Departamentul Automatică şi Informatică Industrială
Facultatea Automatică şi Calculatoare
Universitatea Naţională de Ştiinţă şi Tehnologie
POLITEHNICA Bucureşti

LUCRARE DE DIPLOMĂ

Dezvoltarea unui sistem de conducere pentru un robot industrial cu 5 grade de libertate

Coordonator
Conf.dr.ing Silviu Răileanu

Absolvent
Albert-Gabriel Ungureanu

2025

Cuprins:

1. Introducere

În era producției moderne, automatizarea proceselor industriale joacă un rol foarte important. Roboții industriali sunt utilizați în liniile de asamblare, manipulare sau chiar inspecție, deoarece elimină eroarea umană și cresc atât eficiența cât și randamentul. Astfel, fabricile se pot baza pe o productivitate constantă.

Complexitatea unui robot industrial este strâns corelată cu numărul de grade de libertate pe care acesta le are. Desigur, cu cât acest număr este mai mare, cu atât robotul va fi mai flexibil și totodată capabil să înlocuiască cu ușurință activități pe care le-ar fi făcut un om.

Această lucrare propune prezentarea unui sistem de conducere pentru robotul industrial *Scorbot-ER VII* [1]. Acesta este un robot de tip braț articulat, care dispune de 5 grade de libertate. Fiecare articulație este acționată de un motor electric de curent continuu, în timp ce deplasarea este măsurată cu un encoder relativ dedicat.

Principalul obiectiv al proiectului este reprezentat de realizarea unei placi de control al mișcării și dezvoltarea unei soluții pentru conducerea unui număr de 5 axe.
Obiectivele intermediare sunt următoarele:

- Mișcarea individuală a fiecărei articulații;
- Mișcarea punctului condus în spațiul articulațiilor ($\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$);
- Mișcarea punctului condus în spațiul cartezian (x, y, z).

Placa de control va fi alcătuită din 7 module ce au la bază microcontroller-ul Atmel Atmega328P, programat în limbajul C. Unul dintre module va avea rolul de a le orchestra pe celelalte, care vor rula constant regulatoare proporționale cu scopul de a urmări referințele primite. Utilizatorii vor avea la dispoziție și o interfață grafică realizată folosind biblioteci ale limbajului Python, prin intermediul căreia brațul robotic va putea fi condus cu ușurință.

2. Prezentarea domeniului din care face parte lucrarea

Robotica industrială reprezintă o latură a ingineriei, care se concentrează pe proiectarea și integrarea roboților în procesele de producție. Acest lucru a devenit esențial din mai multe motive, precum reducerea erorilor umane și creșterea productivității. Totodată, aceștia pot fi folosiți în medii care ar putea pune persoanele umane în pericol. În plus, roboții se descurcă mult mai bine la sarcini care necesită un nivel de precizie ridicat.

Conform *IFR* [5], numărul de roboți utilizati crește anual, fapt ce reflectă interesul pentru automatizarea proceselor și trecerea la sisteme inteligente de fabricație.

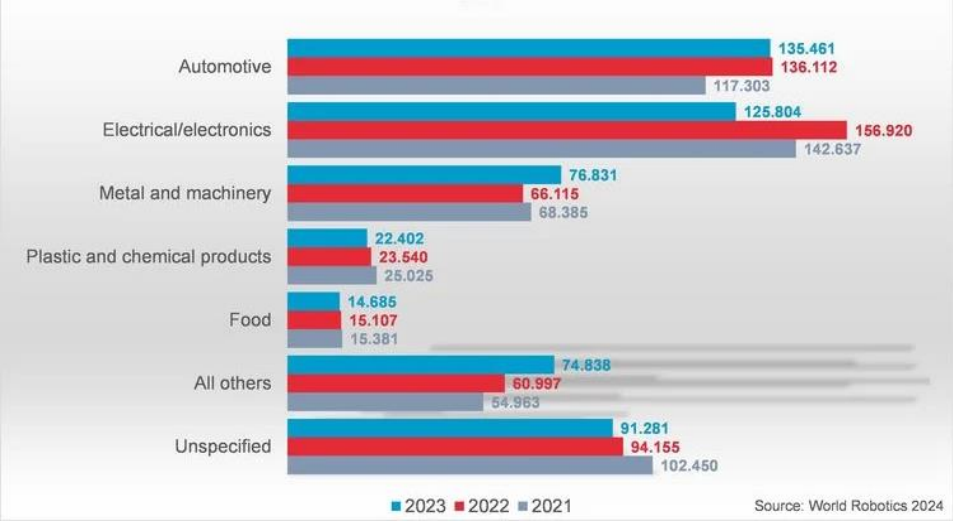

Fig. 2.1. Evoluția numărului de roboți industriali utilizați anual

După cum se poate observa și în *Fig. 2.1*, cele mai automatizate domenii sunt industria auto și cea care acoperă producția sistemelor electrice și electronice.

De asemenea, o altă analiză interesantă este reprezentată de tendința de creștere a numărului de roboți industriali folosiți. Aceasta este prezentată în *Fig. 2.2*.

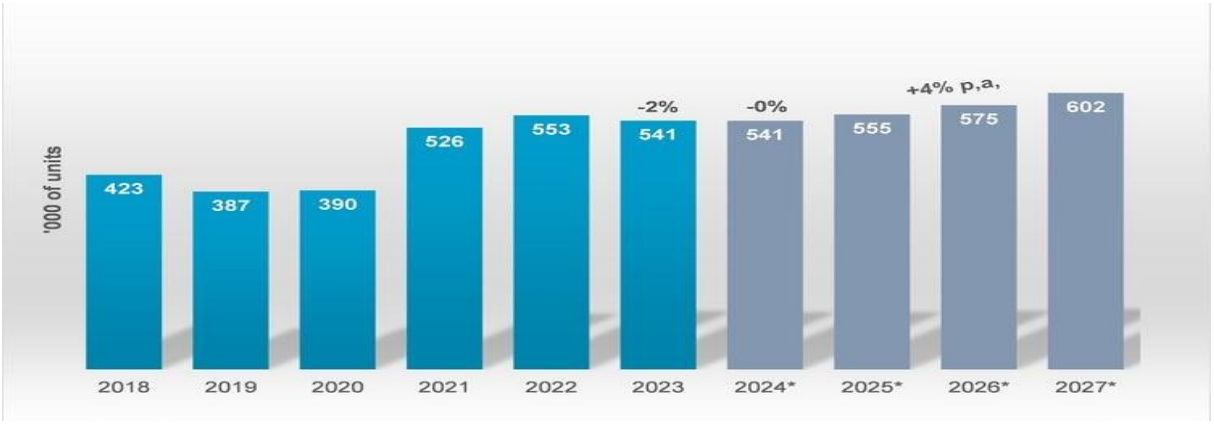

Fig. 2.2. Numărul de instalări anuale ale roboților industriali

În funcție de arhitectura lor, roboții industriali pot fi clasificați într-una dintre următoarele categorii:

- Cartezieni;
- SCARA;
- Delta;
- Cu braț articulat.

Dintre categoriile de mai sus, cei mai întâlniți roboți industriali sunt cei cu braț articulat, pentru că sunt mult mai versatili. Dispunând de mai multe grade de libertate, pot executa mișcări complexe, asemenătoare cu ale unui om.
Robotul folosit în această lucrare, Scorbot-ER VII [1], face parte tot din acestă categorie și are 5 grade de libertate.

Este important de menționat că roboții industriali sunt formați dintr-o structură mecanică și o structură de control și conducere. Cea din urmă este responsabilă în totalitate de deplasarea articulațiilor în funcție de dorința utilizatorului. Există două moduri principale de mișcare:

- În spațiul articulațiilor
- În spațiul cartezian

Mișcarea în spațiul articulațiilor constă în comanda independentă a fiecărei articulații.
Mișcarea în spațiul cartezian necesită rezolvarea problemei de cinematică inversă pentru a putea deduce în ce poziție trebuie să se regăsească fiecare articulație astfel încât efectorul terminal să parcurgă o traiectorie liniară.

Lucrarea mea se încadrează în acest domeniu și are ca obiectiv final dezvoltarea unui sistem de conducere pentru robotul industrial Scorbot-ER VII [1], pentru a face posibil atât controlul în spațiul articulațiilor cât și pe cel în spațiul cartezian.

3. Descrierea problemei si prezentarea solutiei propuse

În procesul actual de automatizare și digitalizare a proceselor industriale, utilizarea roboților este esențială. Astfel, este important ca viitorii ingineri să fie familiarizați încă din anii facultății cu aceștia. Problema identificată în cadrul acestei lucrări pornește de la nevoia de a controla eficient robotul industrial Scorbot-ER VII [1], astfel încât acesta să execute mișcări precise. Mai exact, lucrarea propune dezvoltarea unui sistem de conducere pentru acest robot, folosind numeroase cunoștințe obținute pe parcursul anilor, la materii precum: *Maşini electrice şi acţionări*, *Electronică digitală*, *Modelare și Simulare*, *Ingineria reglării automate* sau *Sisteme de conducere a roboților*.

Având în vedere vechimea acestui robot, sistemul de conducere original a devenit inutilizabil, miscându-se greu și dând numeroase erori de funcționare.
Soluția mea propune un sistem de conducere personalizat, care este alcătuit din următoarele componente:

- Interfața cu utilizatorul care ruleaza pe orice sistem de operare;
- Un modul care comunică cu calculatorul și trimite comanda specifică fiecarui motor;
- Câte un modul care primește o comandă și actionează motorul pentru a o urmări, pentru fiecare articulație.

Mai exact, modulul de tip *master* comunică prin intermediul protocolului de comunicație *UART* [4] cu calculatorul, care calculează folosind cinematica inversă unghiurile la care trebuie să se găsească fiecare articulație și trimite aceste date prin intermediul protocolului de comunicație *I2C* [3]. Cele 5 module de tip *slave* rulează în mod continuu un controler proporțional cu scopul de a urmări referințele primite de la *master*.

Pentru a fi cât mai simplu de folosit, această aplicație dispune și de o interfață cu utilizatorul realizată în limbajul *Python*, prin intermediul căreia se pot efectua următoarele:

- Mișcarea independentă a fiecărei articulații;
- Salvarea punctului curent în spațiul articulațiilor;
- Mișcarea liniară între diferite puncte configurate anterior;
- Poziția curentă în spațiul articulațiilor.
- Afișarea curentului consumat de fiecare articulație.

Prin această lucrare am urmărit demonstrarea capacității de a proiecta, dezvolta și testa un sistem de conducere aplicabil în robotica industrială, având ca scop înțelegerea profundă a principiilor de control.

4. Prezentarea echipamentului

Scorbot-ER VII [1] este un robot industrial de tip braț articulat, având 5 grade de libertate, după cum se poate observa și în *Fig. 4.1*:

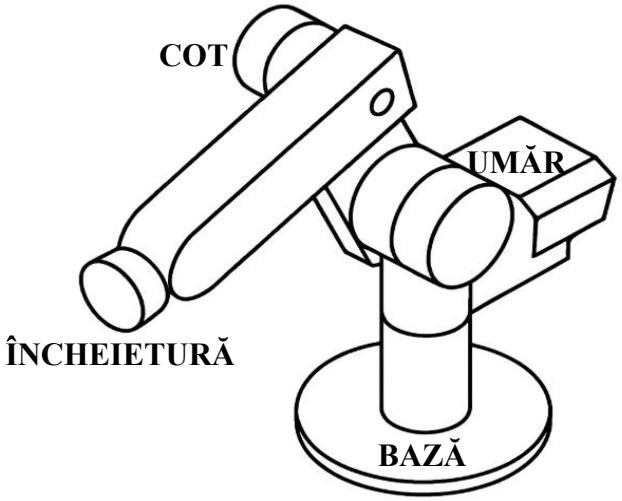

Fig. 4.1. Structură Scorbot-ER VII

Brațul robotic are o sarcină maximă de 2 kg, acesta cântărind 30 kg.

O prezentare tehnică a robotului poate fi observată urmărind *Tabel 5.1*:

Articulație	Cursă maximă	Mișcare efectuată
Bază	310°	Rotație bază
Umăr	170°	Ridicare / Coborâre braț
Cot	225°	Ridicare / Coborâre antebraț
Înclinație încheietură	180°	Ridicare / Coborâre efector terminal
Rotație încheietură	360°	Rotație efector terminal

Tabel 4.1. Detalii articulații Scorbot-ER VII

O altă informație importantă este cea legată de domeniul de funcționare al brațului robotic, prezentat în *Fig. 4.2*:

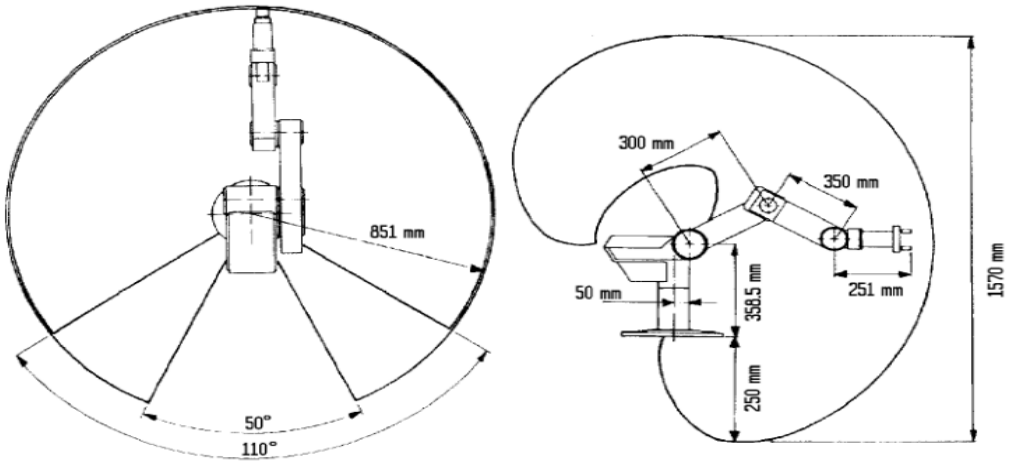

Fig. 4.2. Domeniu de funcționare Scorbot-ER VII

Fiecare articulație este alcătuită din următoarele componente:

- motor electric de curent continuu pentru a se putea deplasa;
- encoder incremental optic dedicat pentru a putea cuantifica deplasările.

Motoarele funcționează la o tensiune de 12 VDC și sunt controlate prin intermediul unei punți H de putere.

Fig. 4.3. Motor DC

Modulul *BTS7960*, prezentat în *Fig. 4.3.*, poate conduce un curent de până la 43A și are atât protecție termală cât și de supracurent. Datorită puterii ridicate, acesta asigură posibilitatea de a accelera și frâna eficient motorul.

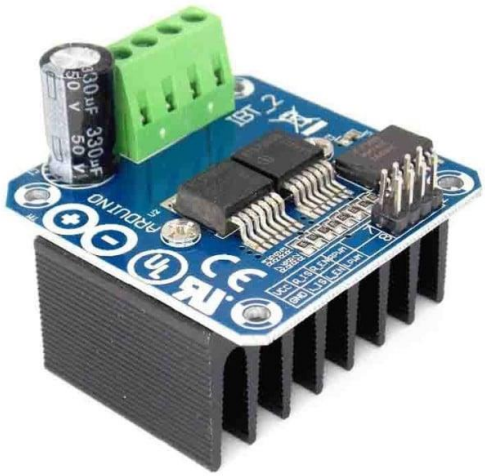

Fig. 4.4. Punte H de putere

Pentru a putea realiza un control în buclă închisă, deplasarea fiecărei articulații este măsurată cu ajutorul encoderului incremental atașat. Acesta este alcătuit din două leduri, două fotodiode și un disc perforat. Principiul de functionare poate fi observat în *Fig. 4.5.*

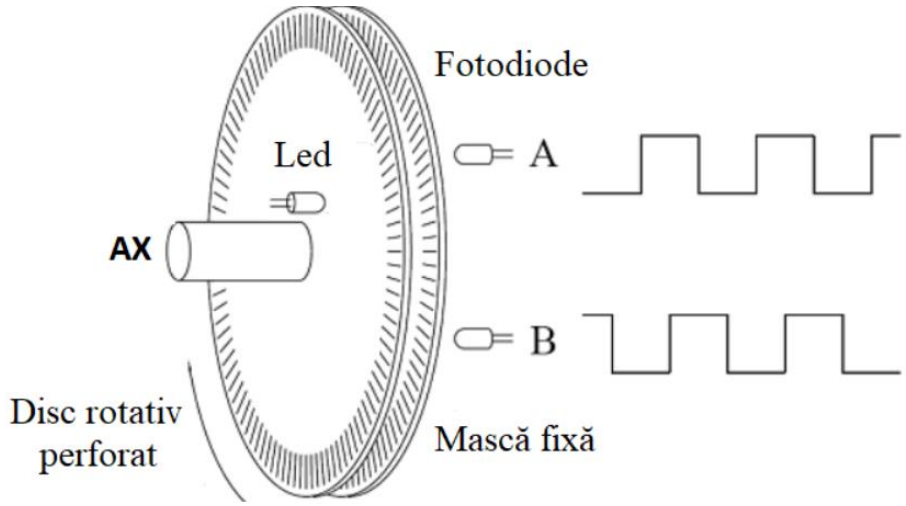

Fig. 4.5. Principiu de funcționare encoder optic

Conform [7], în funcție de faza relativă a impulsurilor **A** și **B** determină sensul de rotație al motorului. Un puls ascendent **A** urmat de un puls ascendent **B** va reprezenta sensul trigonometric în timp ce un puls descendent **A** urmat de un puls ascendent **B** va reprezenta sensul acelor de ceasornic. Trenurile de impulsuri defazate **A** și **B** sunt cunoscute ca semnale în cuadratură.

Pentru a conecta brațul robotic, fiecare articulație a acestuia are câte un conector de tip DB9. Funcționalitatea fiecărui pin poate fi observată în *Fig 4.6.*

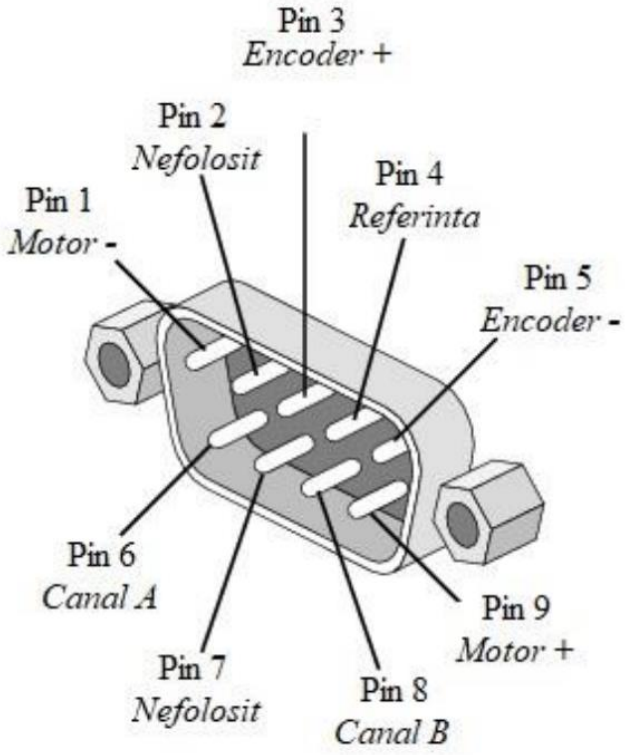

Fig. 4.6. Conector DB9 pentru interfațare

Conform manualului de utilizare [1], cablajul intern al brațului robotic este realizat ca în *Fig. 4.7.*

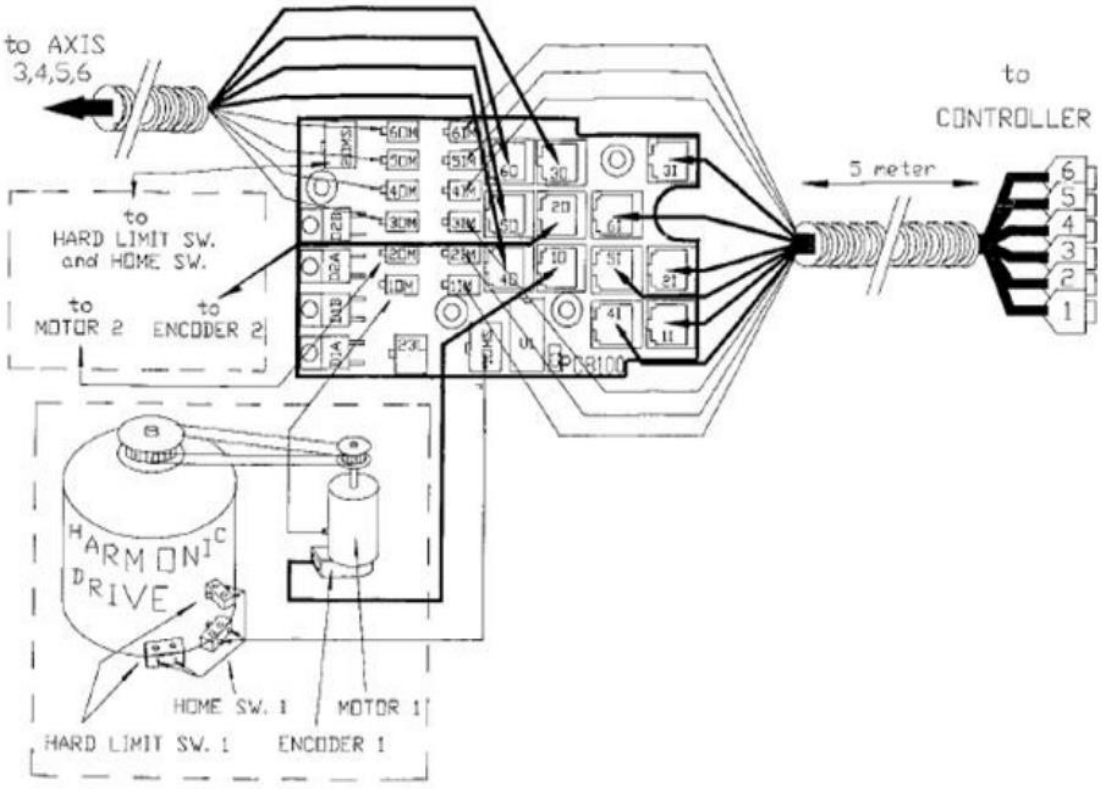

Fig. 4.7 Cablaj intern Scorbot-ER VII

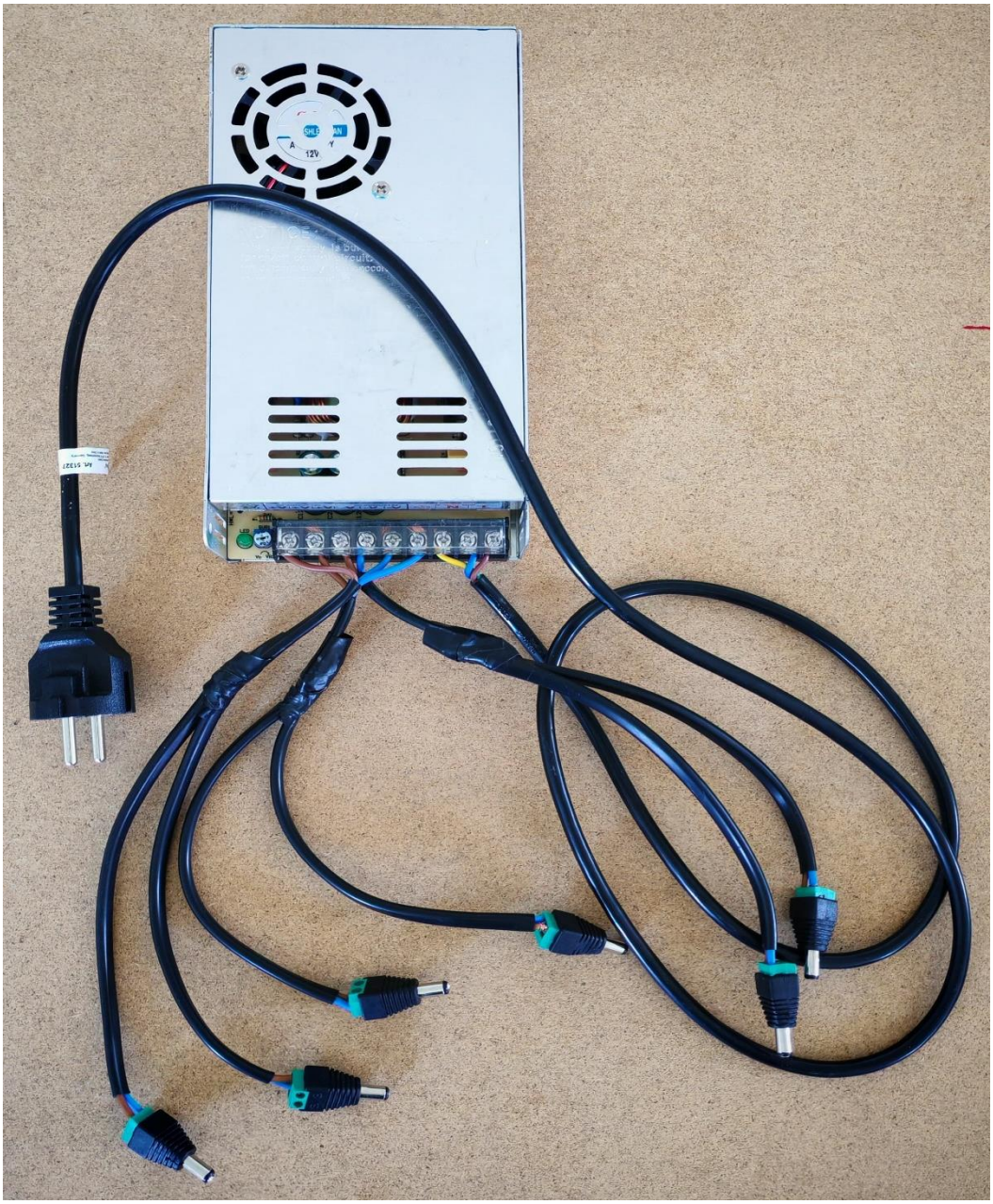

Fig. 4.8. Sursă de alimentare

Pentru alimentarea sistemului, am achizitionat o sursa de alimentare care suporta un curent cosumat de până la 30A, suficient chiar și pentru cazul în care toate cele 5 motoare rulează în paralel la consum maxim.

Din cauza faptului că sursa dispune doar de 3 ieșiri, am legat în paralel 2 câte 2 motoare. Pentru a balansa consumul, am împerecheat câte o articulație care consumă mai mult cu una care consumă mai puțin, astfel:

- baza cu înclinația încheieturii;
- cotul cu rotația încheieturii;
- umărul cu unealta.

De asemenea, pentru a facilita conexiunea cu placa de control, am adaugat conectori de tip mamă (pentru placa de control) și tată (pentru sursa de alimentare).

5. Arhitectura de conducere a brațului robotic

Pentru a manipula brațul robotic, am proiectat un sistem de conducere, compus dintr-un controller de mișcare și o aplicație utilizator prin intermediul căreia utilizatorii să controleze cât mai facil robotul. Diagrama bloc a acestui sistem poate fi observată în *Fig. 5.1.*

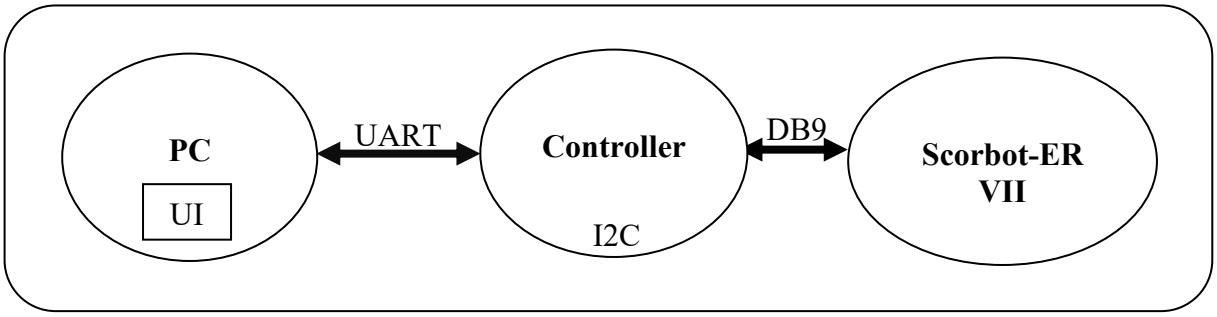

Fig. 5.1. Diagrama bloc sistem de conducere

5.1. Prezentarea aplicației

Prin intermediul aplicației se permite controlul robotului atât în spațiul articulațiilor cât și în spațiul cartezian. Interfața poate fi observată în *Fig. 5.2.*

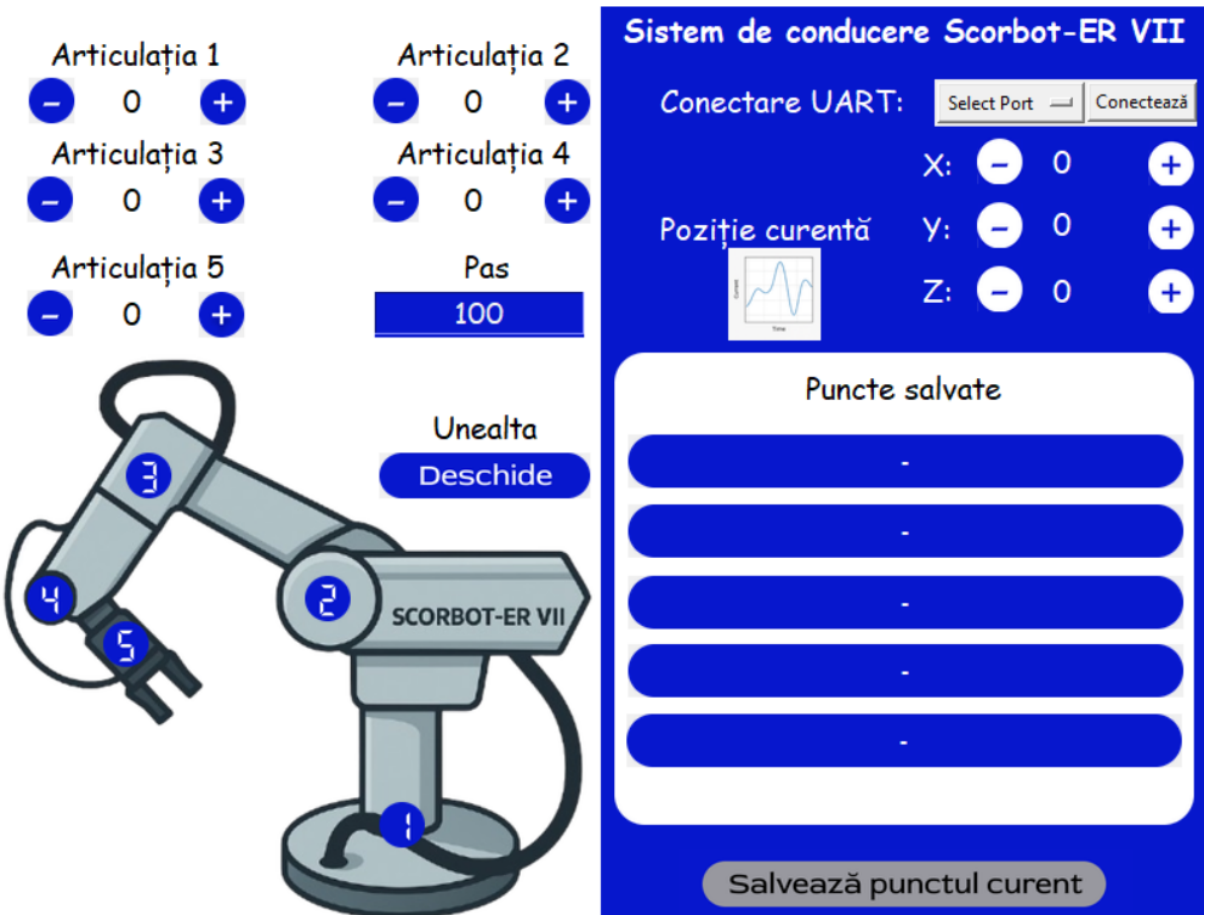

Fig. 5.2. Interfața utilizator

Aplicația permite următoarele acțiuni:

- Conectare prin intermediul UART la placa Arduino Nano;
- Deplasarea fiecărei articulații;
- Controlul uneltei (închidere / deschidere);
- Salvarea configurației curente în spațiul articulațiilor;
- Revenirea la o configurație salvată anterior;
- Afișarea pozitiei efectorului terminal în spațiul cartezian;
- Deplasarea liniară a punctului condus în spațiul catezian;
- Afișarea curentului măsurat de fiecare articulație pe parcursul deplasării.

Pentru a putea controla brațul robotic, este obligatoriu ca inițial să se stabilească o conexiune cu placa Arduino Nano prin intermediul protocolului de comunicație UART. Conexiunea se stabilește selectând din lista de port-uri disponibile pe cel corespunzător și ulterior apăsând butonul *Conectează*.

Pentru a stabili diverse poziții ale punctului condus pe care utilizatorul dorește ca brațul robotic să le atingă, se permite manipularea individuală a articulațiilor. Folosind butoanele aferente **+** și **-**, valoarea curentă se va modifica în funcție de pasul selectat, care poate fi ajustat în funție de granularitatea dorită a deplasării. Valorile afișate pentru fiecare articulație sunt exprimate în unități motor pentru a permite o precizie cât mai ridicată. De asemenea, fiecare articulație a fost reprezentată pe schița robotului și are un interval de valori pe care le poate prelua, stabilit experimental și prezentat în *Tabel 5.1*, pentru a preveni posibile accidente.

Articulație	Valoare minimă	Valoare maximă
1	-500000	500000
2	-400000	800000
3	-200000	400000
4	-150000	150000
5	-100000	100000

Tabel 5.1. Interval valori articulații exprimat în unități motor

Pentru a putea efectua operații de tipul *Pick & Place*, unealta se poate închide / deschide folosind butonul aferent. Când unealta este închisă va fi afișat butonul *Deschide*, iar dacă unealta este deschisă va fi afișat butonul *Închide*.

După ce utilizatorul a ajuns la una dintre configurațiile dorite, acesta o poate salva, folosind butonul *Salvează punctul curent*. Valorile articulațiilor vor fi convertite în grade unghiulare și vor fi afișate pe una dintre cele 5 poziții disponibile pentru salvare. În cazul în care toate pozițiile sunt deja ocupate, sistemul le va suprascrie circular, începând cu prima.
Ratele de conversie din unități motor în grade unghiulare pentru fiecare articulație sunt prezentate în *Tabel 5.2.*

Articulație	$\frac{unități\ motor}{grad\ unghiular}$
1	256
2	260
3	260
4	166
5	256

Tabel 5.2. Rate conversie unități motor – grade unghiulare

Valorile au fost obținute experimental, deplasând fiecare articulație cu câte 90°, măsurând câte unități motor au fost necesare și făcând raportul dintre cele două valori.

Revenirea la o configurație deja salvată se face prin selectarea acesteia din listă. Deplasarea se poate face atât în spațiul articulațiilor cât și în cel cartezian, în funcție de preferința utilizatorului.

O altă funcționalitate interesantă este reprezentată de afișarea în timp real a poziției curente a punctului condus, în spațiul cartezian, care a fost realizată folosind modelul cinematic al robotului. În acest sens, cunoscându-se valorile fiecărei articulații, folosind cinematica directă și caracteristicile brațului robotic, se poate calcula cu precizie poziția efectorului terminal.
Astfel, pe baza următoarelor valori, preluate din *Fig 4.2*:

$$a_1 = 50,$$
$$a_2 = 300,$$
$$a_3 = 350,$$
$$d_1 = 358.5,$$
$$d_{5\_tool} = 251,$$
$$\alpha_4 = \theta_4 - \theta_2 + \theta_3$$

putem calcula inițial coordonatele în plan vertical (ignorând unealta), folosind formulele următoare:

$$PR = a_2 * \cos(\theta_2) + a_3 * \cos(\theta_2 - \theta_3) \quad (5.1)$$
$$PZ = a_2 * \sin(\theta_2) + a_3 * \sin(\theta_2 - \theta_3) \quad (5.2)$$

la care adăugăm efectul uneltei și obținem:

$$r = a_1 + PR + d_{5\_tool} * \cos(\alpha_4) \quad (5.3)$$
$$\mathbf{Z = d_1 + PZ + d_{5\_tool} * \sin(\alpha_4)\ (5.4)}$$

Pentru a afla poziția pe axele X și Y, va trebui să convertim din coordonate polare în coordonate carteziene, folosind următoarele formule:

$$\mathbf{X = r * \sin(\theta_1)\ (5.5)}$$
$$\mathbf{Y = r * \cos(\theta_1)\ (5.6)}$$

Pe lângă posibilitatea de a manipula individual fiecare articulație, utilizatorul poate deplasa punctul condus în spațiul articulațiilor, folosind controalele din zona *Poziție curentă*. Aplicația va trimite punctul curent și noul punct configurat către un script matlab care rezolvă problema cinematicii inverse pentru Scorbot-ER VII. Scripul va returna o serie de valori prin care articulațiile vor trebui să treacă la anumite momente de timp.

După cum sugerează și numele, rolul cinematicii inverse este de a calcula valorile articulațiilor cunoscându-se poziția efectorului terminal în spațiul cartezian (X, Y, Z) bine stabilit încă de la început.

Conform [2] brațul robotic poate fi reprezentat în sistemul de referință O_{xyz} după cum se poate observa în *Fig. 5.3.*

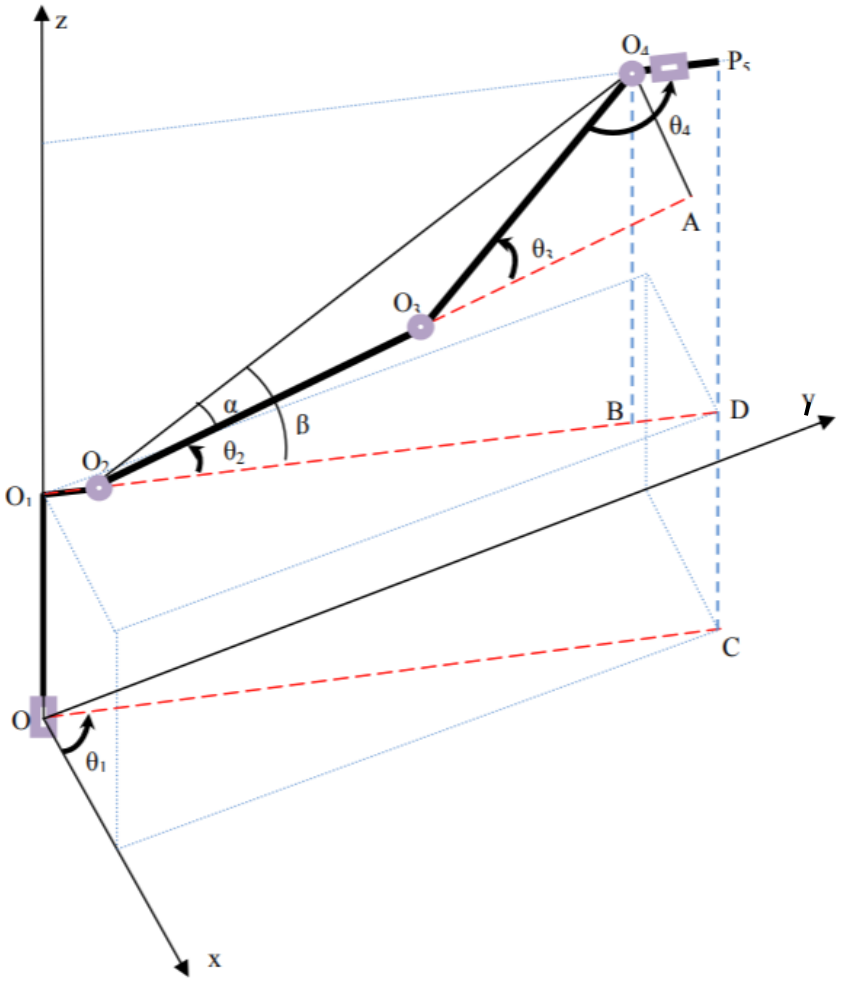

Fig. 5.3. Reprezentare Scorbot-ER VII în sistemul de referință O_{xyz}.

De asemenea, modelul cinematic al acestuia este prezentat în *Fig. 5.4.*

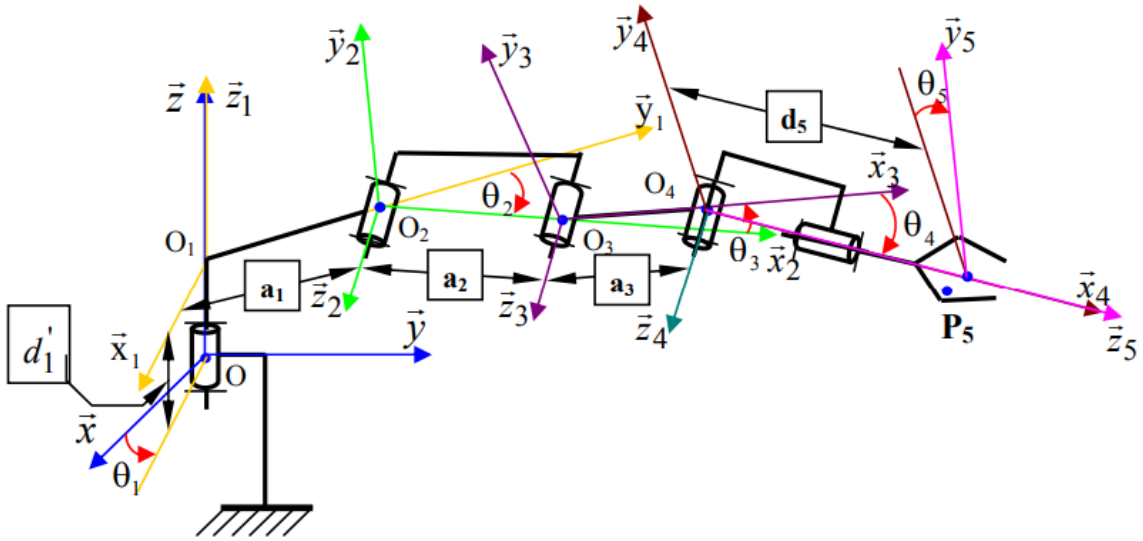

Fig. 5.4. Model cinematic Scorbot-ER VII

Folosind informațiile anterioare și parametrii Denavit-Hartemberg din *Tabel 5.3.* putem rezolva problema de cinematică inversă.

Tip articulație	a_i (mm)	α_i	d_i (mm)	θ_i
1- rotație	50	$-\frac{\Pi}{2}$	358.5	θ_1
2- rotație	300	0	0	θ_2
3- rotație	350	0	0	θ_3
4- rotație	0	$\frac{\Pi}{2}$	0	θ_4
5- rotație	0	0	251	θ_5

Tabel 5.3. Parametrii Denavit-Hartemberg Scorbot-ER VII

Formulele pe baza cărora sunt calculate valorile articulațiilor sunt următoarele:

$$\theta_1 = \arctan(x, y) \quad (5.7)$$

$$PZ = Z - d_1 - d_5 * \sin(\alpha_4) \quad (5.8)$$

$$PR = \sqrt{x^2 + y^2} - a_1 - d_5 * \cos(\alpha_4) \quad (5.9)$$

$$\theta_2 = \arctan(PZ, PR) + \arccos\left(\frac{a2^2 + PR^2 + PZ^2 - a3^2}{2 * a2 * \sqrt{PR^2 + PZ^2}}\right) \quad (5.10)$$

$$\theta_3 = \pi - \arccos\left(\frac{a2^2 + a3^2 - PR^2 - PZ^2}{2 * a2 * a3}\right) \quad (5.11)$$

$$\theta_4 = abs(\alpha_4) + \theta_2 - \theta_3 \quad (5.12)$$

$$\theta_5 = \alpha_5 \quad (5.13)$$

Pentru a realiza o mișcare liniară între două puncte din spațiul cartezian, este necesar să parcurgem următorul set de pași:

1. Stabilirea punctelor și verificarea apartenenței acestora la spațiul de lucru al brațului robotic;
2. Stabilirea unei viteze de croazieră între cele două puncte;
3. Calculul distanței euclidiene dintre cele două puncte;
4. Calculul timpului total necesar pentru deplasare cu o viteză constantă;
5. Calculul numărului de puncte intermediare în funcție de precizia dorită;
6. Definirea cuantelor de timp;
7. Generarea unui profil de viteză pentru traiectorie;
8. Interpolarea traiectoriei între cele două puncte;
9. Calculul cinematicii inverse pentru fiecare punct obținut;
10. Transmiterea informației către placa de control al mișcării.

Această secvență de pași a fost realizată cu ajutorul unui script realizat în limbajul de programare Matlab și poate fi observată în *Fig. 5.5.*

```
% Transfomarea punctelor într-un spațiu 3D cu o rotație de 180 de grade în
% jurul axei x
Tf_P1 = trvec2tform([p1(1) p1(2) p1(3)])*axang2tform([1 0 0 pi]);
Tf_P2 = trvec2tform([p2(1) p2(2) p2(3)])*axang2tform([1 0 0 pi]);
tform2eul(Tf_P1, 'XYZ');
tform2eul(Tf_P2, 'XYZ');

% Calculul timpului total pe baza vitezei maxime și a distanței euclidiene
% dintre cele două puncte între care se dorește deplasarea liniară
viteza=limitare_viteza;
distanta=sqrt((p1(1)-p2(1))^2+(p1(2)-p2(2))^2+(p1(3)-p2(3))^2);
t_total=distanta/viteza;

% Stabilirea numărului de puncte intermediare
N=ceil(distanta);

% Generarea unui vector de timp
tTimes = linspace(0,t_total,N);
tInterval = [0 t_total];

% Generarea traiectoriei
[s,sd,sdd] = trapveltraj([0 1],numel(tTimes));

% Interpolarea traiectoriei
[T,dT,ddT] = transformtraj(Tf_P1,Tf_P2,tInterval,tTimes,'TimeScaling',[s;sd;sdd]);
```

Fig. 5.5. Script Matlab pentru generarea unei traiectorii liniare între 2 puncte

De asemenea, după generarea punctelor, script-ul matlab returnează și un grafic în care se pot observa punctele pe care efectorul terminal al brațului robotic le va atinge. Acesta poate fi observat în *Fig. 5.6.*

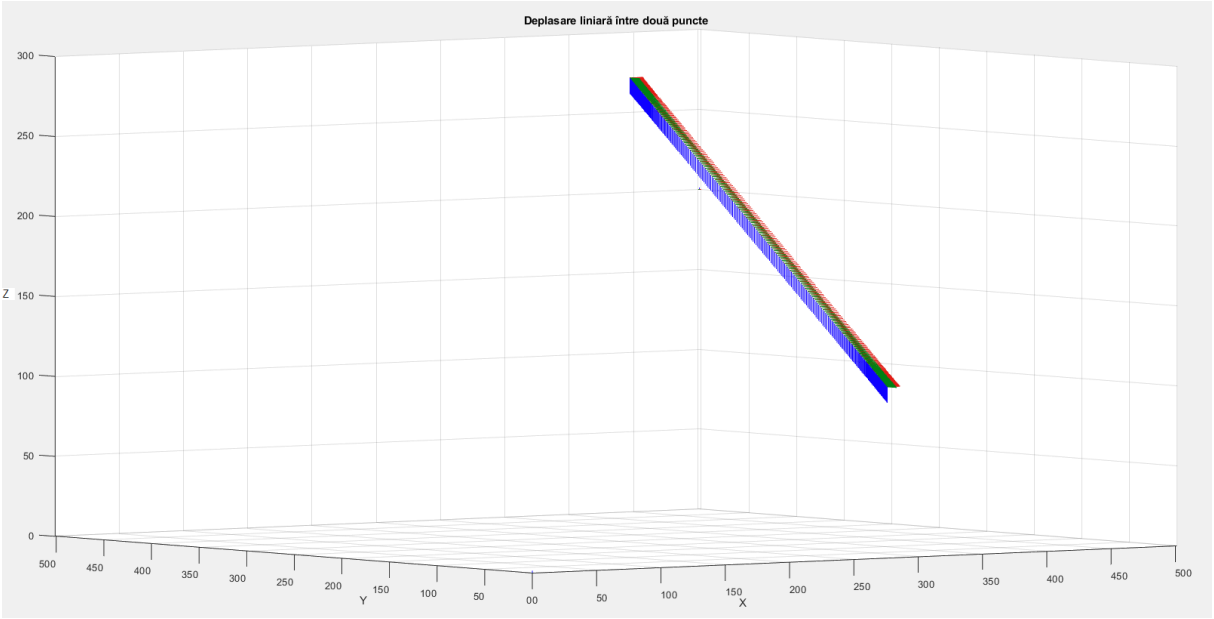

Fig. 5.6. Traiectorie liniară între două puncte în spațiul cartezian O_{xyz}

5.2. Structura plăcii de control al mișcării

Elementul de bază al proiectului este reprezentat de placa de control, prin intermediul căreia aplicația acționează moatoarele brațului robotic în funcție de comenzile utilizatorului.

Un controller de mișcare este un dispozitiv alcătuit atât din hardware cât și software, care coordonează comenzile de poziționare și viteză pentru controlul precis al mișcării unui braț robotic. Acesta interpretează comenzile provenite de la nivelul superior, precum aplicația utilizator prezentată anterior și le traduce în semnale electrice către motoare, asigurând o mișcare precisă și sincronizată a tuturor articulațiilor.

Printre funcțiile principale ale unui astfel de sistem se numără următoarele:

- Controlul traiectoriei;
- Interpolare;
- Sincronizare multi-axă;
- Managementul vitezei și al accelerației.

Diagrama bloc a controllerului de mișcare proiectat de mine poate fi observată în *Fig 5.7.*

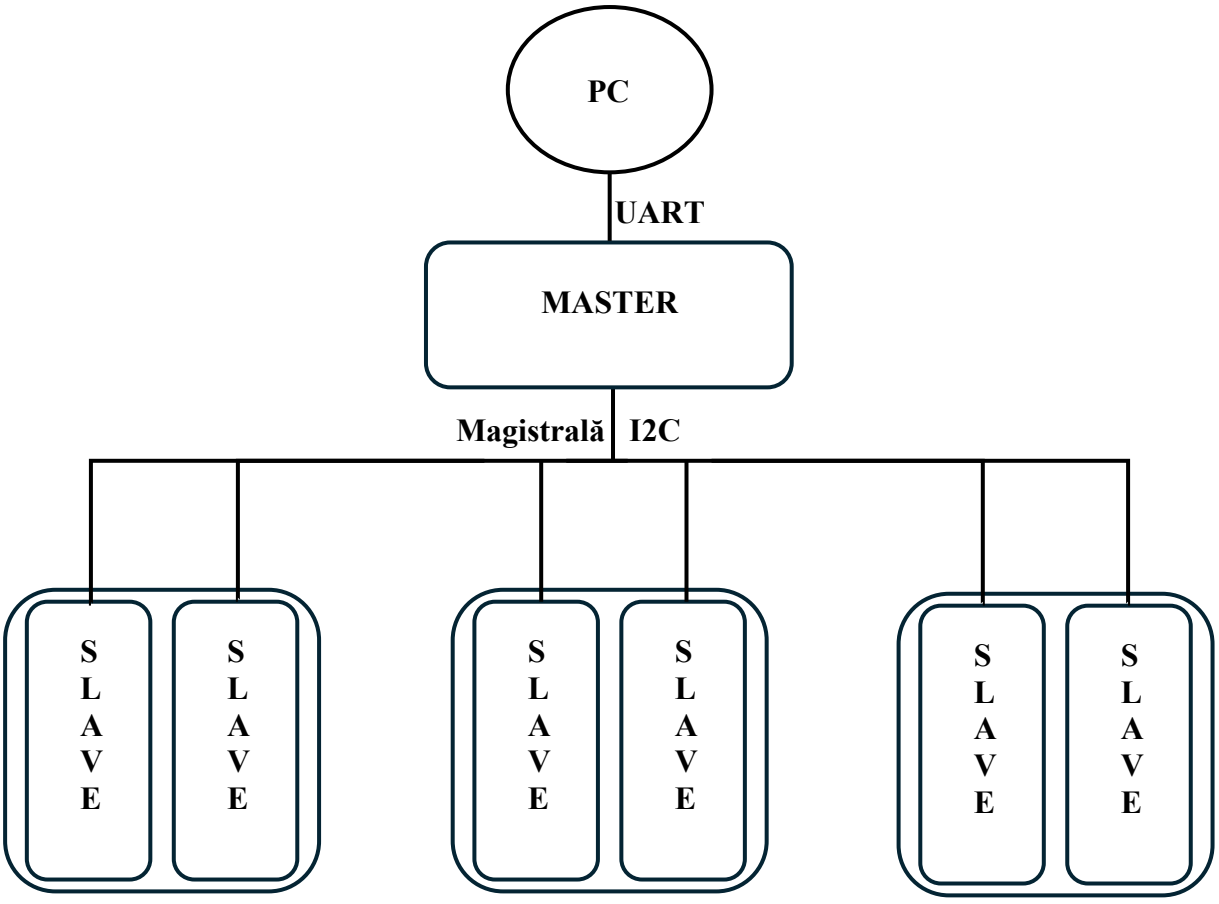

Fig. 5.7. Diagramă bloc a plăcii de control al miscării

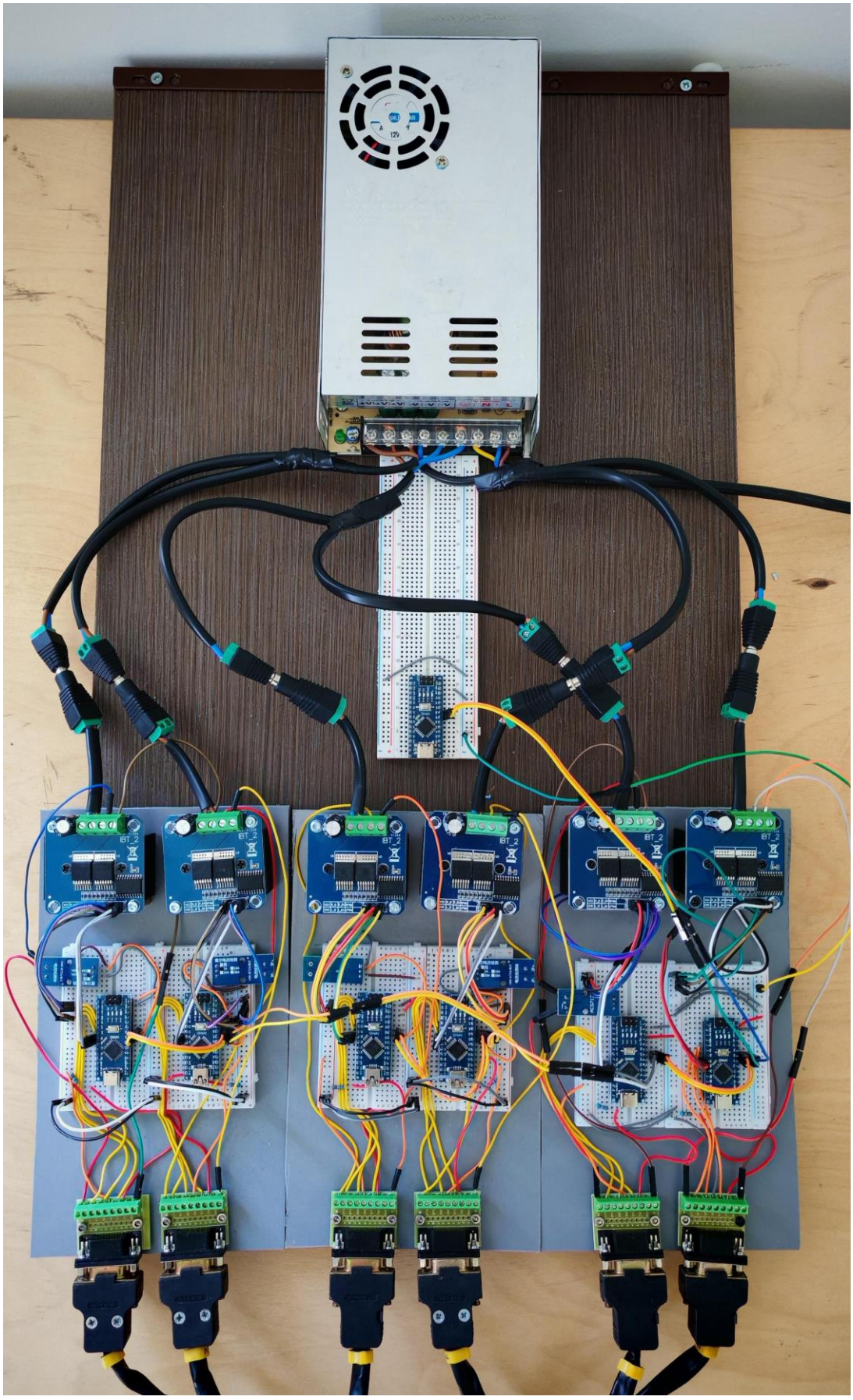

Fig. 5.8. Placa de control al mișcării

În componența acestuia intră următoarele tipuri de module:

- Modul Master;
- Modul Slave.

Modulul de tip *master* se ocupă de împărțirea pozițiilor primite de la aplicația utilizator către fiecare motor și organizarea datelor care reprezintă curentul consumat de fiecare dintre acestea. De asemenea, după finalizarea deplasării tuturor articulațiilor, trimite un semnal către aplicație cu scopul de a anunța ca este disponibil pentru primirea unei noi comenzi.

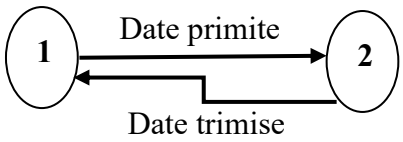

Fig 5.9. Diagrama de stări modul master

Conform *Fig. 5.9.*, acest modul se poate afla într-una din următoarele două stări:

- Starea 1: așteaptă date de la aplicații prin intermediul *UART*;
- Starea 2: trimite datele corespunzătoare fiecărui modul de tip slave prin intermediul *I2C*;

Am ales să folosesc protocolul de comunicație de tip I2C [3] pentru că a fost necesară sincronizarea a 5 dispozitive. De asemenea, cu ajutorul unei magistrale I2C, comunicația cu fiecare modul a fost ușor de implementat. Fiecare modul are setată o adresă unică și astfel va interpreta doar datele care sunt precedate de aceasta. În plus, această metodă a impus folosirea unui număr de doar 3 fire: SCL, SDA și GND.

- SDA reprezintă linia pe care sunt transmise datele;
- SCL reprezintă linia pe care master-ul generează semnalul de ceas pentru sincronizarea transmisiei;
- GND reprezintă linia prin intermediul căreia se asigură că toate modulele vor interpreta nivelele logice în același mod.

În *Fig 5.10.* se poate observa secvența unui mesaj trimis prin intermediul protocolului I2C.

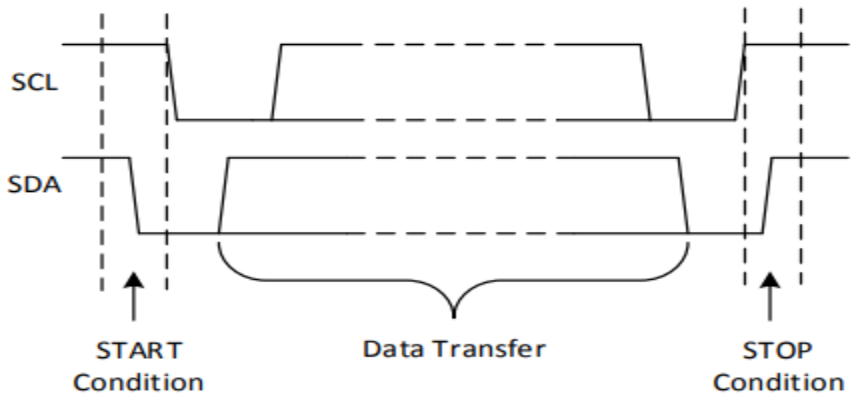

Fig. 5.10. Transfer date prin intermediul I2C

Comunicația va avea loc între un semnal de START și un semnal de STOP trimise de către master, interpretate după cum urmează:

- O tranziție high – low pe linia de date cât timp linia SCL este high va reprezenta START;
- O tranziție low – high pe linia de date cât timp linia SCL este high va reprezenta STOP.

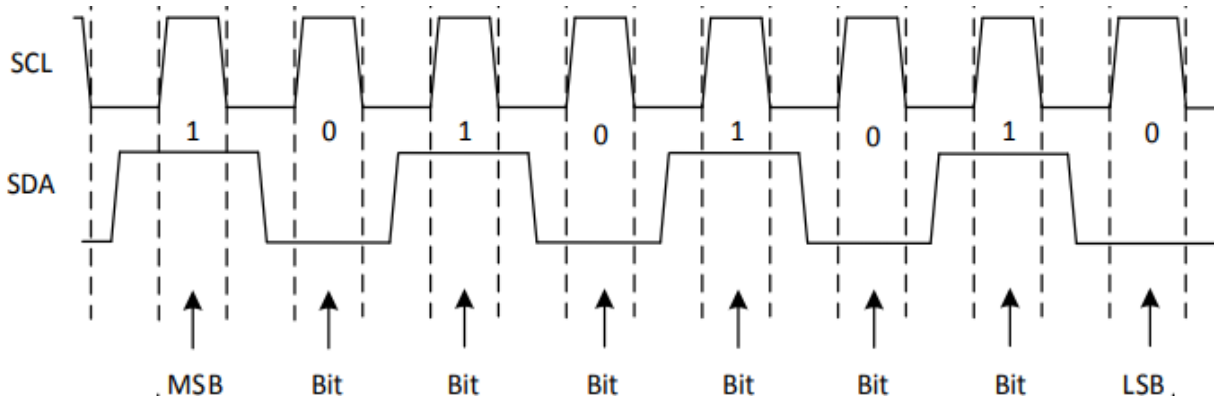

Fig. 5.11. Interpretare date transmite prin intermediul I2C

După cum se observă și în *Fig. 5.11.*, datele vor fi interpretate la fiecare tranzitie low - high a semnalului CLK. Cât timp semnalul de pe SCL este high, nivelul logic de pe linia SDA trebuie să rămână constant. Valoarea respectivă va fi adăugată în sirul de biți, care va fi convertit într-o valoare numerică. După fiecare 8 biți transmiși cu succes, slave-ul va trimite înapoi un semnal de tipul ACK, confirmând recepția datelor. În cazul în care se dorește transmiterea unor date ce necesită reprezentare pe mai mult de 8 biți, se vor trimite fragmentat în intervale de 8 biti, cu mare atenție la ordinea acestora.

În cazul comunicației cu aplicația, a fost ales protocolul UART [4] datorită simplității de interconectare și a eficienței sale, ce s-a dovedit a fi suficientă pentru cerințele acestui caz. De asemenea, nu a necesitat folosirea unui adaptor, ci doar a unui cablu USB.
Conform [4], UART (Universal Asynchronous Receiver – Transmitter) reprezintă un protocol de comunicație serială asincronă, prin intermediul căreia se pot transmite date între două dispozitive.

Acest protocol necesită folosirea a 3 fire:

- TX – RX;
- RX – TX;
- GND comun.

Din punctul de vedere al vitezei maxime, acesta suportă frecvențe de la 9600 până la 250000. Se pot utiliza si valori mai mari însă datele se pot corupe. 250000 reprezintă valoarea maximă pe care am reușit să o folosesc, păstrând în totalitate integritatea datelor transmise.

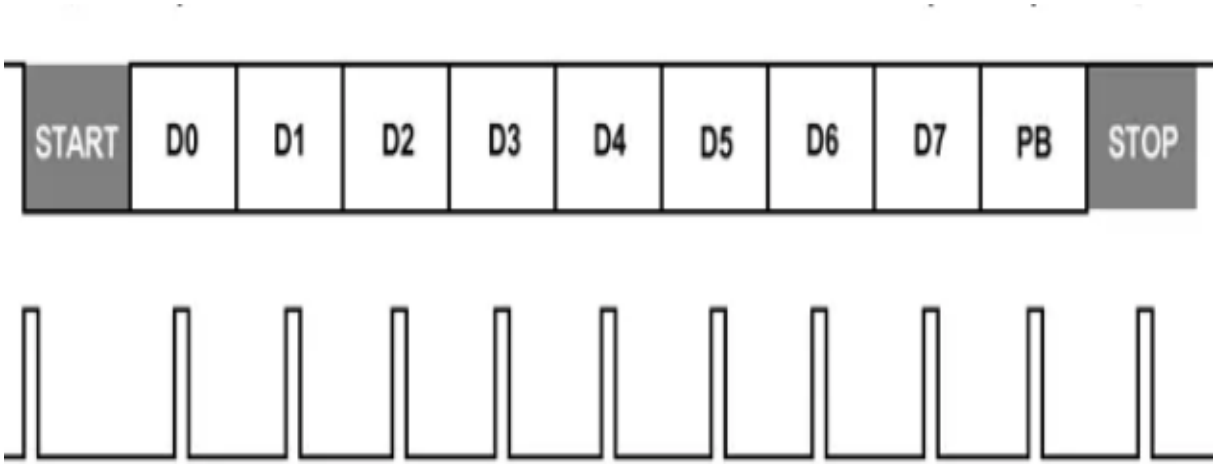

Fig. 5.12. Transmitere date prin intermediul UART

După cum se poate observa în *Fig. 5.12.,* avem la fel ca în cazul I2C un semnal de START și unul de STOP:

- START este reprezentat de coborârea nivelului logic pe linia de transmisie;
- STOP este reprezentat de ridicarea nivelului logic după trasmiterea celor 9 biți.

Între acestea, sunt transmiși biții care fac parte din datele propriu-zise, urmați de un bit de partitate ce deservește la verificarea datelor. Cel care primește va calcula propria valoare pe baza datelor obținute și o va compara cu cea primită. Dacă nu sunt egale, înseamnă că datele au fost corupte.

Caracterul asincron al acestui tip de comunicație este evidențiat de faptul că după primirea bit-ului START, fiecare dispozitiv își va genera un semnal de ceas propriu, pe bază căruia va interpreta datele. Pentru o comunicare corectă, este necesar ca ambele dispozitive să fie configurate cu aceeași rată de transfer.

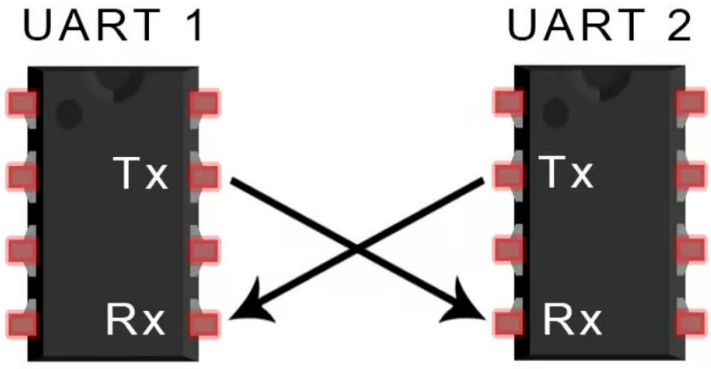

Fig. 5.13. Conexiune între 2 dispozitive pentru a comunica prin intermediul UART

În *Fig. 5.13.* se poate observa cum trebuie să fie conectate 2 dispozitive pentru a comunica prin intermediul UART. De asemenea, pentru a asigura o interpretare identică a nivelelor logice, este necesar și GND comun.

Fiecare modul de tip *slave* se ocupă cu reglarea poziției unei articulații. Folosind modulul *BTS7960*, care înglobează de fapt o punte de putere H și feedback-ul primit de la encoderul incremental optic dedicat se poate efectua o reglare în buclă închisă, a cărei diagramă bloc poate fi observată în *Fig. 5.14.*

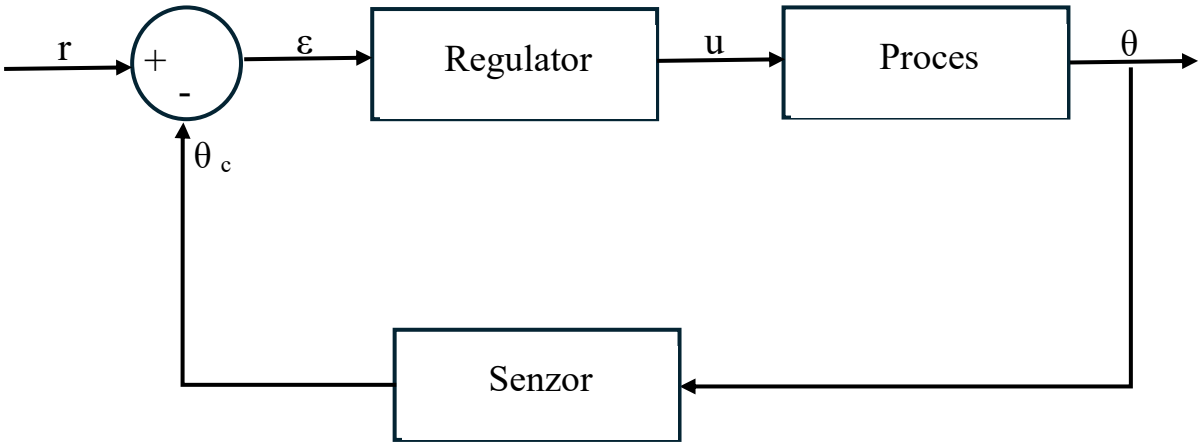

Fig. 5.14. Schemă de reglare SRA

Astfel, primind o referință de la master, exprimată în unități motor, modulul slave va rula un sistem de reglare automata pentru a o urmări cât mai eficient. Un prim pas este compararea referinței cu poziția curentă a articulației, care este citită prin intermediul encoderului incremental optic dedicat. Scăzând valoarea poziției curente din valoarea referinței se obține valoarea erorii pe care trebuie să o reglăm.

Reglarea se efectuează prin intermediul *Regulator*-ului, care primește ca intrare eroarea ε și trimite mai departe comanda *u* către proces. Regulatorul ales a fost unul de tip proporțional, datorită complexității scăzute a cazului de lucru. Acesta aplică mereu o comandă direct proporțională cu eroarea care trebuie să fie reglată, conform formulei *(5.14)*.

$$u = K_P * \varepsilon \quad (5.14)$$

Valoarea factorului de amplificare K_P a fost aleasă experimental astfel încât atât eroarea staționară cât și suprareglajul să fie nule, având un timp de reglare satisfăcător.

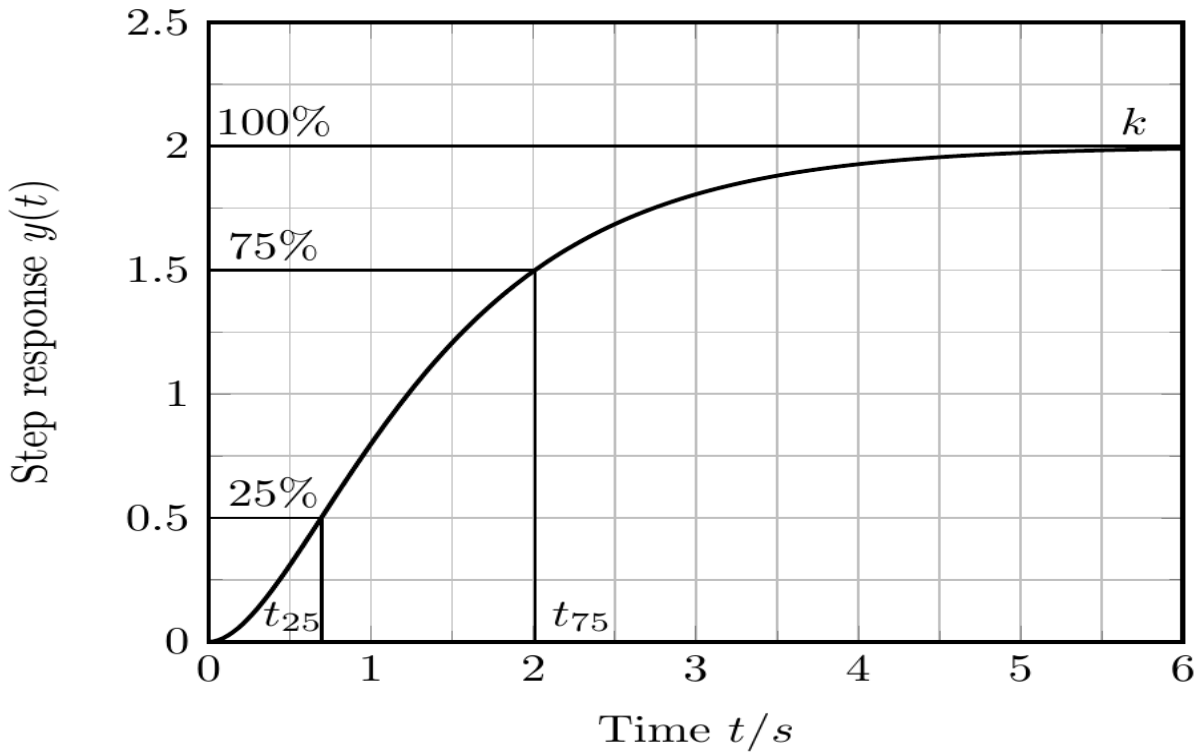

Fig. 5.15. Răspuns SRA la intrare de tip treaptă

Conform *Fig. 5.15.*, performanțele sistemului la o intrare de tip treaptă sunt următoarele:

- eroare staționară (ε_{st}): 0
- suprareglaj (σ): 0

Comanda calculată pe baza erorii (ε) și a factorului de amplificare K_P este aplicată motorului de curent continuu prin intermediul unei punti H. Aceasta este folosită pentru a permite controlul motorului în ambele sensuri.

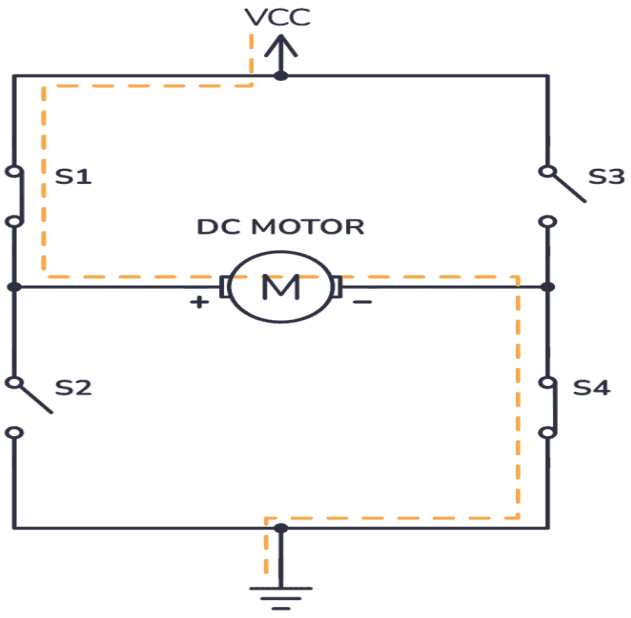

Fig. 5.16. Schemă electrică punte H

După cum se poate observa și în *Fig. 5.16.* o punte H este formată din 4 comutatoare care permit aplicarea la bornele motorului în două sensuri diferite. Astfel, se permit următoarele 3 operații:

- Rotirea motorului în ambele sensuri:
  - Sens orar:
    - Se închid S1 și S4;
    - Se deschid S2 si S3;
  - Sens trigonometric:
    - Se închid S2 și S3;
    - Se deschid S1 și S4;
- Oprirea motorului;
  - Toate comutatoarele se deschid;
- Frânarea motorului.
  - Toate comutatoarele se închid. (oprire rapidă)

Cele 4 comutatoare pot fi implementate folosind tranzistoare sau relee.
Modulul folosit de mine, *BTS7960*, are în structura sa o punte H dublă, fiind capabil să controleze motoare de curent continuu de putere mare. De asemenea, are protectii la supratensiune și supracurent.
Pentru a putea controla viteza motorului, a fost necesară aplicarea comenzilor folosind tehnica PWM (Pulse Width Modulation) [8]. Aceasta reprezintă o succesiune rapidă de impulsuri ON / OFF, conform *Fig. 5.17.*

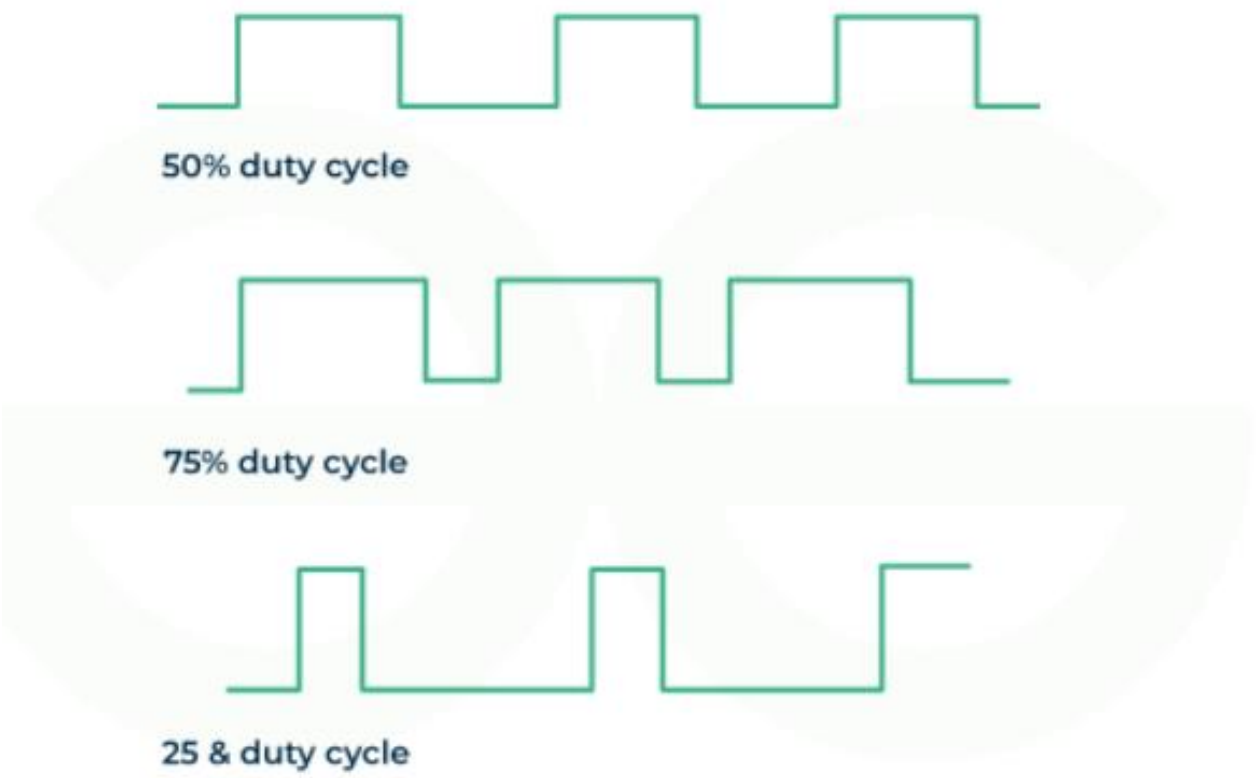

Fig. 5.17. Puls PWM

Valoarea finală este calculată în funcție de valoarea tensiunii maxime și valoare duty-cycle, care reprezintă procentul de timp în care semnalul este ON.
De exemplu, pentru o tensiune de 12V, valorile tensiunii la ieșire ar fi următoarele în cele 3 cazuri prezentate:

- Duty-cycle 50%: 6V
- Duty-cycle 75%: 9V
- Duty-cycle 25%: 3V

Această valoare poate fi ușor calculată folosind formula *6.15.*

$$U_{mediu} = U_{alimentare} * \frac{duty-cycle}{100} \quad (5.15)$$

$$\text{duty-cycle} = \frac{ONtime}{ONtime+OFFtime} \quad (5.16)$$

Feedback-ul sistemului de reglare automată pe baza căruia se calculează comanda este asigurat de prezența encoderului optic incremental în structura fiecărei articulații. Acesta convertește mișcarea mecanică de rotație într-un semnal digital. În aplicația mea, encoderul ajută la determinarea deplasării unghiulare și a direcției de rotație a motorului.
În structura unui encoder optic intră următoarele componente:

- Disc optic cu fante distribuite circular;
- LEDuri pentru a emite lumină;
- Fototranzistori pentru a detecta lumina emisă.

Pe măsură ce motorul se rotește, odată cu el se va roti si discul perforat. Fantele de pe acesta permit intermitent trecerea luminii. Astfel, fototranzistorii se vor deschide și vor genera impulsuri digitale. Fiecare impuls va reprezenta o unitate de mișcare.
În cazul meu sunt prezente două canale digitale A și B, generate în cuadratură, adică decalate în fază cu 90°.
Conform [7], aceste canale pot fi citite într-unul dintre următoarele moduri, în funcție de precizia dorită:

- Rezoluție 1x: se incrementează / decrementează poziția o singură dată după un ciclu complet a semnalelor A și B;
- Rezoluție 2x: se incrementează / decrementează poziția de două ori după un ciclu complet a semnalelor A și B; (atât pe front crescător cât și descrescător)
- Rezoluție 4x: se incrementează / decrementează poziția pentru fiecare tranziție individuală a semnalelor A și B.

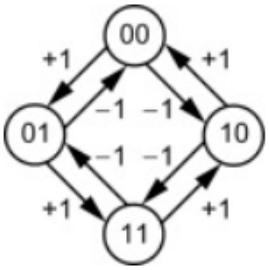

Fig. 5.18. Diagramă de stări pentru semnalele digitale A și B

Datorită preciziei necesară aplicației mele, am ales să măsor la o rezoluție 2x.
Pentru a fi posibil acest lucru, a trebuit să conectez ieșirile digitale ale encoderului la pini de pe placa de dezvoltare Arduino Nano care suportă întreruperi externe.
O întrerupere externă [9] reprezintă un mecanism prin care un microcontroller reacționează instant la un eveniment. Diagrama de stări a unui modul de tip slave care prezintă acest comportament este prezentată în *Fig. 5.19.*

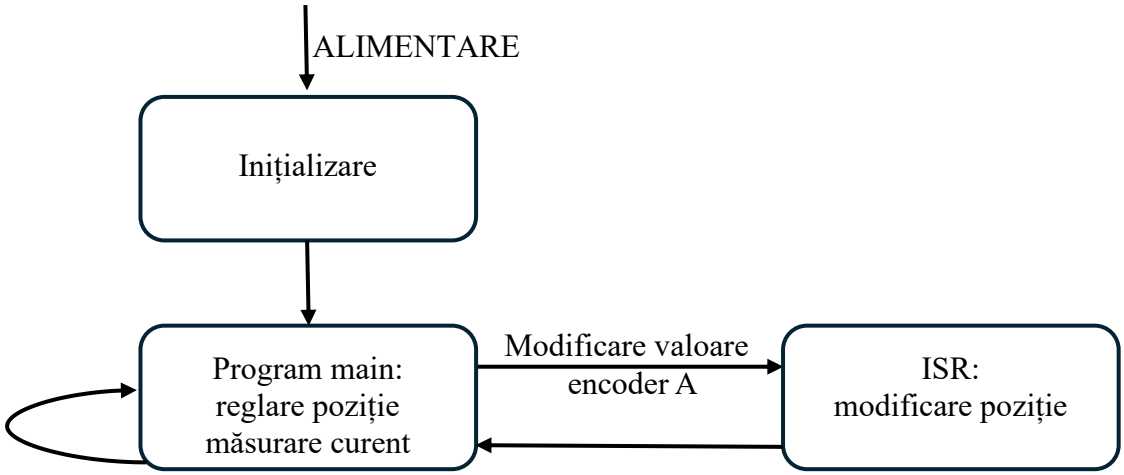

Fig. 5.19. Diagramă stări modul slave

În codul sursă, aceste stări sunt implementate după cum urmează:

```
void setup() {
  Serial.begin(9600);
  Wire.begin(SLAVE_ADDRESS);
  Wire.onReceive(receiveEvent);

  pinMode(13, INPUT);
  PCICR |= 0b00000001;
  PCMSK0 |= 0b00100000;

  pinMode(MOTOR_PIN_1, OUTPUT);
  pinMode(MOTOR_PIN_2, OUTPUT);
  pinMode(ENCODER_A, INPUT_PULLUP);
  pinMode(ENCODER_B, INPUT_PULLUP);
  pinMode(ENCODER_REF, INPUT_PULLUP);

  attachInterrupt(digitalPinToInterrupt(ENCODER_A), Moving, CHANGE);

  time = micros();
  start = false;
  motorMove(0, 0);
}
```

Fig. 5.20. Inițializare modul

După cum poate fi observat și în *Fig. 5.20.,* inițializarea modulului presupune următoarele:

- pornirea comunicației prin intermediul I2C și fixarea adresei corespunzătoare;
- activarea întreruperilor externe prin intermediul registrilor PCICR și PCMSK0;
- setarea modului corespunzător pentru fiecare pin folosit;
- setarea rutinei de tratare a întreruperii;
- resetarea valorilor variabilelor folosite în cod.

```
void loop() {
  currentMillis = micros();
  time = currentMillis;

  long e = target - current_poz;

  if (abs(e) > 10) {
    float u = kp * e;

    pwr = fabs(u);
    pwr = constrain(pwr, MIN_PWM, MAX_PWM);

    if (u < 0)
      motorMove(1, pwr);
    else
      motorMove(-1, pwr);
  } else {
    motorMove(0, 0);
  }
}
```

Fig. 5.21. Reglarea pozitiei folosind un regulator proporțional

Conform *Fig. 5.21.* starea care rulează cel mai mult presupune:

- calculul erorii curente pe baza referintei primate și a pozitiei curente;
- calculul comenzii necesare;
- verificare comenzii rezultate (trebuie să fie într-un interval suportat de motor);
- trimiterea comenzii către puntea H.

```
void motorMove(int dir, int power) {
  power = constrain(power, MIN_PWM, MAX_PWM);
  if (dir == 1) {
    analogWrite(MOTOR_PIN_1, power);
    analogWrite(MOTOR_PIN_2, 0);
  } else if (dir == -1) {
    analogWrite(MOTOR_PIN_1, 0);
    analogWrite(MOTOR_PIN_2, power);
  } else {
    analogWrite(MOTOR_PIN_1, 0);
    analogWrite(MOTOR_PIN_2, 0);
  }
}
```

Fig 5.22. Comanda motorului prin intermediul PWM

În *Fig. 5.22.* sunt prezentate toate cele 3 decizii care pot fi luate în vederea controlului motorului:

- Rotație în sens orar;
- Rotație în sens trigonometric;
- Oprire.

```
void Moving() {
  if (digitalRead(ENCODER_A) == HIGH) {
    current_poz += (digitalRead(ENCODER_B) == LOW) ? -1 : 1;
  } else {
    current_poz += (digitalRead(ENCODER_B) == LOW) ? 1 : -1;
  }
}
```

Fig. 5.23. Funcția de tratare a întreruperii

Tratarea întreruperii externe este realizată cu ajutorul funcției *Moving*, care este apelată la fiecare modificare a semnalului digital emis de encoderul A. Astfel, citind valoarea encoderului B putem deduce sensul rotației și modifica în mod corespunzător valoarea poziției curente.
Valoarea referinței este primită prin intermediul magistralei I2C, astfel:

```
void receiveEvent(int bytes) {
  if (Wire.available() >= 3) {
    byte highB = Wire.read();   // Byte-ul superior (MSB)
    byte midB  = Wire.read();   // Byte-ul mijlociu
    byte lowB  = Wire.read();   // Byte-ul inferior (LSB)

    target = ((long)highB << 16) | ((long)midB << 8) | lowB;

    Serial.print("Am primit: ");
    Serial.println(target);
  }
}
```

Fig. 5.24. Primire date prin I2C

Având în vedere intervalul mare de valori în care valoarea deplasării unghiulare se poate găsi, a fost nevoie de împărțirea acestei valori în 3 bytes. În acest sens, înainte de transmitere, modulul master o va sparge în 3 bucăți în timp ce modul slave o va reconstrui.

Atât la baza modulului de tip MASTER cât și a celor de tip SLAVE, stă microcontroller-ul ATmega328P, folosit prin intermediul plăcuței Arduino Nano [6] prezentată în *Fig 5.25.*, pentru a ușura procesul de dezvoltare. Am ales această placă de dezvoltare datorită dimensiunii și a costului redus. De asemenea, detaliile tehnice ale acesteia sunt suficiente:

- 14 pini digitali I/O;
- 6 pini care suportă PWM;
- 8 pini analogici;
- 32KB memorie flash;
- 16 MHz frecvență de ceas;
- Comunicație I2C;
- Comunicație UART;
- Întreruperi externe.

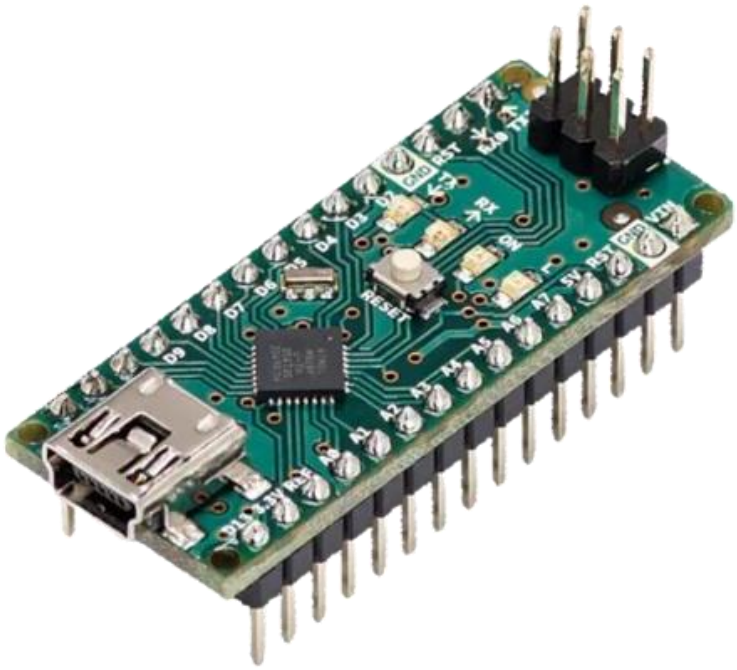

Fig 5.25. Placă de dezvoltare Arduino Nano

Aceste module de tip *SLAVE* sunt în număr de 6, unul pentru fiecare dintre cele 5 articulații și unul pentru unealta din vârful robotului, care poate efectua operații de închidere și deschidere.

Pentru o eficientizare a spațiului ocupat, modulele slave au fost grupate două câte două, după cum se poate observa și în *Fig. 5.26.*

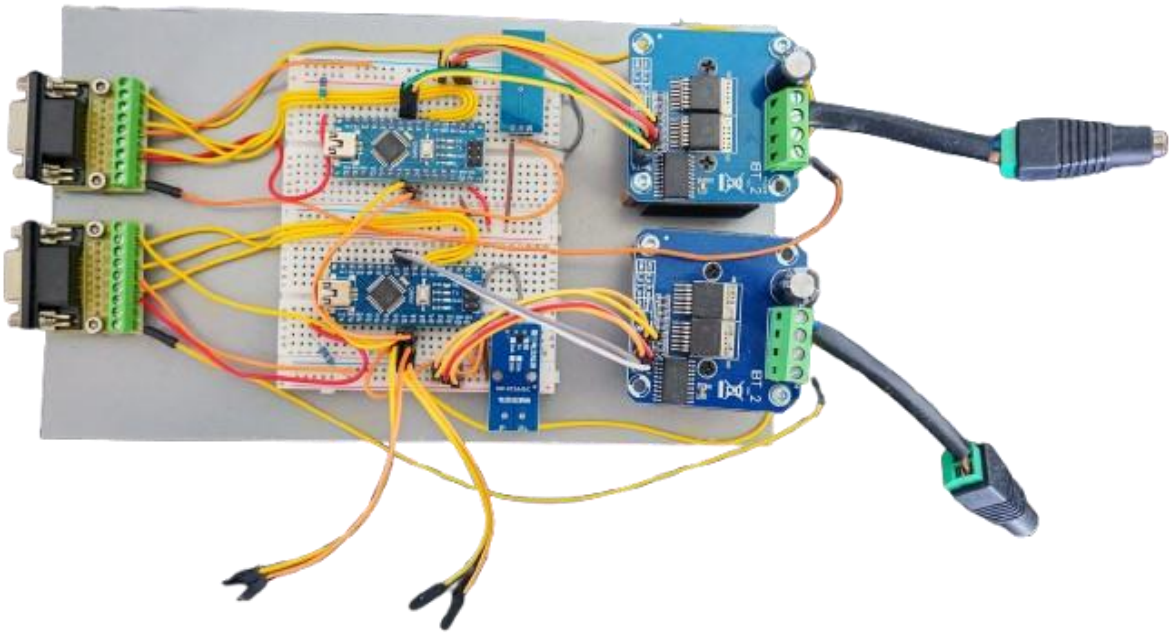

Fig. 5.26. Module Slave

Pe lângă microcontroller-ul deja menționat, în componența acestui modul mai intră și următoarele componente:

- Senzor de curent HALL – ACS712
- Punte H de putere – BTS7960
- Terminal DB9-RS232

Senzorul de curent a fost adăugat pentru a putea măsura consumul pe fiecare articulație a brațului robotic.
Principiul pe bază căruia acesta funcționează este următorul:

- Curentul trece printr-un conductor integrat în cip și generează un câmp magnetic;
- Senzorul HALL detectează câmpul magnetic și generează o tensiune;
- Tensiunea va fi proporțională cu valoarea curentului și este centrată în jurul valorii de 2V5 pentru a putea măsura atât valori pozitive cât și negative.

Modulul folosit de mine, ACS712, are următoarele detalii tehnice:

- Sensibilitate: 66 mV/A;
- Gamă de măsurare: +/-30 A;
- Curent consumat 10mA;
- Tensiune de alimentare: 5V.

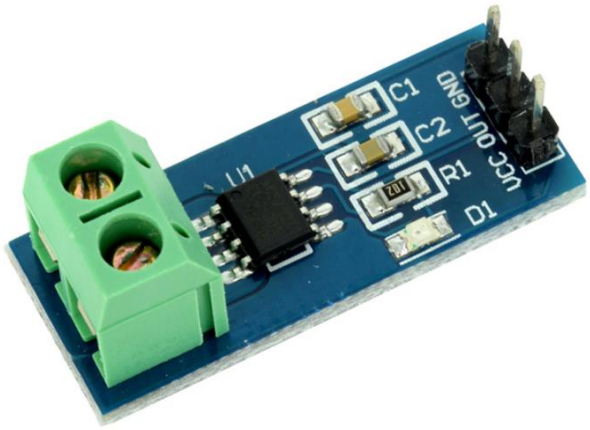

Fig. 5.27. Modul Hall ACS712

Conform *Fig. 5.27.*, modulul ACS712 are 3 pini la care trebuie să se conecteze următoarele:

- GND: masa sistemului;
- OUT: ieșire analogică;
- VCC: alimentare de 5V.

De asemenea, curentul care se dorește a fi măsurat trebuie să înserieze acest modul prin intermediul celor 2 conectori tip borna cu șurub.

Curentul măsurat va fi salvat și afișat prin intermediul unui grafic, precum cel din *Fig. 5.28.*

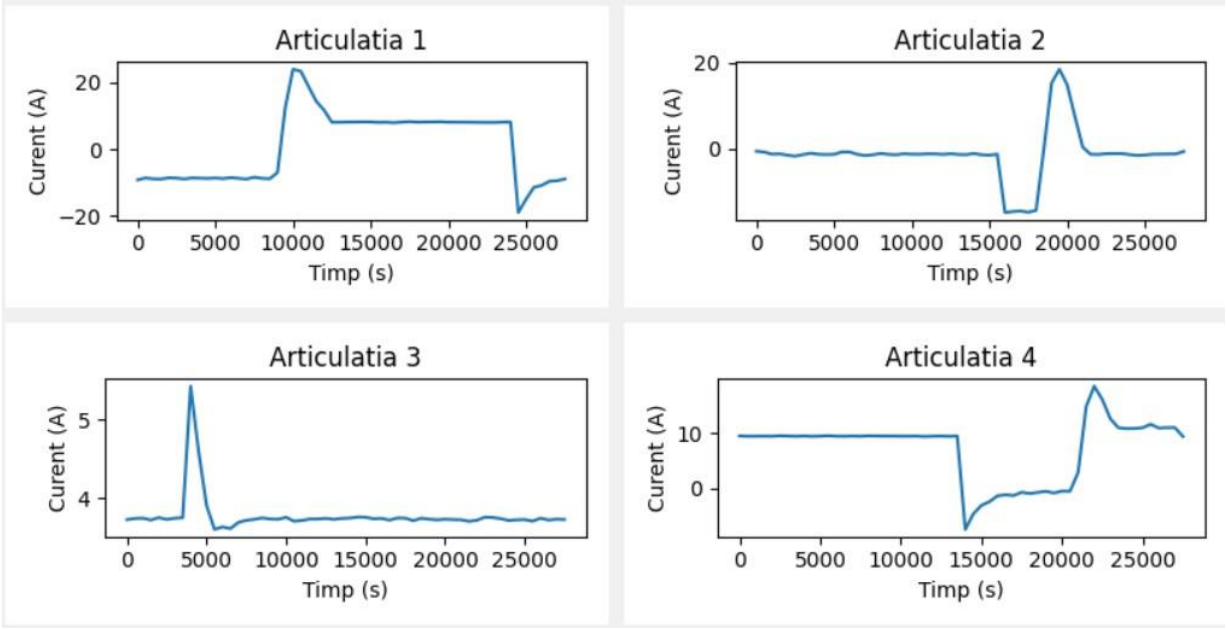

Fig. 5.28. Curent consumat pe parcursul unei deplasări

Pentru a primi aceste grafice, sistemul va trece prin următoarele etape:

- Cerere din parte utilizatorului prin apăsarea butonului corespunzător din interfață;
- Trimiterea unui mesaj prin intermediul UART către modulul master;
- Cererea valorilor curenților consumați de către fiecare motor folosit pentru deplasarea articulațiilor prin intermediul I2C;
- Organizarea valorilor primite;
- Trimiterea valorilor înapoi către aplicație prin intermediul UART.

Această secvență de pași este realizată de master după cum se poate vedea în figurile următoare.

```
case 'X':
    for(int i = 0; i < 4; i++) {
      readCurrentBuffer(SLAVE1_ADDRESS, currentBuffer1, bufferSize, 0);
      readCurrentBuffer(SLAVE2_ADDRESS, currentBuffer2, bufferSize, 1);
      readCurrentBuffer(SLAVE3_ADDRESS, currentBuffer3, bufferSize, 2);
      readCurrentBuffer(SLAVE4_ADDRESS, currentBuffer4, bufferSize, 3);
      readCurrentBuffer(SLAVE5_ADDRESS, currentBuffer5, bufferSize, 4);
      delay(500);
    }
      sendAllBuffersOverSerial();
      break;
```

Fig. 5.29. Primirea cererii de la aplicația utilizator

După cum se poate observa în *Fig. 5.29.*, când comanda primită prin UART este ***X***, modulul va trimite request-uri către fiecare master. Numărul de request-uri este în număr de 4 din cauza limitării trasmisiei prin I2C, prin care se pot trimite secvențe de maxim 32 de octeți.

```
void readCurrentBuffer(uint8_t slaveAddress, float* targetBuffer, int bufferSize, int nR) {
  for (int p = 0; p < 2; p++) {
    Wire.requestFrom(slaveAddress, 8 * sizeof(float));  // 32 bytes

    byte* ptr = (byte*)(targetBuffer + readIndex[nR]);
    int i = 0;
    while (Wire.available() && i < 32) {
      ptr[i++] = Wire.read();
    }

    readIndex[nR] += 8;
    if (readIndex[nR] >= bufferSize)
      readIndex[nR] = 0;
  }
}
```

Fig. 5.30. Trimiterea unei secvențe de request pe I2C către un slave

În *Fig. 5.30.* se poate observa cum se trimit 2 cereri de câte 32 de octeți către fiecare slave și după primire se salvează circular în buffer-ul corespunzător.
În total vom avea câte 56 de valori ale curentului măsurat pentru fiecare dintre cele 5 articulații.

```
void sendAllBuffersOverSerial() {
  for (int i = 0; i < bufferSize; i++) {
    Serial.print(currentBuffer1[i], 3); Serial.print(',');
    Serial.print(currentBuffer2[i], 3); Serial.print(',');
    Serial.print(currentBuffer3[i], 3); Serial.print(',');
    Serial.print(currentBuffer4[i], 3); Serial.print(',');
    Serial.print(currentBuffer5[i], 3); Serial.println();
  }
}
```

Fig. 5.31. Trimiterea datelor către aplicația utilizator

După ce toate datele au fost primite de la modulele slave, modulul master le trimite prin intermediul UART către aplicația utilizator care le va salva și afișa grafic.

În ceea ce privește rolul modului slave în acest mecanism, acesta este următorul:

- Fiecare modul va salva ultimele 56 de valori ale curentului consumat măsurat pe parcursul unei deplasări într-un buffer circular de dimensiune;
- Când va primi o cerere de la master, device-ul va trimite aceste valori.

```
float voltage = 0;

for (int i = 0; i < numSamples; i++) {
  voltage += analogRead(analogPin) * (5.0 / 1023.0);
}

voltage /= numSamples;

float current = (voltage - Vref) / sensitivity;

currentBuffer[writeIndex] = current;
writeIndex = (writeIndex + 1) % bufferSize;
```

Fig. 5.32. Măsurarea curentului consumat

Conform *Fig. 5.32.*, pentru a măsura curentul am efectuat o mediere a mai multor valori consecutive pentru a nu avea foarte mult zgomot. De asemenea, valorile sunt scrise circular în vectorul *currentBuffer*.
Conform specificațiilor senzorului de curent folosit, ACS712, acesta folosește un decalaj de 2V5 pentru a putea măsura atât valori negative cât și pozitive a curentului. Pentru a converti

valoarea tensiunii măsurate în amperi, rezultatul se împarte la valoarea sensitivității, care în cazul meu este 0.66mV/A.

La primirea unei cereri de citire a valorilor din partea modulului master, fiecare slave va apela funcția prezentată în *Fig. 5.33.*

```
void requestEvent() {
  int start = packetIndex * 8;

  if (start + 8 > 56) {
    packetIndex = 0;
    start = 0;
  }

  Wire.write((byte*)(currentBuffer + start), 8 * sizeof(float));

  packetIndex++;
  if (packetIndex >= 7) packetIndex = 0;
}
```

Fig. 5.33. Trasmiterea datelor prin intermediul I2C

Conform protocolului stabilit, vom avea pentru fiecare articulație câte 56 de măsurători, deci 7 pachete de câte 8 măsurători. Prin intermediul unui pachet se trimit doar 8 măsurători din cauza limitei de 56 de octeți pe care o poate atinge dimensiunea unei trasmisiuni prin intermediul protocolului de comunicație I2C.

6. Exemplu de utilizare al sistemului

Un exemplu de utilizare al sistemului dezvoltat poate fi reprezentat de deplasarea liniară între două puncte aflate în spațiul cartezian. Pentru a realiza această mișcare, utilizatorul va trebui să parcurgă următorii pași în acestă ordine:

1. Alimentarea sistemului.
2. Conectarea modului master la calculatorul personal.
3. Deschiderea aplicației utilizator.
4. Conectarea aplicației la modulul master prin selectarea port-ului de comunicație precum este prezentat în *Fig. 6.1.*

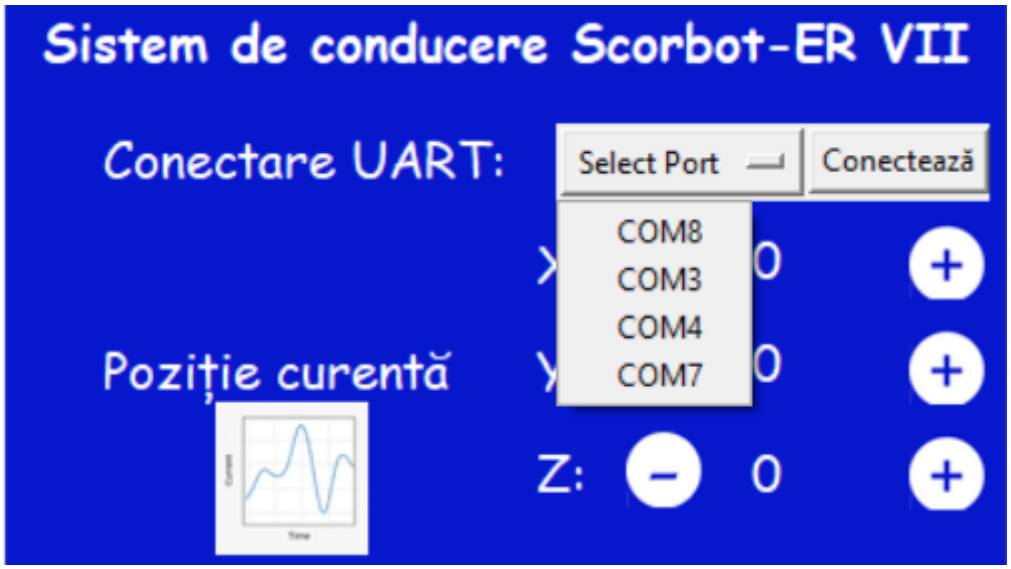

Fig. 6.1 Conectarea modului master la aplicația utilizator

La apăsarea butonului va fi afișată o listă cu toate porturile de comunicație disponibile.

5. Salvarea punctelor între care se dorește să se efectueze mișcarea liniară, folosind controlul robotului în spațiul articulațiilor.

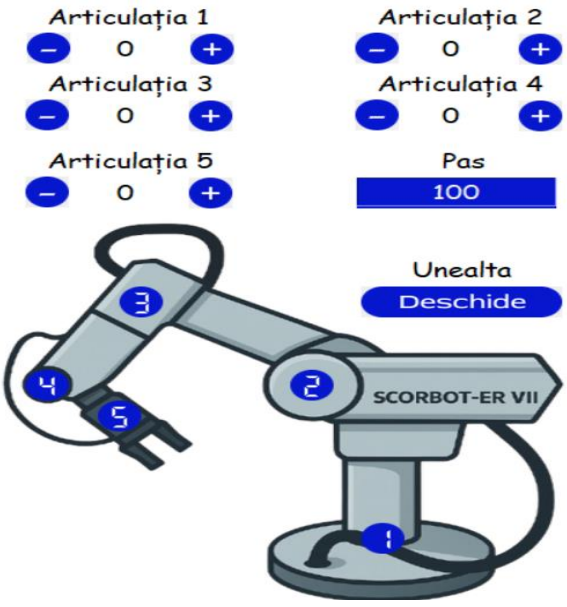

Fig. 6.2. Controlul individual al articulațiilor

Folosind butoanele + și – prezentate în *Fig. 6.2*., utilizatorul va modifica individual pozitiile fiecărei articulații. După ce ajunge la o punctul dorit, îl poate salva folosind butonul ***Salvează punctul curent***.

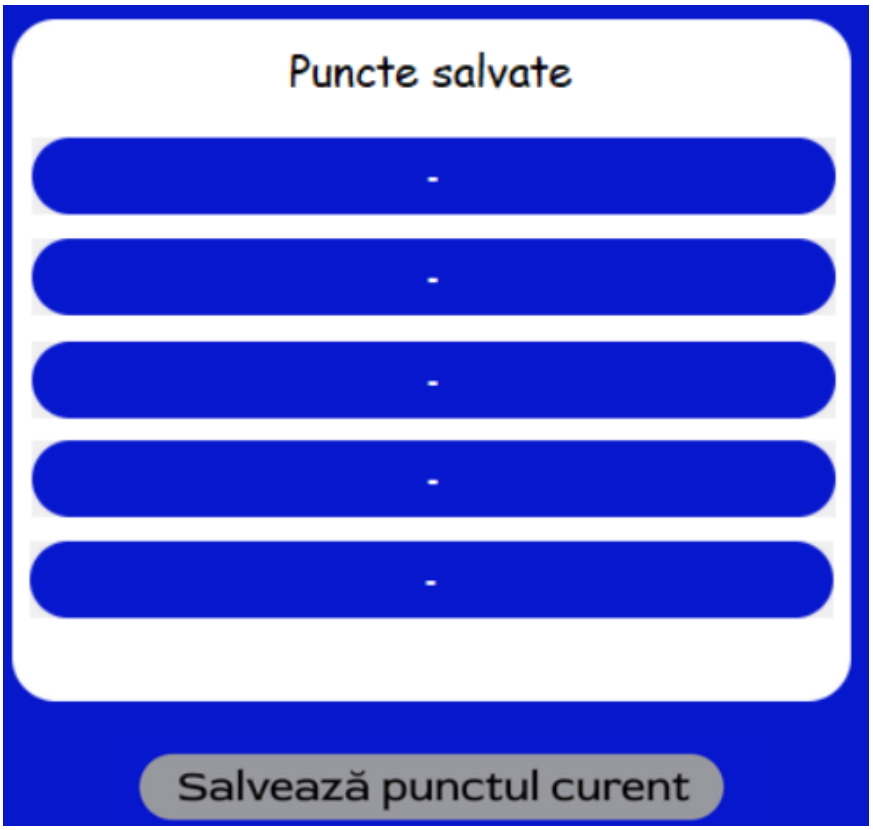

Fig. 6.3. Salvarea configurațiilor

După apăsarea acestui buton, lista care poate fi observată în *Fig. 6.3.* va fi populată în mod circular cu pozițiile articulațiilor curente.

6. După salvarea pozițiilor dorite, lista va arăta astfel:

Fig. 6.4. Exemplu de 2 puncte salvate

7. Să presupunem că robotul se află într-o poziție oarecare și dorim deplasarea liniară a acestuia între cele două puncte salvate. Inițial, utilizatorul va deplasa brațul robotic la configurația 1 folosind spațiul articulațiilor. La apăsarea pe o configurație se vor afișa următoarele posiblități:

Fig. 6.5. Alegerea tipului de deplasare

Inițial se va alege ***Spațiul articulațiilor***.

8. După ce robotul ajunge în configurația 1, utilizatorul poate apăsa pe configurația 2 și să aleagă ***Spațiul cartezian***.

Urmărind pașii menționați, utilizatorul poate efectua o mișcare liniară a efectorului terminal al brațului robotic între două puncte din spațiul cartezian. Poziția acestuia poate fi observată în timp real prin intermediul aplicației și va afișată conform *Fig. 6.6.*

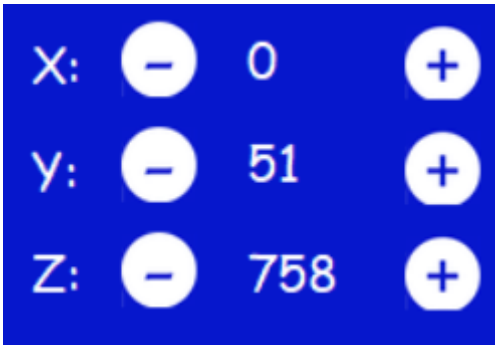

Fig. 6.6 Poziția curentă în spațiul cartezian

De asemenea, utilizatorul poate verifica și consumul de curent pe care l-a avut robotul pe parcursul deplasării respective. Acesta va fi afișat sub forma unor grafice precum cele din *Fig. 6.7.*

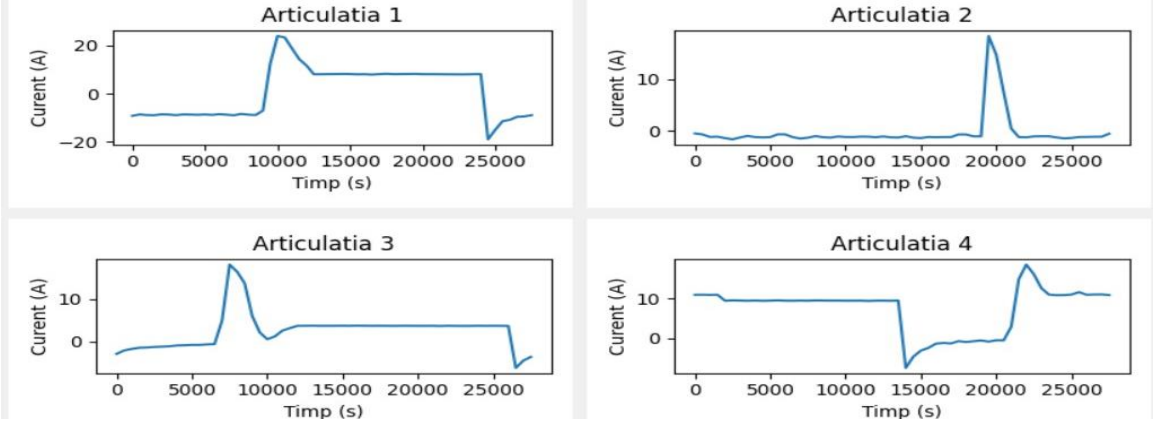

Fig. 6.7. Curent consumat de fiecare articulație

7. Concluzii si planuri de viitor

Această lucrare a avut ca scop implementarea unui sistem de conducere pentru robotul industrial Scorbot-ER VII, cu 5 grade de libertate. Proiectul a integrat atât componente hardware cât și software, permitând controlul precis al fiecărei articulații prin intermediul unei interfețe grafice prietenoase.

Am reușit să obțin următoarele rezultate:

- Control individual al tuturor articulațiilor;
- Deplasare în spațiul articulațiilor;
- Deplasare în spațiul cartezian;
- Măsurarea curentului consumat pe parcursul deplasărilor.

De asemenea, în urma acestui proiect, pot spune că am reușit să înteleg mult mai bine conceptele de cinematica inversă, dar și de integrare hardware-software a unui sistem robotic real plecând de la zero. În plus, a fost necesar să efectuez analize în ceea ce privește componentele utilizate astfel încât să ajung la un rezultat cât mai bun.

În ceea ce privește planurile de viitor, pot menționa automatizarea robotului cu ajutorul unui sistem de vedere artificială.

8. Bibliografie

1. Eshed Robotec. *Scorbot-ER VII User's Manual*. 2nd ed., 1996.
2. Laurenţiu Predescu. *On the kinematics of the Scorbot ER-VII Robot*, 2015.
3. Jonathan Valdez & Jared Becker. *Understanding the I2C Bus*, 2015.
4. Umakanta Nanda & Sushant Kumar Pattnaik. *UART*, 2016.
5. International Federation of Robotics. *Industrial Robots – IFR.* [online] Disponibil la: https://ifr.org/industrial-robot [Accesat la: 01 mai 2025].
6. Arduino. *Arduino NANO – Datasheet*. [online] Disponibil la: https://docs.arduino.cc/resources/datasheets/A000005-datasheet.pdf [Accesat la: 15 februarie 2025].
7. ScienceDirect. *Encoder incremental*. [online] Disponibil la: https://www.sciencedirect.com/topics/engineering/incremental-encoder [Accesat la 20 martie 2025].
8. GeeksForGeeks. *Pulse Width Modulation*. [online] Disponibil la: https://www.geeksforgeeks.org/pulse-width-modulation-pwm/ [Accesat la 5 martie 2025].
9. GeeksForGeeks. *External interrupts*. [online] Disponibil la: https://www.geeksforgeeks.org/external-and-internal-interrupts/ [Accesat la 20 februarie 2025].