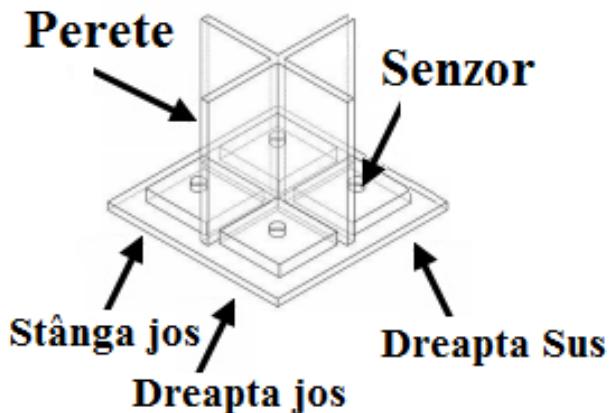
C. Fototranzistorul

“Fototranzistorul este realizat pe o structură de tranzistor, al căruia curent de colector este comandat de un flux luminos. Baza tranzistorului este înlocuită cu o suprafață care poate fi iluminată, asigurând astfel curentul de bază necesar. Unele mai sunt prevăzute cu un electrod de bază. Fototranzistorul este un tranzistor cu joncționă bază-colector fotosensibilă. Pentru fototranzistoare sunt două variante constructive: cu două terminale sau cu trei terminale.”^[5]



Figura 1.2.3 – Fototranzistor

Cei patru senzori sunt asamblați într-o matrice 2x2 și separați printr-un perete în 4 cadrane ca în Figura 1.2.4.

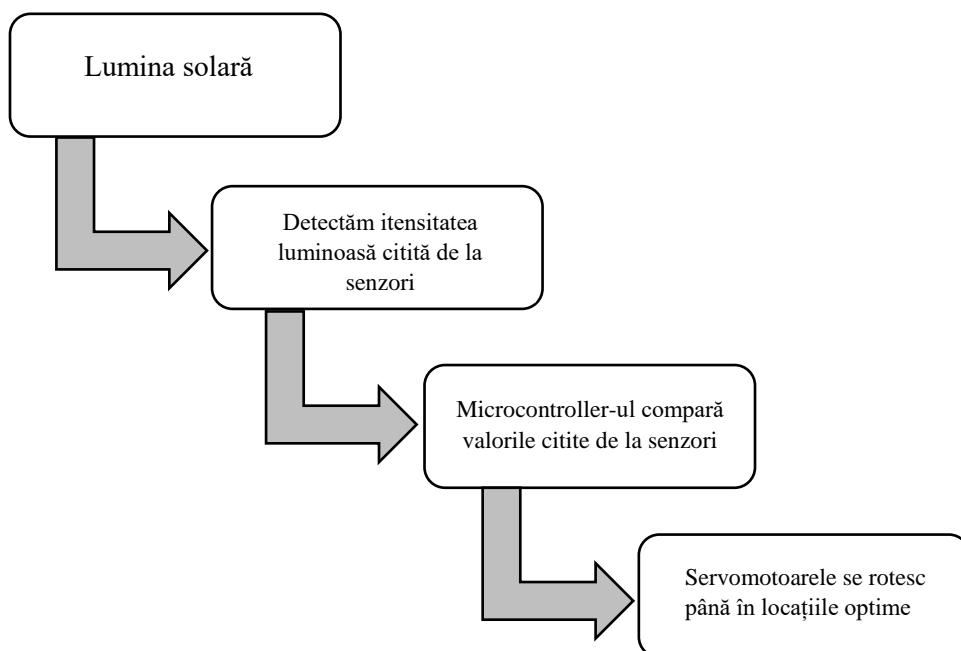


Sursa abdatată: Hossain, et al., pg 3, 2015

Figura 1.2.4 – Sistem de umbrire

Sistemul de umbrire se montează ulterior pe același cadru cu panoul solar pentru a detecta dacă razele de lumină cad perpendicular pe panoul solar. Dacă intensitatea luminoasă preluată de la toți senzorii analogici este egală ca valoare, luând în calcul și toleranța, atunci sistemul este în poziție optimă. Altfel, se modifică orientarea prin rotirea axei orizontale sau verticale, după caz.

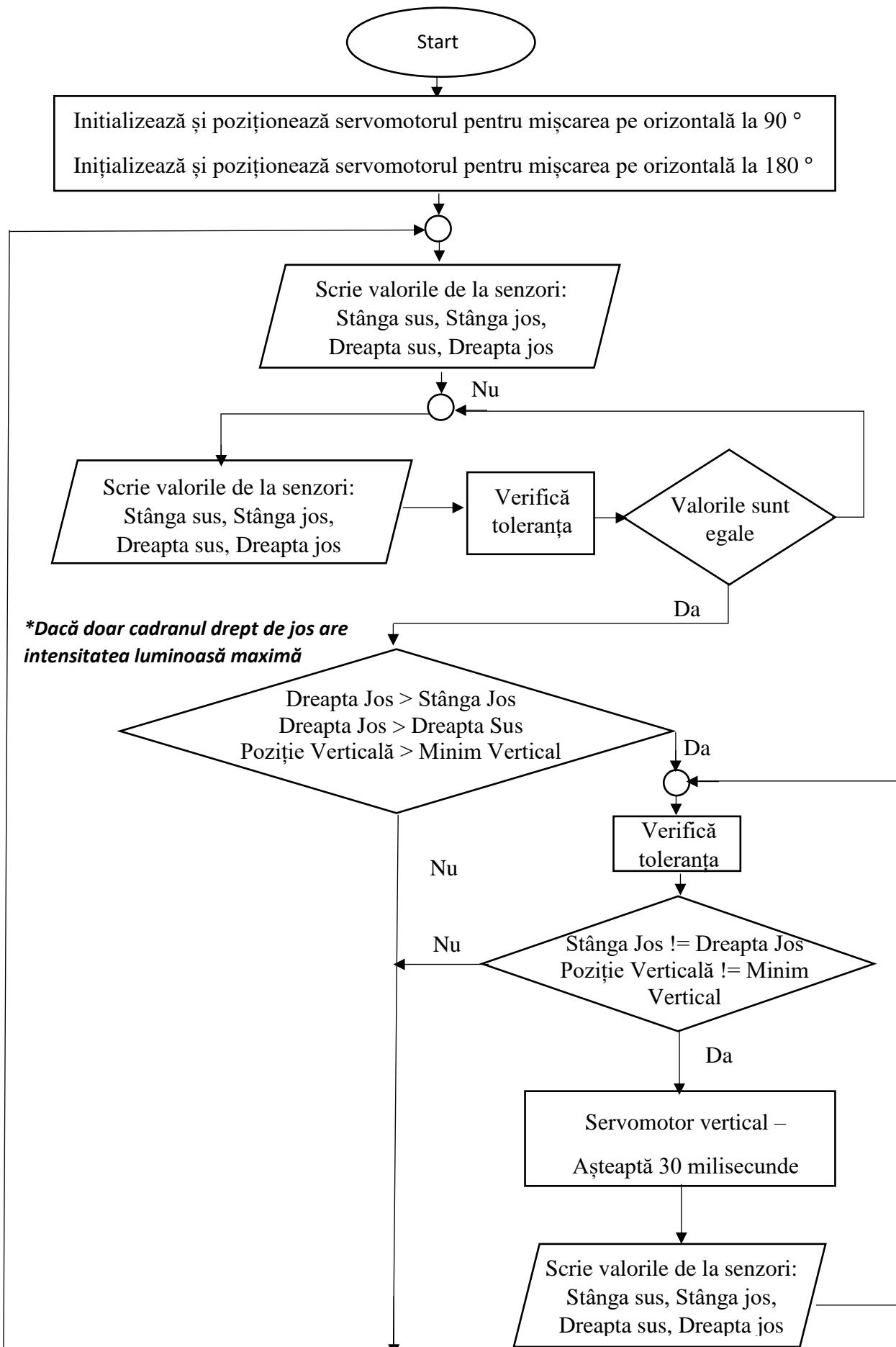
Mecanismul de detecție este prezentat în Figura 1.2.5.

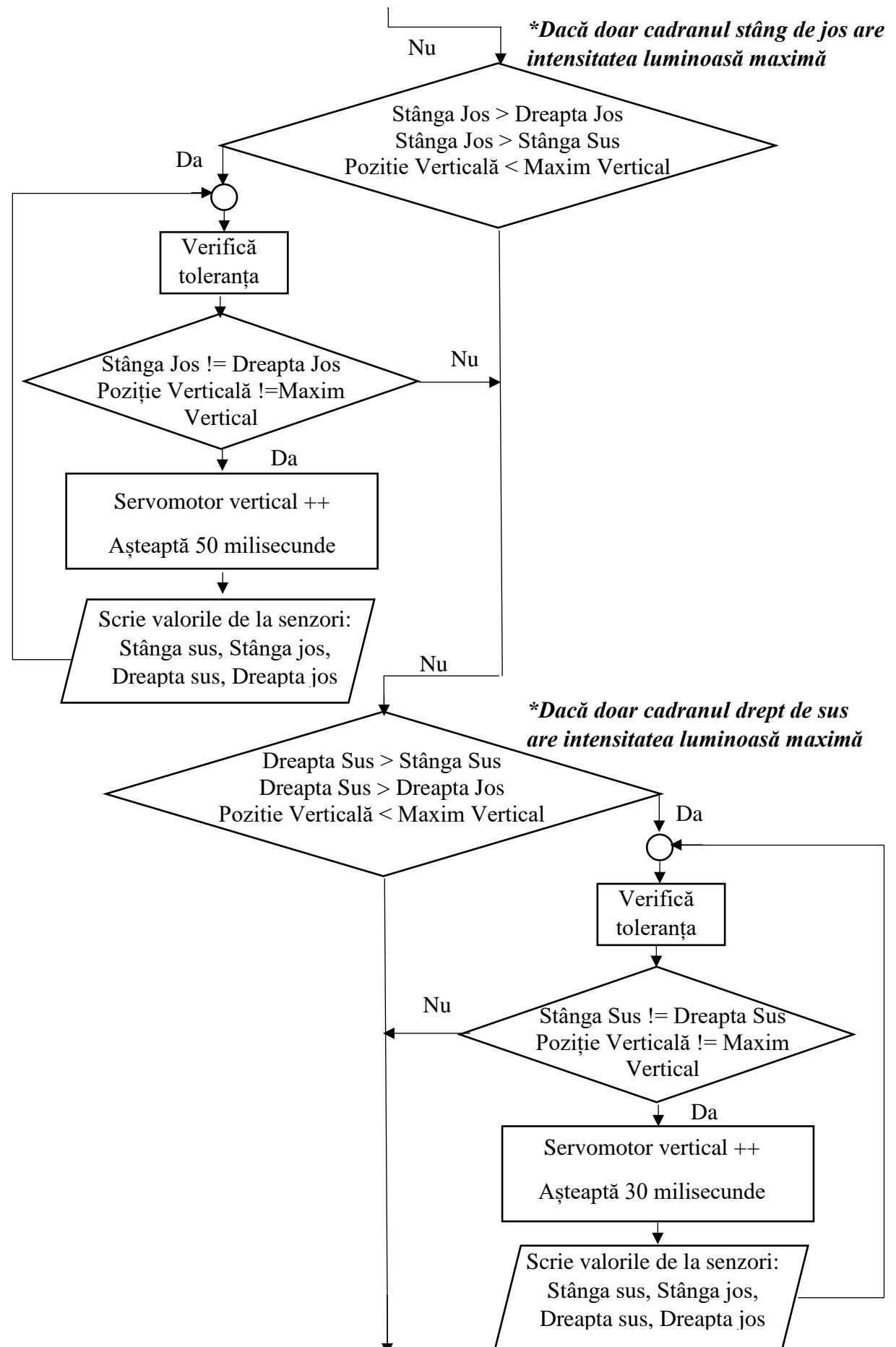


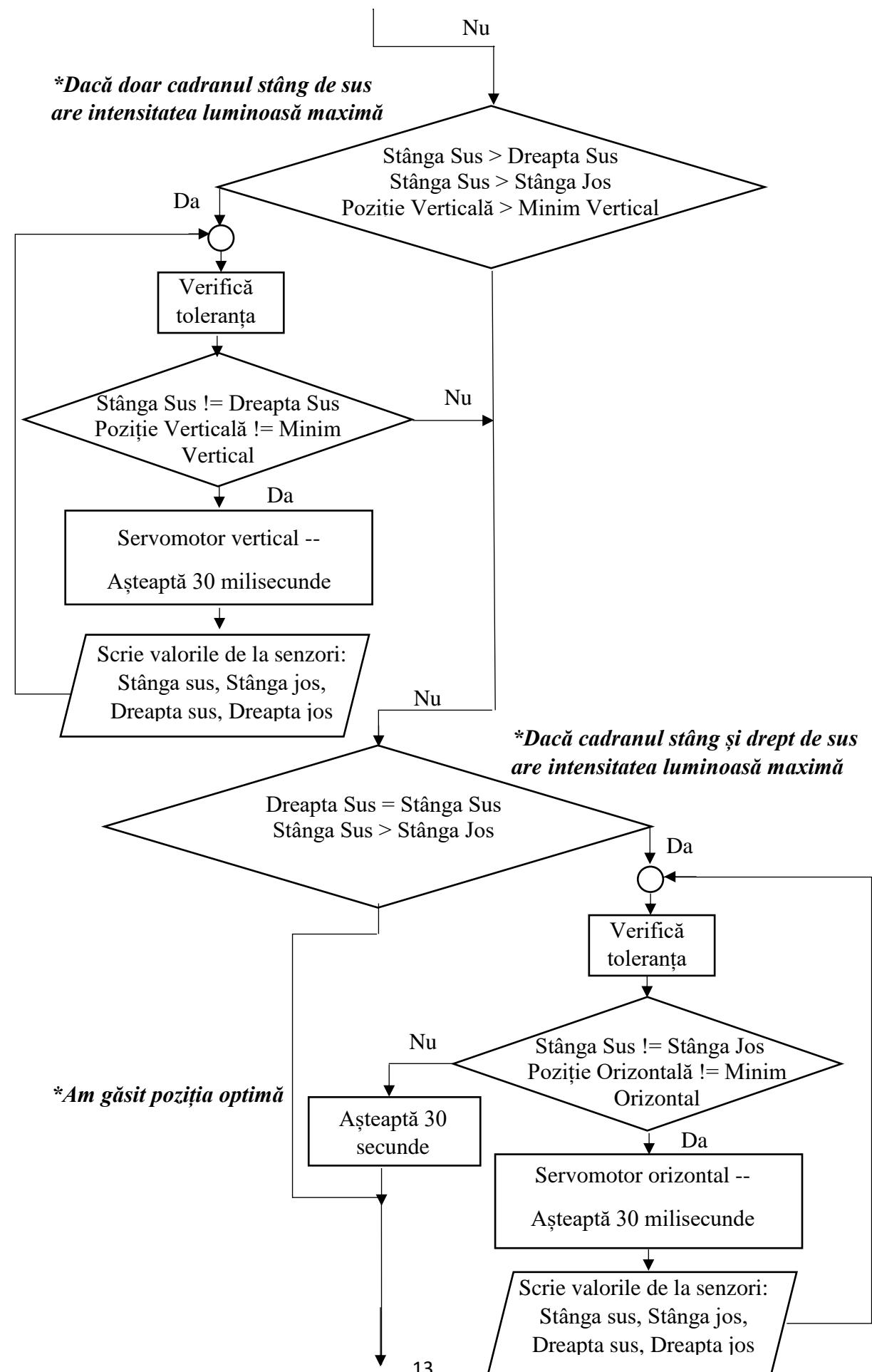
Sursa tradusă: Hossain, et al., pg 3, 2015

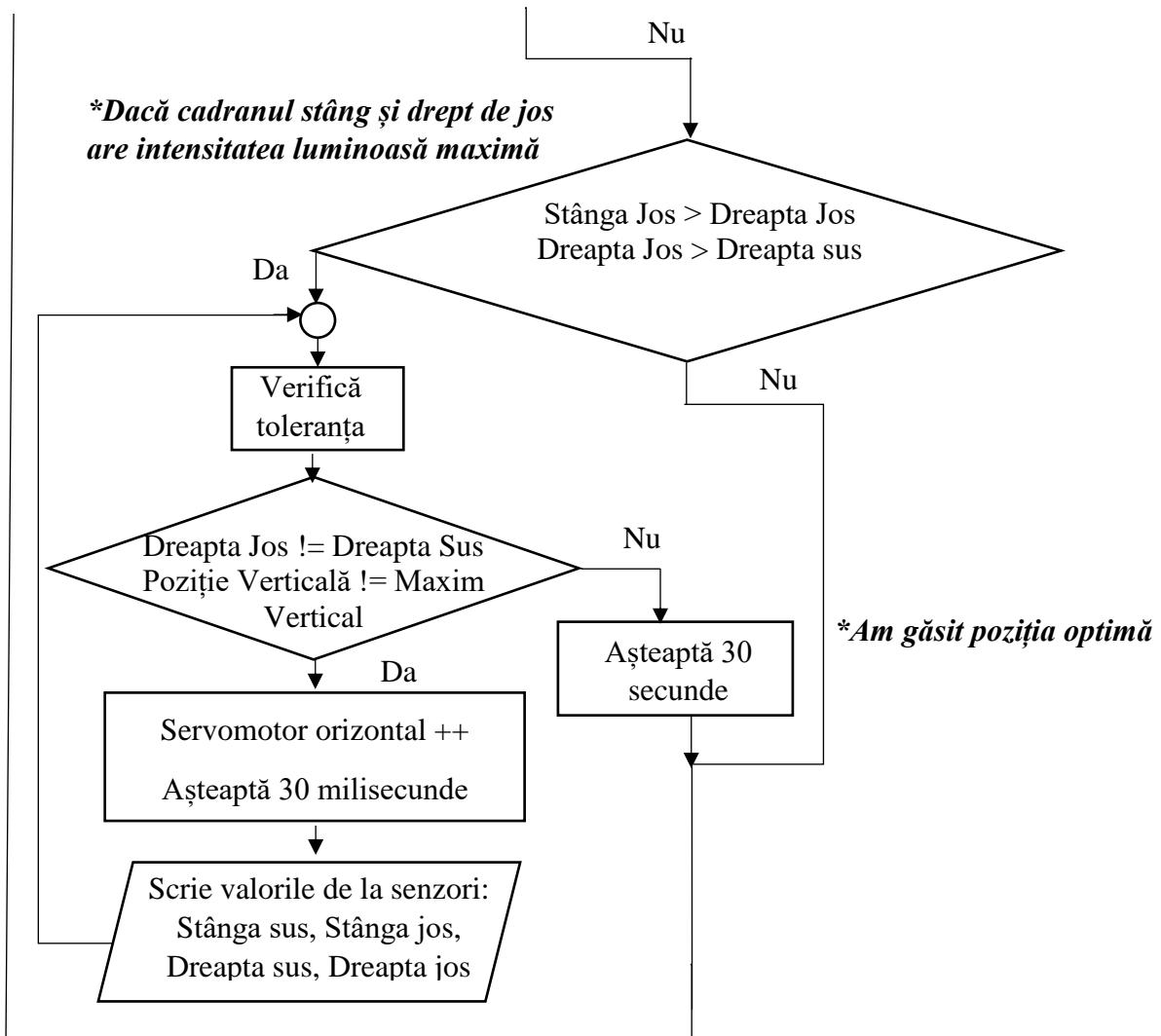
Figura 1.2.5 – Mecanism de detecție

În continuare voi prezenta diagrama pseudocod pe baza căreia se realizează algoritmul propus de mine pentru urmărirea solară a unui sistem Azimuth Altitude DAST.









Pentru sisteme de tip Azimuth Altitude DAST, există și alte abordări folosite în implementarea algoritmului. Abordarea prezentată de mine nu se folosește de calculul altitudinii și a azimutului ci doar de cantitatea de lumină din cele patru cadrane. În articolul *A simple and low-cost active dual-axis solar tracker*, se poate găsi o abordare clasică, mai rapidă dar cu o precizie și un răspuns al sistemului mai scăzut. De asemenea, consumul este considerabil mai ridicat datorită necesității de revenire în punctul inițial, după depășirea limitei de rotație maxime a servomotorului vertical.

După cum se poate observa, am ales să poziționez servomotoarele la jumătatea intervalului de rotație pentru a permite deplasarea în mod egal la stânga și la dreapta în jurul axului vertical precum în sus și în jos în jurul axului orizontal. Pentru servomotorul vertical am planificat să folosesc un model ce se poate roti până la 270° , și nu doar până la 180° aşa cum se folosește la majoritatea mecanismelor DAST. Astfel urmărirea se face continuu până la depistarea maximului de lumină, unde se așteaptă 30 de secunde (pentru a avea eficiență conform Tabelului 1.1.1)

Capitolul 2

Proiectarea și implementarea sistemului de urmărire solară ales

Sistemul de urmărire solară proiectat se bazează pe cel din lucrarea *A simple and low-cost active dual-axis solar tracker*, la care s-au adus îmbunătățiri de proiectare și design cu scopul de a facilita montarea sistemului pe o platformă mobilă, și pentru a putea ataşa un panou solar mai mare. Se pot observa diferențele în Figura 2.1 respectiv Figura 2.2

Proiectarea s-a făcut pentru un prototip ce poate suporta un panou solar de dimensiunea $45 \cdot 20$ cm. Pentru realizarea design-ului, am folosit aplicația web SmartDraw, ce permite proiectarea de obiecte 2D.

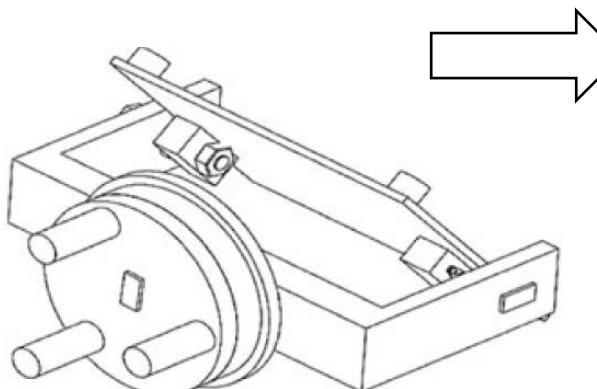


Figura 2.1 – Design vechi

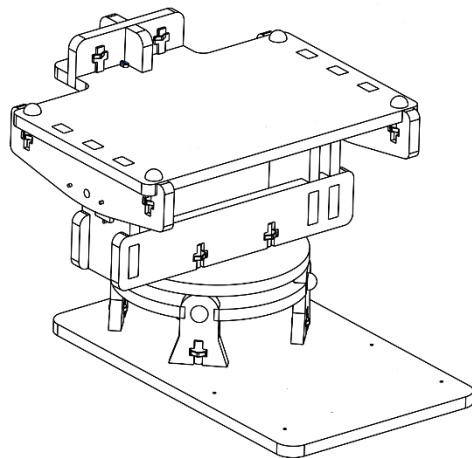


Figura 2.2 – Design nou

2.1 Modelarea fizică a sistemului

Piesele au fost realizate din plexiglas, pentru a oferi rezistență și durabilitate sistemului. S-a folosit un CNC pentru a decupa fiecare piesă, pe baza unor fișiere .gcode.

CNC-ul, sau mașina-unealtă cu comandă numerică, permite tăierea, frezarea, și decuparea automată a materialelor precum: lemnul, diverse metale, sticlă, materiale plastice, etc. “De la industria aeronaumatică, medicală și militară până la industria chimică, cea a confecțiilor sau cea alimentară, mașinile unealtă cu CNC eficientizează procesele de producție prin precizie înaltă, capacitate de lucru mai mare și durată de activitate mai scurtă.”^[7]

Pentru programarea CNC-urilor, se folosește un limbaj de programare numit G. Programele scrise în limbajul G se salvează în format GCODE și sunt compilate de către mașina de calcul a utilajului cu scopul ca brațul robotic al acestuia să efectueze un traseu în propriul spațiu de lucru. În funcție de program și de gripper, CNC-ul realizează diverse operațiuni.

Programul G comandă mișcarea utilajului prin anumite instrucțiuni, ceea ce îi permite să se deplaseze și să efectueze sarcini în spațiu de lucru pe toate cele trei axe, în sistemul de

coordonate world al acestuia. Se poate modifica originea acestui sistem de coordonate pe baza programui G, în cazul în care se dorește realizarea de piese dificile din punct de vedere al design-ului.

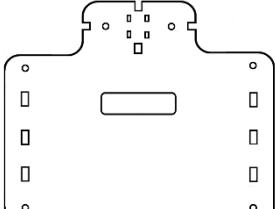
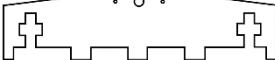
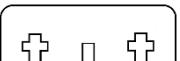
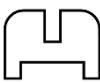
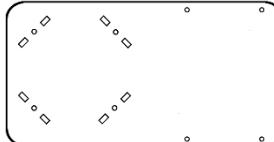
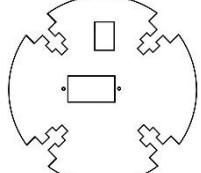
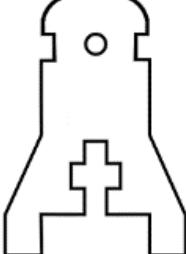
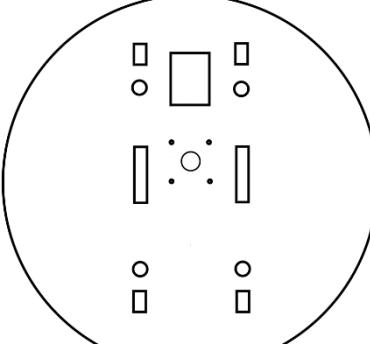
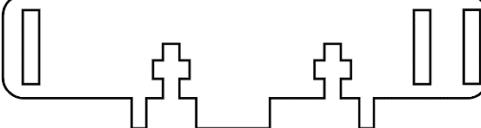
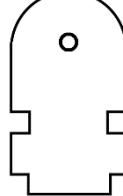
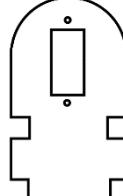
În general, un program G conține comenzi pentru pornirea și oprirea mașinii, setarea punctului de start, coborârea și ridicarea gripper-ului sau a efectuatorului terminal, impunerea unei întârzieri (delay), deplasarea liniară, deplasarea bruscă și deplasarea interpolată în spațiul cartezian (în puncte de coordonate P(x,y,z)).

```
1 G21 (metric ftw)
2 G90 (absolute mode)
3 G92 X0 Y0 Z0 (zero all axes)
4 G92 Z0.00 F149.90 (go to printing level)
5
6 (Polyline consisting of 2 segments.)
7 G1 X-93.91 Y26.01 F3500.00 (linear)
8 G1 X-90.99 Y19.56 F3500.00
9 M300 S50.00 (pen up)
10 G4 P150 (wait 150ms)
11
12
13 (end of print job)
14 M300 S50.00 (pen up)
15 G4 P150 (wait 150ms)
16 M300 S255 (turn off servo)
17 G1 X0 Y0 F3500.00
18 G1 Z0.10 F149.90 (go up to finished level)
19 G1 X0 Y0 F149.90 (go back to center)
20 M18 (drives off)
```

Figura 2.1.1 – Exemplu de cod G

Pentru generarea fișierelor .gcode a fost nevoie să realizez design-ul 2D al fiecărei componente, folosind aplicația SmartDraw. Aplicația permite alegerea metricii și calculul de distanțe între obiectele desenate în spațiu, alegând anterior template-ul Floor Plans din secțiunea New Documents. La salvarea fișierelor în format SVG se păstrează metrica aleasă. Fișierele cu extensia .svg se convertesc ușor în .gcode prin folosirea unei aplicații dedicate cunoscute sub numele de Inkscape. Această aplicație desktop, are o paletă largă de tool-uri utile și există posibilitatea de conversie a imaginilor sau a altor fișiere compatibile, în fișiere cu extensia .gcode.

Pentru a proiecta piese de dimensiuni mai mari se pot modifica datele din fișierele SVG și se generează din nou programele G, cu specificația că în cazul orificiilor pentru instalarea servomotoarelor, trebuie modificate dimensiunile și metrica, în funcție de forma și mărimea motoarelor. Așadar, având acces la date, se poate realiza un sistem la scara 1:1 pentru panouri solare mari, dar trebuie să se țină cont cu rigurozitate de calcule ce țin de rezistența materialelor pentru alegerea servomotoarelor, a dimensiunii pieselor și a gabaritului maxim acceptat.

Sistem fizic				
Subsistem	Componente			
Plan Orizontal	 Bază (1)	 Margine de susținere a bazei și prindere servomotor orizontal (2)	 Axa est-vest de umbră (1)	 Axa nord-sud de umbră (1)
Plan Vertical	 Bază (1)	 Suport de rotire pe verticală cu prindere servomotor vertical (1)	 Picior suport (4)	
Andocare		 Bază de prindere (1)		
Prindere	 Margine de prindere plan vertical și orizontal (2)	 Suport de rotire pe orizontală (2)	 Suport de rotire pe orizontală cu prindere servomotor orizontal (2)	

Tabel 2.1.1 – Componentele fizice ale sistemului de urmărire

În Tabelul 2.1.1 se pot observa toate subsistemele din care este alcătuit mecanismul de urmărire din punct de vedere mechanic. Fiecare componentă a subsistemului este deașabilă și ușor de schimbat la nevoie. Pentru a înțelege cum se îmbină aceste componente și cum obținem structura fizică a sistemului DAST final, trebuie să ținem cont de următoarele aspecte:

- Baza planului orizontal este prevăzută cu un orificiu de evacuare folosit pentru a trece prin acesta firele electrice ce provin de la panoul solar. De asemenea există 4 orificii dreptunghiulare folosite pentru fixarea senzorilor de lumină.
- Marginea de susținere a bazei are trei orificii rotunde folosite pentru a fixa servomotorul ce asigură rotirea în jurul axei orizontale.
- Baza planului vertical are patru orificii folosite pentru prinderea sistemului de platformă mobilă.
- Suportul de rotire pe verticală prezintă două orificii dreptunghiulare. Orificiul din centrul piesei, prevăzut cu alte două orificii rotunde, are ca scop prinderea servomotorului ce asigură rotirea în jurul axei verticale. Orificiul de sus este pentru evacuare, folosit pentru a trece firele electrice prin acesta.
- Baza de prindere are un orificiu de evacuare identic, cu același scop. Ea mai prezintă și 5 orificii rotunde folosite pentru a fixa servomotorul de pe axa verticală, în funcție de tipul de motor ales.
- Suportul de rotire orizontală cu prindere are un orificiu dreptunghiular prevăzut cu alte două orificii rotunde ce au ca scop prinderea servomotorului de pe axa orizontală.
- În cazul în care se dorește reimplementarea la scară mai mare a sistemului, este necesar să se modifice și metrica pentru toate orificiile de prindere și de fixare.

Există structuri fizice pentru sisteme DAST care au orificiile pentru senzorii de lumină poziționate ca în Figura 2.1.1, sau cu pereții de umbrire (axele de umbrire) foarte înalte, cu scopul de a calcula mai precis cantitatea de lumină din fiecare cadran. Cu toate acestea, dacă se optează pentru poziționarea senzorilor în extremitățile planului orizontal, suntem obligați să atașăm un panou solar mai mic decât baza de susținere. Pentru sisteme la scară mare, este o metodă utilă și eficientă din punct de vedere al consumului deoarece este nevoie de un cuplu mai mic pentru modificarea orientării deoarece nu se imprimă forțe la marginea planului orizontal ca în Figura 2.1.2.

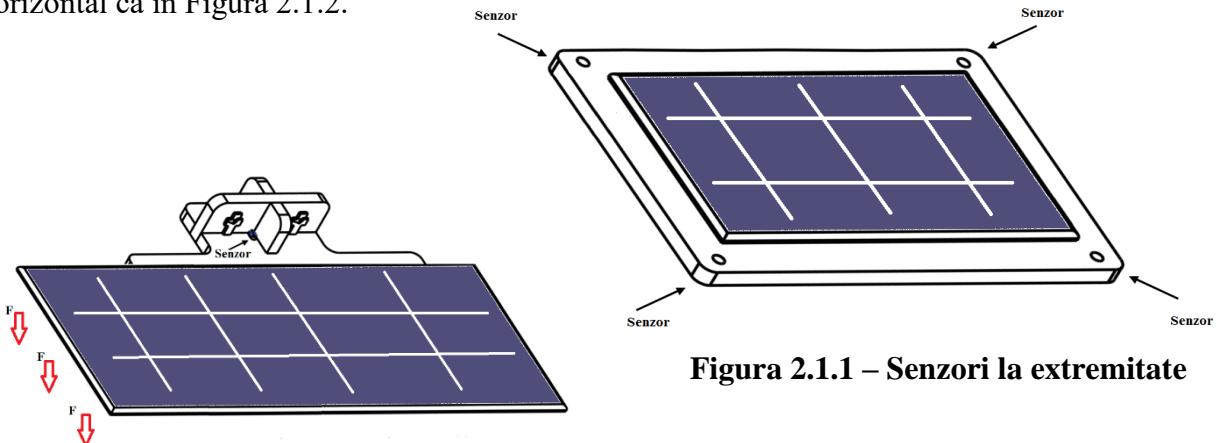


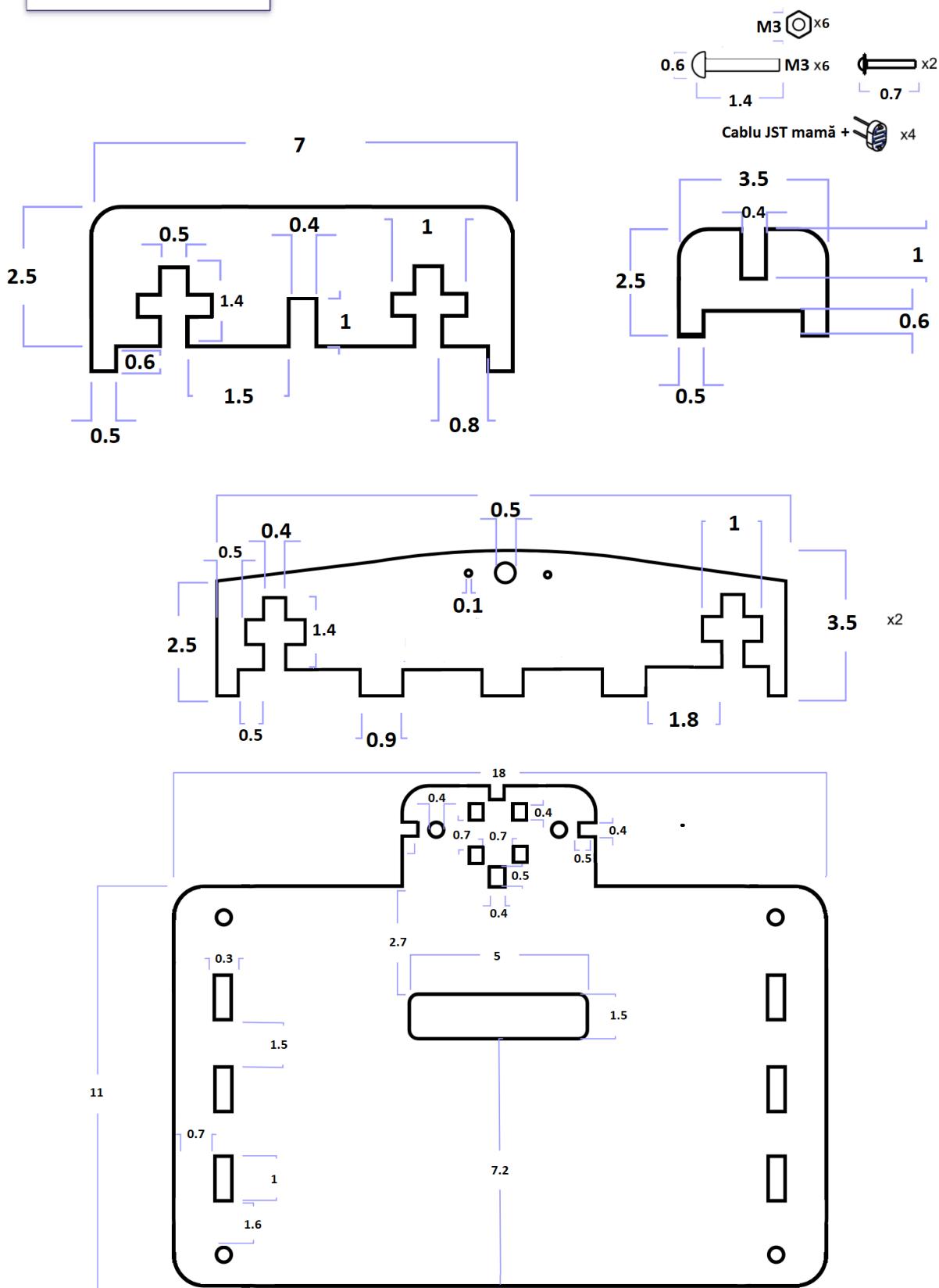
Figura 2.1.1 – Senzori la extremitate

Figura 2.1.2 – Imprimare forțe în margini

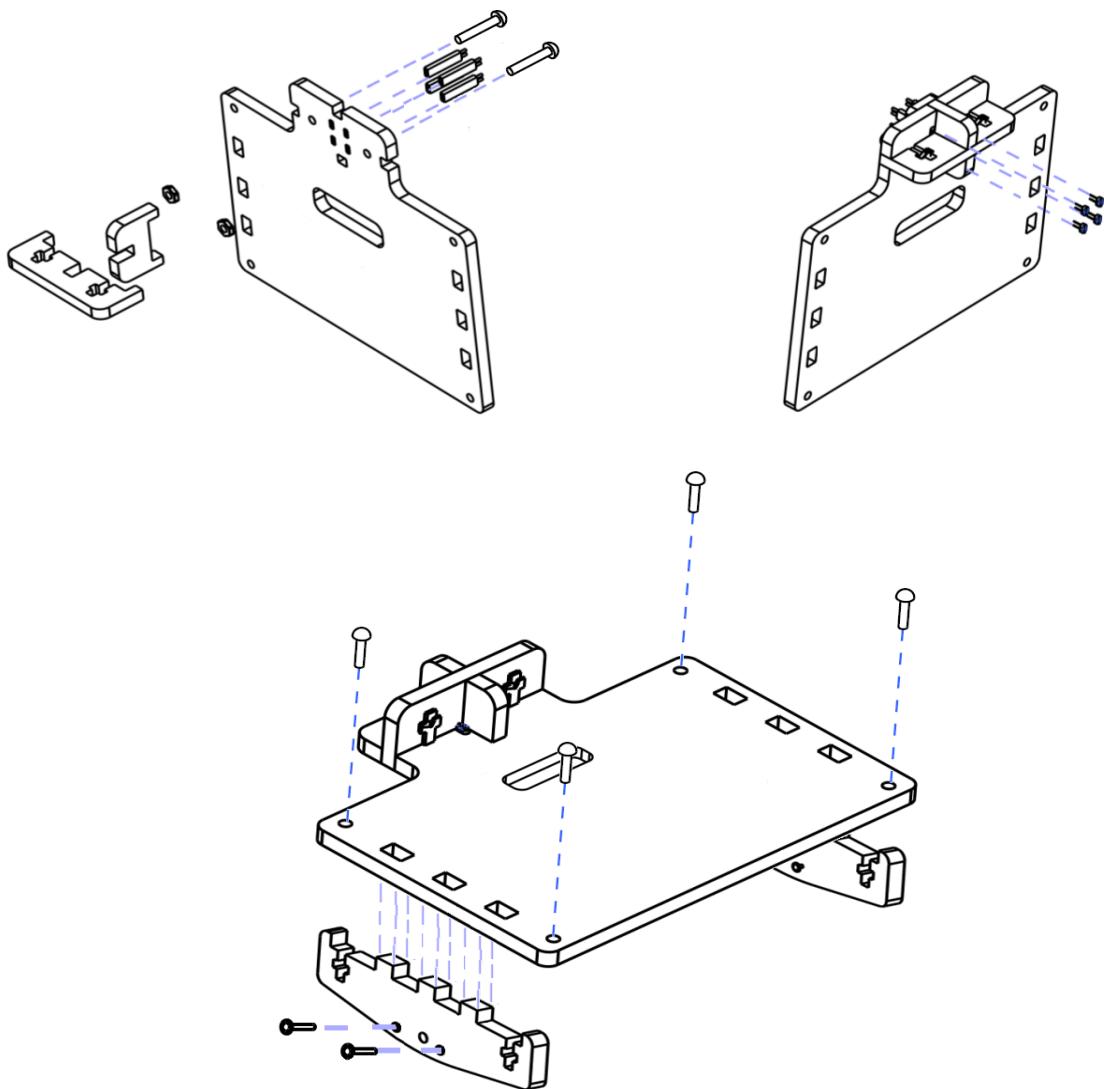
În continuare voi prezenta fiecare componentă în parte, dimensiunile și modul de asamblare al acestora.

A. Plan orizontal

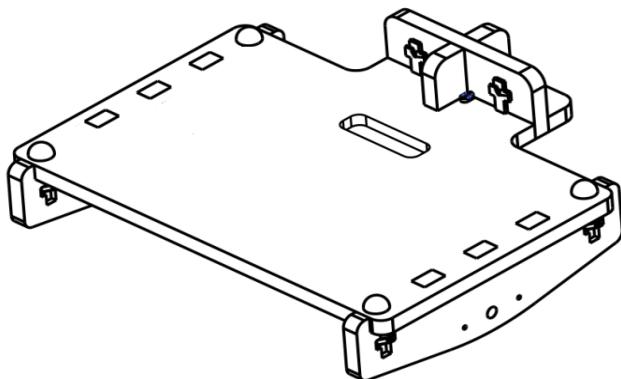
1. Componente și măsurători



2. Asamblare componente

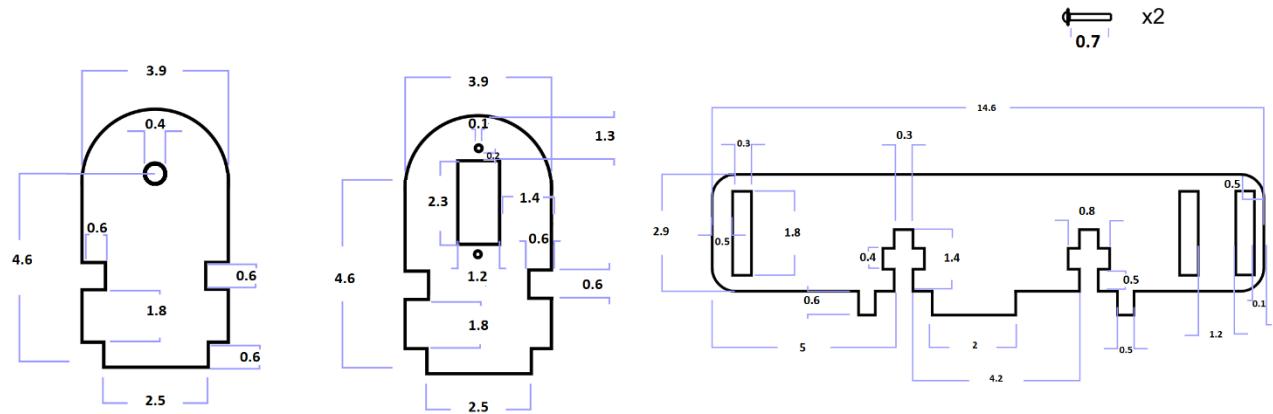


3. Rezultat

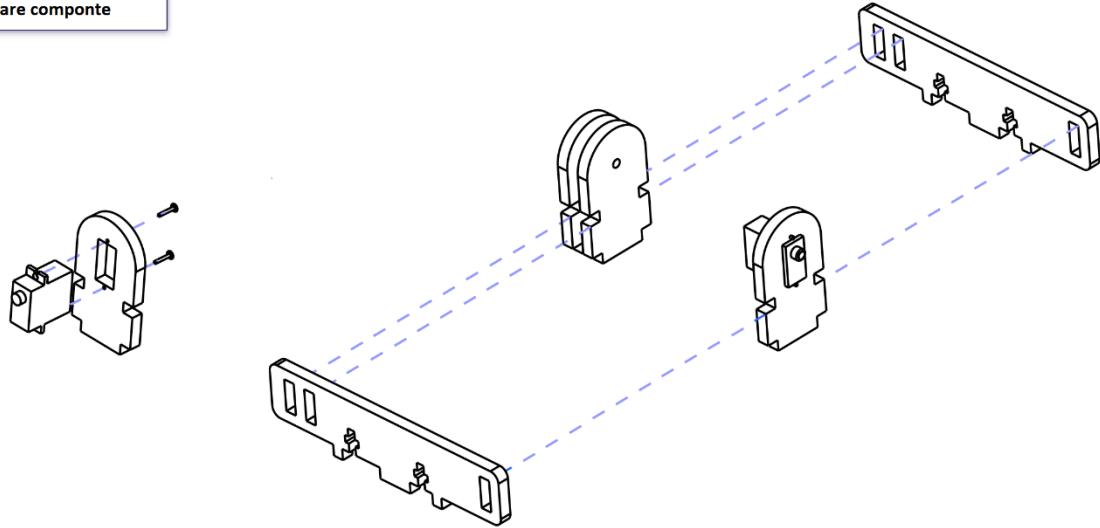


B. Prindere

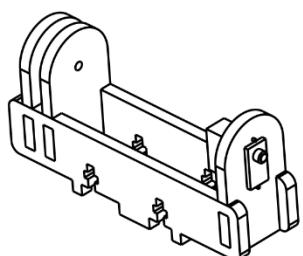
1. Componente și măsurători



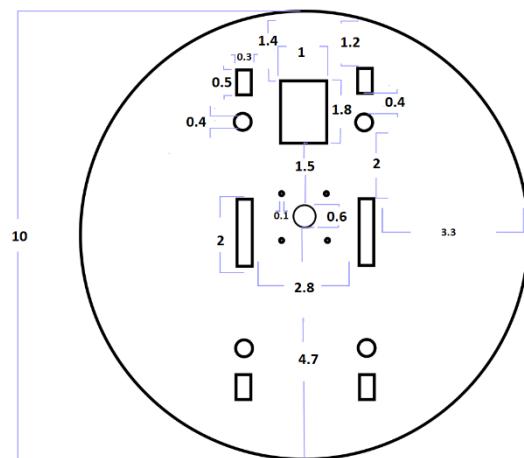
2. Asamblare componente



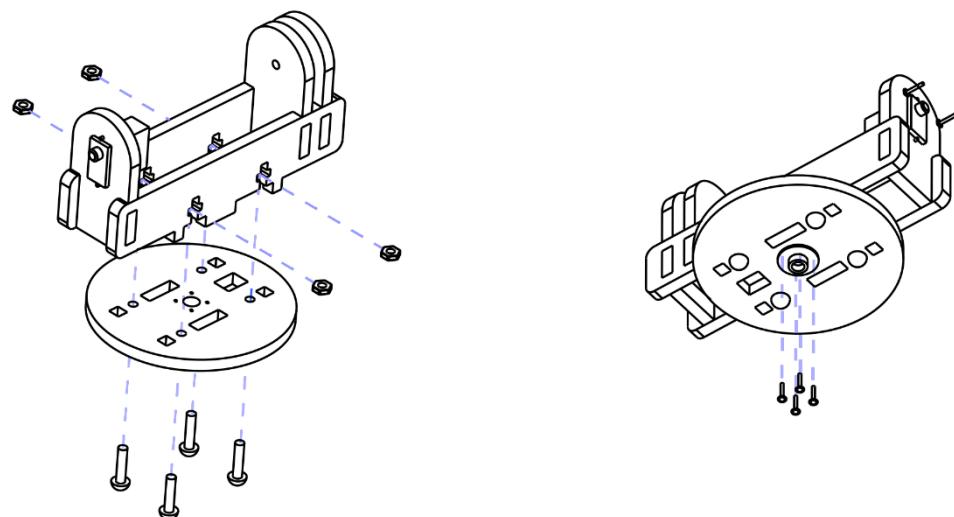
3. Rezultat



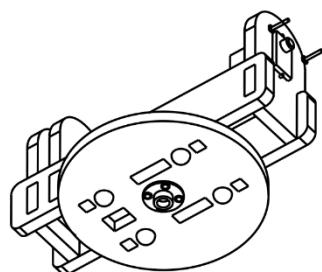
1. Componente și măsurători



2. Asamblare componente

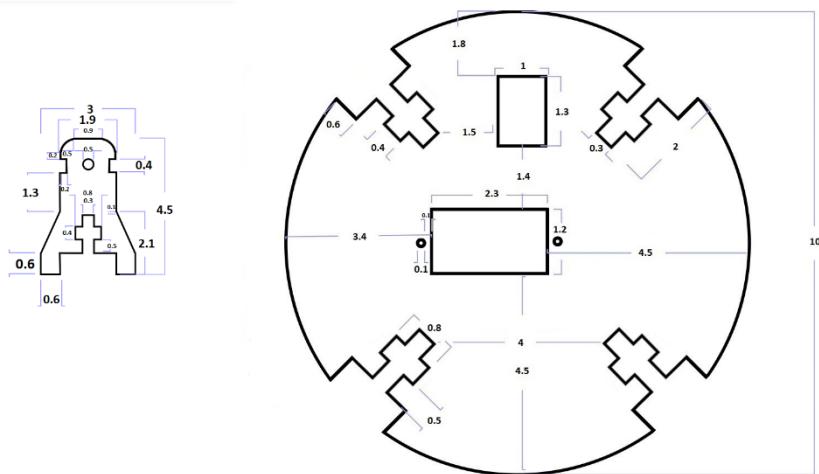


3. Rezultat

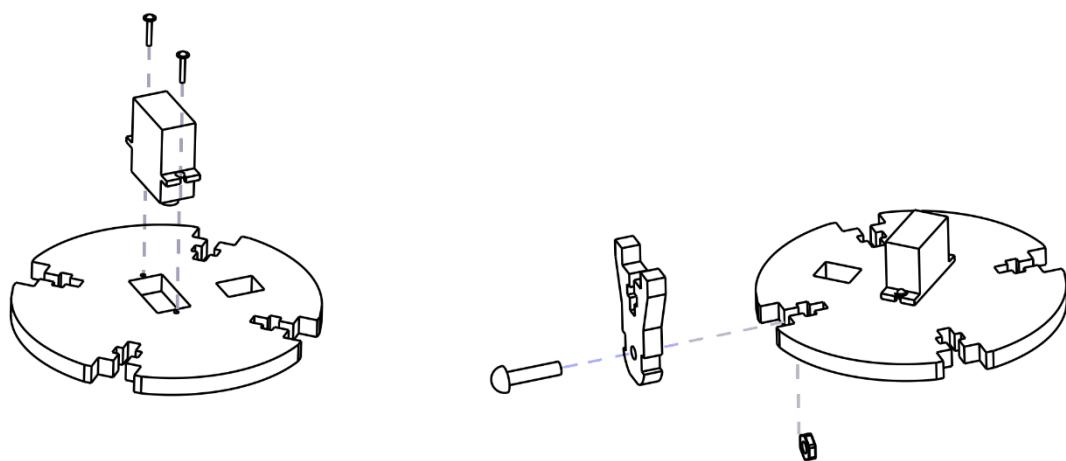


C. Plan vertical

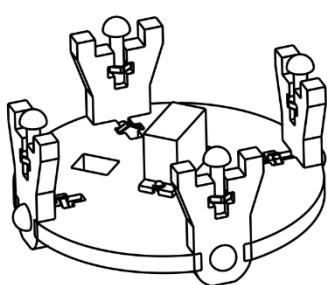
1. Componente și măsurători



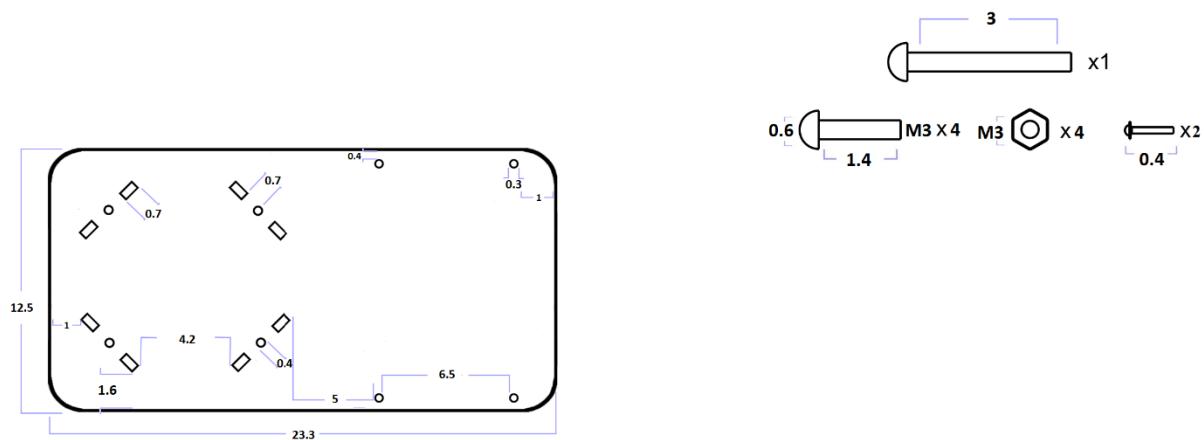
2. Asamblare componente



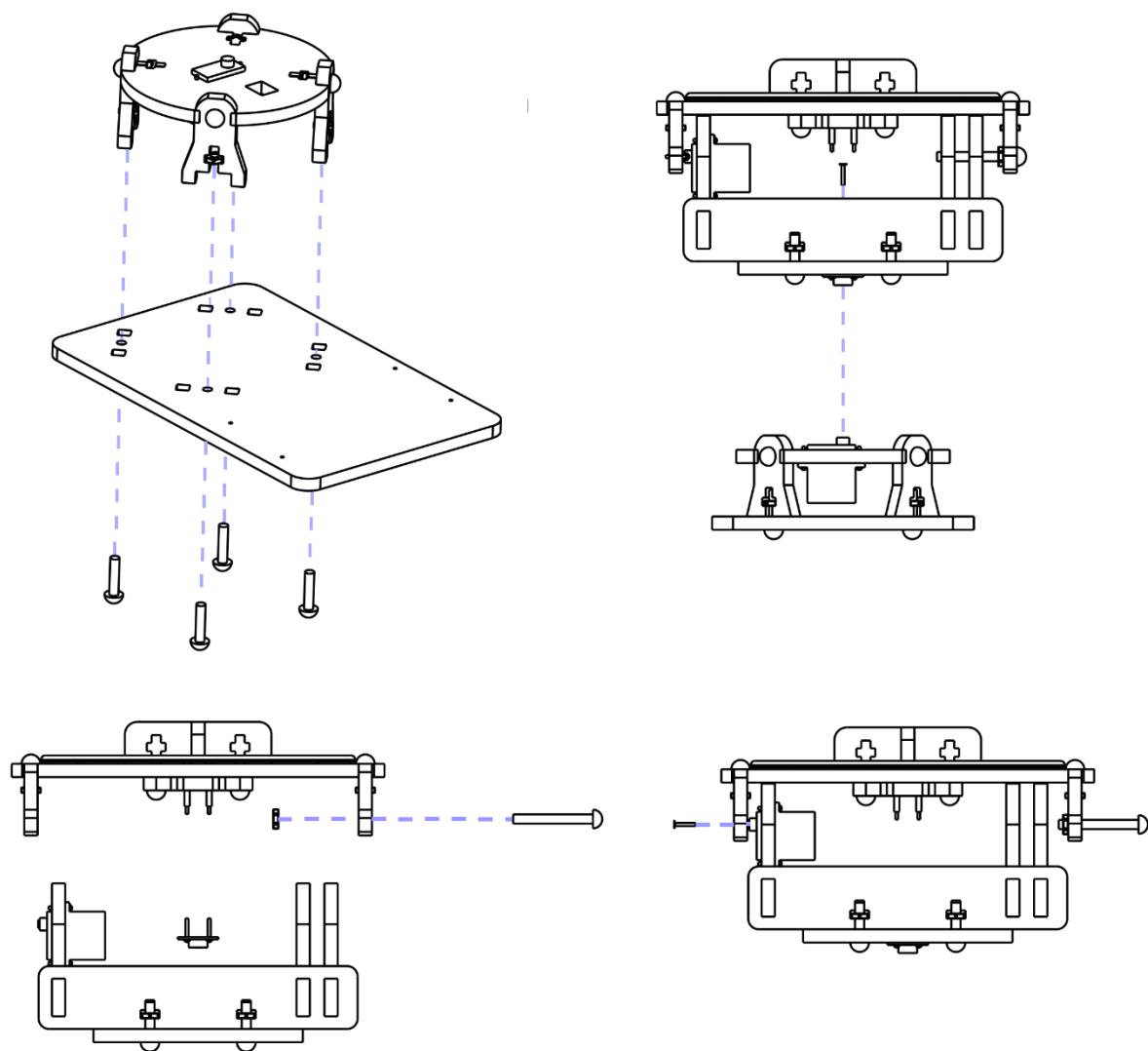
3. Rezultat



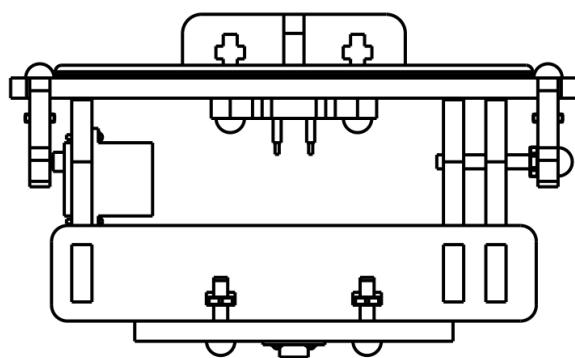
1. Componente și măsurători



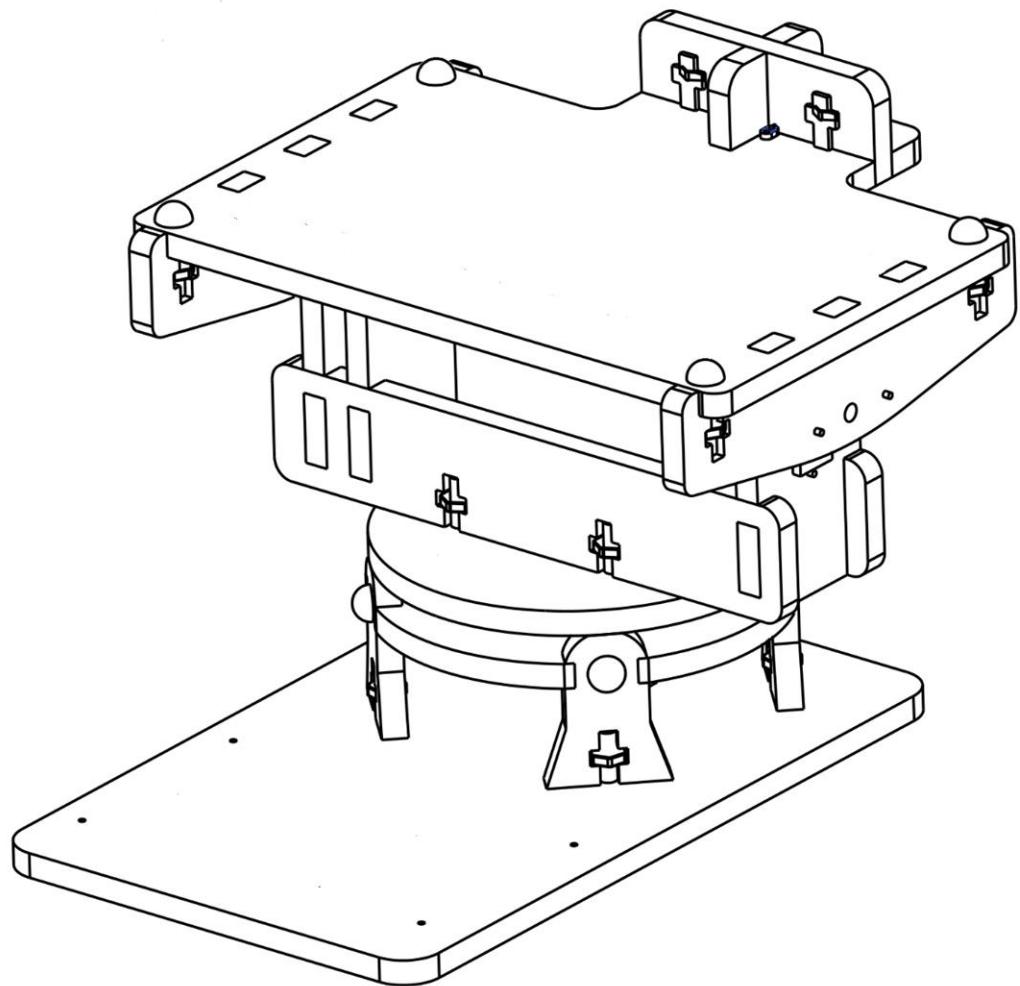
2. Asamblare componente



3. Rezultat



D. Mecanismul final



2.2 Alegerea mecanismului de urmărire

Folosind structura fizică creată și detaliată în subcapitolul anterior, se va obține sistemul DAST integrat și funcțional pe baza unei plăci de dezvoltare ce comandă prin intermediul microcontroller-ului două servomotoare în funcție de datele primite de la senzorii de lumină.

2.2.1 Alegerea componentelor hardware

Pentru comanda sistemului am ales să folosesc placă de dezvoltare Arduino Uno. Structura fizică a fost concepută pentru a încadra la fix servomotorul FT90M. Pentru detecția luminii am folosit patru fotorezistențe, 4 rezistențe și patru cabluri JST. Pentru a putea conecta direct servomotoarele la placă și pentru a realiza un sistem compact, a fost nevoie de folosirea unui plăci de expansiune Senzor Shield. Întreg sistemul este protejat de mediul exterior printr-o carcăsă metalică special concepută pentru protecția plăcii Arduino Uno la care s-a conectat un Senzor Shield.

Arduino Uno



Figura 2.2.1.1 – Arduino Uno

Arduino Uno este o placă de dezvoltare compatibilă Arduino bazată pe microcontrollerul ATmega328P, “microcontroller pe 8 biți cu 32K bytes de memorie programabilă, 1K byte de memorie EEPROM și 2K bytes de memorie SRAM 32x8 registrii, două timer-e pe 8 biti și unul pe 16 biți. Microcontroller-ul este unul performant ce face parte din familia AVR Atmel, folosită în numeroase aplicații practice ca și unitate centrală de prelucrare (CPU).”^[8]

O placă de dezvoltare este o întreagă platformă hardware ce integrează mai multe componente utile în dezvoltarea de sisteme practice. Arduino Uno are 14 pini I/O digitali din care 6 pot fi folosiți pentru control PWM (Pulse With Modulation – tehnică de variație a tensiunii date la ieșire unui alt dispozitiv), 6 pini analogici, un buton de reset, o mufă jack pentru alimentarea cu curent continuu și un port USB. Există de asemenea și pini ce furnizează 3.3V respectiv 5V la ieșire, pini pentru împământare (GND) precum și pinul Vin folosit pentru alimentarea plăcii de dezvoltare de la o sursă externă. Pinii 0 (RX) și 1 (TX) pot fi folosiți ca porturi seriale pentru comunicarea cu alte sisteme de calcul. De asemenea există și doi pini pentru protocolul I2C, numite SCL și SDA, pe care le vom folosi ulterior la sincronizarea mecanismului de urmărire, descrisă în Capitolul 5.

Pentru acest proiect am folosit doi pini digitali, patru pini analogici, portul USB, un pin 5V și unul GND.

Senzor Shield



Figura 2.2.1.2 – Senzor Shield V4

Placa de expansiune Senzor Shield V4 este compatibilă cu numeroase plăci de dezvoltare Arduino și se utilizează pentru conectarea directă a mai multor tipuri de senzori sau dispozitive electrice. Am folosit acest Shield pentru a conecta cele două servomotoare și a compacta întregul circuit.

Feetech FT90M



Figura 2.2.1.3 – Servomotor FT90M

Servomotorul Feetech FT90M este un servomotor cu roți dințate din metal și poate efectua o mișcare de 270 °. Acest servomotor a fost cea mai bună soluție ținând cont că se fabrică foarte puține servomotoare capabile să efectueze o rotație completă cu oprire, sunt greu de achiziționat datorită costurilor și nu sunt proiectate la dimensiunile prototipului. Scopul sistemului DAST este de a alimenta platforme ce efectuează mișcări relativ continue. Așadar, există șanse foarte mici ca sistemul DAST să ajungă la limita maximă sau minimă a servomotorului vertical și ca după 30 de secunde platforma să se afle în aceeași poziție. Cu toate acestea, în cazul în care acest lucru se întâmplă, pentru a asigura funcționalitatea corectă, în momentul în care se depășeste limita de rotație minimă sau maximă admisibilă de servomotorul vertical, sistemul poate fi setat să revină în poziția inițială pentru a se putea recalibra. Așa cum am menționat în Capitolul 1 al lucrării, această funcționare aduce cu ea din păcate pierderea eficienței și scade randamentul.

Din punct de vedere al specificațiilor tehnice, FT90M are o greutate de 13 g dimensiunea de 23.2·12.5·22.0 mm, un cuplu de 1.9Kg · cm la 4.8V și poate fi controlat digital cu PWM. Viteza sa de rotație la 4.8V este de 0.9 secunde la fiecare 60° parcurse. FT90M consumă 150mA la 6V în timpul efectuării unei mișcări. Acest servomotor este foarte fin, silentios și robust.

Senzori de lumină

Am ales ca senzori patru fotorezistențe GL5528 și pentru fiecare câte o rezistență de 10K Ω. GL5528 măsoară nivelul de iluminare a unei suprafețe, având o rezistență între 8 și 20K Ω la un flux luminos distribuit uniform de 10 lx (lux) măsurat. Ca și timp de răspuns, senzorul are un timp de creștere de 20 ms și un timp de cădere de 30 ms. Lumina soarelui emite în contact direct între 32000 lx și 100000 lx. Capacitatea de măsurare cu precizie a intensității luminoase pe baza unei fotorezistențe scade odată cu creșterea cantității de lumină la care este expus senzorul. Pentru un răspuns și o precizie mai bună în detecția modificărilor fluxului de lumină, se recomandă la proiectarea sistemului la scară 1:1 folosirea unor senzori de calitatea superioară precum senzorul digital BH1750 care măsoară între 1 lx și 65535 lx.^[9]

Cu toate că senzorul folosit în proiect nu este unul de înaltă calitate, sistemul de urmărire creat se bazează pe detecția umbrelor, motiv pentru care în momentul modificării poziției platformei, cantitatea de lumină din cadranul (cadranele) umbrite va fi cu certitudine sub valoarea maximă măsurată de fotorezistență. Prin urmare sistemul va funcționa corespunzător în orice situație ce implică mișcarea sistemului, dar vor apărea mici probleme

dacă sistemul este staționar și își modifică starea doar după modul de “deplasare” al soarelui pe cer.

Pentru conectarea la sistem a fotorezistențelor, am folosit cabluri JST tip mamă.

Carcasă de protecție



Figura 2.2.1.4 – Carcasă de protecție

Carcasa de protecție este folosită pentru a putea fixa placa de dezvoltare de platforma mobilă și pentru a asigura ecranare totală față de campurile electromagnetice perturbatoare externe. Ea este confectionată din tablă din otel dublu cu grosimea de 0.5 mm și permite folosirea unui shield cu înălțimea de maxim 34 mm. Este prevăzut cu o fântă de ieșire pentru fire. De asemenea, carcasa vine cu o bridă metalică ce permite fixarea în aplicații staționare (carcasa platformei în cazul de față, după cum voi specifica în capitolul următor).

2.2.2 Integrare

Odată cu alegerea senzorilor de lumină, se dorește stabilirea toleranței senzorilor și alegerea acelor senzori care supuși la același grad de iluminare, oferă la ieșire o valoarea analogică apropiată, cu o diferență de până la 50. Pe baza unui cod scris în Arduino IDE, am generat 5 seturi de date, a către 20 de măsurători fiecare, cu ajutorul cărora am măsurat toleranțele între senzori. Dacă valoarea măsurată la aceeași cantitate de lumină pentru cadranele Stânga Sus - Stânga Jos depășea valoarea analogică de 50, se impune schimbarea unuia din senzori și recalcularea toleranței. Aceeași metodă am folosit-o și pentru cadranele Stânga Sus – Dreapta Sus, Dreapta Sus – Dreapta Jos precum și Stânga Jos – Dreapta Jos. Rezultatele au fost scrise în tabele și salvate în Excel. Tabelul 2.2.2.1 și Tabelul 2.2.2.2 prezintă rezultatele atunci când senzorii sunt supuși la lumină foarte puternică și lumină slabă.

Stânga Sus (SS)	Dreapta Sus (DS)	Stânga Jos (SJ)	Dreapta Jos (DJ)	Toleranță SS - DS	Toleranță SS - SJ	Toleranță SJ - DJ	Toleranță DS - DJ
824	834	821	810	10	3	11	24
822	831	820	809	9	2	11	22
819	829	818	808	10	1	10	21
818	828	817	806	10	1	11	22
819	830	818	809	11	1	9	21
822	833	819	810	11	3	9	23
823	834	821	810	11	2	11	24
822	832	819	809	10	3	10	23
819	830	817	808	11	2	9	22
818	828	816	807	10	2	9	21
819	830	817	807	11	2	10	23
821	834	819	809	13	2	10	25
823	834	822	810	11	1	12	24
824	832	820	810	8	4	10	22

Figura 2.2.2.1 descriere circuitul final, fără utilizarea unui Senzor Shield.

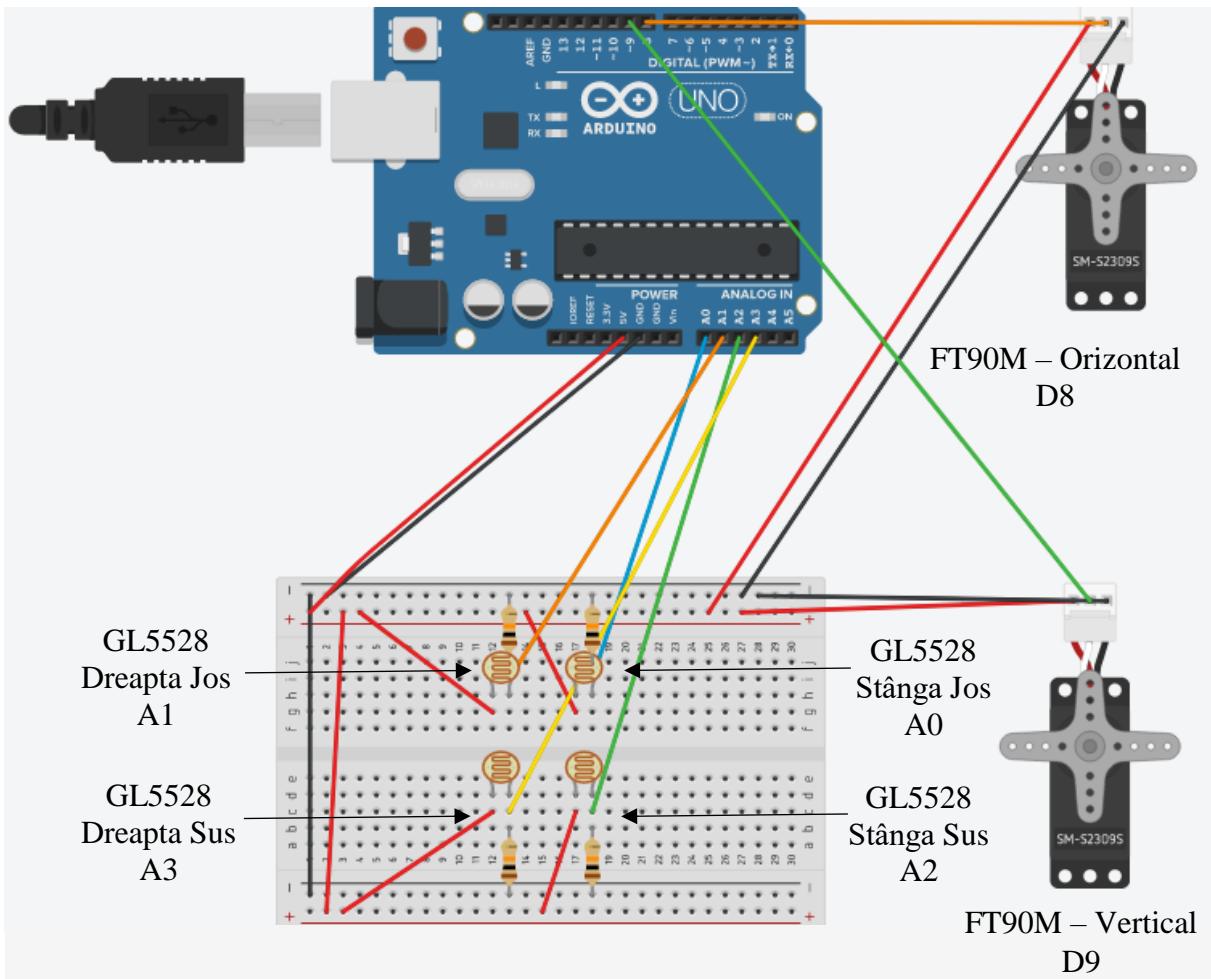


Figura 2.2.2.1 – Circuitul sistemului de urmărire

Ca sistemul să poată funcționa normal, trebuie setate limitele de rotire pentru servomotoare. Planul orizontal se va putea înclina 45° Sud și 45° Nord, servomotorul fiind adus inițial la 90° . Așadar intervalul de deplasare admis pentru servomotorul orizontal este $[45^{\circ}; 135^{\circ}]$. Planul vertical se poate rota 135° Vest și 135° Est, servomotorul fiind adus inițial la 135° . Așadar intervalul de deplasare admis pentru servomotorul vertical este $[0^{\circ}; 270^{\circ}]$. Dacă se depășesc limitele orizontale, sistemul rămâne în punctul de maxim sau minim orizontal, după caz, dar se poate deplasa pe verticală în orice direcție până la atingerea limitelor verticale, și în direcție opusă pe orizontală. Dacă se ating limitele verticale, sistemul respectă aceleași reguli: sistemul rămâne în punctul de maxim sau minim vertical, dar se poate deplasa pe orizontală în orice direcție până la atingerea limitelor orizontale, și în direcție opusă pe verticală.

Toleranțele, limitele și pozițiile curente se salvează ca variabile globale în codul sursă ca în Figura 2.2.2.2. Dacă după atingerea limitelor verticale se dorește revenirea la origine pentru recalibrare, modificăm pseudocodul prezentat în Capitolul 1 conform cu Figura 2.2.2.3.

```

//Toleranțe
double toleranceLRT = 10.0;
double toleranceLTLD = 10.0;
double toleranceLDRD = 30.0;
double toleranceRTRD = 30.0;

int vertical_limitLow = 0;
int vertical_limitHigh = 180; // echivalent cu 270
int verticalValue = 90; // echivalent cu 135

int horizontal_limitLow = 45;
int horizontal_limitHigh = 135;
int horizontalValue = 90;

```

Figura 2.2.2.2 – Variabile globale

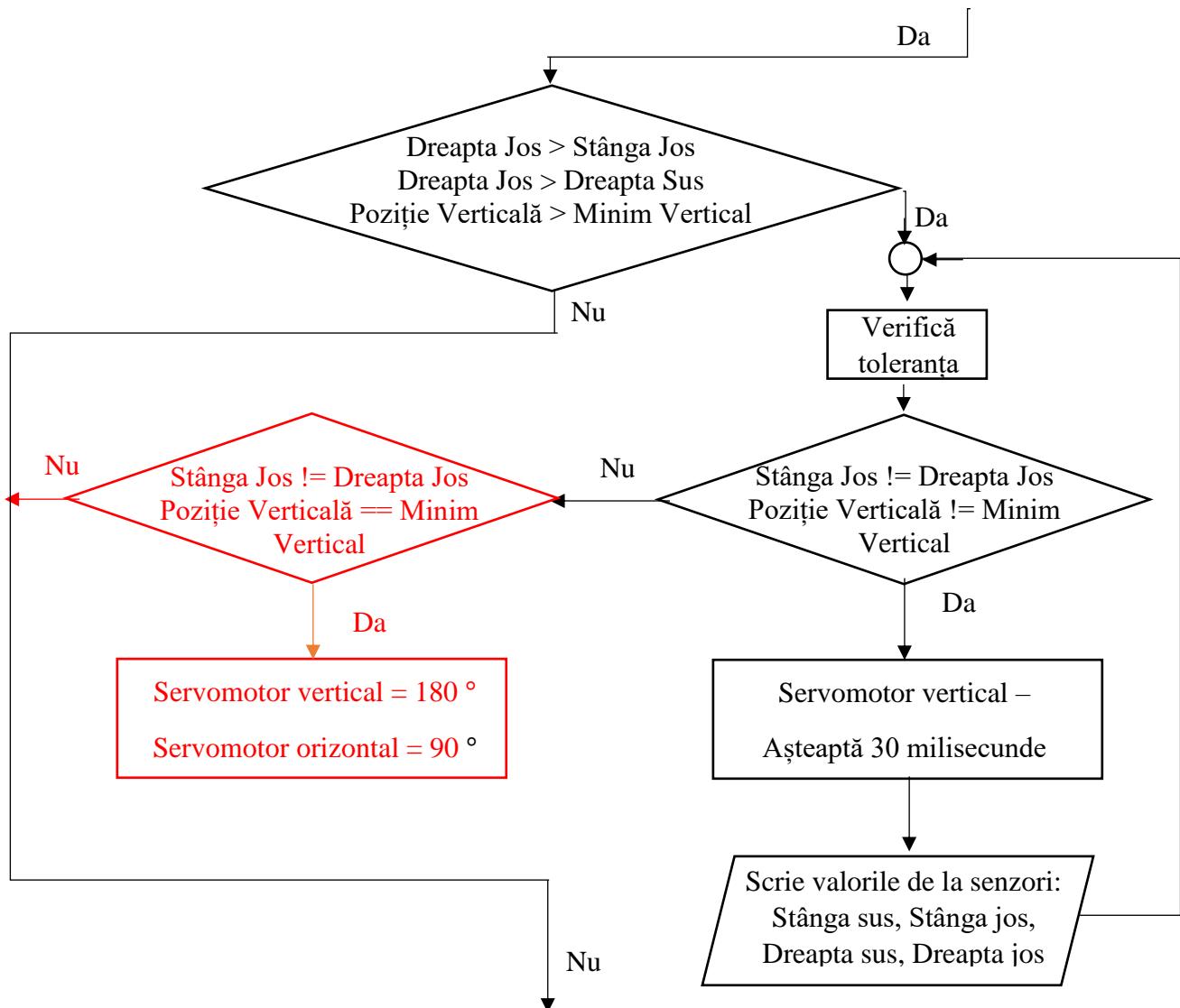
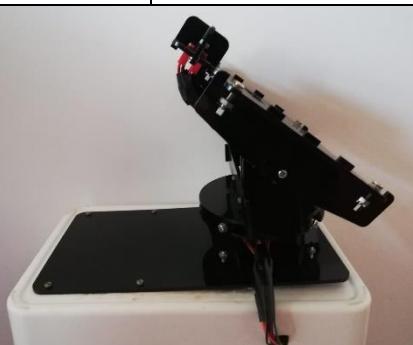
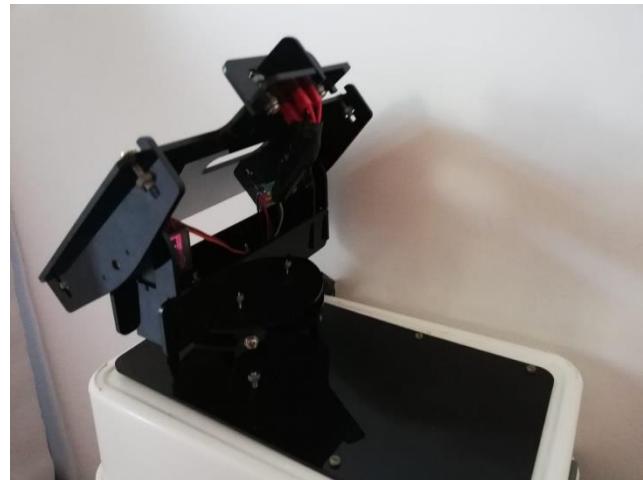
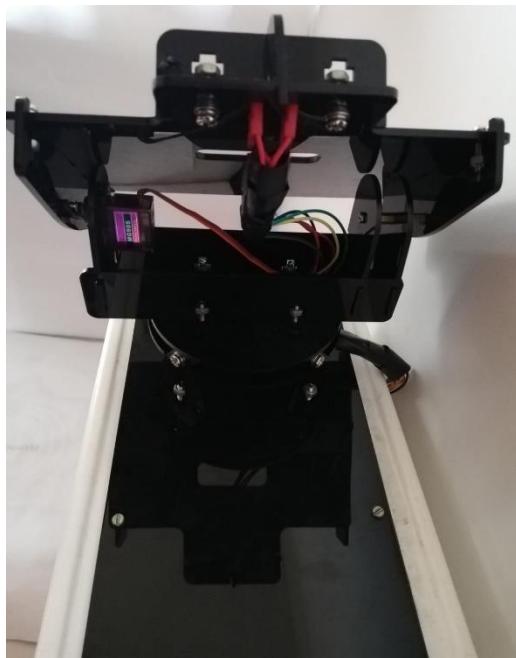
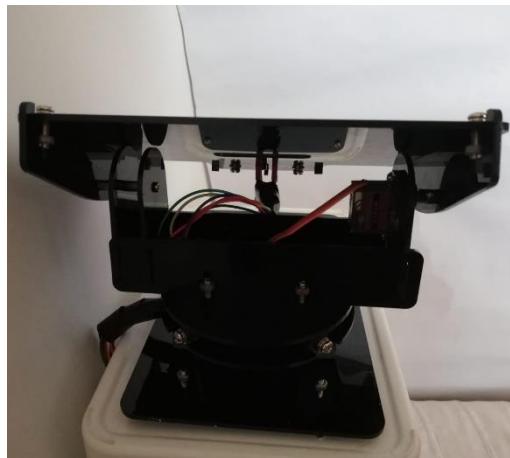


Figura 2.2.2.3 – Ajustare algoritm

2.3 Rezultat

Sistemul DAST final



Capitolul 3

Proiectarea și implementarea platformei controlată la distanță

Scopul proiectului este realizarea unei prototip pentru o platformă mobilă, alimentată de sistemul DAST. Ulterior, în cazul în care proiectul intră în producție și se realizează la scară 1:1 se poate atașa pe suprafața acestuia anumiți roboți industriali care execută sarcinii ce necesită deplasarea în locații întâi la intervale precise de timp.

Cel mai util domeniu în care se poate folosi SmartPlatform este în dezvoltarea de roboți agricoli pentru cultivarea digitală a plantelor și îngrijirea automatizată a pământului. Există o varietate de roboți folosiți în prezent pentru a realiza o serie de sarcini ce au fost întotdeauna mai dificil de executat de către om. Există roboți capabili să îngrijească pământul, să planteze semințe, să irige sau să stropească cu pesticide plantele, să culeagă fructele din livezi și legumele din pământ și să monitorizeze o grădină.

Prin utilizarea de machine learning, a senzorilor de înaltă tehnologie, a simulatoarelor pentru controlul pesticidelor și a serviciilor IoT se pot dezvolta roboți pentru tratarea culturilor, pentru recoltare și plantare, precum și aplicații ce oferă date în timp real referitoare la umiditatea solului, cantitatea de lumină, evoluția producției, activitatea roboților și alte rezultate. De asemenea se pot realiza sisteme de control a platformelor și roboților agricoli, iar utilizarea de statistici și grafice poate ajuta la depistarea perioadei potrivite pentru culegerea roadelor, în funcție de tipul de plantă, locație, vreme și alte date.

În lucrarea *Research and development in agricultural robotics: A perspective of digital farming*, inginerii de la universitatea Putra din Malaysia vorbesc despre primii roboți folosiți pentru recoltare precum și cele mai moderne tehnologii și metode de automatizare.



[11]

Figura 3.1 – SWEEPER EU H2020

“Sweeper EU H2020 este prima platformă din lume complet automatizată folosită pentru recoltarea ardeilor dulci. Această platformă mobilă are atașat manipulatorul LRMate 200iD prevăzut cu un gripper pentru prinderea recoltelor de pe tulpinile plantelor. Folosind camera fixată în vârful efectuatorului terminal, SWEEPER poate scană planta și să detecteze care din ele sunt ardei, precum și maturitatea acestora pe baza unor algoritmi ce se folosesc de culori, lumină, dimensiuni și clasificări. Pentru a învăța, robotul a fost supus unor teste bazate pe algoritmi avansați și diverse simulatoare.” (Shamshiri et al, pg. 3, 2018)

Există roboți folosiți pentru a controla creșterea vegetației sălbaticice, stropirea uniformă a pământului, monitorizarea și colectarea de date, la un preț acceptabil raportat la eficiența lor ridicată. Colectarea datelor are ca scop digitalizarea proceselor agricole.

Platformele alimentate cu energie solară sunt atât eficiente cât și prietenoase mediului. Un exemplu de astfel de platformă este RIPPA (Robot for Intelligent Perception and Precision Application) dezvoltat de Centrul de Robotica Australiană al Universității din Sydney. Acest robot este capabil să stropească cu pesticide cu viteză și precizie, pe baza unor dozări controlate de lichid.^[12]



Figura 3.2 – RIPPA

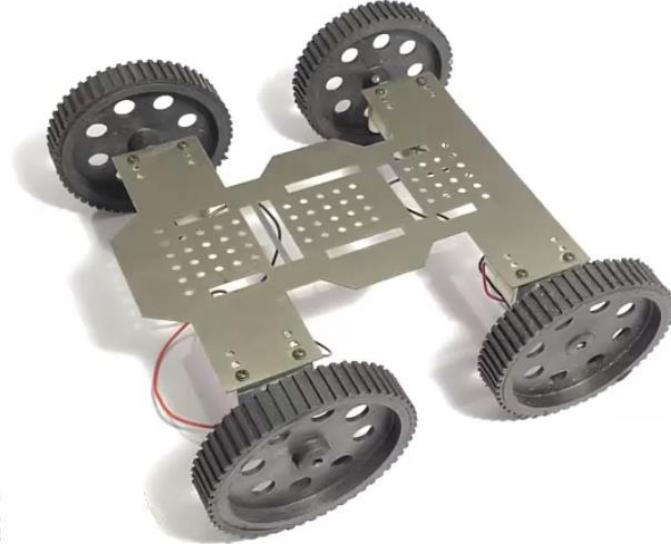
Roboții agricoli sunt folosiți într-o varietate de aplicații practice și eficientizează producția. Digitalizarea agriculturii aduce creșteri economice mari, costuri scăzute și control sporit. Mai multe detalii despre importanța automatizării proceselor agricole se pot citi în lucrarea mai sus menționată.

Scopul prototipului SmartPlatform este de a crea un sistem autonom din punct de vedere energetic pentru deplasarea unor astfel de roboți prin medii agricole (câmp, livezi, păduri). Platforma este deținută și poate fi folosită pentru roboți ce trebuie să se deplaseze pe trasee cunoscute sau care trebuie să se poziționeze în diverse locații. Controlul se realizează pe baza unei aplicații IoT. Aplicația vine în ajutor permitând agricultorilor care o utilizează să controleze platforma de la distanță pe baza unei console și să creeze programe sau cicli de deplasare. Pentru protecție, în cazul apariției unor obstacole, atât aplicația cât și platforma răspunde prin generarea unor alerte sonore și vizuale și ulterior prin oprirea sistemului.

Fiind un prototip, sistemul funcționează în condiții ideale și nu s-au luat în calcul toți factorii externi. Turația motoarelor este constantă și nu se modifică în funcție de natura terenului pe care se deplasează platforma. Ulterior, pe baza unor algoritmi se poate dezvolta un mecanism de depistare a factorilor externi perturbatori și să se ajusteze sistemul prin crearea de regulatoare. S-a luat în calcul doar sursa de alimentare folosită, modalitatea de încărcare, modalitate de control, mecanismul de control și comportamentul în mediul ideal.

3.1 Alegerea componentelor hardware

Pentru realizarea prototipului platformei s-a folosit un șasiu metalic care vine cu patru motoare în curent continuu și patru roți fabricate special pentru teren accidentat după cum se poate observa în Figura 3.1.1. Cadrul are dimensiunea de $25 \cdot 30$ cm și o înălțime la sol de 7 cm. Diametrul unei roți este de 10,5 cm și întregul ansamblu poate suporta până la 10 kg.



[13]

Figura 3.1.1 – Șasiul platformei

Ca și model de motoare, platforma vine cu modelul RF-500TB-12560. Acest motor este unul DC ce funcționează la 12 V cu un curent la solicitare maximă de 0,71 A, o viteză de 4630 rot/min și o greutate de 43 g. Pentru a reduce turația motorului s-a folosit un reductor format dintr-un mecanism de roți dințate atașat motorului, scăzându-i viteza considerabil, la 72,5 rot/min. Pentru proiectul de față, această configurație mecanică nu generează probleme, iar în subcapitolul următor voi explica cum folosesc detalii referitoare la turația motoarelor pentru a măsura distanțe și tempi specifici. În Figura 3.1.2 se poate observa motorul și mecanismul de roți dințate atașat.

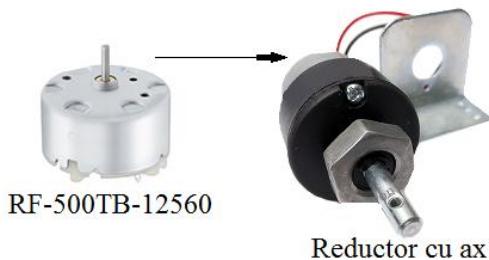


Figura 3.1.2 – Motorul RF-500TB-12560

Am ales să folosesc acest tip de șasiu cu patru motoare pentru a avea aceeași turație pe față și pe spate și prin urmare același putere în roți. Acest lucru permite ca în cazul unor denivelări puternice, platforma să poată fi redresată indiferent de sensul de deplasare ales de utilizator, prin folosirea controller-ului manual.

Pentru a alimenta motoarele, am folosit o baterie din plumb de 12 V care se încarcă pe baza unui regulator pentru încărcare solară sau aşa cum în circuitul de încărcare solară, direct de la panoul solar ataşat sistemului DAST dezbatut în capitolul anterior.

Contrulul propriu-zis al platformei este asigurat de folosirea a două controller-e pentru motoare ce se folosesc de driver-ul L298N (Figura 3.1.3) și a unei platforme hardware IoT NodeMCU ESP8266 (Figura 3.1.4), ce are integrat un microchip Wi-Fi și un microprocesor.

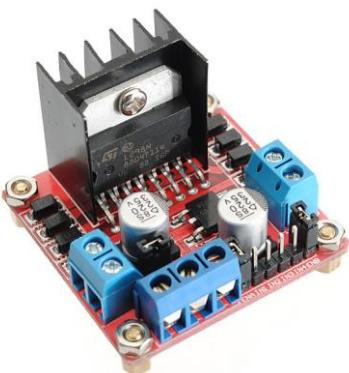


Figura 3.1.3 – Controller cu L298N

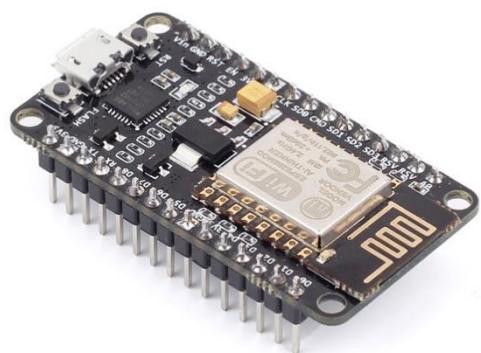


Figura 3.1.4 – NodeMCU ESP8266

“Driver-ul L298N poate fi utilizat pentru a controla două motoare de curent continuu de până la 2 A fiecare, cu o tensiune între 5 și 35 V DC sau un motor pas cu pas. Regulatorul este rapid și are protecție pentru scurt-circuit și un radiator pentru a menține L298N la o temperatură normală de funcționare. Există, de asemenea, un regulator de 5 V on bord.”^[14]

Se va putea observa în Subcapitolul 3.2 că m-am folosit și de faptul că la alimentarea controller-ului la tensiuni mai mari de 12 V, putem folosi pinul de input de 5 V ca o sursă de tensiune.

ESP8266 este folosit într-o gamă foarte largă de proiecte IoT și este unul din cele mai cunoscute module Wi-Fi. Dacă folosim NodeMCU, putem utiliza modulul Wi-Fi fără a avea nevoie de un alt microcontroller sau placă de dezvoltare.

“Are 9 pini (D0-D8) digitali I/O din care 8 suportă PWM (D1-D8) și unul analogic (doar intrare A0). Cu logica pe 3.3V, la baza acestei plăci stă chip-ul ESP8266 care rulează la 26 Mhz, are 4 MB flash și 160 KB RAM. Firmware-ul implicit al chip-ului permite rularea de scripturi LUA dar poate fi adaptat pentru cod Arduino. Practic, acesta îmbină conectivitatea Wi-Fi cu versatilitatea Arduino, fiind compatibil pentru încărcarea codului sursă pe microcontroller prin intermediul aplicației software Arduino IDE.”^[15]

Pentru a măsura distanțele și pentru detecția obstacolelor, am folosit doi senzori ultrasonici HC-SR04 (Figura 3.1.5). Acești senzori se folosesc de undele sonore pentru a măsura distanța față de obiectele aflate direct în raza undei. Practic se măsoară distanța pe baza vitezei sunetului, considerată 343 m/s și timpul mediu parcurs de la transmisie la recepție (Δt). Mai multe detalii despre cum se transmite semnalul și cum se folosește acest senzor vor fi descrise în Subcapitolul 3.4.



Figura 3.1.5 – Senzorul HC-SR04

3.2 Integrare

În Figura 3.2.1 se poate observa întregul circuit și modul de conectare al componentelor.

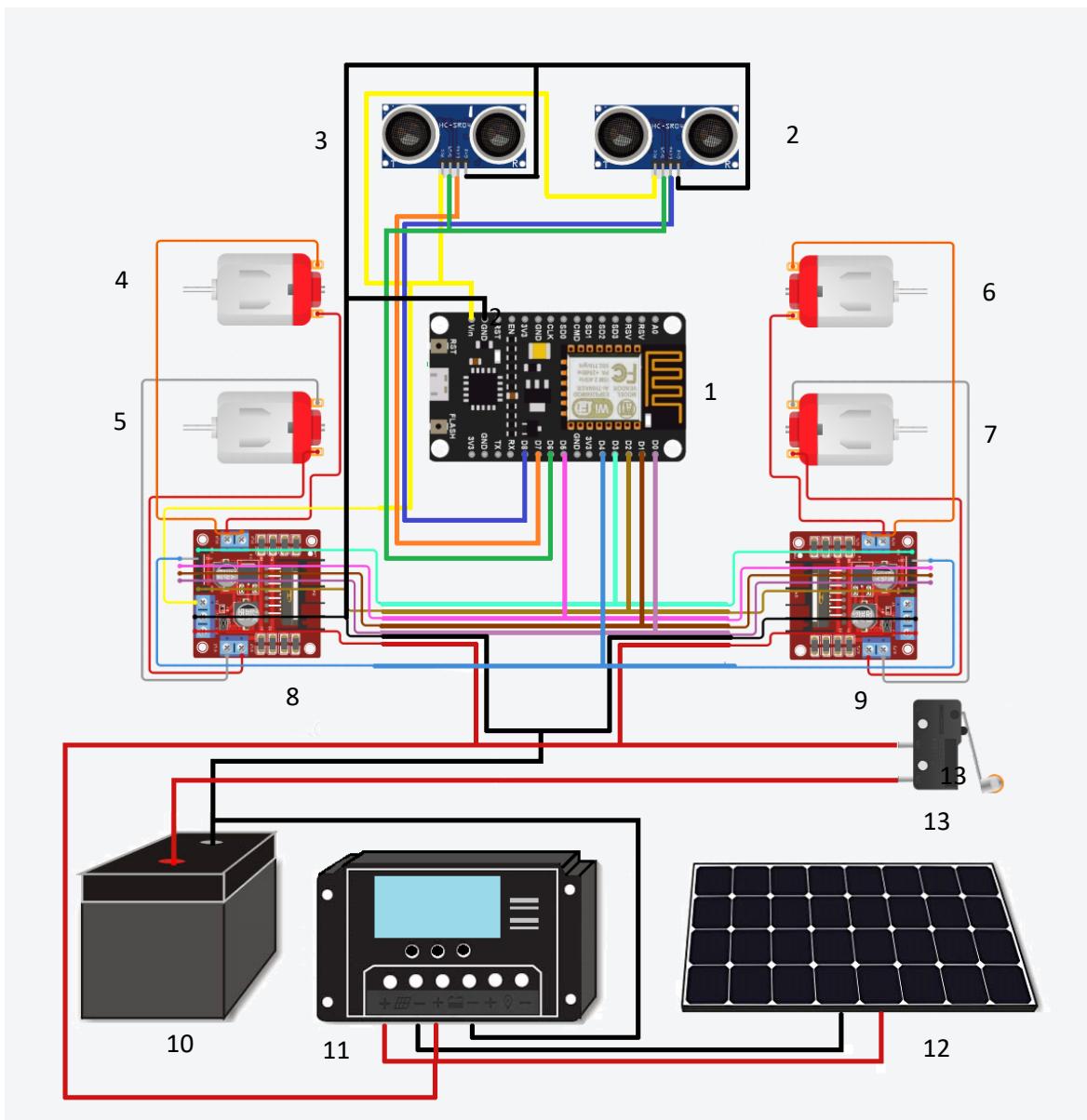


Figura 3.2.1 – Circuitul platformei

1. NodeMCU ESP8266

Datorită necesității de a folosi toți pinii digitali de care dispune placa de dezvoltare, a fost nevoie de împărțirea semnalelor între cele două seturi de motoare (4, 5 – stânga, 6, 7 – dreapta) și cei doi senzori optici (2 – senzor detectie față, 3 – senzor detectie spate). Deoarece atât driverele de control ale motoarelor (8 – față, 9 – spate) și senzorii ultrasonici au logica pe 5 V (1 logic la 5 V), iar NodeMCU o are pe 3.3 V a trebuit să ne folosim de pinul Vin 5V al unuia dintre cele două controller-e, care alimentat la 12 V (10 – baterie) poate fi folosit ca un output ce furnizează 5 V. Am conectat pinul Vin al placii de dezvoltare la acest output și împământarea GND la GND-ul driver-ului pentru a avea o logică pe 5 V.

2. Senzor HC-SR04 față

3. Senzor HC-SR04 spate

Acești doi senzori permit depistarea obstacolelor din față sau din spatele platformei, însă doar pentru un singur sens odată. Dacă platforma înaintează se va folosi senzorul de pe față, iar dacă dă înapoi, senzorul de pe spate. Cei doi senzori folosesc în comun pinul de Trig (output) care permite trimiterea undei sonore scurte și de frecvență înaltă (40000Hz). Se poate face acest lucru deoarece semnalul transmis este același pentru ambii senzori (se transmite pe o durată de de 10 microsecunde o undă sonoră digitală). Acest pin se conectează la placa de dezvoltare prin pinul D6. Pinul de Echo care recepționează semnalul este un pin de input și prin urmare nu poate fi folosit la comun deoarece trebuie să știm cine recepționează semnalul. Pentru senzorul față s-a ales conectarea la placă prin pinul D8, iar pentru senzorul spate prin pinul D7.

4. Motor 12 V DC dreapta față (Motor B)

5. Motor 12V DC stânga față (Motor A)

6. Motor 12V DC dreapta spate (Motor B)

7. Motor 12V DC stânga spate (Motor A)

Motoarele se alimentează și controlează pe baza lui 8 și 9.

8. Controller motoare cu L298N față

9. Controller motoare cu L298N spate

Controller-ul 8 este folosit pentru motoarele de la 4 și 5, iar controller-ul 9 pentru motoarele de la 6 și 7. Cele două folosesc la comun pinii pe placa de dezvoltare atât pentru pinii enable (EnA, EnB), cât și pentru pinii de input (Int1, Int2, Int3, Int4) de pe aceștia.

Motoarele 4 și 6 din dreapta folosesc la comun pe placa de dezvoltare pinul D3 pentru pinii EnB ai celor două drivere și pinii D4 și D5 pentru Int3 (sens de deplasare înainte), respectiv In4 (sens de deplasare înapoi), ce permit controlul direcției motoarelor B pentru cele două drivere.

Motoarele 5 și 7 de pe stânga folosesc la comun pe placa de dezvoltare pinul D2 pentru pinii EnA ai celor două drivere și pinii D0 și D1 pentru Int1 (sens de deplasare înainte), respectiv In2 (sens de deplasare înapoi) ce permit controlul direcției motoarelor A pentru cele două drivere.

Este foarte important ca pinii EnA, EnB să fie conectați la placa de dezvoltare prin pinii digitali cu PWM deoarece la baza funcționării motoarelor stă acest tip de semnal. Se evită astfel folosirea pinului D0 pentru EnA respectiv EnB.

Pinii D0, D1, D2, D3, D4 și D5 vor fi pini de output care transmit semnal de la placă de dezvoltare la controller-e.

Dacă se dorește deplasarea înainte, atunci se setează viteza de rotație a motoarelor din stânga și din dreapta - D2, D3 (analogWrite(enA, viteză), analogWrite(enB, viteză)) și se activează pinii pentru mișcarea înainte - D0, D4 (digitalWrite(Int1, HIGH), digitalWrite(Int2, LOW), digitalWrite(Int3, HIGH), digitalWrite(Int4, LOW)).

Pentru deplasarea cu spatele se procedează la fel dar se activează D1 și D5 în loc de D0 și D4 (digitalWrite(Int1, LOW), digitalWrite(Int2, HIGH), digitalWrite(Int3, LOW), digitalWrite(Int4, HIGH)).

Pentru viraje la stânga se dezactivează motoarele din stânga (digitalWrite(Int1, LOW), digitalWrite(Int2, LOW)), iar pentru viraje la dreapta, cele din dreapta (digitalWrite(Int3, LOW), digitalWrite(Int4, LOW)), totul cu condiția ca sensul de deplasare să fie deja setat.

În Figura 3.2.2 se poate vedea ce valoare numerică se asociază pinilor digital de pe placă în Arduino IDE, iar în Figura 3.2.3 se poate observa care sunt pinii cu PWM.

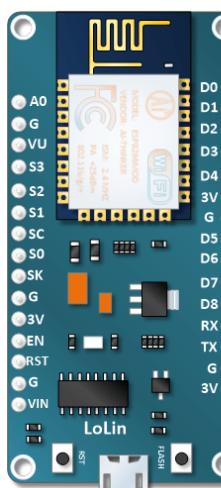


Figura 3.2.2 – Asocierea numerică a pinilor

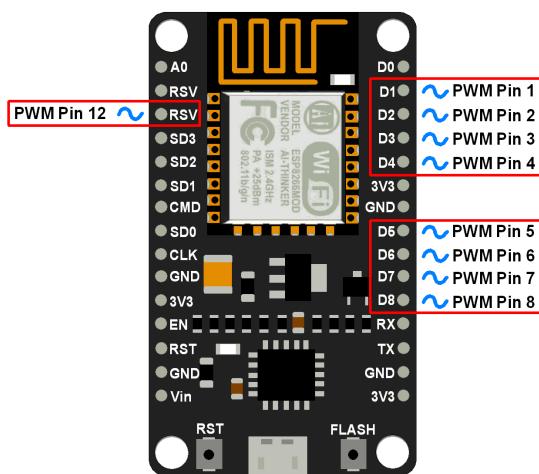


Figura 3.2.3 – PWM pentru NodeMCU

10. Sursa de alimentare directă – Baterie Pb 12V

11. Circuit de încărcare

12. Sursa de alimentare indirectă – Panou solar montat pe mecanism DAST

Despre componente 11 și 12 se va discuta detaliat în Capitoul 5

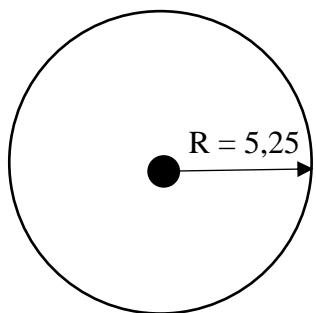
13. Întrerupător

Întreaga platformă va primi tensiune de la baterie doar după apăsarea unui întrerupător montat pe carcasa realizată ulterior.

Așa cum am menționat, scopul proiectului este de a permite atât controlul manual cât și cel automatizat al sistemului mobil. Pentru a putea automatiza mecanismul de deplasare, trebuie

să știm exact viteza de rotație a roților. Pe baza acestei informații și știind raza roților prinse pe axul motor, se poate calcula cu exactitate în cât timp se deplasează platforma pe o distanță dorită în condiții ideale (teren drept fără intreruperi).

Pentru parcurgerea unor cicli de deplasare, utilizatorul nu poate seta viteza platformei și aceasta va fi setată ca fiind maximă. Deoarece microprocesorul de la ESP8266 folosește un registru pe 10 biți, valoare analogică maximă va fi de 1023 ($2^{10} - 1$). La valoare de 1023, rotația axului motor va fi de 72,5 rot/min.



Cunoscând raza unei roți ca fiind 5,25 cm, obținem că lungimea totală a cercului este $2 \cdot \pi \cdot R = 32,99$ cm.

Așadar, la o rotație completă platforma se deplasează 32,99 cm și într-un minut $72,5 \cdot 32,99 = 2391$ cm. Pentru a ne putea folosi această informație, trebuie să convertim din minute în milisecunde.

Astfel obținem că mașina se deplasează 0,039 cm pe milisecundă și pentru a calcula timpul de deplasare pe distanță x ne vom folosi mai departe de următoarea formulă $t = x \text{ cm} \cdot 60 \cdot 1000 \text{ ms}/2391 \text{ cm} = x \cdot 25,1 \text{ ms}$.

Aceste calcule le voi utiliza la implementarea algoritmului pentru efectuarea de cicli de deplasare precum și la algoritmul de revenire.

Pe baza mai multor teste, s-a constatat că la viteza maximă și în mod ideal, platforma efectuează o întoarcere la 90° în 1735 ms, iar pentru aceasta se pierd 24 cm în raport cu roata din spate stânga și 10 cm în raport cu poziția roții din spate dreapta. Așadar, la fiecare pas efectuat de către platformă, vom fi nevoiți ca înainte de viraj să dăm cu spatele 24 de cm și după viraj, încă alți 10 cm. Astfel se vor impune trei delay-uri:

1. Delay de viraj: 1735 ms.
2. Delay înainte de viraj: $24 \cdot 25,1 = 602$ ms
3. Delay după viraj: $10 \cdot 25,1 = 251$ ms

De asemenea, pentru implementare trebuie avut în vedere și distanța de la axul roților din față până la cel din spate de 16 cm, considerând că aceasta este dimensiunea mașinii într-un plan XoY virtual conform cu Subcapitolul 3.4.

Pentru controlul manual, utilizatorul poate seta din aplicație viteza de deplasare a platformei. Există 5 viteze : I – 57,4 rot/min, II – 60,2 rot/min, III – 63,8 rot/min, IV – 68 rot/min, V – 72,5 rot/min.

3.3 Alegerea serviciului BaaS

Deoarece aplicația IoT este realizată în Android, în loc să mă folosesc de protocolul http, am ales să folosesc un serviciu BaaS care este compatibil atât cu Android cât și cu placa de dezvoltare NodeMCU ESP8266. Placa de dezvoltare joacă local rol de server când clientul (aplicația Android) așteaptă finalizarea programului și transmiterea distanțelor de la senzori, cât și de client când aplicația Android este folosită la controlul manual sau când se setează altă configurație pentru conectarea la internet a platformei. În Capitoul 4 se vor explica mai pe larg aceste detalii.

Servicul Backend as a Service este un intermediar, folosit ca și server pentru cele două sisteme (aplicație și platformă) unde se stochează datele venite din ambele părți pe cloud sau în baza de date. Folosind un serviciu BaaS nu trebuie să ne facem griji de scrierea de API-uri, sau de gestionarea serverelor.



Figura 3.3.1 – Serviciul BaaS

Serviciul BaaS folosit este de la Google și poartă numele de Firebase, dezvoltat pe baza platformei Google Cloud. Pentru proiectul SmartPlatform se putea folosi doar Google Cloud, însă “Firebase are integrate funcționalități foarte avansate și robuste de care are nevoie o aplicație web sau mobilă precum procesul de autentificare, la care se adaugă trimiterea unui email de validare a contului, resetarea parolei, baza de date în timp real, cloud, hosting și monitorizarea de activități, toate realizate foarte generic pentru a putea fi folosite în multe limbaje de programare.”^[16]

Am ales Firebase deoarece am dorit să creez o aplicație de calitate, neavând cunoștințele necesare pentru creare tuturor API-urilor care permit folosirea serviciilor integrate în aplicația SmartPlatform. Există firme foarte mari care folosesc acest serviciu pentru dezvoltarea de aplicații și am avut încredere în el.

De regulă, dacă se optează pentru o aplicație Android, se alege Firebase deoarece el este integrat în Android Studio sub forma unui SDK (Software Development Kit) și ne permite conectarea directă la baza de date.

Baza de date de la Firebase este una NoSQL, având un format JSON, ușor de serializat și deserializat. Folosind clasele speciale din Java pentru Firebase, nu trebuie să ne mai facem griji legate de apariția de excepții sau execuții în paralel de task-uri asincrone.

JSON (JavaScript Object Notation) este un format structurat de date ce permite interschimbul de informații între aplicații. În cazul nostru folosim acest tip de structură pentru a trimite în baza de date informații legate de starea platformei sau acțiunile utilizatorului ce folosește aplicația.

3.4 Dezvoltarea mecanismelor de control tip IoT compatibil cu serviciul BaaS

Microchip-ul Wi-Fi ESP8266 împreună cu microprocesorul său este utilizat într-o gamă foarte extinsă de aplicații IoT și platforme hardware, fiind compatibilă cu numeroarase servicii de Cloud precum Azure, Google Cloud sau Amazon Cloud Services. Există și alte chipuri ce pot fi folosite pentru aceste aplicații, însă pentru compatibilitatea între serviciul de control oferit clientului (aplicația Android) și platforma fizică, s-a păstrat serviciul Firebase. Deși serviciul are și posibilitatea de stocare pe cloud, pentru dezvoltarea mecanismelor de control de tip IoT, atât platforma cât și aplicația comunică prin intermediul bazei de date în timp real, pusă la dispoziție de serviciul BaaS ales.

De asemenea, pentru modulul Wi-Fi ESP8266 integrat în platforma IoT NodeMCU 1.0 folosită de noi, s-a dezvoltat în urmă cu câțiva ani o librărie specială numita FirebaseArduino, ce permite transferul și receptia securizată de date către Firebase, pe baza protocolului TCP/IP și a datelor în format JSON.

Deoarece baza de date oferită serviciul BaaS este una NoSQL, aşa cum am explicitat în subcapitolul anterior, pentru a trimite date, putem să serializăm și să deserializăm date doar în format JSON. Înținând cont de toate acestea, pentru ca librăria FirebaseArduino să poată funcționa, avem nevoie de librăria ArduinoJson (ce ne permite creare de json-uri statice sau dinamice, scrierea în baza de date sau citirea datelor din baza de date), precum și de librăria ESP8266WiFi folosită pentru a asigura conexiunea la internet a modului Wi-Fi.

Pentru conectarea la internet avem nevoie se furnizăm un SSID și un Password care să fie valid, iar pentru conectarea la Firebase, avem nevoie de numele bazei de date, precum și de certificatul unic oferit de Firebase pentru fiecare proiect în parte. Acestea se obțin din interiorul aplicației. Întreg procesul de conectarea a fost realizat de mine printr-o funcție prezentată în Figura 3.4.1.

```
void wifiConnect() {
    WiFi.begin(My_SSID, My_PASS); // Începe conexiunea
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000); // dacă încă nu s-a conectat așteaptă o secundă și reîncearcă
    }

    Firebase.begin(My_FIREBASE_HOST, My_FIREBASE_AUTH); // Începe conexiunea cu Firebase

    StaticJsonBuffer<200> jsonBuffer; // Crează un buffer static
    JsonObject& root = jsonBuffer.createObject(); // Stocăză într-un nou obiect Json
    root["blocked"] = ""; // adaugă noduri
    root["distanceFront"] = 1000000.0;
    root["distanceBack"] = 1000000.0;
    root["state"] = "R-Stop";
    JsonObject& defaultSettings = root.createNestedObject("default"); // crează un nod cu alte subnoduri
    defaultSettings["ssid"] = My_SSID; // adaugă subnoduri
    defaultSettings["password"] = My_PASS;
    Firebase.set("/devices/SmartPlatform00001TX", root);
    while (Firebase.failed()) {
        Firebase.set("/devices/SmartPlatform00001TX", root);
    }
}
```

Figura 3.4.1 – Conectarea platformei la Firebase

Înțial, după ce s-a alimentat platforma, aceasta nu este blocată pe nicio direcție deoarece distanța față de obiectele din față și din spate este considerată ca fiind foarte mare (1000000.0) și nu există obstacole în apropiere. Înțial conexiunea la internet este considerată puternică. De asemenea, se stabilește care este parola și SSID-ul Wi-Fi-ului la care se conectează platforma în mod default, fiind cele stabilite pentru fiecare produs în parte.

SSID-ul inițial corespunde cu numele platformei sau cum se va numi în Capitolul 4, deviceID. Acesta este SmartPlatformNumărDeOrdineTX, unde numărul de ordine începe de la 00001, iar parola este una specifică pentru fiecare dispozitiv în parte. Se observă că platforma este oprită inițial fiind în stare „R-Stop”.

Dacă operațiunea s-a realizat cu succes, în baza de date va apărea structura din Figura 3.4.2 și o va suprascrie pe cea anterioră.

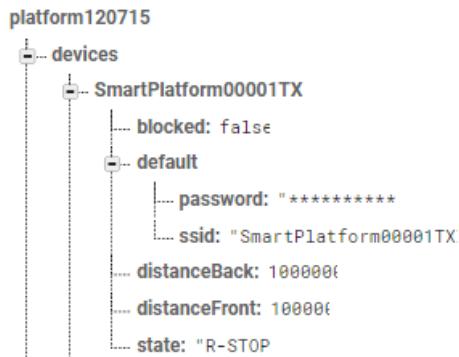


Figura 3.4.2 – Starea inițială a platformei conectate

Ulterior, prin conectarea la platformă din aplicația IoT realizată în Android, structura se modifică. De exemplu, după conectare apare flag-ul “connected”, care ne spune dacă există un utilizator conectat la sistemul fizic. De asemenea, dacă utilizatorul alege să controleze platforma manual, starea se va modifica în funcție de direcția, sensul și viteza mașinii (se folosesc metode precum goFront, goBack, turnLeft, turnRight, stopCar, setSpeed și combinații între acestea).

La controlul automatizat, utilizatorul poate încărca un program care rulează pe platformă. Un program conține un nume, mai mulți pași cu un counter pentru a putea parcurge ulterior fiecare pas în vederea execuției programului și un flag “comeback” care dacă este activ (true) ne spune că utilizatorul dorește revenirea înapoi la punctul de start, în aceeași poziție. Algoritmii de execuție și de revenire vor fi detaliați în acest capitol. Un pas (step) conține detalii referitoare la tipul de mișcare executată de platformă. El are o direcție de deplasare (înainte, înapoi, stânga, dreapta), o distanță (centimetri sau metri), un număr de intervale la care platforma se oprește și timpul de execuție al procesului industrial pentru care este folosită platforma, pentru fiecare interval (secunde, minute, ore). Aceste date se convertesc ulterior în centimetri pentru distanță și milisecunde pentru întârziere, pentru a putea transmite corect datele platformei, așa cum am explicat la finalul Subcapitolului 3.2.

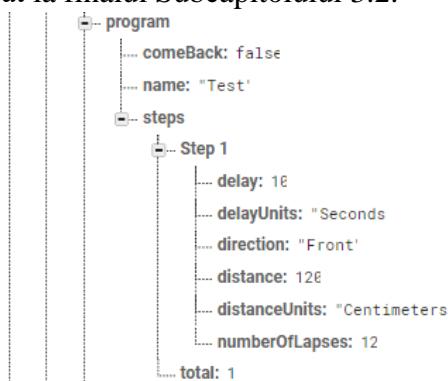


Figura 3.4.3 – Program încărcat

Dacă nu există un program încărcat pe platformă, platforma poate fi controlată manual pe baza stărilor diferite în care se poate afla și o funcție ce se apelează după acea stare, setată din aplicația Android. La controlul manual platforma trimite date în timp real la server de unde sunt preluate de utilizator, referitoare la distanța față de obiectele aflate în direcția de mers curentă a platformei. Dacă se depășește limita de siguranță acceptată platforma se oprește.

Dacă starea este „R_Speed1”, „R_Speed2”, „R_Speed3”, „R_Speed4” sau „R_Speed5”, se apelează funcția setCarSpeed care primește ca parametru viteza de croazieră a mașinii. Este bine de reținut că la viraje, viteza este mereu maximă („R_Speed5”) și nu poate fi schimbată. Se salvează viteza curentă ca după revenirea dintr-un viraj, platforma să se deplaseze în continuare cu viteza setată din aplicația IoT. Metoda setCarSpeed setează viteza de rotație a motoarelor din stânga și dreapta.

```
void setCarSpeed(int speed) {
    carSpeed = speed;
    analogWrite(rightEn, speed);
    analogWrite(leftEn, speed);
}
```

Dacă starea este „H-F” (high front) se apelează funcția goFront. Metoda setează direcția de mers înainte pentru cele două seturi de motoare.

```
void goFront() {
    digitalWrite(leftFront, HIGH);
    digitalWrite(leftBack, LOW);
    digitalWrite(rightFront, HIGH);
    digitalWrite(rightBack, LOW);
}
```

Când starea se modifică din „H-F” în „L-F” (low front), se apelează funcția stopCar care primește ca parametru un index ce ne spune ce roți vrem să deblocăm pentru a ieși dintr-un viraj sau dacă dorim să blocăm toate roțile. În acest caz, se apelează cu un index default (0), și se opresc toate roțile.

```
void stopCar(int type) {
    switch(type) {
        case 1 : digitalWrite(rightFront, HIGH); // pornesc doar roțile din stânga și merg înainte
        digitalWrite(rightBack, LOW);
        break;
        case 2 : digitalWrite(rightFront, LOW); // pornesc doar roțile din stânga și merg înapoi
        digitalWrite(rightBack, HIGH);
        break;
        case 3 : digitalWrite(leftFront, HIGH); // pornesc doar roțile din dreapta și merg înainte
        digitalWrite(leftBack, LOW);
        break;
        case 4 : digitalWrite(leftFront, LOW); // pornesc doar roțile din dreapta și merg înapoi
        digitalWrite(leftBack, HIGH);
        break;
        default: digitalWrite(leftFront, LOW); // opresc toate roțile
        digitalWrite(leftBack, LOW);
        digitalWrite(rightFront, LOW);
        digitalWrite(rightBack, LOW);
        break;
    }
}
```

Figura 3.4.4 – Oprire / Pornire roți

Pentru starea „H-B” (high back) folosim funcția goBack, care este opusă din punct de vedere al semnalelor active din metoda goFront și pentru starea „L-B” (low back) se folosește funcția stopCar, tot cu index default pentru a opri toate roțile.

```
void goBack() {  
    digitalWrite(leftFront, LOW);  
    digitalWrite(leftBack, HIGH);  
    digitalWrite(rightFront, LOW);  
    digitalWrite(rightBack, HIGH);  
}
```

Pentru starea „H-L_F” (high left front) sau „H-L_B” (high left back), se folosește funcția turnLeft() cu condiția ca inițial platforma controlată manual să fie în starea „H-F” respectiv „H-B”. Metoda blochează roțile din stânga ale mașinii.

```
void turnLeft() {  
    analogWrite(rightEn, speed5);  
    digitalWrite(leftFront, LOW);  
    digitalWrite(leftBack, LOW);  
}
```

La trecerea în starea „L-L_F” (low left front) sau „L-L_B” (low left back) se apelează funcția stopCar cu indexul 1 respectiv 2 și mașina continuă să se deplaseze înainte respectiv înapoi.

Pentru starea „H-R_F” (high righ front) sau „H-R_B” (high right back) se folosește funcția turnRight() cu condiția ca inițial platforma controlată manual să fie în starea „H-F” respectiv „H-B”. Metoda blochează roțile din dreapta ale mașinii.

```
void turnRight() {  
    analogWrite(leftEn, speed5);  
    digitalWrite(rightFront, LOW);  
    digitalWrite(rightBack, LOW);  
}
```

La trecerea în stările „L-R_F” (low left front), „L-R_B” (low left back), se apelează funcția stopCar cu indexul 3 respectiv 4 și mașina continuă să se deplaseze înainte respectiv înapoi.

Tot la controlul manual, platforma permite înregistrarea distanțelor față de obstacole și transmiterea lor către server. Deoarece ESP8266 are un microprocessor cu două timere, din care unul este folosit de microchip-ul Wi-Fi, și prin urmare nu se poate realiza un sistem de întreruperi care să permită trimiterea de date de la senzori continuu. De aceea, datele se trimit cât timp ne aflăm în aceeași poziție și se înregistrează doar măsurătorile de pe sensul curent de deplasarea al platformei pentru a scurta timpul de calcul și răspuns al sistemului. Știind sensul, utilizatorul este avertizat dacă există obstacole ce pot obstrucționa deplasarea platformei și care sunt de interes la momentul respectiv.

Conform cu Subcapitolul 3.2, știm că senzorii HC-SR04 folosesc la comun pinul de Trig pentru transmiterea undelor sonore și astfel este imposibilă transmiterea simultană de date de la ambii senzori. Pentru calculul propriu-zis al distanței procedăm ca în Figura 3.4.5.

```
digitalWrite(trig, LOW);
delayMicroseconds(2);
digitalWrite(trig, HIGH);
delayMicroseconds(10);
digitalWrite(trig, LOW);

// Măsoară răspunsul de la senzori

float duration = pulseIn(echo, HIGH);
pulseIn(echo, LOW);
// Determină distanța pe baza duratei
// Folosește 343 metri pe secundă ca viteză a sunetului

float distance = (duration / 2) * 0.0343;
```

Figura 3.4.5 – Calculul distanței

Știind sensul de deplasare, trimitem la senzor date referitoare la acea direcție folosindu-ne de metoda setFloat din librăria FirebaseAndroid scrisă în C++.

Aceste metode de tip set (setBool, setString, setInt, setFloat) primesc ca parametru un JSON și o valoare a nodului. Există metode de tip get care primesc ca parametru doar un JSON și permit accesul la valoarea din nodul respectiv. Mai multe detalii se pot găsi în documentație.^[17]

Exemplu pentru deplasarea în față:

```
Firebase.setFloat("/devices/SmartPlatform00001TX/distanceFront", distance);
// ne asigurăm că s-a putut comunica cu Firebase
while (Firebase.failed()) {
    Firebase.setFloat("/devices/SmartPlatform00001TX/distanceFront",distance);
}
```

Funcția de calcul a distanțelor efectuează câte trei măsurători repetitive, iar în cazul în care distanța medie a celor 3 valori obținute este mai mică decât o valoare minimă stabilită (30 cm în cazul de față), platforma se oprește și atributul blocked ia valoarea „FRONT” sau „BACK” pentru a ști pe ce sens de deplasare a apărut obstacolul. Starea platformei devine „R-Stop”.

După cum am specificat anterior, dacă din aplicația mobilă se încarcă un program, atunci utilizatorul nu mai are acces la controlul manual și este nevoie să aștepte finalizarea programului sau să opreasă programul. Programul este format din pași execuției în linie dreaptă (față-spate sau stânga-dreapta). Așa cum am menționat, un pas are o orientare, o distanță parcursă cu un ordin de mărime specificat, un număr de intervale în care se împarte distanța și o întârziere efectuată la fiecare interval cu un ordin de mărime specificat.

Pentru ca platforma să poată reveni în poziția de start la cererea utilizatorului, a trebuit să se realizeze un algoritm pentru a ști cu exactitate poziția curentă și cea anterioară ei, cu scopul de a identifica direcția și sensul de deplasare curent. Pentru aceasta m-am folosit de un sistem de coordonate virtual XoY.

Algoritmul consideră punctul de coordonate (0, 0) ca fiind perpendiculara dintre roata din spate stânga și sol la momentul de start al programului. La fiecare pas parcurs se salvează coordonatele punctului anterior $P_A(a_x, a_y)$, iar în funcție de sensul și direcția de deplasare se modifică coordonatele adunând sau scăzând distanța de deplasare a pasului convertită în centimetric, pe OX sau pe OY după caz și se salvează ca un nou punct curent $P(x, y)$. La final se începe execuția pasului.

Pentru a înțelege mai bine mecanismul vom considera următorul exemplu: Fie un program ce trebuie să execute trei pași ca în Figura 3.4.6. Algoritmul va înregistra la fiecare pas cele două perechi de coordonate și va efectua mișcarea aferentă pasului după cum este prezentat în Figura 3.4.7.

- **Step 1**
 - delay: 1
 - delayUnits: Seconds
 - direction: Front
 - distance: 50
 - distanceUnits: Centimeters
 - numberOfLapses: 2
- **Step 2**
 - delay: 0
 - delayUnits: Seconds
 - direction: Left
 - distance: 0.5
 - distanceUnits: Meters
 - numberOfLapses: 1
- **Step 3**
 - delay: 5
 - delayUnits: Seconds
 - direction: Right
 - distance: 100
 - distanceUnits: Centimeters
 - numberOfLapses: 4

Figura 3.4.6 – Pașii unui program de test

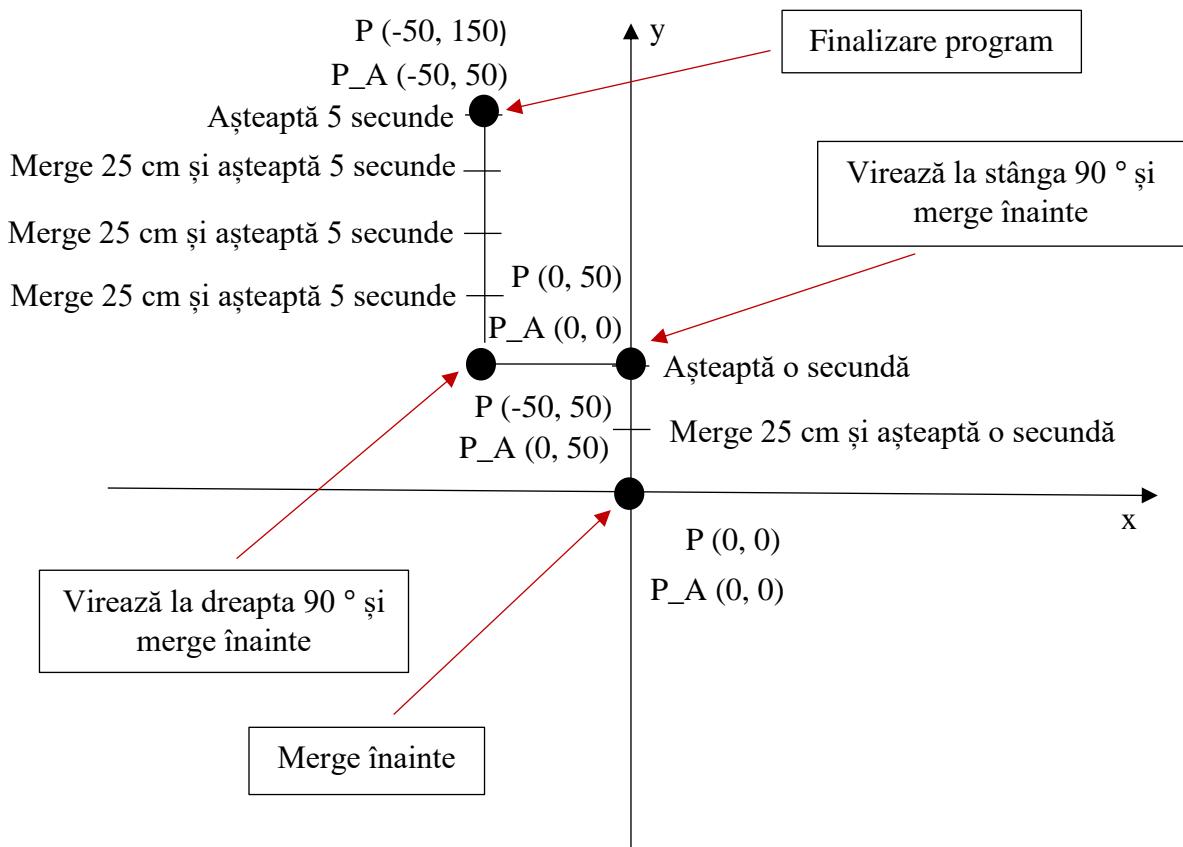
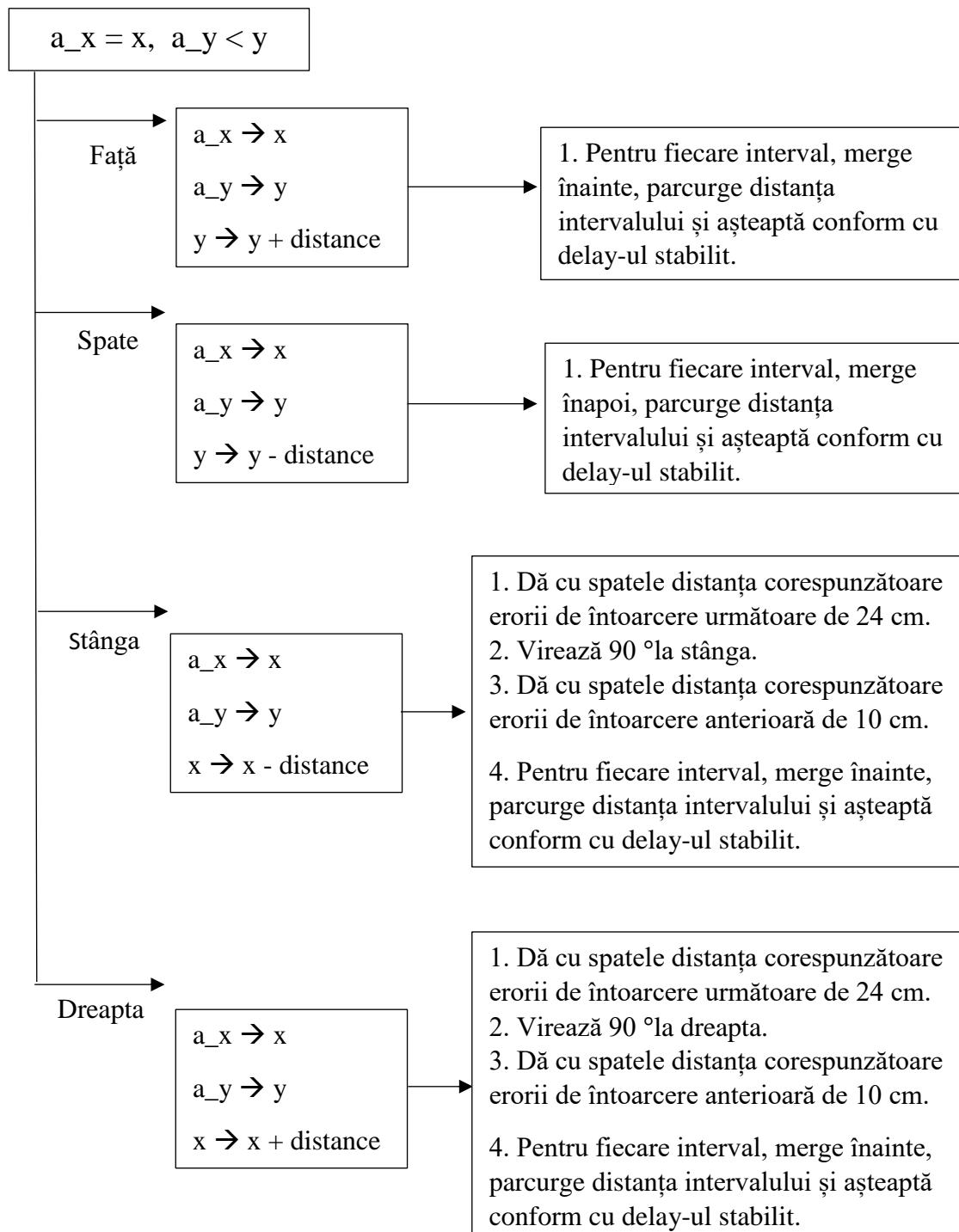


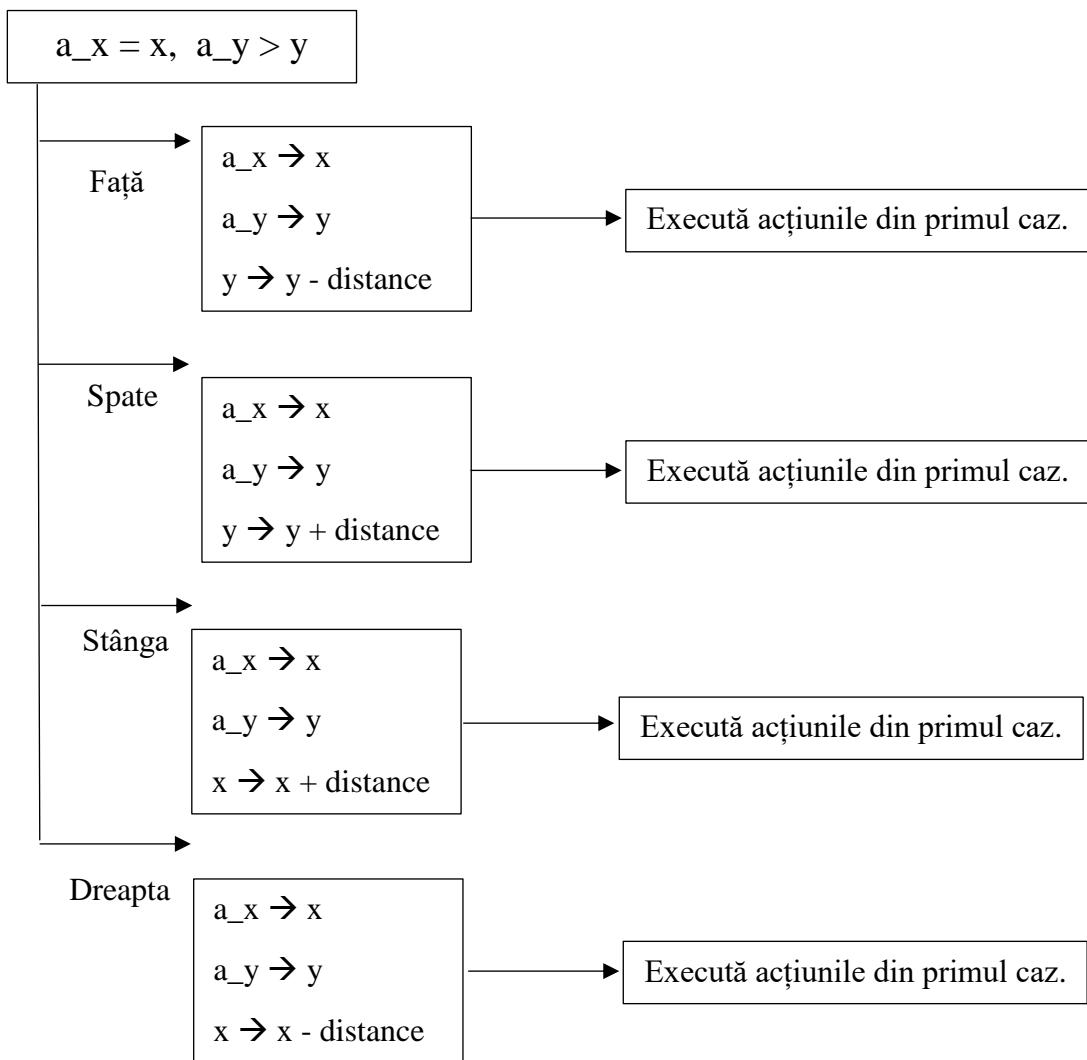
Figura 3.4.7 – Interpretarea programului de test

Pentru a asigura funcționalitatea corectă a algoritmului, se iau în calcul cele două puncte la fiecare pas. Scopul este de a obține coordonata pentru roata din spate stânga, în raport cu planul virtual. Algoritmul este prezentat mai jos și ia în calcul toate pozițiile în care se află platforma în raport cu planul XoY.

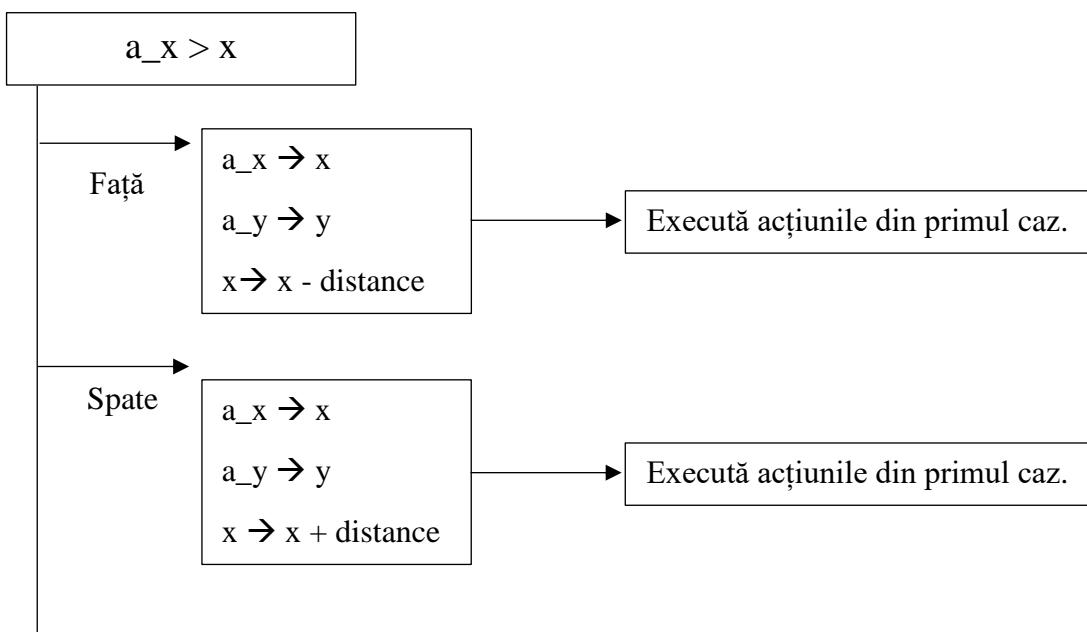
Dacă:

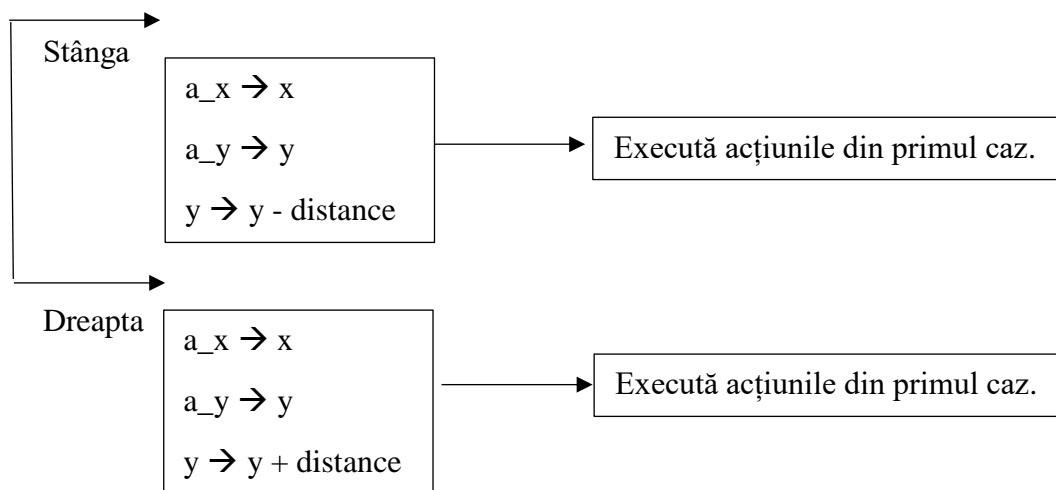


Dacă:

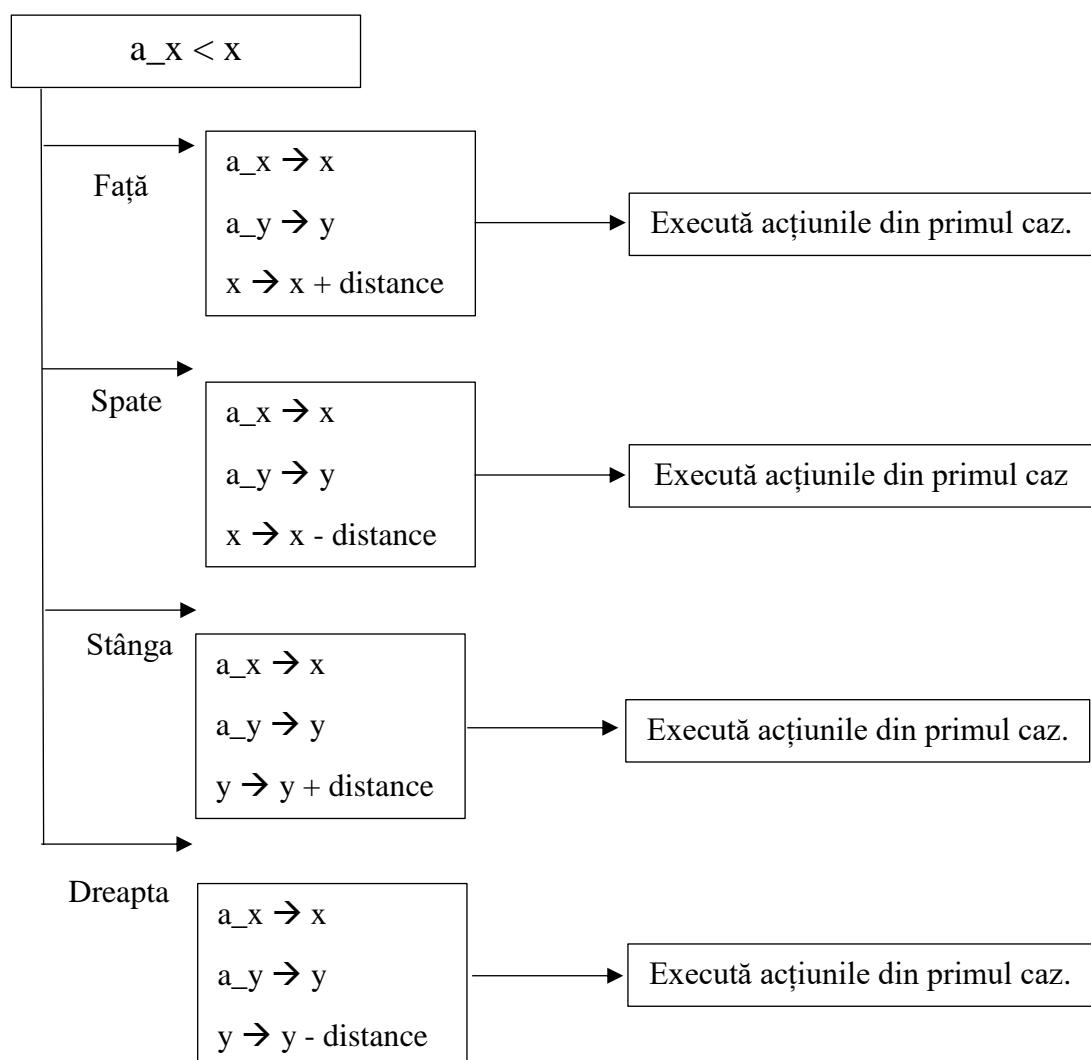


Dacă:





Dacă:



De exemplu, pentru cazul în care x anterior este mai mare decât x și mașina este poziționată pe direcția OX, cu sensul orientat în dreapta, avem următoare secțiune de cod:

```
if (*lx > *x) {  
    if (stepDirection == 1) { // dacă merge înainte  
        *lx = *x;  
        *ly = *y;  
        *x = *x - stepDistance;  
        for (int j = 1; j <= numberOfLapses; j++) {  
            goFront();  
            delay((long)timePerLapse);  
            stopCar(front_back);  
            delay((long)stepDelay);  
        }  
    } else {  
        if (stepDirection == 2) { // dacă merge înapoi  
            *lx = *x;  
            *ly = *y;  
            *x = *x + stepDistance;  
            for (int j = 1; j <= numberOfLapses; j++) {  
                goBack();  
                delay((long)timePerLapse);  
                stopCar(front_back);  
                delay((long)stepDelay);  
            }  
        } else {  
            if (stepDirection == 3) { // dacă merge la stânga  
                *ly = *y;  
                *lx = *x;  
                *y = *y - stepDistance;  
                goBack();  
                delay(turnTimeFront_Back);  
                stopCar(front_back);  
                goFront();  
                turnLeft();  
                delay(turnTime);  
                stopCar(front_back);  
                goBack();  
                delay(turnTimeLeft_Right);  
            }  
            delay((long)timePerLapse);  
            stopCar(front_back);  
            delay((long)stepDelay);  
        }  
    }  
}  
}  
}  
}

delay((long)timePerLapse);  
stopCar(front_back);  
for (int j = 1; j <= numberOfLapses; j++) {  
    goFront();  
    delay((long)timePerLapse);  
    stopCar(front_back);  
    delay((long)stepDelay);  
}  
}  
} else {  
    if (stepDirection == 4) { // dacă merge la dreapta  
        *ly = *y;  
        *lx = *x;  
        *y = *y + stepDistance;  
        goBack();  
        delay(turnTimeFront_Back);  
        stopCar(front_back);  
        goFront();  
        turnRight();  
        delay(turnTime);  
        stopCar(front_back);  
        goBack();  
        delay(turnTimeLeft_Right);  
        stopCar(front_back);  
        delay((long)stepDelay);  
    }  
}  
}  
}
```

Pentru a reveni în poziția de start, se implementează algoritmul descris mai jos.

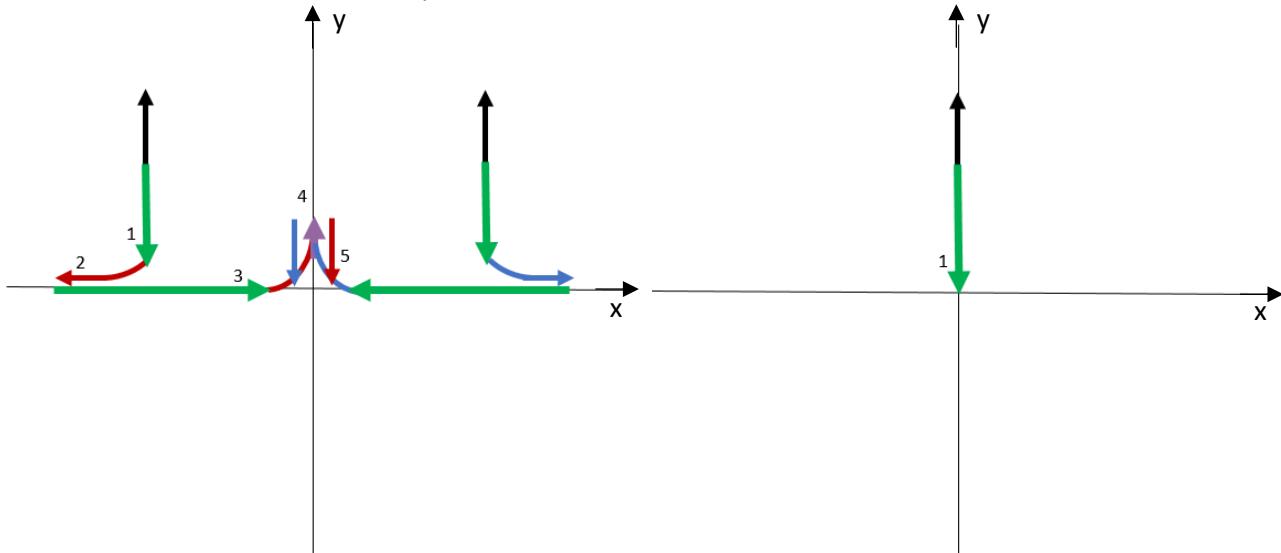


Figura 3.4.8 – Întoarcere pe yx^{\uparrow}

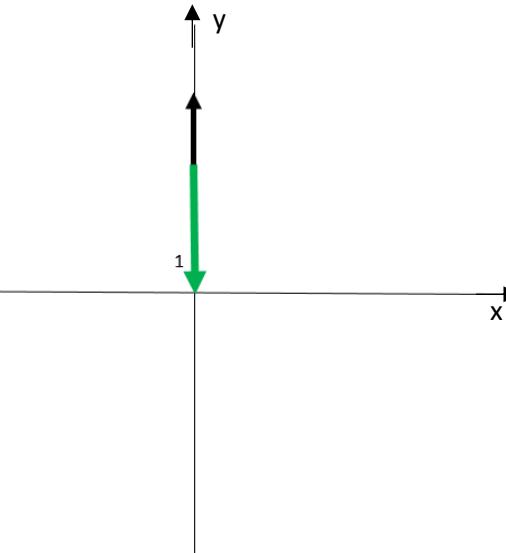


Figura 3.4.9 – Întoarcere pe y^{\uparrow}

Dacă platforma este orientată ca în Figura 3.4.8, pentru a reveni în poziția inițială se execută următorii pași:

1. Merge înapoi până la parcurgerea distanței totale de pe OY, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm.
2. Realizează o întoarcere cu spatele la 90° către dreapta / stânga în funcție de cadran.
3. Merge înainte până la parcurgerea distanței totale de pe OX, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm și adunând eroarea la întoarcerea anterioară de 10 cm.
4. Realizează o întoarcere cu față la 90° către stânga / dreapta în funcție de cadran.
5. Merge înapoi pentru a parcurge eroarea la întoarcerea anterioară de 10 cm.

Dacă platforma nu s-a deplasat pe OX în programul rulat, conform Figurii 3.4.9, pentru a reveni în poziția inițială se execută o mișcare înapoi în linie dreaptă pe distanța OY.

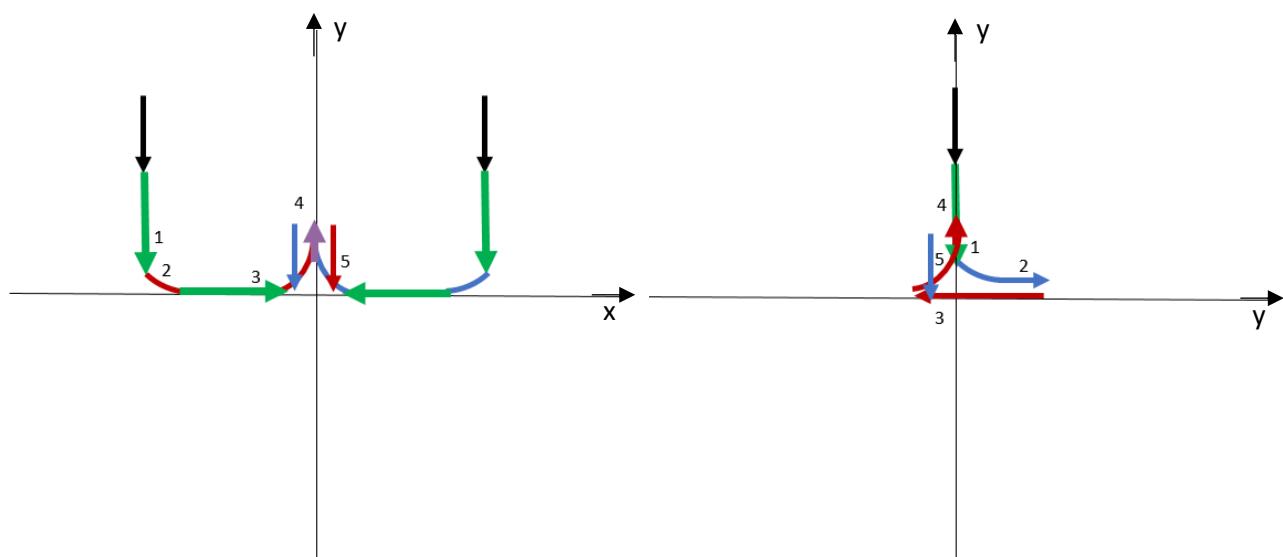


Figura 3.4.10 – Întoarcere pe yx^{\downarrow}

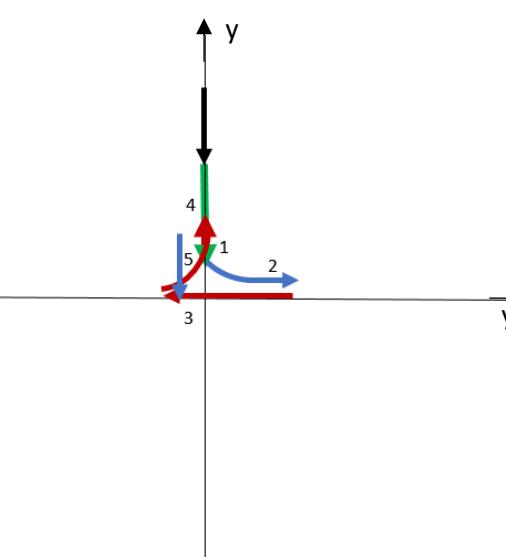


Figura 3.4.11 – Întoarcere pe y^{\downarrow}

Dacă platforma este orientată ca în Figura 3.4.10, pentru a reveni în poziția inițială se execută următorii pași:

1. Merge înainte până la parcurgerea distanței totale de pe OY, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm precum și distanța dintre axurile roțiilor (lungimea platformei).
2. Realizează o întoarcere cu față la 90° către stânga / dreapta în funcție de cadran.
3. Merge înainte până la parcurgerea distanței totale de pe OX, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm și adunând eroarea la întoarcerea anterioară de 10 cm.
4. Realizează o întoarcere cu față la 90° către stânga / dreapta în funcție de cadran.
5. Merge înapoi pentru a parcurge eroarea la întoarcerea anterioară de 10 cm.

Dacă platforma nu s-a deplasat pe OX în programul rulat, conform Figurii 3.4.11, pentru a reveni în poziția inițială aceasta execută următorii pași.

1. Merge înainte până la parcurgerea distanței totale de pe OY, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm.
2. Realizează o întoarcere cu față la 90° către stânga.
3. Merge înapoi până la parcugerea distanței formată din suma dintre eroarea de întoarcere anterioară de 10 cm și eroarea de întoarcere următoare de 24 cm.
4. Realizează o întoarcere cu față la 90° către stânga.
5. Merge înapoi pentru a parcurge eroarea la întoarcerea anterioară de 10 cm.

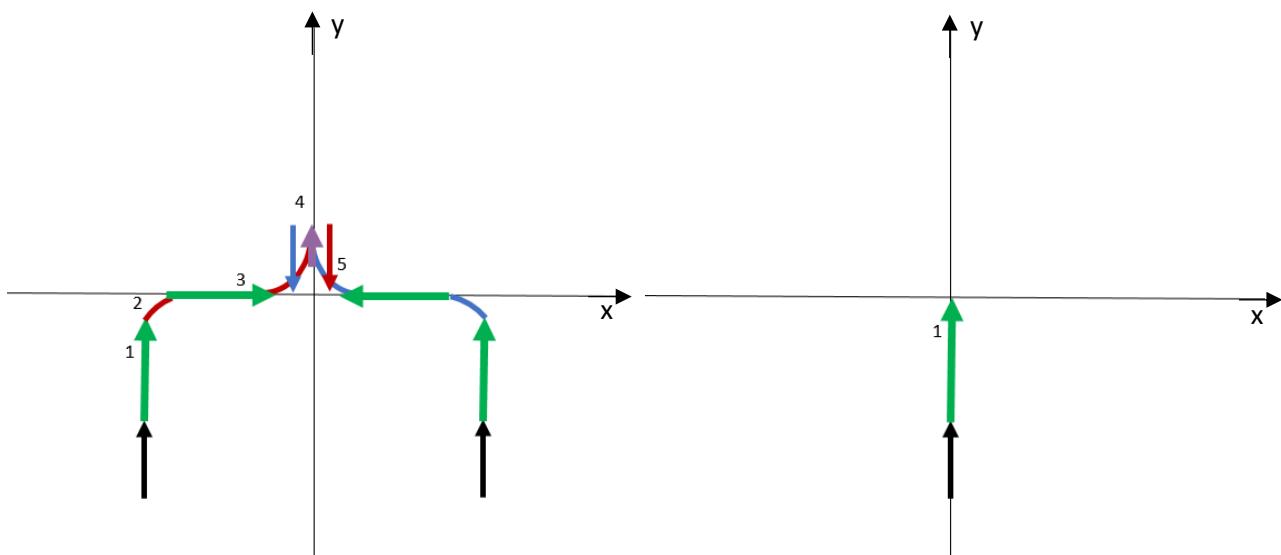


Figura 3.4.12 – Întoarcere pe -yx

Figura 3.4.13 – Întoarcere pe -y

Dacă platforma este orientată ca în Figura 3.4.12, pentru a reveni în poziția inițială se execută următorii pași:

1. Merge înainte până la parcugerea distanței totale de pe OY, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm precum și distanța dintre axurile roțiilor (lungimea platformei).
2. Realizează o întoarcere cu față la 90° către dreapta / stânga în funcție de cadran.

3. Merge înainte până la parcurgerea distanței totale de pe OX, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm și adunând eroarea la întoarcerea anterioară de 10 cm.
4. Realizează o întoarcere cu față la 90° către stânga / dreapta în funcție de cadran.
5. Merge înapoi pentru a parcurge eroarea la întoarcerea anterioară de 10 cm.

Dacă platforma nu s-a deplasat pe OX în programul rulat, conform Figurii 3.4.13, pentru a reveni în poziția inițială se execută o mișcare înapoi în linie dreaptă pe distanța OY din care se scade distanța dintre axul roților (lungimea platformei).

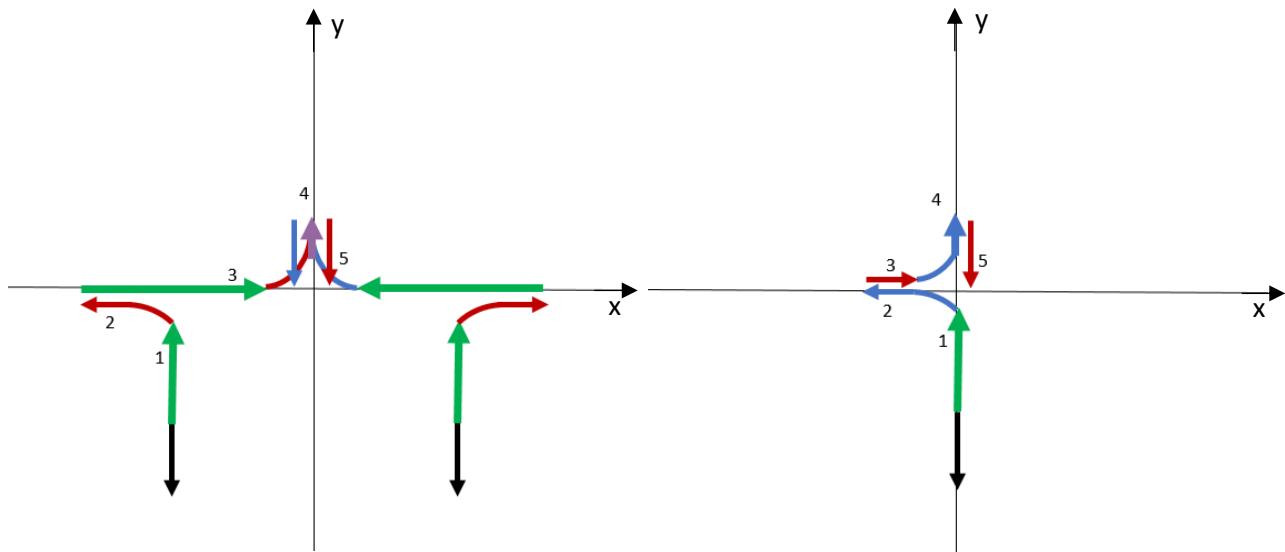


Figura 3.4.14 – Întoarcere pe -yx ↓

Figura 3.4.15 – Întoarcere pe -y ↓

Dacă platforma este orientată ca în Figura 3.4.14, pentru a reveni în poziția inițială se execută următorii pași:

1. Merge înapoi până la parcurgerea distanței totale de pe OY, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm.
2. Realizează o întoarcere cu spatele la 90° către dreapta / stânga în funcție de cadran.
3. Merge înainte până la parcurgerea distanței totale de pe OX, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm și adunând eroarea la întoarcerea anterioară de 10 cm.
4. Realizează o întoarcere cu față la 90° către stânga / dreapta în funcție de cadran.
5. Merge înapoi pentru a parcurge eroarea la întoarcerea anterioară de 10 cm.

Dacă platforma nu s-a deplasat pe OX în programul rulat, conform Figurii 3.4.15, pentru a reveni în poziția inițială aceasta execută următorii pași.

1. Merge înapoi până la parcurgerea distanței totale de pe OY, scăzând din aceasta eroarea la întoarcere următoare de 24 cm.
2. Realizează o întoarcere cu spatele la 90° către stânga.
3. Merge înainte până la parcurgerea distanței formată din eroarea întoarcerii anterioare de 10 cm.
4. Realizează o întoarcere cu față la 90° către stânga.
5. Merge înapoi pentru a parcurge eroarea la întoarcerea anterioară de 10 cm.

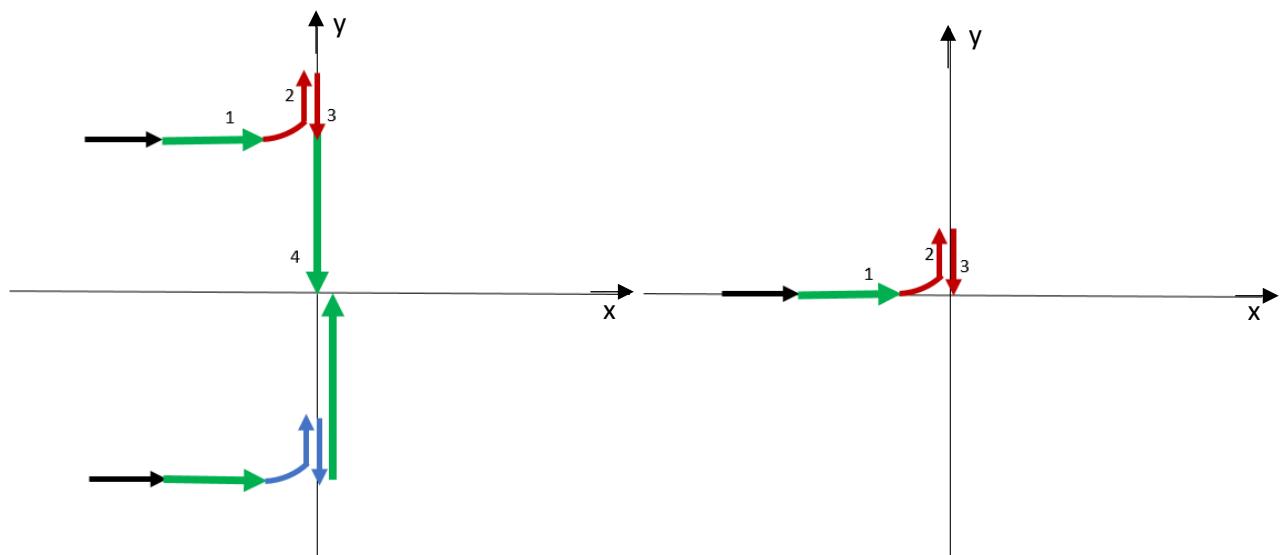


Figura 3.4.16 – Întoarcere pe -xy →

Figura 3.4.17 – Întoarcere pe -x →

Dacă platforma este orientată ca în Figura 3.4.16, pentru a reveni în poziția inițială se execută următorii pași:

1. Merge înainte până la parcurgerea distanței totale de pe OX, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm precum și distanța dintre axurile roților (lungimea platformei).
2. Realizează o întoarcere cu față la 90° către stânga.
3. Merge înapoi pentru a parcurge eroarea la întoarcerea anterioară de 10 cm.
4. Merge înapoi / înainte în funcție de cadran, până la parcurgerea distanței totale de pe OY.

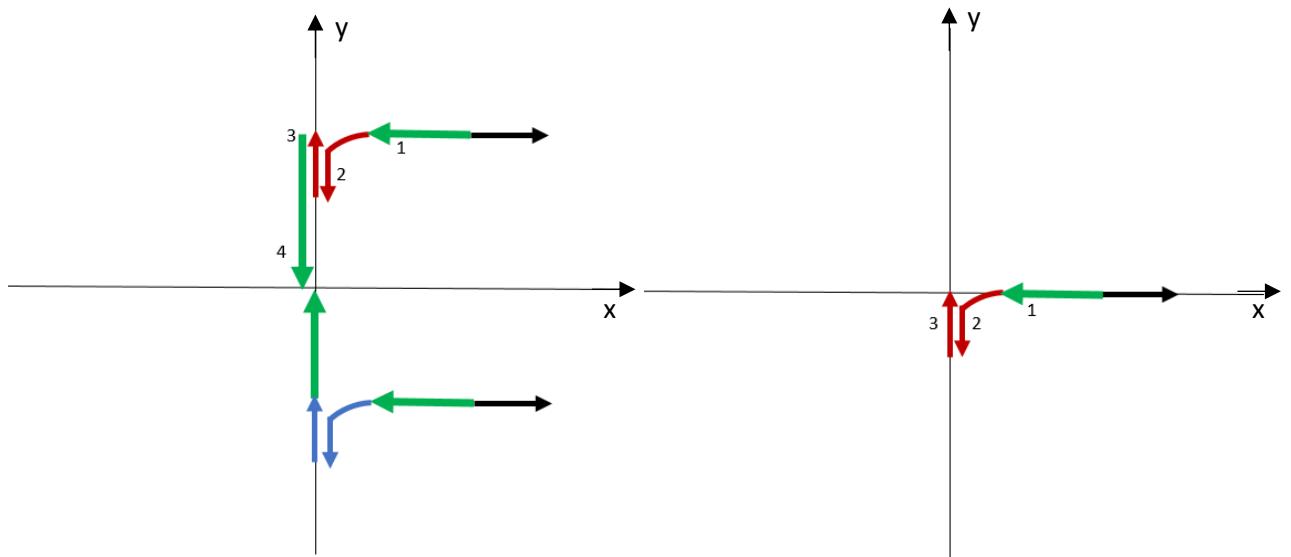


Figura 3.4.18 – Întoarcere pe xy →

Figura 3.4.19 – Întoarcere pe x →

Dacă platforma este orientată ca în Figura 3.4.18, pentru a reveni în poziția inițială se execută următorii pași:

1. Merge înapoi până la parcurgerea distanței totale de pe OX, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm.

2. Realizează o întoarcere cu spatele la 90° către dreapta.
3. Merge înainte pentru a parurge eroarea la întoarcerea anterioară de 10 cm.
4. Merge înapoi / înainte în funcție de cadran, până la parurgerea distanței totale de pe OY.

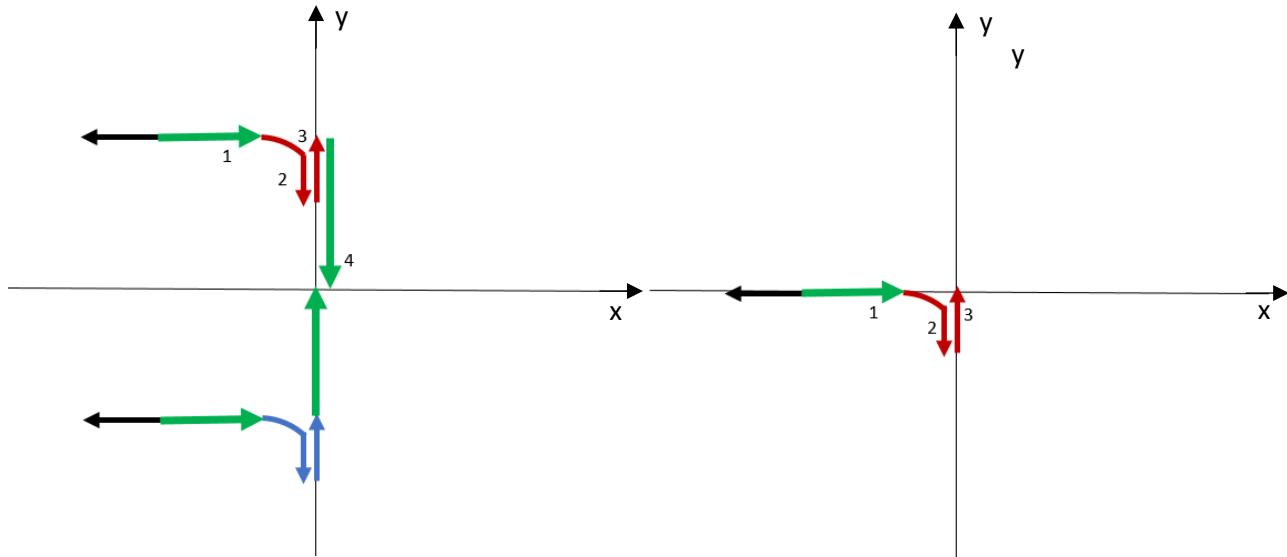


Figura 3.4.20 – Întoarcere pe -xy ←

Figura 3.4.21 – Întoarcere pe -x ←

Dacă platforma este orientată ca în Figura 3.4.20, pentru a reveni în poziția inițială se execută următorii pași:

1. Merge înapoi până la parurgerea distanței totale de pe OX, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm.
2. Realizează o întoarcere cu spatele la 90° către stânga.
3. Merge înainte pentru a parurge eroarea la întoarcerea anterioară de 10 cm.
4. Merge înapoi / înainte în funcție de cadran, până la parurgerea distanței totale de pe OY.

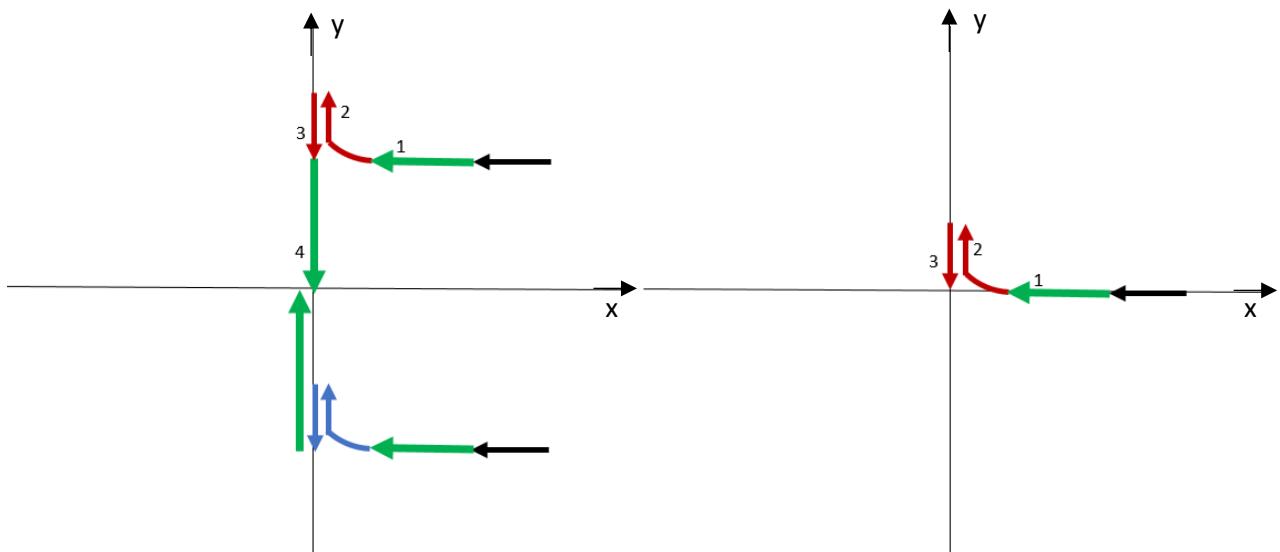


Figura 3.4.22 – Întoarcere pe xy ←

Figura 3.4.23 – Întoarcere pe x ←

Dacă platforma este orientată ca în Figura 3.4.22, pentru a reveni în poziția inițială se execută următorii pași:

1. Merge înapoi până la parcurgerea distanței totale de pe OX, scăzând din aceasta eroarea la întoarcerea următoare de 24 cm precum și distanța dintre axurile roților (lungimea platformei).
2. Realizează o întoarcere cu față la 90° către dreapta.
3. Merge înapoi pentru a parcurge eroarea la întoarcerea anterioară de 10 cm.
4. Merge înapoi / înaainte în funcție de cadran, până la parcurgerea distanței totale de pe OY.

Conform cu Figurile 3.4.17, 3.4.19, 3.4.21 și 3.4.23, dacă platforma nu s-a deplasat pe OY în programul rulat, pentru a reveni în poziția inițială aceasta execută primii 3 pași prezenți.

De exemplu, pentru revenirea la punctul de start în condiția în care după parcurgerea ultimului pas platforma este în poziția din Figura 3.4.10, se folosește următoarea secțiune.

```

if (x == lx && ly > y && y > 0) {
    oyTime = 60000*(abs(y)-24-carSize)/realSpeed;
    oxTime = 60000*(abs(x)-14)/realSpeed;
    goFront();
    delay((long)oyTime);
    if (x != 0) {
        if (x < 0) {
            turnLeft();
            delay(turnTime);
            stopCar(left_front);
        } else {
            turnRight();
            delay(turnTime);
            stopCar(right_front);
        }
        delay((long)oxTime);
        if (x < 0) {
            turnLeft();
        } else {
            turnRight();
        }
    }
}

```

<pre> delay(turnTime); stopCar(front_back); goBack(); delay(turnTimeLeft_Right); stopCar(front_back); } else { turnLeft(); delay(turnTime); stopCar(front_back); goBack(); delay(turnTimeLeft_Right); delay(turnTimeFront_Back); stopCar(front_back); goFront(); turnLeft(); delay(turnTime); stopCar(front_back); goBack(); delay(turnTimeLeft_Right); stopCar(front_back); } } </pre>	<pre> delay(turnTime); stopCar(front_back); goBack(); delay(turnTimeLeft_Right); stopCar(front_back); } else { turnLeft(); delay(turnTime); stopCar(front_back); goBack(); delay(turnTimeLeft_Right); stopCar(front_back); } } </pre>
---	---

Pentru explicații detaliate, luăm de exemplu cazul revenirii la punctul de start în condiția în care după parcurgerea ultimului pas platforma este în poziția din Figura 3.4.16.

```
if (lx < x && x < 0) {  
    oyTime = 60000*(abs(y))/realSpeed; // convertim în milisecunde  
    oxTime = 60000*(abs(x)-24-carSize)/realSpeed; // scăzând eroare la întoarcere 24 cm  
        // precum și distanța dintre  
        // axurile roțiilor pentru axa OX  
  
    goFront(); // 1. Merge înainte  
  
    delay((long)oxTime); // până la parcurgerea distanței totale de pe OX  
    turnLeft(); // 2. Realizează o întoarcere cu față la 90 ° către stânga.  
    delay(turnTime); // așteaptă până face o întoarcere completă  
    stopCar(front_back); // oprește platforma de tot  
  
    goBack(); // 3. Merge înapoi  
  
    delay(turnTimeLeft_Right); // pentru a parurge eroarea la întoarcerea anterioară de 10 cm.  
    stopCar(front_back); // oprește platforma de tot  
  
    // 4. Merge  
  
    if (y != 0) {  
        if (y > 0) {  
            goBack(); // înapoi  
        } else {  
            goFront(); // sau înainte  
        } // în funcție de cadran  
  
        delay((long)oyTime); // până la parcurgerea distanței totale de pe OY  
        stopCar(front_back); // oprește platforma de tot  
    }  
}
```

La finalizarea pașilor unui program platforma efectuează următoarele operații.

1. Vede dacă utilizatorul a închis forțat programul.
2. Dacă nu, verifică dacă utilizatorul a optat pentru revenirea la punctul de start.
3. Dacă s-a optat pentru revenire, rulează algoritmul de revenire, oprește mașina și înștiințează utilizatorul la finalizare, specificând numele programului care s-a încheiat. Din aplicație, utilizatorul va fi înștiințat într-o fereastră că platforma a terminat programul și a revenit la start.

4. Dacă nu s-a optat, spune utilizatorului să aștepte și permite afișarea unei ferestre (popup) unde utilizatorul poate alege ca platforma să rămână în punctul final sau să revină la punctul de start, așteptând ca utilizatorul să fie înștiințat și să răspundă.
5. Dacă utilizatorul a ales să se întoarcă, efectuează din nou pasul 2.
6. La final, șterge programul din baza de date.

Din aplicația IoT SmartPlatform, utilizatorul poate seta o nouă configurație pentru conectarea platformei la internet, furnizând un nou SSID și Password. Utilizatorul poate face asta doar dacă este conectat la platformă.

Deoarece programul ce rulează pe mașină este în buclă, trebuie să avem grijă ca conexiunea să nu se facă continuu și să așteptăm de fiecare dată conectarea la noua configurație. Pentru aceasta se activează din aplicație un flag care ne spune dacă s-au schimbat setările, numit otherNetwork. Din programul mașinii se verifică dacă există cineva conectat la platformă și dacă s-a modificat configurația pe baza flag-urilor otherNetwork și connected. Dacă configurația s-a schimbat, atunci modulul Wi-Fi începe conectarea cu noua parolă și noul SSID și se dezactivează flag-ul otherNetwork ca la următoarea buclă să nu mai facă reconectarea inutil.

Se poate schimba parola doar dacă pe platformă nu rulează niciun program. Utilizatorul este înștiințat că se schimbă configurația și pus în așteptare.

Dacă trec mai mult de 10 secunde și platforma nu este conectată, utilizatorul este rugat să se deconecteze de la platformă printr-o fereastră și să repornească platforma pentru a se conecta la configurația inițială. Mai multe detalii apar în aplicație la secțiunea Tips din pagina Settings despre care voi discuta în Capitolul 4.

Dacă utilizatorul se deconectează de la platformă (doar dacă nu rulează un program deja) și platforma este pornită, platforma se reconectează la internet folosind setările prestabilite.

```

if (!program.compareTo("")) {

    // vede dacă există utilizator conectat la platformă și dacă s-a schimbat configurația

    // dacă nu se poate face conexiunea la Firebase, reîncearcă de fiecare dată

    bool otherNetwork = Firebase.getBool("devices/SmartPlatform00001TX/otherNetwork");

    while (Firebase.failed()) {

        otherNetwork = Firebase.getBool("devices/SmartPlatform00001TX/otherNetwork");

    }

    bool connected = Firebase.getBool("devices/SmartPlatform00001TX/connected");

    while (Firebase.failed()) {

        connected = Firebase.getBool("devices/SmartPlatform00001TX/connected");
}
}

```

```

}

// dacă da

if (otherNetwork && connected) {

    // ia parola și ssid-ul modificat din baza de date

    String ssid = Firebase.getString("/devices/SmartPlatform00001TX/ssid");

    while (Firebase.failed()) {

        ssid = Firebase.getString("/devices/SmartPlatform00001TX/ssid");

    }

    String password = Firebase.getString("/devices/SmartPlatform00001TX/password");

    while (Firebase.failed()) {

        password = Firebase.getString("/devices/SmartPlatform00001TX/password");

    }

    char newSSID[ssid.length() + 1]; // convertește la char[]

    strcpy(newSSID, ssid.c_str());

    char newPass[password.length() + 1];

    strcpy(newPass, password.c_str());

    // punе utilizatoul să aștepte

    Firebase.setBool("devices/SmartPlatform00001TX/wait", true);

    while (Firebase.failed()) {

        Firebase.setBool("devices/SmartPlatform00001TX/wait", true);

    }

    // setează counter-ul la 0

    int time = 0;

    WiFi.begin(newSSID, newPass); // Începe conexiunea și verifică dacă se poate conecta

    while (WiFi.waitForConnectResult() && WiFi.status() != WL_CONNECTED) {

```

```

delay(1000); // dacă încă nu s-a conectat aşteaptă o secundă și reîncearcă

// crește counter

time++;

Firebase.setInt("devices/SmartPlatform00001TX/time", time);

while (Firebase.failed()) {

    Firebase.setInt("devices/SmartPlatform00001TX/time", time);

}

// dacă trec 10 secunde, se încearcă conectarea cu configurația default

if(time == 10) {

    delay(12000); // aşteaptă răspunsul de la aplicație

    Firebase.remove("devices/SmartPlatform00001TX/time");

    while (Firebase.failed()) {

        Firebase.remove("devices/SmartPlatform00001TX/time");

    }

    WiFi.begin(My_SSID, My_PASS);

    while (WiFi.status() != WL_CONNECTED) {

        delay(1000); // dacă încă nu s-a conectat aşteaptă o secundă și

    } // reîncearcă

    break;

}

// scoate clientul de pe aşteptare
Firebase.remove("devices/SmartPlatform00001TX/wait");

while (Firebase.failed()) {

    Firebase.remove("devices/SmartPlatform00001TX/wait");

}

```

```

Firebase.remove("devices/SmartPlatform00001TX/time");

while (Firebase.failed()) {

    Firebase.remove("devices/SmartPlatform00001TX/time");

}

Firebase.setBool("devices/SmartPlatform00001TX/otherNetwork", false);

while (Firebase.failed()) {

    Firebase.setBool("devices/SmartPlatform00001TX/otherNetwork", false);

}

}

// dacă utilizatorul se deconectează

if (!connected) {

    WiFi.begin(My_SSID, My_PASS);

    while (WiFi.status() != WL_CONNECTED) {

        delay(1000); // dacă încă nu s-a conectat aşteaptă o secundă și reîncearcă

    }

}

```

Capitoul 4

Dezolvarea aplicației de control

Acet capitol are ca rol înțelegerea funcționalității aplicației IoT și scopul ei în funcționarea acestui proiect. Aplicația IoT, dezvoltată în Android Studio folosind Java, este dedicată controlului acestui tip de device dar poate fi folosită de mai mulți utilizatori și pentru mai multe platforme mobile de tip SmartPlatform dezvoltate ulterior. Aplicația poartă același nume ca și platforma și este esențială pentru utilizarea prototipului și ulterior a sistemului la scară 1:1.

Folosind serviciul BaaS integrat cu Android^[18], am securizat autentificarea și am realizat un sistem care permite accesul sincronizat al utilizatorilor și controlul prioritizat al acelora care au asignat în cont același deviceID, deci prin urmare împart aceeași platformă.

Dacă cei ce folosesc aplicația lucrează ca administratori pentru astfel platforme cu scopul deplasării robotilor industriali în anumite locații pentru efectuarea de sarcini, aceștia pot accesa sistemul doar pe rând.

Utilizatorii pot folosi aplicația în două moduri și în două stări descrise mai jos.

Primul mod este cel al controlului manual, unde pe baza unui joystick virtual, utilizatorul poate controla platforma de la distanță cât timp aceasta se află în aria vizuală și există conexiune la internet. Controlul manual este prevăzut și cu un sistem de atenționare asupra obstacolelor ce se folosește de datele înregistrate de la senzorii ultrasonici. Voi detalia în Subcapitolul 4.3 acest mecanism. De asemenea utilizatorul poate seta din joystick viteza platformei, câtă vreme aceasta nu se află într-un viraj, deoarece în viraje, aşa cum am menționat în Capitolul 3, viteza e maximă.

Al doilea mod este cel al controlului automatizat. Controlul automatizat permite utilizatorului să încarce în codul platformei un program care ulterior va fi rulat de către aceasta. Câtă vreme există un program pornit, utilizatorul nu poate modifica setările de configurare, să se deconecteze, să se delogheze sau să adauge noi programe pe care să le ruleze ulterior.

Legat de cele două stări menționate, o stare este cea de stand by care este activă la momentul conectării în sistem, până la conectarea cu sistemul fizic ce trebuie controlat. În această stare, utilizatorul nu poate modifica setările de conectivitate (SSID și Password), dar poate crea noi programe, le poate vizualiza sau poate să își șteargă contul.

Pentru controlul manual și automatizat, este necesar ca utilizatorul să nu mai fie în starea stand by ci în starea connected.

4.1 Procesul de autentificare

Pentru autentificarea în aplicația SmartPlatform, utilizatorul este nevoit să introducă ID-ul platformei asociat contului ce urmează să fie creat. Există două variante de alegere a ID-ului pentru platforma asociată contului. Ea poate fi introdusă manual din pagina Registration, sau pe baza scănerării unui QR code din interiorul aplicației.



Figura 4.1.1

This is a screenshot of the "User Registration" screen from the SmartPlatform app. It has a dark background with white text. At the top, it says "User Registration" next to the "SMART PLATFORM" logo. Below that are several input fields: "First Name" and "Last Name", "Email address", "Enter password", "Re-enter password", "Phone number", and "Device ID". At the bottom of the form is a green "SIGN UP" button. There is also a link "Already Registered? Sign In Here" at the very bottom.

Figura 4.1.2

This screenshot shows the same registration screen as Figura 4.1.2, but with an error message. The "Device ID" field contains the value "4-078-787-7888", which is invalid. A callout bubble appears over this field with the text "Device invalid! There is no platform with this ID...Try another". The rest of the form and buttons are identical to the previous figure.

Figura 4.1.3

Conform Figurii 4.1.1, se observă că la deschiderea aplicației pentru un nou utilizator, acesta poate apăsa pe butonul „SCAN YOUR PLATFORM” și alege să scaneze codul QR atașat de sistemul fizic achiziționat. Dacă acesta preferă să adauge deviceID-ul manual, el poate apăsa pe link-ul de sub buton. Pentru acei utilizatori care au deja cont, se poate apăsa pe link-ul din zona inferioară a ecranului care îl va duce direct pe pagina de Login.

După ce utilizatorul scanează platforma, el va fi redirectionat pe pagina Registration, cu DeviceID-ul deja setat, și nu îl mai poate modifica.

Dacă cel ce folosește aplicația preferă să adauge manual deviceID-ul, există riscul ca contul să nu poată fi creat dacă valoarea ID-ului nu corespunde cu nicio platformă înregistrată în baza de date, ca în Figura 4.1.3. Este bine de știut că o platformă nouă creată și dată unui client spre folosință, se va înregistra în baza de date din Firebase cu numele deviceID-ului. De asemenea, utilizatorul va ști deviceID-ul sistemului fizic deoarece acesta va fi scris în ghidul de utilizare al sistemului IoT achiziționat.

Scanarea propriu-zisă s-a realizat cu ajutorul bibliotecii ce permite scanarea de coduri de bară numită me.dm7.barcodescanner.zxing.ZXingScannerView.

După ce s-au setat dependențele în build.gradle din folderul app, și permisiunile din fișierul xml AndroidManifest ("Android.permission.CAMERA"), se moștenește interfața ZXingScannerView.ResultHandler. Folosind clasa ZXingScannerView din pachet putem crea un eveniment pentru a deschide camera. Dacă nu au fost setate permisiunile de acces la cameră,

va apărea o fereastră prin care se poate accepta accesul. Astfel se pornește camera și se permite scanarea. Pe baza metodei suprascrisă a interfeței onRequestPermissionsResult(...) se permite citirea și interpretarea codului QR, iar cu metoda suprascrisă handleResult(..) se trimită în activitatea de creare a contului, deviceID-ul și se trece în noua pagină cu deviceID-ul deja setat (a se vedea Figura 4.1.4).

```

public void generate() {
    scannerView = new ZXingScannerView(this);
    if (checkPermission()){
        setContentView(scannerView);
        scannerView.setResultHandler(this);
        scannerView.startCamera();
    } else {
        requestPermission();
    }
}

private boolean checkPermission()
{
    return (ContextCompat.checkSelfPermission(getApplicationContext(), CAMERA) == PackageManager.PERMISSION_GRANTED);
}

public void requestPermission() {
    ActivityCompat.requestPermissions(this, new String[]{CAMERA}, 0);
}

@Override
protected void onPause() {
    super.onPause();
}

@Override
public void handleResult(Result result) {
    String code;
    code = result.getText();
    finish();
    Intent registerWithID = new Intent(ActivityMain.this, ActivityRegister.class);
    registerWithID.putExtra(getResources().getStringArray(R.array.Extra)[0], code);
    startActivity(registerWithID);
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (!requestCode == 0 && grantResults.length < 1) {
        scannerView = new ZXingScannerView(this);
        if (checkPermission()) {
            setContentView(scannerView);
            scannerView.setResultHandler(this);
            scannerView.startCamera();
        }
    }
}

```

Figura 4.1.4 – Scanare QR

Pentru crearea contului, pe lângă device ID, utilizatorul trebuie să ofere un nume, prenume, email parolă și un număr de telefon. Pentru toate datele adăugate în form se testează dacă acestea corespund cu tipul de input al field-ului și în caz contrar apar mesaje de eroare ca în Tabelul 4.1.1

Toate erorile de prevenire prezentate în Tabelul 4.1.1 au ca rol imposibilitatea de creare a contului dacă datele introduse nu respectă criterii de integritate și siguranță.

Field	Tipul erorii	
Nume, Prenume		
Email		
Parolă		
Reintroducere Parolă		
Număr de telefon		
DeviceID		

Tabel 4.1.1 – Erori de prevenire la Sign Up

Conform cu Figura 4.1.2, dacă utilizatorul are un cont, poate apăsa pe link-ul din josul ecranului pentru a fi redirecționat la pagina de Login. Dacă toate field-urile sunt acceptate, atunci se merge mai departe și se crează contul prin apăsarea butonului „SIGN UP”. La crearea contului, în Firebase se generează un cont nou în secțiunea Authentication, ca în Figura 4.1.5. Acest cont are un field UID unic pe care l-am folosit ca în momentul în care înregistrarea s-a efectuat cu succes, să se adauge un nou utilizator cu numele UID și în baza de date, în tabela users, așa cum apare în Figura 4.1.5.

Search by email address, phone number, or user UID					Add user	⋮
Identifier	Providers	Created	Signed In	User UID ↑		
radu.pelin7@gmail.com	✉	Jun 16, 2019	Jun 17, 2019	tLTgnYMXkC0tyL9c321ohgAv2aV2		

Figura 4.1.5 – Autentificare



Figura 4.1.6 – Adăugare utilizator în baza de date

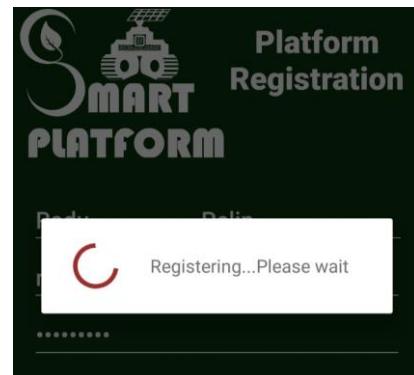


Figura 4.1.7 – Înregistrare

În Figura 4.1.7 se observă că la apăsarea butonului „SIGN UP” apare un circle process bar care îンștiințează utilizatorul cât mai are de așteptat până se verifică procesul de autentificare de către Firebase. Dacă rezultatul este pozitiv și task-ul se încheie cu succes, pe ecran apare un mesaj de confirmare „Successfully registered...” și utilizatorul este redirecționat la pagina de Login. Altfel, apare un warning „Failed to register...Try again” prin care utilizatorul este rugat să mai încece odată. Această eroare poate apărea și în cazul existenței unui utilizator cu același email ca cel cu care dorim să ne înregistram. În acest caz se atenționează utilizatorul că există alt cont cu email-ul furnizat în form-ul de autentificare completat de către acesta.

La redirecționarea pe pagina de Login, apare un mesaj care specifică că trebuie să se verifice email-ul pentru confirmarea contului. De aceea, în figura 4.1.6 se vede că flag-ul valid este setat pe false, deoarece încă nu s-a realizat confirmarea. Firebase se ocupă de trimiterea email-ului de confirmare care conține un link. La apăsarea link-ului utilizator poate să se conecteze din aplicație cu noile credențiale.

În Figura 4.1.8 se poate observa pagina de Login a aplicației.

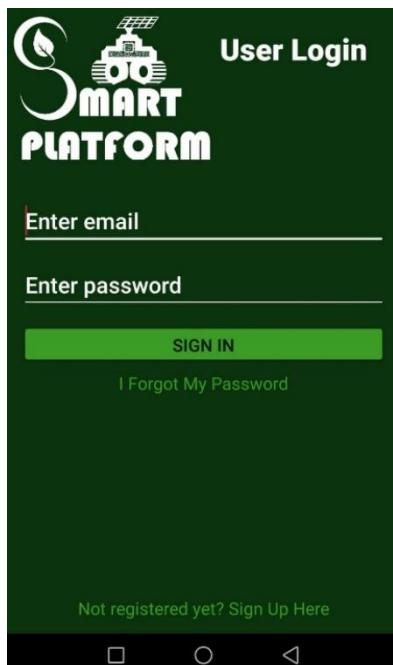


Figura 4.1.8

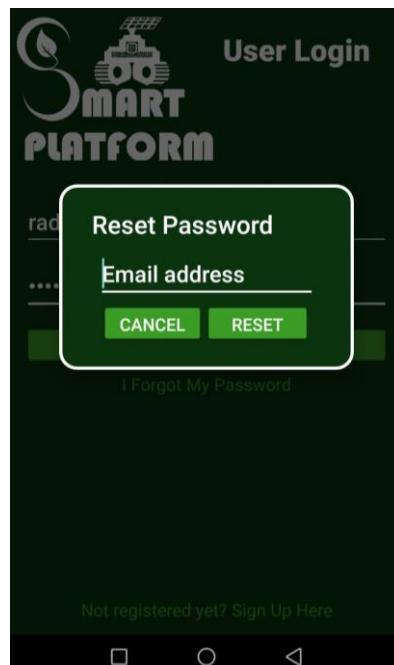


Figura 4.1.9

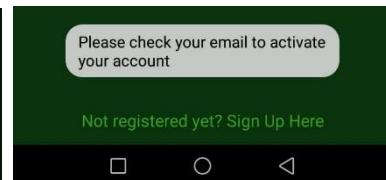


Figura 4.1.12

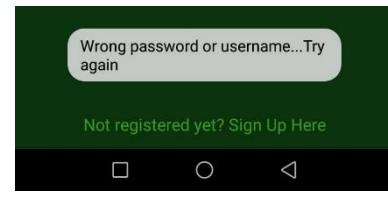


Figura 4.1.11

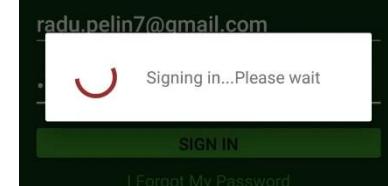


Figura 4.1.10

Dacă utilizatorul vrea să creeze un nou cont din pagina de Login, el poate reveni la pagina de autentificare prin apăsarea link-ului din josul ecranului.

Când acesta a uitat parola, poate apăsa pe link-ul „I Forgot My Password” de sub butonul „SIGN IN”. Odată apăsat acest buton, se deschide o fereastră ca cea din Figura 4.1.9 unde utilizatorul are un field în care poate introduce email-ul contului pentru care dorește resetarea parolei. Dacă email-ul nu este corect sau dacă nu s-a putut finaliza cu succes task-ul de resetarea a email-ului datorită unor probleme legate de server sau alte servicii, apar mesaje de eroare corespunzătoare problemei întâlnite la apăsarea butonului „Reset”. Apăsând pe butonul „Cancel” se închide fereastra fără trimiterea unui email de resetare a parolei.

La efectuarea cu succes a task-ului, pe ecran apare un mesaj care anunță utilizatorul să își verifice adresa de email pentru resetarea parolei. Firebase gestionează singur aceste evenimente și trimit către utilizator un link care îi permite acestuia resetarea parolei. Link-ul redirecționează clientul către o pagină cu un buton “ok” și un field “New password”. Prin completarea acelui field-ului și apăsarea butonului se resetează parola contului, modificare ce apare în Firebase în secțiunea Authentication.

La apăsarea butonului „SIGN IN” se realizează conectarea propriu-zisă. După apăsare, Firebase verifică dacă conectarea se poate realiza cu succes și utilizatorul este rugat să aștepte conform cu Figura 4.1.10. Dacă parola și email-ul nu corespund sau sunt invalide, va apărea un mesaj de eroare ca în Figura 4.1.11 și Figura 4.1.12.

Dacă autentificarea s-a realizat cu succes clientul este redirecționat pe pagina de control automatizat a platformei, considerată pagina de start a contului utilizator. Dacă contul a fost activat înainte de autentificare, apare un mesaj pentru confirmarea activării contului, iar flag-ul valid din baza de date devine true, ca în Figura 4.1.13.



Figura 4.1.13 – User în baza de date

După cum se poate observa, pentru a putea folosi aplicația, trebuie să fim în permanență conectați la o rețea. Orice acțiune trebuie efectuată doar după ce se verifică că există o conexiune stabilă. De aceea, pentru fiecare eveniment și acțiune se urmărește dacă s-a putut stabili conexiunea cu Firebase și dacă există conexiune la internet. În capitolele următoare voi explica detaliat acest mecanism prin mai multe exemple concrete.

4.2 Procesul de acces și conectare la platformă

În momentul în care utilizatorul intră în cont, el este pus direct în starea stand by. Această stare îi permite utilizatorului doar să creeze, editeze și vizualizeze programe noi în secțiunea de control automatizat. Cu toate acestea, el nu poate porni un program, modifica setările sau să controleze manual platforma, deoarece nu este conectat la device.

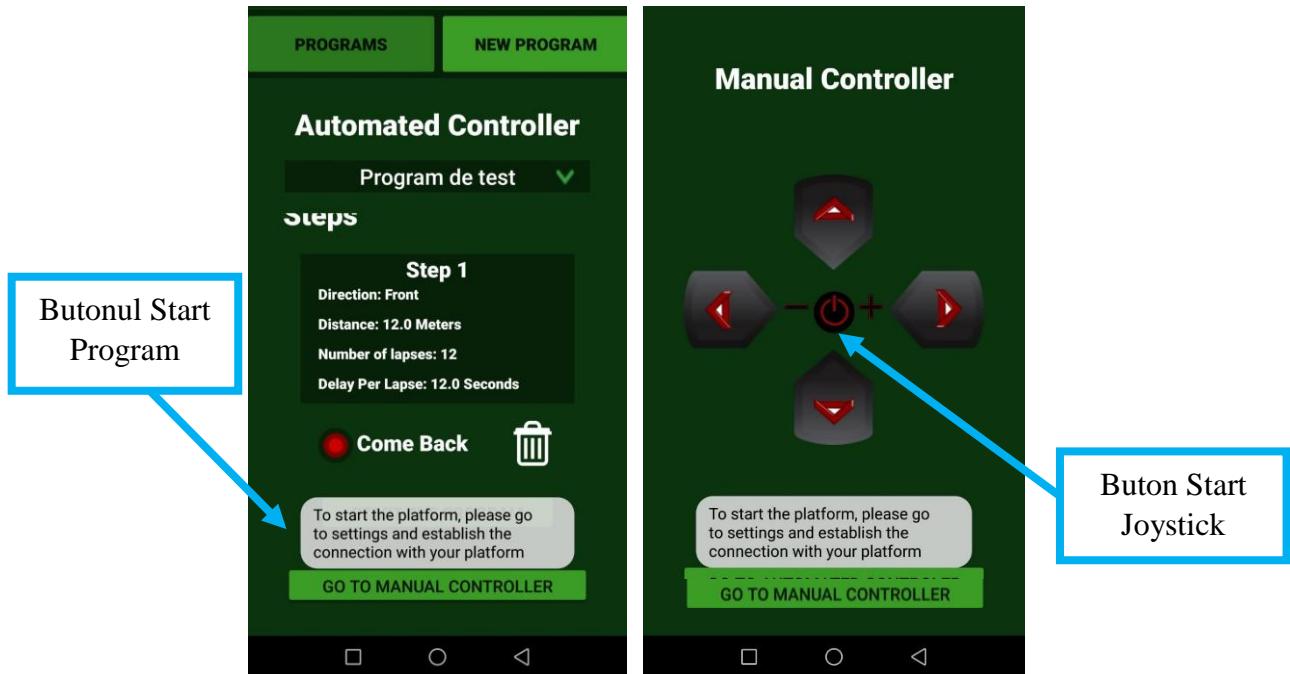


Figura 4.2.1 – Eroare la control

În Figura 4.2.1 se poate observa că dacă nu s-a stabilit conexiunea între aplicația IoT și platforma fizică, în cazul pornirii controller-ului manual sau la încercarea de pornire a unui program, aplicația înștiințează utilizatorul despre cum poate stabili conexiunea cu sistemul fizic pentru a efectua operațiunea dorită.

Conform cu mesajul, trebuie să ajungem la pagina de setări a contului curent. Pagina de setări se poate accesa prin apăsarea pe navigatorul din colțul stânga sus al aplicației mobile, ca în Figura 4.2.2.

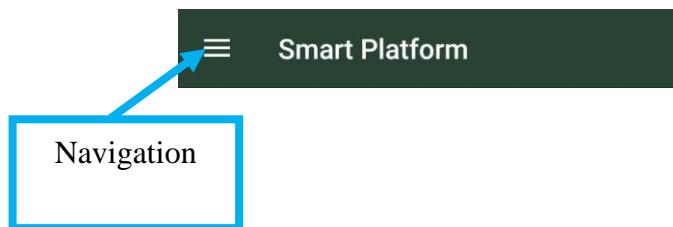


Figura 4.2.2 – Navigation button

La apăsarea butonului de navigare va apărea meniul de tip navigation drawer. Navigatorul este folosit pentru a facilita accesul la anumite zone ale aplicației ce fac referire la setările contului și nu numai. Un sertar de navigare (navigation drawer) este o fereastră ce poate fi permanent vizibilă în partea stânga a ecranului, sau apare la apăsarea butonului de navigare. În acest caz fereastra își face gradual și dispare gradual (fade in / fade out).

Un navigator are de obicei un header și mai multe item-uri deosebite prin diferite iconițe ca în Figura 4.2.3. Header-ul poate fi o poză cu utilizatorul sau inițialele prenumelui și numelui ca în cazul de față.

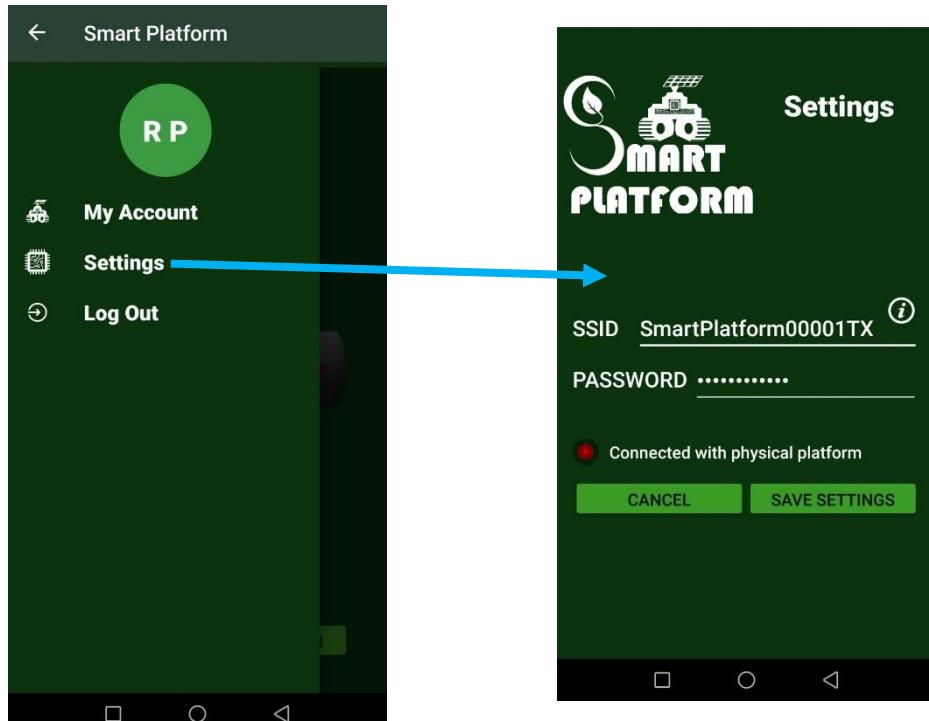


Figura 4.2.3 – Navigation Drawer

Figura 4.2.4 – Setările de cont

La accesarea paginii de Settings, utilizatorul poate vizualiza care este SSID-ul curent al rețelei Wi-Fi la care este platforma conectată, însă pentru securitate nu poate vedea parola. Modificarea configurației se poate face doar atunci când utilizatorul este conectat la platformă. Pentru a face asta se apăsa pe butonul „Connected with physical platform”. După apăsarea acestui buton, va apărea o fereastră de atenționare ca în imaginea stângă din Figura 4.2.4.

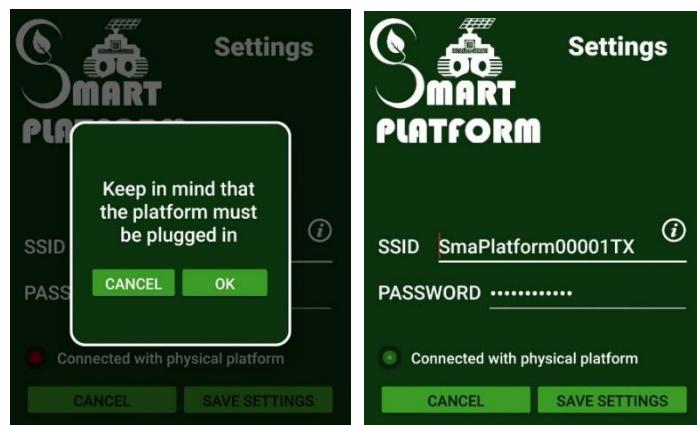


Figura 4.2.4 – Stabilirea conexiunii cu platformă

Pentru a putea comunica cu platforma odată ce ea este alimentată, trebuie să se ofere acces la internet. Ca acest lucru să fie posibil, trebuie folosită funcția hotspot a telefonului mobil, unde se modifică SSID-ul și Password-ul cu valorile care se află în Settings la momentul curent ca în Figura 4.2.4, urmând să se deschidă datele mobile. Acestea nu se închid cât timp

nu s-a modificat configurația la internet sau cât timp s-a modificat configurația și ulterior platforma răspunde la comenzi date fie din controller-ul manual fie din cel automat.

După ce utilizatorul a confirmat dorința de a se conecta și butonul devine verde ca în Figura 4.2.4, el va avea opțiunea de a schimba configurația de conectare sau să păstreze configurația actuală care este cea default.

Pentru a salva noile setări și a stabili conexiunea se apasă pe butonul „SAVE SETTINGS”. La apăsare, pagina de setări se închide și utilizatorul este redirecționat către pagina de control automatizat. Pe ecran va apărea un mesaj care ne anunță că suntem conectați, iar dacă am optat și pentru schimbarea setărilor de internet, va exista și un mesaj pentru a anunța asta. Se poate reuși la modificări prin apăsarea butonului „Cancel”.

La realizarea acestei operațiuni, în baza de date pentru device-ul la care s-a conectat utilizatorul apare flag-ul connected activ, iar utilizatorul ieșe din starea stand by. Dacă se repornește platforma, utilizatorul va fi deconectat și redirecționat la pagina de start, funcționalitate detaliată în Subcapitolul 4.4. Programul care rulează se va opri și joystick-ul se va închide.

La intrarea în cont pentru prima dată, utilizatorul are câmpurile ssid și password identice cu cele ale platformei cu care și-a asociat contul, în mod default. Dacă s-a optat pentru schimbarea configurației, în baza de date, se modifică ssid-ul și password-ul pentru utilizatorul curent și rămâne neschimbăt până la modificarea configurației, indiferent dacă el este conectat sau deconectat de la platformă.



Figura 4.2.5 – No Internet

Dacă după 10 secunde de la modificarea configurației platforma nu a reușit să se conecteze, se va actiona un semnal sonor și va apărea o fereastră cu un mesaj care ne atenționează că nu există conexiune la internet, ca în figura 4.2.5 și ssid-ul și password-ul utilizatorului se vor suprascrie cu valorile default pentru ssid și password ale platformei.

```
public class NoInternet extends TimerTask {
    @Override
    public void run() {
        ConnectivityManager conManager = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        if (deviceID != null && conManager.getActiveNetworkInfo() != null && !isInStandBy) {
            FirebaseDatabase.getInstance().getReference(getString(R.string.DeviceDB))
                .child(deviceID).addListenerForSingleValueEvent(new ValueEventListener() {
                    @Override
                    public void onDataChange(DataSnapshot dataSnapshot) {
                        if (dataSnapshot.child("time").exists()) {
                            long time = (long) dataSnapshot.child("time").getValue();
                            if(time == 10) {
                                timer.cancel();
                                startActivity(new Intent(getApplicationContext(), PopupNoInternet.class));
                            }
                        }
                    }
                });
        }
    }
}
```

Figura 4.2.6 – NoInternet Task

Pentru a se afișa popup-ul s-a rulat un task odată la 10 secunde extinzând clasa TimerTask, pe toate paginile contului. Se verifică dacă s-a realizat conexiunea cu Firebase (deviceID != NULL), dacă există conexiune la internet sau dacă utilizatorul care are asociată platforma este conectat la ea (!isInStandBy).

Conform cu Figura 4.2.6, dacă platforma a activat timer-ul și utilizatorul nu este în stand by (este conectat), atunci va apărea mesajul de avertizare „No Internet”. În thread-ul principal, pentru rularea task-ului putem scrie astfel:

```
TimerTask timerNoInternet = new NoInternet();
Timer timer = new Timer(true);
timer.scheduleAtFixedRate(timerNoInternet, 0, 10000);
```

Dacă după schimbarea credențialelor încă se așteaptă pentru conectarea la internet a platformei, utilizatorul nu poate modifica setările și apare un mesaj de eroare asociat în pagina de Settings: „You can't change settings while the platform is connecting to the Internet”.

De asemenea, în baza de date pentru platforma asociată se modifică câmpurile ssid și password care sunt populate cu valorile ssid și password ale utilizatorului conectat și apare flag-ul otherNetwork, folosit pentru ca platforma să știe că trebuie să se conecteze la noua rețea Wi-Fi, aşa cum am explicitat în Capitoul 3.

La deconectare, câmpurile ssid și password ale platformei iau valoarea default, dar utilizatorul rămâne cu ele neschimbate și vizibile în pagina de setări. Dacă la deconectare se modifică credențialele, după apăsarea butonului de salvare a setărilor modificări nu vor fi luate în seamă din motive de siguranță.

Toate aceste detalii sunt expuse utilizatorului într-o fereastră, la apăsarea butonului „Info” din pagina Settings. În Figura 4.2.7 se poate vedea fereastra deschisă.

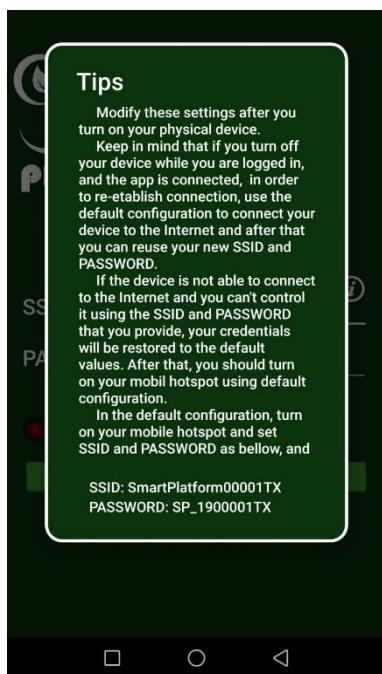


Figura 4.2.7 – Tips

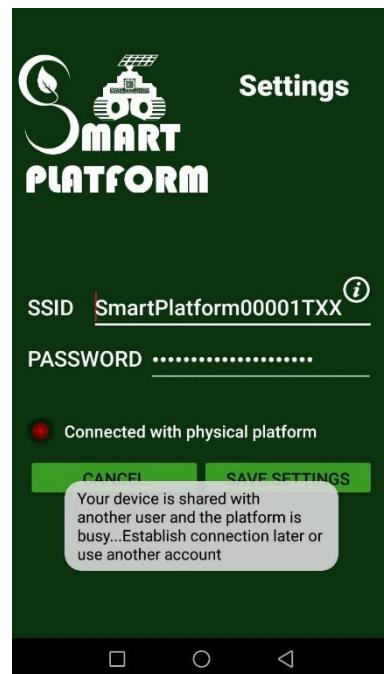


Figura 4.2.8 – Shared Error

În cazul în care un alt utilizator este conectat la platformă și clientul încearcă totuși să se conecteze va apărea un mesaj de eroare ca în Figura 4.2.8.

4.3 Procesul de control

Rolul aplicației SmartPlatform este de interfață de control a platformei fizice. Interfața de control și mecanismul de conectare la platformă descris în Subcapitolul 4.2, împreună cu serviciile furnizate de Firebase, formează împreună sistemul IoT.

Procesul de control este esențial pentru acest sistem și la baza lui stă întreaga funcționalitate a platformei. Există diferite modalități de control ale platformei, aşa cum am discutat în capitolele anterioare.

O primă metodă este cea a controlului manual, pe baza unui joystick virtual aflat în aplicație, o metodă simplă din punct de vedere a operațiilor ce trebuie să le efectueză mașina de calcul a platformei fizice.

Cea de-a doua metodă implică utilizarea unor programe formate din pași ce pot fi execuți în linie dreaptă. De această dată, procesarea este mai grea și mai laborioasă, fiind nevoie de algoritmi bazați pe un sistem de coordonate virtuale, descris în Capitolul 3. De asemenea, la controlul automatizat, pot apărea numeroase probleme de sincronizare în cazul unei conexiuni slabe la internet. Schimbul de informații între aplicația mobilă și platformă este mult mai mare ca în cazul controlului manual.

Dintre cele două metode, doar controlul manual prevede depistarea obstacolelor întâlnite pe traseu, în sensul de deplasare al platformei. Condiția ca platforma să efectueze programul ce a fost încărcat este să nu existe obstacole pe traseu.

4.3.1 Control manual

La conectare, utilizatorul este direcționat pe pagina de control automatizat. Pentru a avea acces la controller-ul manual, se apasă pe butonul „GO TO MANUAL CONTROLLER” din partea de jos a ecranului ca în Figura 4.3.1.1

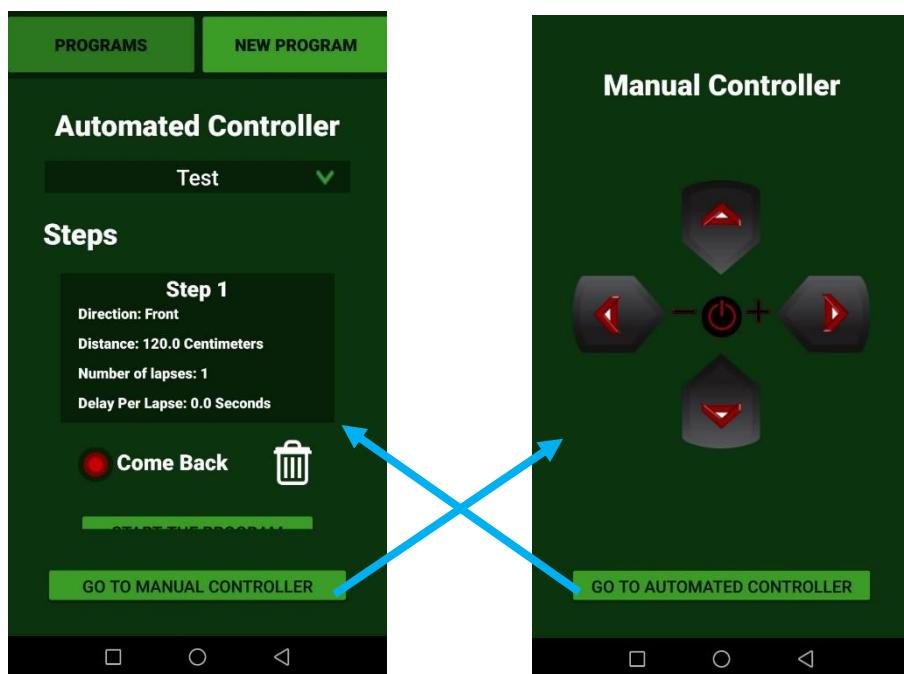


Figura 4.3.1.1 – Switch Controllers

La intrarea în pagina „Manual Controller” va apărea un joystick virtual pentru controlul la distanță al platformei, ca în Figura 4.3.1.1.

Pentru pornirea joystick-ului, utilizatorul trebuie să fie conectat, aşa cum am menționat în subcapitolul anterior, iar butonul de power să fie activ ca în imaginea stângă din Figura 4.3.1.2



Figura 4.3.1.2 – Pornire și oprire joystick

Conform cu figura 4.3.1.2, la apăsarea butonului de power, va apărea un mesaj ce ne spune că joystick-ul poate fi folosit, iar starea platformei devine „R_Speed1” care semnifică că platforma poate să se deplaseze cu viteza a I – a. Pentru a se modifica starea, în baza de date în tabela devices pentru platforma asociată contului, se modifică atributul state în valoarea „R_Speed1”. Dacă se reapașă butonul de power, joystick-ul se resetează și oprește și platforma intră în starea „R_Stop”. În aplicație se afișează un mesaj corespunzător ca în Figura 4.3.1.2.

Din consolă se poate schimba viteza de deplasare a mașinii, până la cinci viteză.

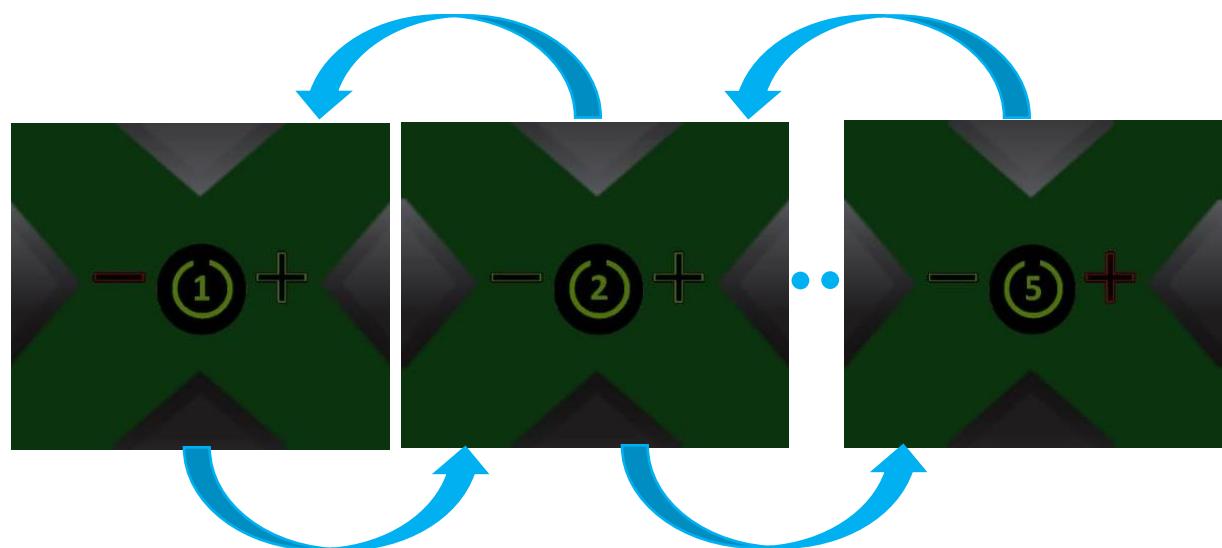
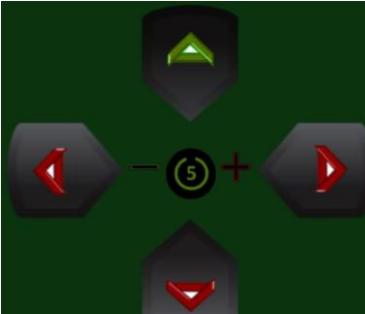
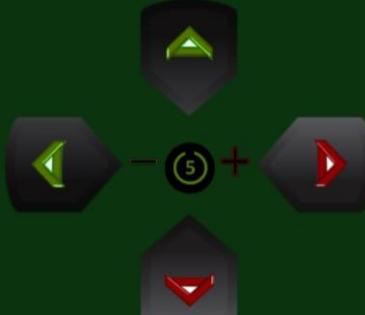
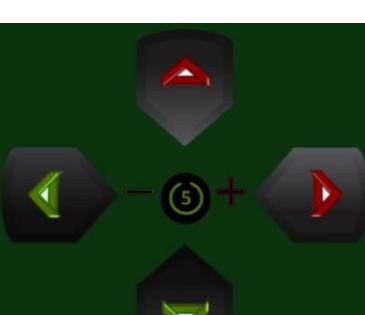


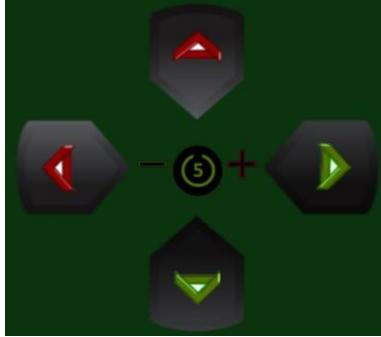
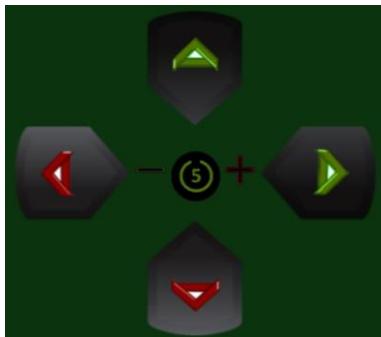
Figura 4.3.1.3 – Modificarea vitezei

La schimbarea vitezei, se modifică starea platformei, trecând prin stările „R_Speed1”, „R_Speed2”, „R_Speed3”, „R_Speed4” și „R_Speed5” aşa cum am explicitat în capitolul precedent. Dacă platforma se deplasează deja înainte sau înapoi, la modificarea vitezei, platforma își schimbă starea dar continuă să își mențină deplasarea și sensul.

În cazul virajelor, nu este permisă modificarea vitezei deoarece acestea se fac la viteza maximă (a se vedea Figura 4.3.1.).

În Tabelul 4.3.1.1, se pot observa toate modalitățile de control.

Modalitate de control	Eveniment	Observații
Control înainte		<p>La apăsarea butonului „Front”, se schimbă starea platformei în „H-F” (High Front) și aceasta începe deplasarea înainte. Reapăsarea butonului duce la oprirea platformei și intrarea în starea „L-F” (Low Front).</p> <p>Pentru virajele la stânga respectiv dreapta cu sensul de deplasare înainte, trebuie mai întâi setată starea „H-F”.</p>
Control înapoi		<p>La apăsarea butonului „Back”, se schimbă starea platformei în „H-B” (High Back) și aceasta începe deplasarea înapoi. Reapăsarea butonului duce la oprirea platformei și intrarea în starea „L-B” (Low Back).</p> <p>Pentru virajele la stânga respectiv dreapta cu sensul de deplasare înapoi, trebuie mai întâi setată starea „H-B”.</p>
Viraj stânga cu deplasare înainte		<p>Dacă butonul „Front” este activ, la apăsarea ulterioară a butonului „Left” se schimbă starea platformei în „H-L_F” (High Left Front) și se începe deplasarea cu față către stânga.</p> <p>Apăsând din nou pe butonul „Left”, platforma intră în starea „L-L_F” (Low Left Front) care este echivalentă cu starea „H-F” și platforma continuă să se deplaseze înainte.</p>
Viraj stânga cu deplasare înapoi		<p>Dacă butonul „Back” este activ, la apăsarea ulterioară a butonului „Left” se schimbă starea platformei în „H-L_B” (High Left Back) și se începe deplasarea cu spatele către stânga.</p> <p>Apăsând din nou pe butonul „Left”, platforma intră în starea „L-L_B” (Low Left Back) care este echivalentă cu starea „H-B” și platforma continuă să se deplaseze înapoi.</p>

Viraj dreapta cu deplasare înapoi		Dacă butonul „Back” este activ, la apăsarea ulterioară a butonului „Right” se schimbă starea platformei în „H-R_B” (High Right Back) și se începe deplasarea cu față către stânga. Apăsând din nou pe butonul „Right”, platforma intră în starea „L-R_B” (Low Right Back) care este echivalentă cu starea „H-B” și platforma continuă să se deplaceze înapoi.
Viraj dreapta cu deplasare înainte		Dacă butonul „Front” este activ, la apăsarea ulterioară a butonului „Right” se schimbă starea platformei în „H-R_F” (High Right Front) și se începe deplasarea cu față către stânga. Apăsând din nou pe butonul „Right”, platforma intră în starea „L-R_F” (Low Right Front) care este echivalentă cu starea „H-F” și platforma continuă să se deplaceze înainte.

Tabelul 4.3.1.1 – Modalități de control

După cum am menționat anterior, nu se poate modifica viteza în viraje și la încercarea de a o modifica, apar mesaje de eroare corespunzătoare ca în Figura 4.3.1.3

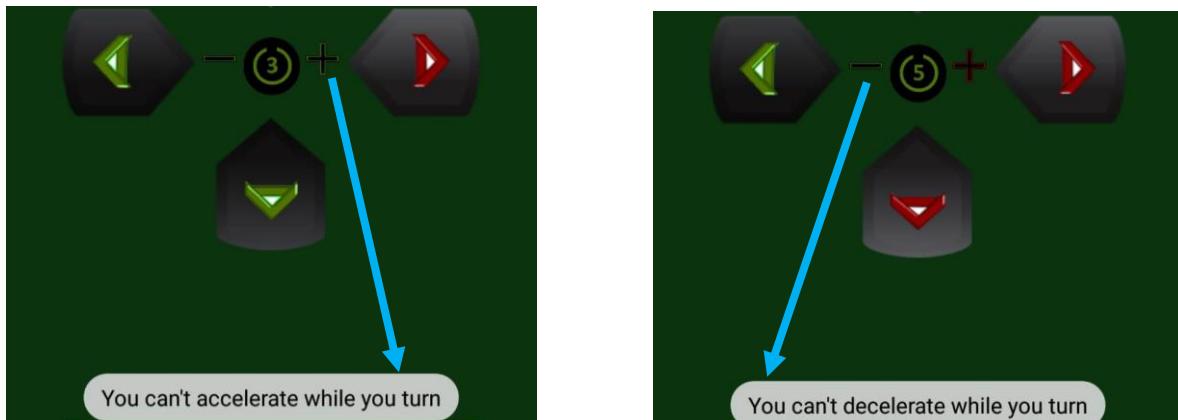


Figura 4.3.1.4 – Acelerare și decelerare la viraj

Pentru modificarea stării m-am folosit de o clasă nasted numită State, care are un atribut String state, un constructor cu un parametru și o metoda pentru modificare, modify() ce returnează true în caz de reușită. În aplicarea metodei modify ne folosim și de metoda displayMessage(...) a clasei principale care are ca rol afișarea unui mesaj în cazul în care sistemul încă nu e configurat și nu se poate face modificarea. Această problemă poate apărea și dacă nu există conexiune bună la internet sau dacă nu se poate stabili conexiunea Firebase. Mesajul se poate vedea în Figura 3.4.1.5.

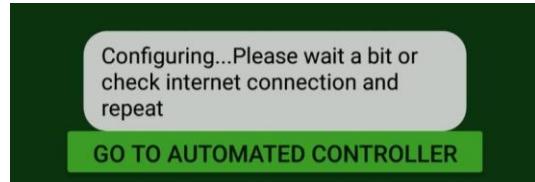


Figura 4.3.1.5 – Conexiune slabă sau în lipsă

```
public class State {
    private String state;

    public State(String state) {
        this.state = state;
    }

    protected boolean modify() {
        ConnectivityManager conManager = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);
        if (deviceID != null && conManager.getActiveNetworkInfo() != null){
            databaseDevices.child(deviceID).child("state").setValue(state);
            return true;
        } else {
            return false;
        }
    }
}
```

Figura 4.3.1.6 – State Class

Un exemplu de apel ar fi:

- If (new State(“R_Stop”).modify()) {

 } else {
 displayMessage(“Configuring...please wait...”, 3000);
 }

Metoda displayMessage(...) este folosită în toate activitățile din aplicația Android creată, cu scopul de a afișa mesaje pe durate de timp precise.

```
private void displayMessage(String message, int duration) {
    final Toast toast = Toast.makeText(ActivityManualController.this,
                                         message, Toast.LENGTH_LONG);
    toast.show();
    Handler handler = new Handler();
    handler.postDelayed(new Runnable() {
        @Override
        public void run() {
            toast.cancel();
        }
    }, duration);
}
```

Figura 4.3.1.7 – Metodă de afișare a mesajelor

Un exemplu de apel ar fi :

- displayMessage(“Configuring...please wait...”, 3000) și se va afișa mesajul timp de 3 secunde

Așa cum am menționat, se pot detecta obstacolele întâlnite pe sensul de mers al platformei. Dacă există un obstacol la mai puțin de jumătate de metru, pe baza distanței înregistrate în timp real în Firebase, utilizatorul conectat la sistemul fizic va primi o alertă pe pagina de control manual, printr-o vibrație a cărei intensitate crește cu cât obiectul este tot mai aproape. De asemenea, se modifică layout-ul care va avea un comportament de blink ca în Figura 4.3.1.8.

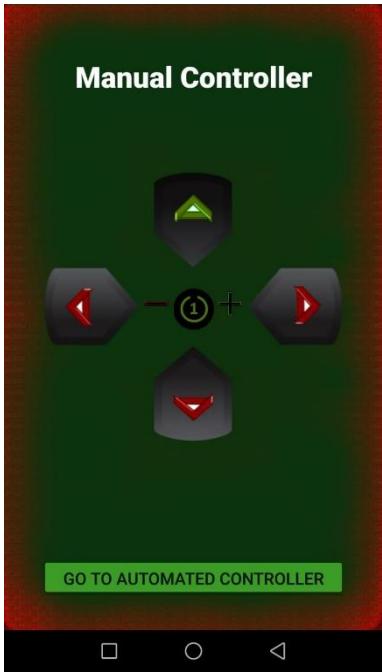


Figura 4.3.1.8

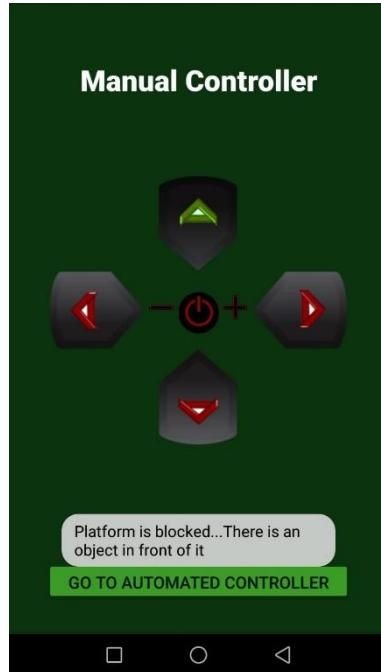


Figura 4.3.1.9

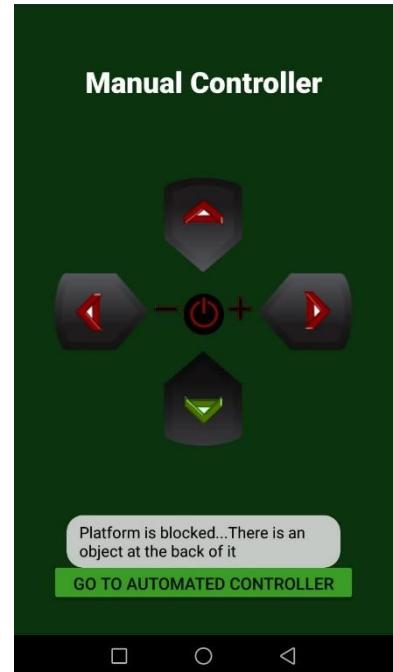


Figura 4.3.1.10

Pentru ca vibrația și alerta vizuală să se sincronizeze parțial, am folosit trei task-uri ce rulează în fundal la intervale de timp potrivite, extinzând clasa TimerTask.

Clasa Alert descrisă în Figura 4.3.1.11 verifică conexiunea cu Firebase, la internet și dacă utilizatorul e conectat la platformă, ia din tabela devices pentru platformă asociată utilizatorului datele referitoare la distanța față de obiectele din față și din spate înregistrate de senzorii ultrasonici și le interpretează corespunzător. Dacă una din distanțe este mai mică de 50 cm, dar mai mare ca distanța de siguranță la care platforma se oprește singură (30 cm), se începe vibrația cu 20 de grade de intensitate diferite în intervalul [50; 1050], la amplitudine maximă.

Clasa AlertShow descrisă în Figura 4.3.12 se ocupă de realizarea alertei vizuale prin modificarea layout-ului. Se verifică ca și la clasa Alert conexiunile, conectarea la platformă și distanțele și în cazul în care se încadrează în intervalul de alertă, și se modifică background-ul.

Clasa LayoutShow descrisă în Figura 4.3.13 asigură că layout-ul revine la background-ul normal după ce nu se mai respectă intervalul de alertă.

Acest mecanism de alertă se oprește dacă distanța măsurată crește peste valoarea maximă de alertă, (> 50), fie dacă scade sub ea (< 30).

```

public class Alert extends TimerTask {
    private double distanceFront;
    private double distanceBack;

    @Override
    public void run() {
        ConnectivityManager conManager = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        if (deviceID != null && conManager.getActiveNetworkInfo() != null && !isInStandBy) {
            FirebaseDatabase.getInstance().getReference(getString(R.string.DeviceDB))
                .child(deviceID).addValueEventListener(new ValueEventListener() {
                    @Override
                    public void onDataChange(DataSnapshot dataSnapshot) {
                        Device myDevice = dataSnapshot.getValue(Device.class);
                        distanceFront = myDevice.getDistanceFront();
                        distanceBack = myDevice.getDistanceBack();
                        if ((distanceFront < 50 || distanceBack < 50)
                            && distanceFront >= 30 && distanceBack >= 30) {
                            for (int i = 50; i > 0; i--) {
                                if (distanceFront < i && distanceFront >= i - 5
                                    || distanceBack < i && distanceBack >= i - 5) {
                                    Vibrator vibrator;
                                    vibrator = (Vibrator) getSystemService(VIBRATOR_SERVICE);
                                    vibrator.vibrate(VibrationEffect.createOneShot((51 - i) * 50, 255));
                                }
                            }
                        }
                    }
                });
    }
}

```

Figura 4.3.1.11 – Alert Task

```

public class AlertShow extends TimerTask {
    private double distanceFront;
    private double distanceBack;

    @Override
    public void run() {
        ConnectivityManager conManager = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        if (deviceID != null && conManager.getActiveNetworkInfo() != null && !isInStandBy) {
            FirebaseDatabase.getInstance().getReference(getString(R.string.DeviceDB))
                .child(deviceID).addValueEventListener(new ValueEventListener() {
                    @Override
                    public void onDataChange(DataSnapshot dataSnapshot) {
                        Device myDevice = dataSnapshot.getValue(Device.class);
                        distanceFront = myDevice.getDistanceFront();
                        distanceBack = myDevice.getDistanceBack();
                        layout = findViewById(R.id.layout);
                        if ((distanceFront < 50 || distanceBack < 50)
                            && distanceFront >= 30 && distanceBack >= 30) {
                            layout.setBackgroundResource(R.drawable.red_shadow);
                        }
                    }
                });
    }
}

```

Figura 4.3.1.12 – AlertShow Task

```

public class LayoutShow extends TimerTask {
    @Override
    public void run() {
        if(!isInStandBy) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    layout = findViewById(R.id.layout);
                    layout.setBackgroundResource(0);
                }
            });
        }
    }
}

```

Figura 4.3.1.11 – LayoutShow Task

Pentru sincronizare, am gestionat perioada de rulare a task-urilor, folosind un singur thread. Programând rularea la perioade fixe de timp, am ales ca task-ul Alert să ruleze la fiecare secundă, iar task-ul de modificare a layout-ului AlertShow și cel al revenirii la background-ul default, LayoutShow, la fiecare 0,5 secunde.

```
Timer timer = new Timer(true);
timer.scheduleAtFixedRate(timerAlert, 0, 1000);
timer.scheduleAtFixedRate(timerAlertShow, 0, 500);
timer.scheduleAtFixedRate(timerLayoutShow, 0, 500);
```

Astfel la o secundă, telefonul va vibra cu intensitatea specifică distanței măsurate, conform cu agloritmul din task-ul Alert. La fiecare 0,5 secunde, va apărea fundalul de alertă, iar la fiecare 0,5 secunde el va reveni la forma default. Fiind același timer, avem un singur thread și modificarea layout-ului se sincronizează, având un efect de blink la fiecare jumătate de secundă. Ținând cont că semnalul care permite vibrația este unul armonic, de amplitudine 255 și de perioadă variabilă de $(51 - i) \cdot 50$ conform cu task-ul Alert, nu se poate realiza o sincronizare cu efectul de blink, dar se sincronizează cu momentul de început al vibrației, prin alegerea unui delay de o secundă.

Dacă se depășeste limita inferioară de 30 cm față de obstacol, atunci alerta anterioară se oprește, platforma se oprește și atributul blocked ia valoare „FRONT” sau „BACK” în funcție de sensul de deplasare al platformei în momentul depistării obstacolului aflat prea aproape pentru a putea să se continue mișcarea curentă.

La oprire apar mesaje de avertizare ca în Figura 4.3.1.9 respectiv Figura 4.3.1.10 în funcție de sensul deplasării, iar din sensul care a generat oprirea, se va observa cum butonul filează pentru a atenționa utilizatorul care este motivul opririi. Acestea nu va putea să se deplaseze în sensul dictat de butonul ce filează și va fi nevoie să se deplaseze opus până ce depășește zona de pericol. La ieșirea din zona de pericol se deblochează butonul și apare un mesaj de înștiințare ca în Figura 4.3.1.14. Dacă se rămâne în intervalul de alertă, aplicația se comportă din nou conform cu Figura 4.3.1.8.

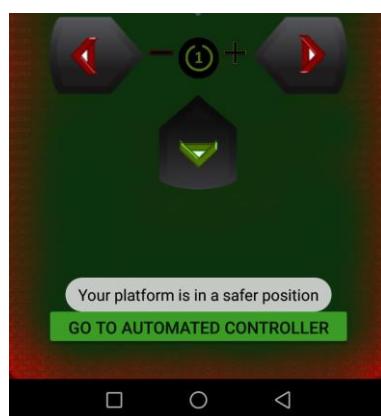


Figura 4.3.1.14 – Ieșire din zona de pericol

Pentru a putea realiza comportamentul descris, am folosit două task-uri sincronizate, moștenind din nou clasa TimerTask.

Primul task, denumit BlockedShow, permite depistarea sensului din care apare obstacolul în momentul în care atributul blocked al platformei este setat la valoarea „FRONT” sau „BACK”, închide consola, blochează butonul din sensul de mers curent și îl aprinde pentru a permite utilizatorului să intuiască ce mișcare trebuie să facă la reaprenderea consolei pentru a ieși din blocaj.

Cel de al doilea task este folosit pentru a stinge butonul cu scopul de a crea un efect de blink asupra acestuia. Acest comportament este foarte important deoarece, dacă utilizatorul care a dus platforma în starea blocked pe „FRONT” sau „BACK” se deconectează și platforma este încă deschisă, utilizatorul următor care se va conecta la platformă va putea să verifice dacă mai există obstacolul pe sensul indicat de aplicație și să schimbe sensul de mers pentru evitare.

Cele două task-uri s-au sincronizat asemănător cu cele precedente.

Este bine de reținut că în condițiile în care starea blocked este activă pe „FRONT” sau „BACK” și se oprește sistemul fizic, se recomandă ca platforma să fie pornită doar atunci când există cel puțin un metru pătrat de pământ fără obstacole în jurul acesteia.

Prin apăsarea butonului „Go To Automated Controller” se poate accesa din nou pagina de control automatizat ce va fi descrisă în subcapitolul următor.

4.3.2 Control automatizat

Așa cum am menționat la începutul Subcapitolului 4.3 și în Capitolul 3, la controlul automatizat, se pot crea programe care presupun deplasarea linie dreaptă cu viraje la 90 °. Platforma se poate deplasa pe distanță mare doar înainte și înapoi, iar deplasarea la stânga sau la dreapta presupune un viraj strâns de 90 ° în direcția indicată, urmată de o deplasare înainte. Fiecare linie parcursă este transpusă în cod ca fiind un pas al programului.

Conform cu clasa Step, un pas are structura următoare :

```
public class Step {
    public String direction; // direcția de deplasare (Front, Back, Left,
                           // Right
    public double distance; // distanța parcursă
    public String distanceUnits; // Centimeters / Meters
    public int numberOfLapses; // numărul de intervale în care se împarte
                           // distanța
    public double delay; // timpul de așteptare după fiecare interval
                           // parcurs
    public String delayUnits; // Seconds / Minutes / Hours
```

Clasa Program este formată dintr-o listă de astfel de pași și are următoarea structură:

```
public class Program {
    public List<Step> steps; // lista cu pașii ce trebuie efectuați
    public boolean comeBack; // flag care indică dacă se dorește ca
                           // platforma să revină la punctul de start
                           // după finalizarea programului
    public boolean on; // flag care indică dacă programul este
                      // pornit
    public String name; // numele programului
```

Pentru crearea și gestionarea pașilor unui program, utilizatorului i se pune la dispoziție o interfață utilă, unde se pot edita sau șterge pașii noi creați. Accesul la pagina de creare se face din pagina de start a aplicației. Când utilizatorul se conectează pentru prima oară în aplicație, pagina de start arată diferit față de cea prezentată anterior, în condițiile în care nu există niciun program înregistrat în baza de date pentru acest utilizator.

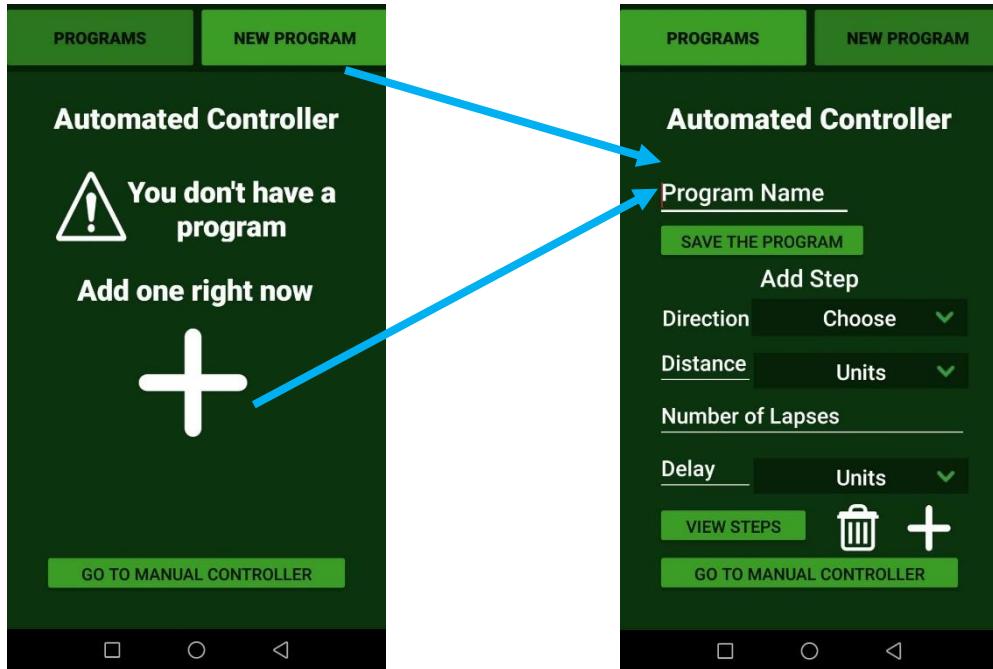


Figura 4.3.2.1 – Adaugare program nou

Prin apăsarea butonului „NEW PROGRAM” din toolbar sau prin apăsarea butonului +, se poate accesa pagina de creare a programelor.

Utilizatorul poate alege pe baza a trei dropdown-uri, direcția de deplasare, ordinul de mărime pentru distanță precum și ordinul de mărime pentru delay ca în Figura 4.3.2.2.

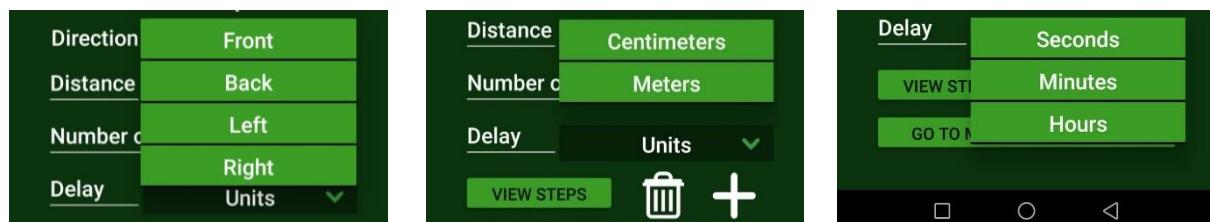
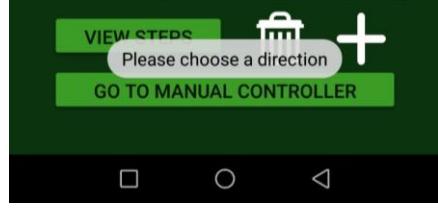
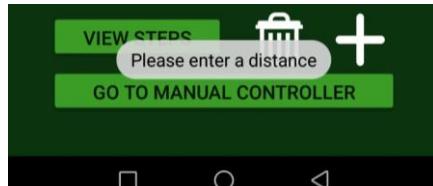
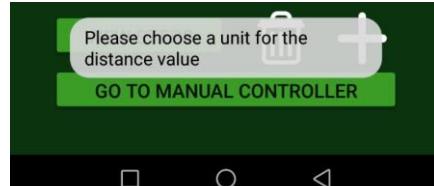
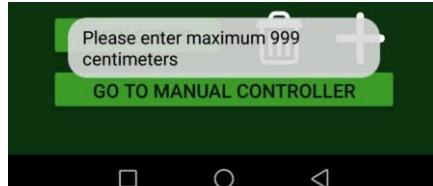
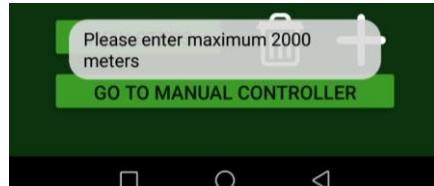
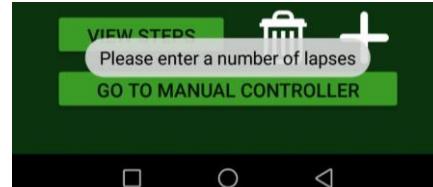
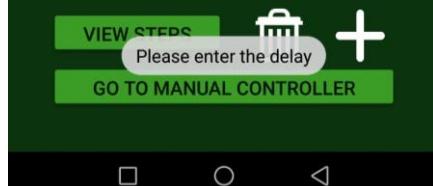
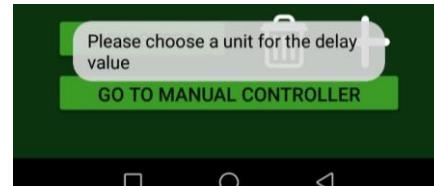
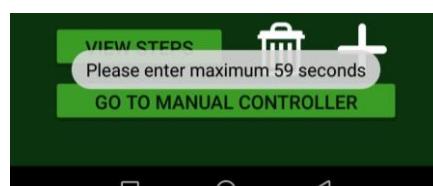
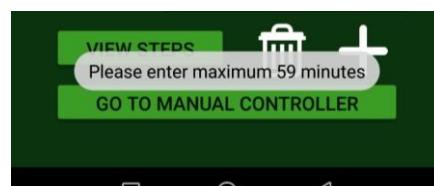


Figura 4.3.2.2 – Alegere din dropdown

Pentru toate field-urile din form, utilizatorul trebuie să introducă o valoare pentru a putea să apese pe butonul + și să adauge pasul nou al programului. La completarea field-urilor, se verifică corectitudinea datelor și se afișează ca și în cazul înregistrării în sistem, mesaje de eroare corespunzătoare prezentate în Tabelul 4.3.2.1.

Field	Tipul erorii	
Direcție		Nu s-a ales o direcție
Distanță		 Nu s-a introdus distanță
Distanță și ordin		 Prea mulți centimetri
Număr de intervale		
Întârziere		 Nu s-a introdus întârzierea pe interval
Întârziere și ordin		 Prea multe secunde

Teabel 4.3.2.1 – Erori de prevenire la adăugarea de pași

Dacă se apasă pe butonul + și nu apare nicio eroare, un nou pas va fi adăugat în listă și va apărea un mesaj de confirmare „A new step was added to your program”. Dacă se va apăsa pe butonul de ștergere, atunci valorile tuturor field-urilor cu excepția field-ului „Program Name” vor fi resetate.

Dacă se părăsește pagina, nu s-a setat un nume pentru program și nu s-a apăsat pe butonul „SAVE THE PROGRAM”, pașii creați nu vor fi salvați.

Pentru vizualizarea și editarea pașilor deja creați se apasă pe butonul „VIEW STEPS” care va deschide o fereastră în care se vor afișa într-un scroll panel toți pașii ca în Figura 4.3.2.3. Dacă nu există niciun pas adăugat, va apărea un mesajul „No step was added”.

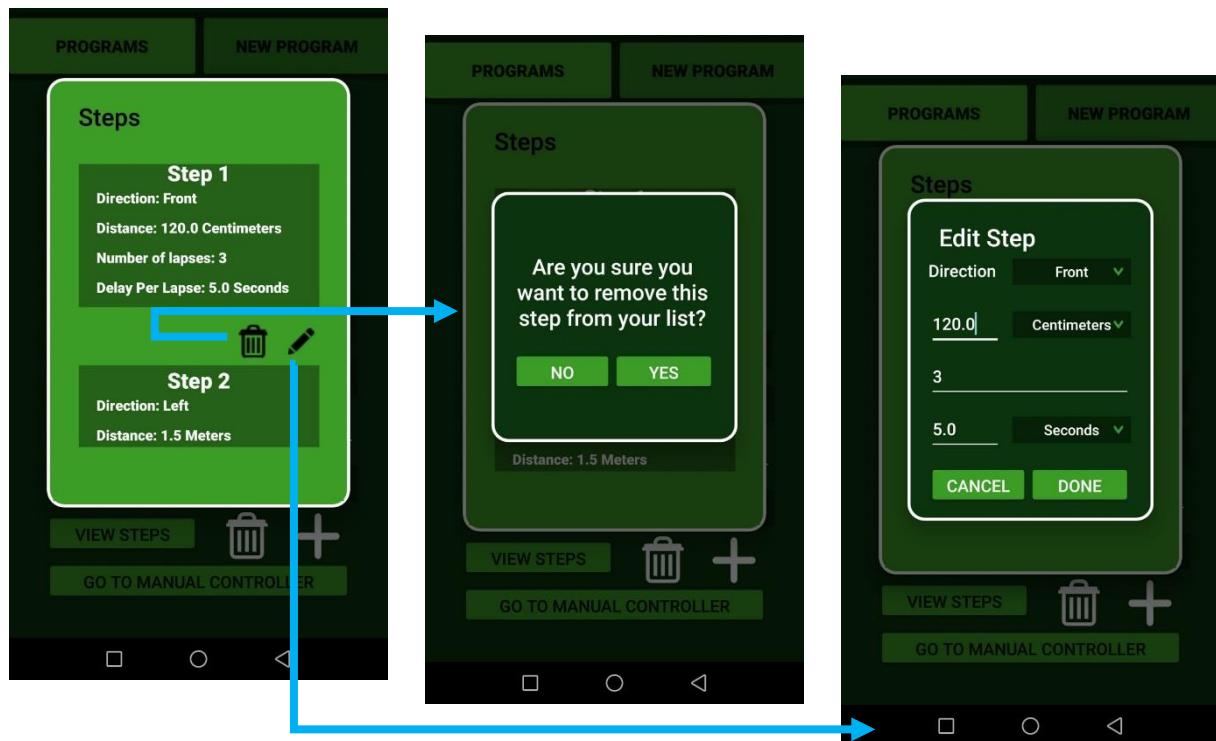


Figura 4.3.2.3 – Editarea și vizualizarea pașilor

Această pagină s-a creat dinamic și prezintă detaliile referitoare la fiecare pas în parte. De asemenea, pentru orice pas se asociază un buton de ștergere și unul de editare conform cu Figura 4.3.2.3.

Dacă se optează pentru ștergerea pasului, va apărea o fereastră de asigurare care cere permisiunea de ștergere sau anulare a comenzi de ștergere. La ștergere, dacă există mai mulți pași în listă, pagina de vizualizare și editare rămâne deschisă și se reactualizează lista de pași. La ștergerea tuturor pașilor se va închide fereastra de vizualizare. Apăsarea ulterioră a butonului „VIEW STEPS” va genera mesajul de avertizare corespunzător („No step was added”), dar dacă s-a ales un nume pentru program, acesta se va păstra.

La apăsarea butonui de edit, peste fereastra deschisă va apărea o nouă fereastră create tot dinamic, cu aspect asemănător paginii de creare a programelor și se vor putea edita fieldurile la cerere. (A se vedea Figura 4.3.2.4). La apăsarea butonului „Done” se testează

corectitudinea datelor pe baza Tabelului 4.3.2.1. Dacă datele introduse sunt corecte, atunci pasul se editează, se închide fereastra și se actualizează pasul din listă.



Figura 4.3.2.4 – Editarea unui pas

După finalizarea editări pașilor, utilizatorul poate crea programul și să îl trimită în baza de date după completarea câmpului Program Name și apăsarea pe butonul „SAVE THE PROGRAM”. Dacă nu există niciun pas adăugat programul nu se va crea deoarece acesta trebuie să aibă cel puțin un pas adăugat în listă și prin urmare va apărea un mesaj de eroare corespunzător.

La salvare, se trimit către Firebase programul cu lista de pași și câmpurile on, comeBack setate pe false. Numele va deva nod așa cum se poate observa în Figura 4.3.2.5. Pentru un program de test numit „Test”, vom avea o structură ca cea din Figura 4.3.2.5. În plus, la nodul steps se va adăuga și câmpul total care ne spune numărul total de pași ai programului.

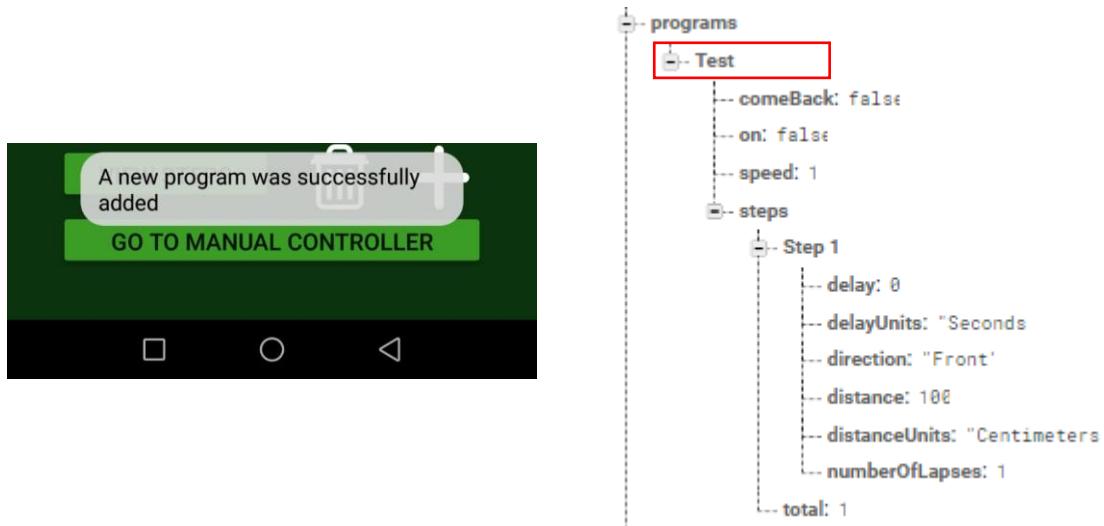


Figura 4.3.2.5 – Salvarea și adăugarea programului în baza de date

La revenirea pe pagina principală prin apăsarea butonului „PROGRAMS” se poate vedea o nouă structură a paginii ca în Figura 4.3.2.6.

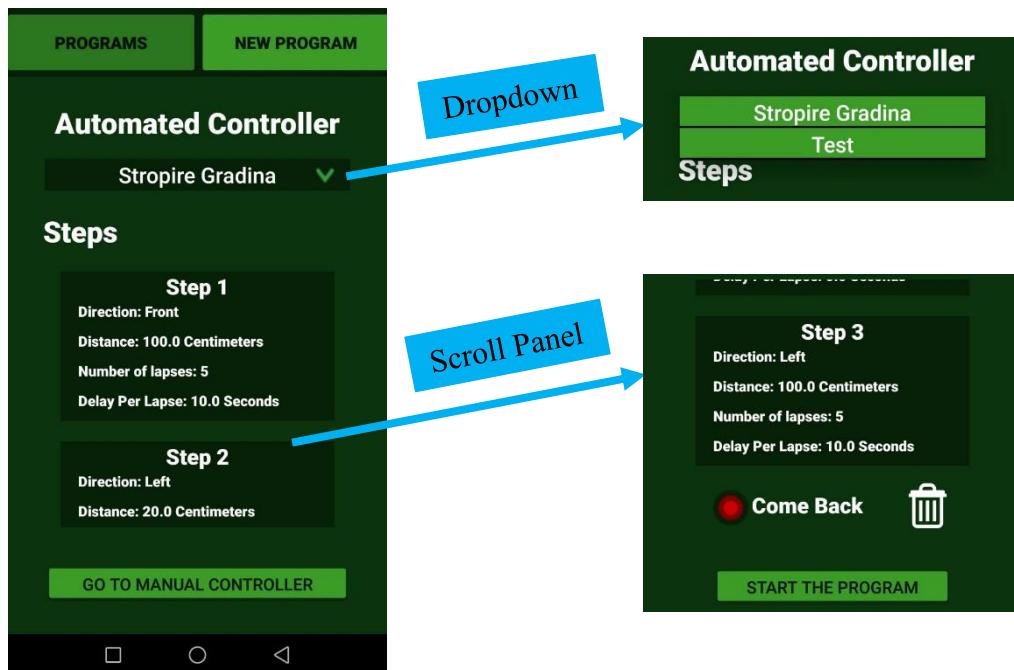


Figura 4.3.2.6 Pagina de pornire cu program adăugat

Se poate observa că după adăugarea de programe în baza de date, este permisă selectarea programului dorit din dropdown și se poate da scroll pentru a vedea toți pașii și setările. La final se poate opta pentru ștergerea programului sau să se ceară platformei întoarcerea înapoi în punctul de start după terminarea programului din butonul „Come Back” care va deveni verde ca în Figura 4.3.2.7.

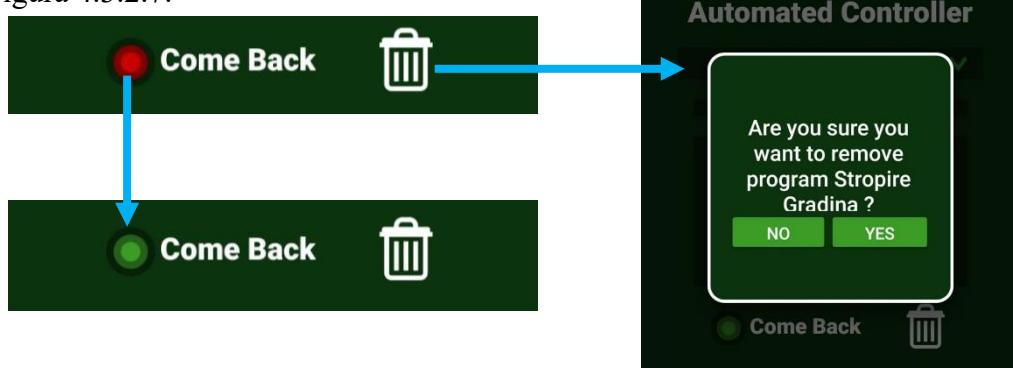


Figura 4.3.2.7 – Setări

La apăsarea pe butonul „Come Back”, în baza de date pentru programul corespunzător se schimbă flagul comeBack în true.

Dacă se dorește ștergerea prin apăsarea butonului delete va apărea o fereastră de asigurare care oferă opțiunea de ștergere sau de anulare.

În cele din urmă, prin apăsarea butonului „START THE PROGRAM” va porni programul, butonul de start va deveni buton de oprire al programului cu textul „STOP THE

PROGRAM”, iar dacă se dorește ștergerea sau editarea, va apărea un mesaj de eroare ca în Figura 4.3.2.8.

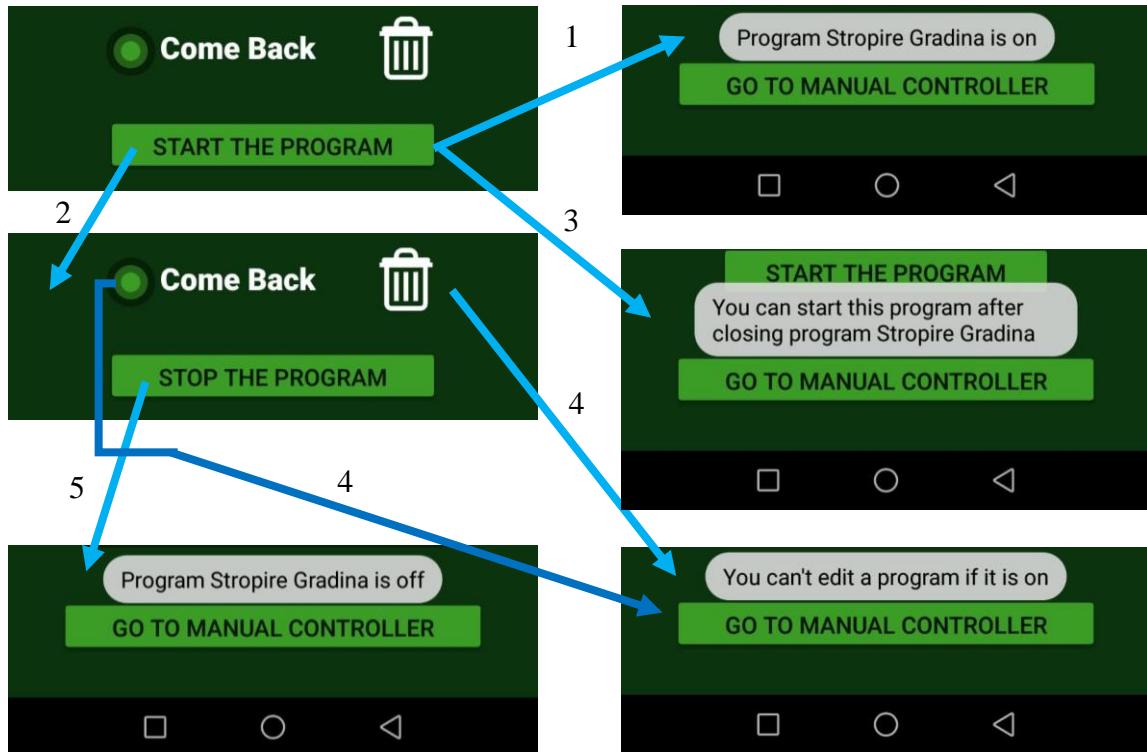


Figura 4.3.2.8 – Oprirea și pornirea programelor

Figura 4.3.2.8 descrie mecanismul de pornire și oprire al programelor și acțiunile permise sau interzise.

1. La pornirea programului apare un mesaj corespunzător și se salvează programul în baza de date la tabela devices pentru platforma la care este utilizatorul conectat, iar în tabela users pentru utilizatorul în cauză, la programul dat spre execuție, flag-ul on devine true.
2. La pornirea programului, butonul de start devine buton de stop.
3. Dacă se dorește pornirea unui alt program în timp ce altul este încărcat pe platformă, apare un mesaj de eroare.
4. Dacă se dorește modificarea setărilor în timp ce un program este pornit, atunci apare un mesaj de eroare.
5. La oprirea programului apare un mesaj corespunzător și se șterge programul din tabela devices pentru platforma la care este utilizatorul conectat, și flag-ul on corespunzător programului din tabela users devine false.

Așa cum am explicitat în Capitolul 3 și la subpunctul 1, pornirea programului duce la încărcarea acestuia în baza de date în tabela devices, la platformă cu care și-a asociat utilizatorul contul și la care este conectat în prezent. Ulterior programul este preluat și încărcat în codul mașinii.

Pe baza unei clase scrise în C++, s-a creat un model pentru un pas al programului și în codul sursă încărcat pe microcontroller. Clasa Step scrisă în C++ este diferită față de cea din Java deoarece a trebuit să se transpună informațiile referitoare la direcție, distanță, întârziere și unități în informații utile și ușor de procesat de mașină. Distanța trebuie convertită mereu în

centimetri, iar timpul în milisecunde, prin rotunjire. Direcțiile vor fi convertite la valori numerice de la 1 la 4 (1 – față, 2 – spate, 3 – stânga, 4 – dreapta), ordinul de mărime al distanței pasului va fi convertit în valori de la 1 la 2 (1 – centimetri, 2 – metri), iar ordinul de mărime al întârziere în valori de la 1 la 3 (1 – secunde, 2 – minute, 3 – ore).

Ulterior se va face conversia la centrimetri și milisecunde, în funcție de valorile numerice returnate. Dacă dorim să calculăm distanța și ordinul de mărime asociat cu valoarea numerică 2 (metri) se va înmulți valoarea corespunzătoare distanței cu 100. Dacă dorim să calculăm întârziere pentru fiecare interval, dacă valoarea numerică asociată ordinului de mărime a întârzierii este 1 (secunde) se va înmulți valoarea corespunzătoare întârzierii cu 1000, dacă este 2 (minute) se va înmulți cu 60000 și dacă este 3 (ore) se va înmulți cu 3600000, urmând să convertească din double în long pentru a putea apela funcția delay(long millis).

Microcontroller-ul verifică la fiecare pas dacă programul încă mai există în baza de date, și dacă da, ia pe rând numele programului, flag-ul comeBack pentru a ști dacă sistemul fizic trebuie să revină la punctul de start, și lista de pași. După ce s-a salvat fiecare pas într-un obiect de tip Step, se verifică dacă programul încă este pornit și dacă da, pe baza algoritmului descris în Capitolul 3 el execută pasul. La finalul programului, dacă utilizatorul nu oprește forțat și nu a optat pentru revenirea la punctul de start, nodului „program” din tabelă i se adaugă atributul booleean wait și alertOn active (true), și platforma intră în starea de așteptare până primește răspuns de la utilizator din aplicația mobilă. Altfel, dacă trebuie să revină la start el continuă să se deplaseze după algoritmul de revenire descris în Capitolul 3 și la final salvează în baza de date numele programului care tocmai s-a terminat într-un atribut numit comeBackFrom.

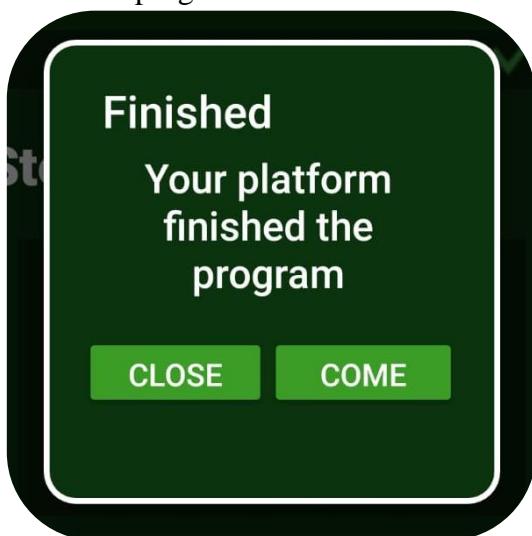


Figura 4.3.2.9 – Finalizare fără revenire

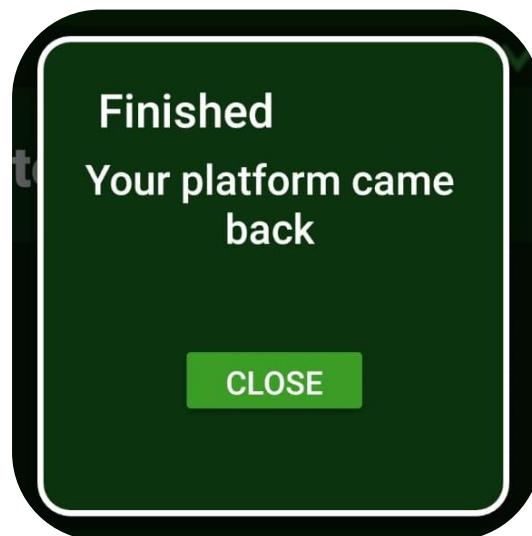


Figura 4.3.2.10 – Finalizare cu revenire

În momentul în care flag-urile alertOn și wait sunt activate, se va deschide în aplicație o fereastră ca în Figura 4.3.2.9 care ne avertizează și printr-un semnal sonor corespunzător, de finalizarea programului cu opțiunea de a opri programul de tot „Cancel” sau să cerem platformei să revină la punctul de start prin apăsarea butonului „Come” și flag-ul wait devine false în ambele cazuri. La deschiderea ferestrei flag-ul alertOn devine false, dar până la confirmarea utilizatorului, flag-ul wait rămâne true. După primirea răspunsului platforma fizică verifică dacă programul încă mai e pornit și dacă trebuie să revină în punctul de start. Dacă nu, se oprește și șterge programul din tabela devices. Tot la apăsarea butonului „Cancel”, în tabela devices pentru programul care tocmai s-a terminat, flag-ul on devine false.

În momentul în care utilizatorul cere revenirea și aceasta se finalizează, se salvează în baza de date numele programului care s-a finalizat, se șterge programul și în aplicație se primește o notificare printr-un semnal sonor și o fereastră care ne spune că platforma a revenit, ca în Figura 4.3.2.10. La închiderea ferestrei sau la apăsarea butonului cancel, în tabela devices pentru programul care tocmai s-a terminat, flag-ul on devine false. La apăsarea butonului „Cancel” se șterge atributul comeBackFrom

Același comportament descris în ultimul paragraf îl întâlnim și atunci când, înainte de începerea programului, s-a optat pentru revenire și evident că fereastra corespunzătoare cu Figura 4.3.2.9 nu mai apare.

Afișarea acestor mesaje de finalizare se realizează cu ajutorul unui thread de tip timer care rulează în orice pagină a contului, la fiecare 20 de secunde două task-uri. Unul denumit ProgramFinished și altul CameBack, ambele moștenind clasa TimerTask.

Task-ul ProgramFinished verifică dacă programul este încă pornit și dacă da, în cazul în care flag-urile wait și alertOn sunt active, se afișeză popup-ul corespunzător cu Figura 4.3.2.9.

Task-ul CameBack verifică dacă programul este încă pornit și dacă da, se caută atributul comeBackFrom. Dacă acesta este găsit, se deschide popup-ul corespunzător.

Cele două task-uri sunt descrise mai jos în Figura 4.3.2.11 respectiv Figura 4.3.2.12.

```
public class ProgramFinished extends TimerTask {
    @Override
    public void run() {
        ConnectivityManager conManager = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        if (deviceID != null && conManager.getActiveNetworkInfo() != null && !isInStandBy) {
            FirebaseDatabase.getInstance().getReference(getString(R.string.DeviceDB))
                .child(deviceID).addListenerForSingleValueEvent(new ValueEventListener() {
                    @Override
                    public void onDataChange(DataSnapshot dataSnapshot) {
                        if (dataSnapshot.child("program").exists()) {
                            if (dataSnapshot.child("program").hasChild("wait") && dataSnapshot.child("program").hasChild("alertOn")) {
                                boolean wait = (boolean) dataSnapshot.child("program").child("wait").getValue();
                                boolean alertOn = (boolean) dataSnapshot.child("program").child("alertOn").getValue();
                                if (wait && alertOn) {
                                    timer.cancel();
                                    startActivity(new Intent(getApplicationContext(), PopupProgramFinished.class));
                                }
                            }
                        }
                    }
                });
    }
}
```

Figura 4.3.2.11 – ProgramFinished Task

```
public class CameBack extends TimerTask {
    @Override
    public void run() {
        ConnectivityManager conManager = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        if (deviceID != null && conManager.getActiveNetworkInfo() != null && !isInStandBy) {
            @OverrideS
            public void onDataChange(DataSnapshot dataSnapshot) {
                if (dataSnapshot.child("cameBackFrom").exists()){
                    timer.cancel();
                    startActivity(new Intent(getApplicationContext(), PopupCameBack.class));
                }
            }
            @Override
            public void onCancelled(DatabaseError databaseError) {
            }
        }
    }
}
```

Figura 4.3.2.12 – CameBack Task

4.4 Alte funcționalități

Există și alte funcționalități ce nu pot fi clasificate. De exemplu în meniul de navigație din Figura 4.2.3 se pot observa butoanele Log out și My Account.

La apăsarea butonului „Log Out” utilizatorul este delegat cu condiția ca acesta să nu fie conectat la sistemul fizic și prin urmare să nu existe niciun program care rulează în momentul respectiv. (a se vedea Figura 4.4.1). Acest lucru este valabil și la apăsare link-ului „Delete My Account” din pagina My Account (Figura 4.4.4). De asemenea, dacă conexiunea la internet este slabă, apare un mesaj de eroare ca în Figura 4.3.1.5 și nu se poate accesa nicio pagină din meniul de navigare sau deconectarea propriu-zisă.

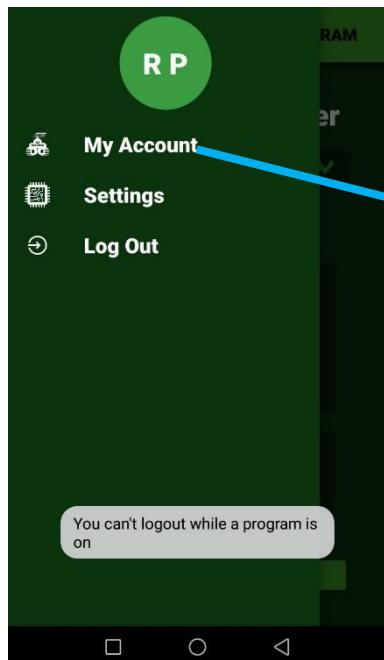


Figura 4.4.1

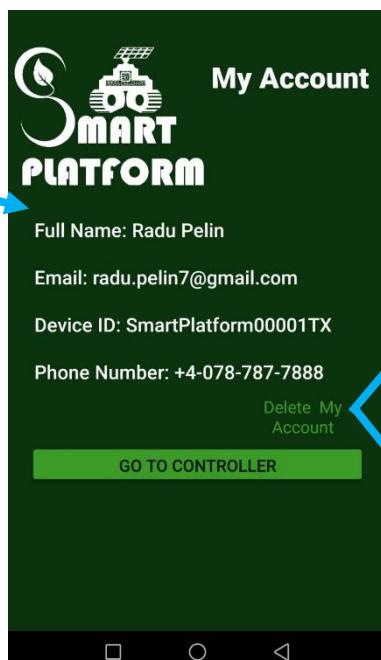


Figura 4.4.2

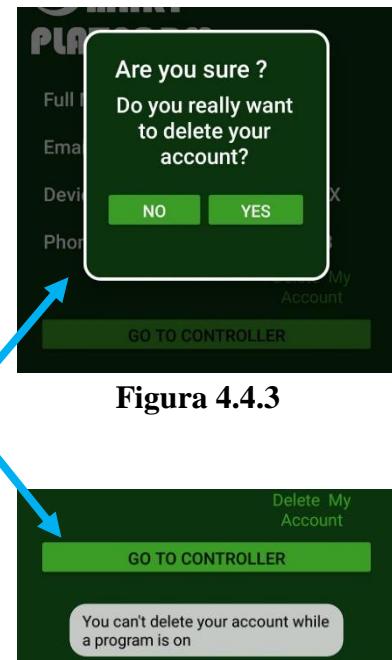


Figura 4.4.3

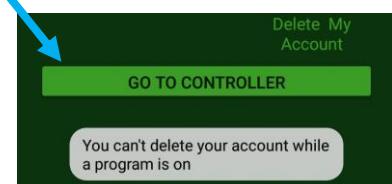


Figura 4.4.4

După cum se poate observa în Figura 4.4.2, la apăsarea pe link-ul „Delete My Account”, dacă nu există un program pornit, va apărea un popup de confirmare ca în Figura 4.4.3 și dacă ștergerea contului s-a realizat cu succes se afișează un mesaj de confirmare și delegarea de la cont.

Pentru pagina Settings, în condițiile în care există un program care rulează pe platformă, se va afișa un mesaj de eroare la încercarea de modificare a setărilor, ca în Figura 4.4.5.

De asemenea, pagina de control manual este inaccesibilă pe toată durata rulării unui program, ca în Figura 4.4.6.

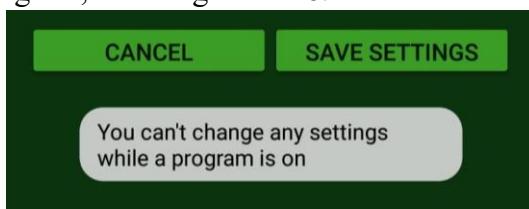


Figura 4.4.5

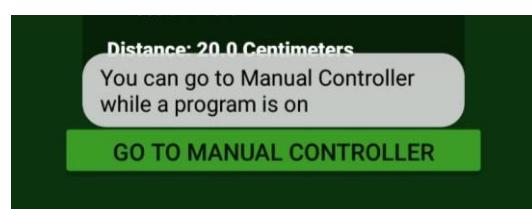


Figura 4.4.6

Există un task care rulează pe toate paginile contului pentru a determina dacă nu cumva platforma a fost deconectată și reaprinsă iar, iar utilizatorul conectat în aplicație, nu mai apare conectat în baza de date și nu poate controla platforma.

Acest task numit Connected ce moștenește clasa TimerTask rulează în background la fiecare secundă pentru a ne asigura că în momentul repornirii sistemului fizic, utilizatorul este pus în stand by și orice program care rula este oprit, acesta fiind redirecționat pe pagina principală de control automatizat.

```
public class Connected extends TimerTask {
    @Override
    public void run() {
        if (deviceID != null && !isInStandBy) {
            FirebaseDatabase.getInstance().getReference(getString(R.string.DevicesDB))
                .child(deviceID).addSingleValueEventListener(new ValueEventListener() {
                    @Override
                    public void onDataChange(DataSnapshot dataSnapshot) {
                        if (!dataSnapshot.child("connected").exists()) {
                            databaseUsers.child(user.getUid()).child("standBy").setValue(true);
                            databaseUsers.child(user.getUid()).child("programs").addValueEventListener(new ValueEventListener() {
                                @Override
                                public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                                    for(DataSnapshot data1 : dataSnapshot.getChildren()) {
                                        for(DataSnapshot data2 : data1.getChildren()) {
                                            if(data2.getKey().equals("on")) {
                                                if((boolean) data2.getValue()) {
                                                    databaseUsers.child(user.getUid()).child("programs")
                                                        .child(data1.getKey()).child("on").setValue(false);
                                                }
                                            }
                                        }
                                    }
                                }
                            });
                        }
                    }
                });
            timer.cancel();
            finishAffinity();
            startActivity(new Intent(getApplicationContext(), ActivityAutomatedControlStart.class));
        }
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
    }
}
}
```

Figura 4.4.7 – Connected Task

Capitolul 5

Obținerea sistemului fizic final

Pentru obținerea prototipului final, a trebuit să se țină cont de ajustarea sistemului de urmărire solară în condițiile în care acesta trebuie să se sincronizeze cu mișcarea platformei.

De asemenea, este foarte important modul în care se încarcă bateria ce alimentează platforma și panoul solar ales. Pentru încărcare s-a folosit un controller de încărcare sau așa numitul circuit de încărcare solară.

5.1 Alegerea circuitului de încărcare și a panoului solar

A. Alegerea circuitului de încărcare

Există o gamă largă de circuite de încărcare pe piață. Acestea pot fi folosite pentru orice tip de panou solar sau baterie, cu condiția de a alege panoul în funcție de bateria care alimentează sistemul.

Pentru a putea fi folosite în orice aplicație practică este bine de știut că nu orice model se potrivește cu cerințele aplicației în care se cere utilizarea circuitelor de încărcare. De aceea trebuie ales cu grijă tipul de circuit: PWM (Pulse Width Modulation) sau MPPT (Maximum Power Point Tracking).

“Controlerul PWM este în esență un intrerupător care conectează o matrice de celule fotovoltaice la o baterie. Rezultatul este că tensiunea rețelei va fi trasă în jos aproape de cea a bateriei. Controlerul MPPT este mai sofisticat (și mai scump): își va ajusta tensiunea de intrare pentru a recupera puterea maximă a panoului solar și apoi va transforma această putere pentru a satisface cerința de tensiune variabilă a bateriei. Astfel, acesta decuplează tensiunile de tensiunea bateriei, astfel încât să existe, de exemplu, o baterie de 12 volți pe o parte a regulatorului de încărcare MPPT și o sursă de 36 de volți pe cealaltă parte prin conectarea în serie a celulelor fotovoltaice.”^[19]

“PWM este cea mai eficientă metodă de obținere a unei tensiuni constante de încărcare a bateriei.” (Morningstar Corporation, pg. 1, 2014). Se recomandă folosirea controller-ului de încărcare PWM pentru sistemele mici, când temperatura celulelor solare este moderată până la înaltă (între 45 ° C și 75 ° C).

Având un panou solar mic care alimentează un prototip, am putut folosi pentru acest proiect un circuit de încărcare PWM, cu un curent de încărcare de 10 A, bun pentru baterii de 12 V, cu un LCD și trei butoane pentru interfață cu utilizatorul și două porturi USB de ieșire (5 V, 2.5 A). Am folosit cele două porturi pentru a alimenta direct placă de dezvoltare Arduino Uno și NodeMCU ESP8266.

Controller-ul funcționează la temperaturi cuprinse între 35 ° C și 60 ° C .



Figura 5.1.1 – Circuit de încărcare

B. Alegerea panoului solar

Panoul solar ales este foarte bun pentru încărcarea bateriilor de 12 V atât cu plumb cât și cu litiu, și a fost ales datorită dimensiunilor și greutății reduse. Am ales acest panou solar deoarece sistemul DAST creat este un prototip și trebuie avut grijă ca gabaritul planului orizontal să nu depășească 1,8 kg pentru a nu solicita peste măsură servomotorul FT90M.

Panoul are o greutate de doar 120 g și este de asemenea flexibil. Ca și dimensiuni acesta are $440 \cdot 190 \cdot 3$ mm. Din punct de vedere al specificațiilor tehnice, panoul are o putere maximă de 10 W și o tensiune la putere maximă de 18 V, având celule monocristaline de eficiență ridicată.



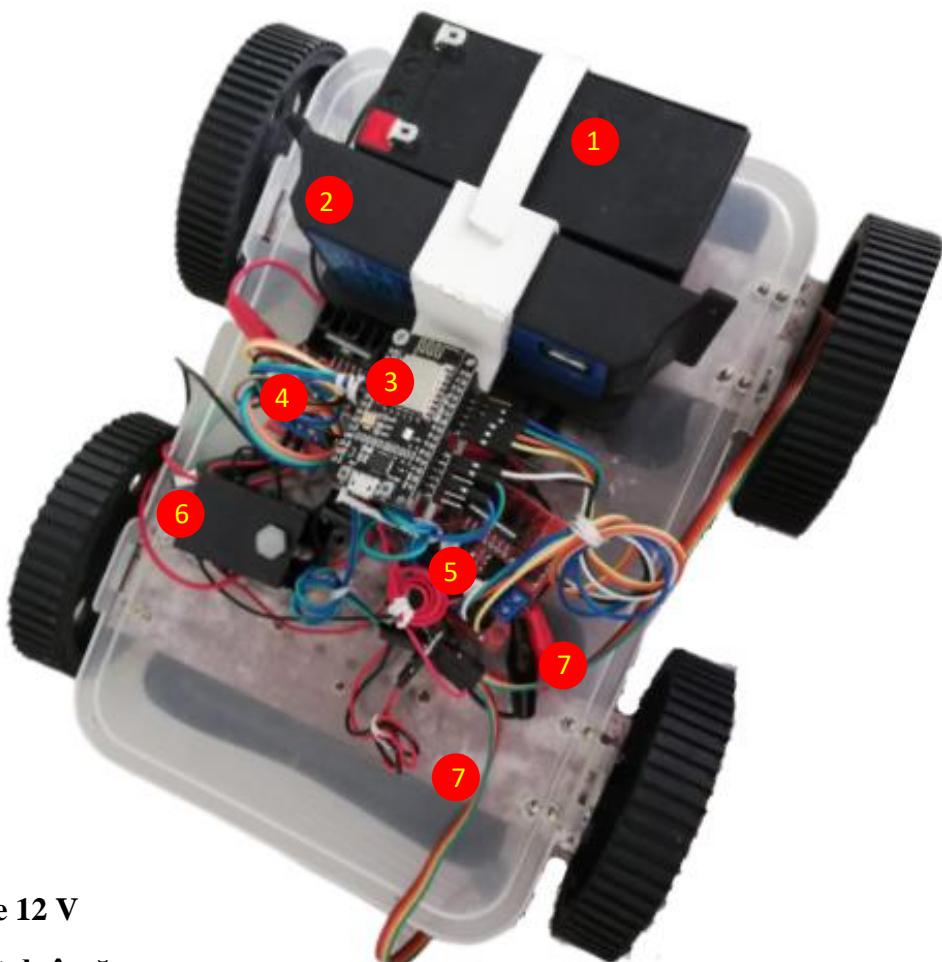
Figura 5.1.2 – Panou solar

În Subcapitolul 3.2 în care s-a dezbatut integrarea, se poate vedea cum se conectează circuitul de încărcare și panoul solar cu restul sistemului.

Pentru obținerea sistemului fizic final s-a confectionat o carcăsa atașată de platformă, deasupra căreia s-a prins sistemul de urmărire ce poate fi vizualizat în Subcapitolul 2.3. Panoul solar se atașează de sistemul de urmărire, aşa cum se va observa în ultimul subcapitol al acestei lucrări.

Circuitul de încărcare a rămas atașat direct de șasiul platformei alături de baterie, driverele de motoare și placa de dezvoltare NodeMCU ESP8266. Am realizat și o cutie izolată de unde se face distribuția tensiunii de la baterie la cele două drivere folosite. Senzorii ultrasonici, împreună cu înterupătorul au fost incorporați în carcăsa, aşa cum se va putea vedea în Subcapitolul 5.3.

În Figura 5.1.3 se poate vedea șasiul, împreună cu toate componentele atașate pe acesta.



1 – Baterie 12 V

2 – Circuit de încărcare

3 – NodeMCU ESP8266

4 – Controller motoare față

5 – Controller motoare spate

6 – Cutie izolată

7 – Conectarea senzorilor ultrasonici

Figura 5.1.3 – Șasiu

5.2 Sincronizarea cu sistemul DAST

Algoritmul de urmărire bazat pe umbre, a fost descris în Subcapitolul 1.2 al lucrării, unde am vorbit despre soluția de urmărire aleasă.

În algoritmul prezentat s-a ales ca după depistarea maximului de lumină, sistemul să nu mai urmărească timp de 30 de secunde, pentru că soluția să rămână mai eficientă decât alte sisteme fixe sau pe o axă. Această valoare am obținut-o prin calcul în Tabelul 1.1.1, unde am observat că eficiența față de sistemele fixe rămâne mai mare la o întârziere de aproximativ 30 de secunde. Datele au fost generate pe baza analizei rezultatelor obținute în lucrarea *A simple and low-cost active dual-axis solar tracker*.

Pentru urmărirea solară în mișcare continuă, metoda care presupune o întârziere după depistarea maximului de lumină nu permite sincronizarea mișcării platformei cu mecanismul de urmărire. Pentru a putea realiza această sincronizare, avem nevoie să știm în permanentă care este poziția mașinii față de un punct de referință.

În momentul în care sistemul DAST a realizat urmărirea și se oprește, se înregistrează unghiul mașinii față de referință dată. Sistemul rămâne oprit, fie până trec 30 de minute fie până când unghiul platformei față de cel înregistrat anterior este mai mare de 10° . Astfel, ne asigurăm că eficiența sistemului DAST se păstrează deoarece pentru deplasări în linie dreaptă sistemul de urmărire se activează doar odată la jumătate de oră, când se depistează modificarea unghiului razelor solare față de planul orizontal. Am ales o astfel de temporizare pentru a compensa consumul mare de energie necesar la mișcarea continuă a platformei, când se efectuează un număr mare de viraje. Unghiul de 10° , deși este la prima vedere unul destul de mare, a fost ales ca în cazul apariției unei suprafete neregulate pe traseu sistemul să nu efectueze urmărirea la cea mai mică modificare a unghiului, pentru a ne asigura că nu se realizează urmăririri inutile.

Valorile alese pentru reimplementarea algoritmului au fost luate fără să se realizeze o cercetare în prealabil referitoare la eficiență ($\frac{\text{putere câștigată}}{\text{putere pierdută}}$). Dacă se va dori ca pe baza proiectului să se realizeze un studiu asupra eficienței sistemului DAST creat în această lucrare pentru crearea unui mecanism de urmărire mai robust, este foarte posibil să fie nevoie de ajustarea toleranțelor atât pentru unghi (10°) cât și pentru întârziere (30 minute).

Pentru calculul diferenței de unghi a fost nevoie de folosirea magnetometrului HMC5883L ce joacă rolul de compas digital în această aplicație.

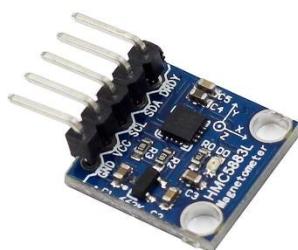


Figura 5.2.1 – HMC885L

Magnetometrul HMC5883L măsoară direcția și magnitudinea câmpului magnetic al Pământului și, prin urmare, este utilizat pentru calculul de unghiuri la costuri reduse și pentru magnetometrie. Măsoară valoarea câmpului magnetic al Pământului de-a lungul axelor X, Y și Z. Poate fi folosit ca busolă pentru a găsi direcția sau direcția de poziționare a unui dispozitiv așa cum îl voi folosi în această lucrare. Folosește protocolul I2C pentru a comunica cu un microcontroler. Valorile câmpului magnetic al terenului de-a lungul axelor X, Y și Z pot fi găsite prin citirea valorilor din adresele anumitor registre utilizând comunicația I2C.

Așadar, schema electrică prezentată în Subcapitolul 2.2.2 se va modifica ca în Figura 5.2.2.

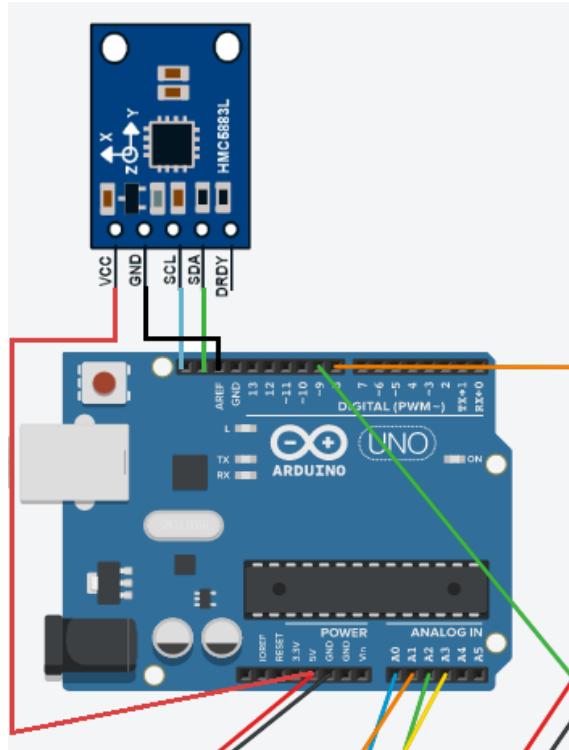


Figura 5.2.2 – Modificarea circuitului

Pentru interfațarea cu HMC5883L și folosirea protocolului I2C este nevoie de instalarea bibliotecilor Wire.h și HMC5883L_Simple.h.

Pentru folosirea compasului digital, avem nevoie să cunoaștem declinația câmpului magnetic din zona în care ne aflăm pe glob. “Declinația magnetică este unghiul dintre câmpul magnetic al Terrei în punctul din care se calculează și nordul geografic. Declinația este pozitivă atunci când nordul magnetic este la est de nordul geografic și variază în timp și spațiu.”^[21]

Pentru București, declinația magnetică este de $5^{\circ} 45'$, iar declinația propriu-zisă este Pozitiv Est conform cu Figura 5.2.3. Așadar, este recomandat să se cunoască locația în care va fi folosit sistemul de urmărire pentru a avea un calcul de precizie.

Setup-ul folosit pentru a putea utiliza HMC5883L este următorul:

1. Se declară global un obiect de tip HMC5883L_Simple numit de exemplu compass.

2. Se declară două variabile float globale last_heading și next_heading folosite pentru comparație. Variabila last_heading se actualizează când sistemul DAST a găsit punctul optim, iar next_heading se va actualiza cât timp sistemul DAST este în repaus.
3. Wire.begin();
4. Se setează declinația: compass.SetDeclination(5, 45, 'E') în cazul nostru.
5. Se setează modul de citire a valorii de la senzor. Se poate citi continuu sau doar o valoare: compass.SetSamplingMode(COMPASS_SINGLE)
COMPASS_SINGLE - citește o singură valoare,
COMPASS_CONTINUOUS – citește valori continuu.
6. Se setează gradul de sensibilitate al senzorului pe baza unor scări de precizie. Există opțiunile 088, 130 (default), 190, 250, 400, 470, 560, 810 , și cu cât valoarea este mai mică cu atât sensibilitatea este mai mare. În cazul nostru compass.setScale(COMPASS_SCALE_130) .
7. Se setează orientarea compasului, în cazul nostru orizontal pe x, conform și cu sistemul de axe virtuale descriși în Capitolul 3:
compass.setOrientation(COMPASS_HORIZONTAL_X_NORTH);
8. Se citește de la senzor: last_heading = compass.GetHeadingDegrees() la finalizarea urmăririi.
9. Se citește de la senzor: next_heading = compass.GetHeadingDegrees() cât timp sistemul de urmărire e în repaus.

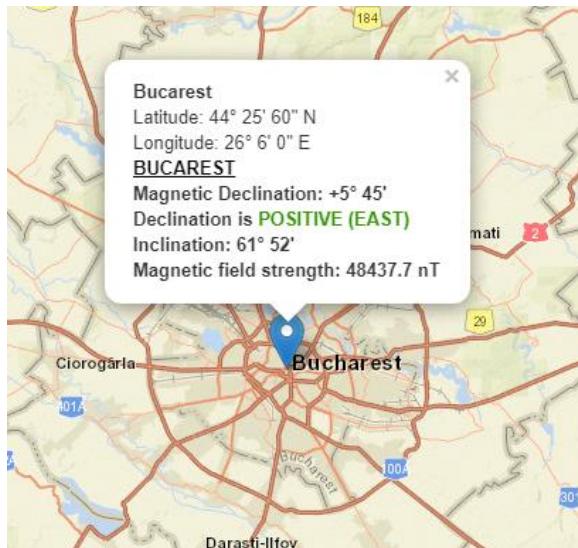


Figura 5.2.3 – Declinația magnetică în București

5.3 Rezultat final



Bibliografie

- [1] Kamrul Islam Chowdhury, Md.Iftekhar-ul-Alam, Promit Shams Bakshi Performance Comparison Between Fixed Panel, Single-axis and Dual-axis Sun Tracking Solar Panel System, 2017.
- [2] Aboubakr El Hammoumi, Saad Motahhir, Abdelaziz El Ghzizal, Abdelilah Chalh, Aziz Derouich A simple and low-cost active dual-axis solar tracker, 2018
- [3] FOTOREZISTENTA. (LDR - Light Dependent Resistor). (05.06.2019) Sursa: http://cursuri.flexform.ro/courses/L2/document/Cluj-Napoca/grupa1/Aranyi_Iulia/site/fotorezistenta.html
- [4] FOTODIODA. (05.06.2019) Sursa: http://cursuri.flexform.ro/courses/L2/document/Cluj-Napoca/grupa1/Aranyi_Iulia/site/fotodiода.html
- [5] FOTOTRANZISTOR. (05.06.2019) Sursa: http://cursuri.flexform.ro/courses/L2/document/Cluj-Napoca/grupa1/Aranyi_Iulia/site/fototranzistor.html
- [6] Sabir Hossain, Bodius Salam, Al Shahriar, Manash Chakraborty AZIMUTH ALTITUDE DUAL AXIS SOLAR TRACKER, 2015
- [7] Ce sunt sistemele CNC și care sunt principalele lor avantaje. (06.06.2019)
Sursa: <https://www.meprouutilaje.ro/blog/despre-mepro-utilaje/>
- [8] Atmel Corporation 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash DATASHEET, 2019; 1:2
- [9] BH1750 Digital Light Sensor. (08.06.2019) Sursa: <https://www.instructables.com/id/BH1750-Digital-Light-Sensor/>
- [10] Redmond Ramin Shamshiri, Cornelia Weltzien, Ibrahim A. Hameed, Ian J. Yule, Tony E. Grift, Siva K. Balasundram, Lenka Pitonakova, Desa Ahmad, Girish Chowdhary Research and development in agricultural robotics: A perspective of digital farming, 2018
- [11] Sweet Pepper Harvesting Robot. (09.06.2019)
Sursa: <http://www.sweeper-robot.eu/>
- [12] The University of Sydney IGNITE October 2015. (09.06.2019)
Sursa: <https://wordvine.sydney.edu.au/files/1733/10311/>
- [13] Rough Terrain Metal Monster Truck Chassis Body. (15.06.2019)
Sursa: <https://nevoneexpress.com/Rough-Terrain-Metal-Monster-Truck-Chassis-Body.php>
- [14] How to Use L298N Motor Driver. (17.06.2019)
Sursa: <https://www.teachmemicro.com/use-l298n-motor-driver/>
- [15] Node MCU V3 - LoLin (Wi-Fi). (17.06.2019) Sursa: <https://ardushop.ro/ro/home/108-node-mcu.html>
- [16] What is Firebase?. (17.06.2019) Sursa: <https://howtofirebase.com/what-is-firebase-fcb8614ba442>

[17] FirebaseArduino Class Documentation. (18.06.2019)

Sursa: <https://firebase-arduino.readthedocs.io/en/latest/>

[18] Add Firebase to your Android project. (20.06.2019)

Sursa: <https://firebase.google.com/docs/Android/setup>

[19] Which solar charge controller: PWM or MPPT?. (21.06.2019)

Sursa: <https://www.victronenergy.com/blog/2014/07/21/which-solar-charge-controller-pwm-or-mppt/>

[20] Morningstar Corporation Why PWM?, 2014; 1:1

[21] Câmpul magnetic al Pământului. (22.06.2019) Sursa: <https://www.scientia.ro/univers/40-terra/215-campul-magnetic-al-pamantului.html>