Universitatea Politehnică Bucureşti Facultatea de Automatică şi Calculatoare
Departamentul de Automatică și Ingineria Sistemelor

LUCRARE DE LICENŢĂ

Proiectarea unui sistem informatic autonom mobil cu capabilități de recunoaștere a traseului

Absolvent

Dincă Marius Cătălin

Coordonator

Prof. dr. ing. Sgârciu Valentin

București 2019

CUPRINS

INTRODUCERE

Încă din antichitate omul a încercat pe cât posibil să își îmbunătățească cât mai mult nivelul de trai. În acest sens, el și-a contruit de-a lungul timpului diferite unelte ce l-au ajutat să îi simplifice sau chiar înlocuiască munca fizică. Un aport important în acest proces l-a avut robotul, descris astăzi ca, citez, "*mecanism automat care poate executa, după un program stabilit, operații complicate, asemănătoare cu cele realizate de om*".

Cuvântul "*robot*" este de origine slavă și se traduce prin "*muncă*". A fost folosit pentru prima dată de scenaristul ceh Karel Capek într-o piesă de teatru numită "*Rossum's Universal Robots*". Tema acestei piese de teatru a fost despre dezumanizarea persoanei într-o civilizaţie bazată din ce în ce mai mult pe tehnologie.

Pentru realizarea de sisteme autonome (care să găsească singure soluții) este necesară legătura a mai multor discipline de robotică (termen utilizat prima dată de scriitorul şi omul de ştiinţă american de origine rusă, Isaac Asimov, într-o scurtă povestioară numită "*Runaround*" în anul 1942). În general, roboții sunt realizați mai ales prin combinația disciplinelor mecanică, electrotehnică și informatică, care a dus la apariția mecatronicii, o disciplină integrata în robotică.

Această lucrare prezintă modul de proiectare si realizare a unui sistem autonom echipat cu o placă de bază Arduino UNO, senzori de reflexie QTR-8RC și alte componente, capabil să găsească cel mai scurt drum pentru un traseu dat.

CAPITOLUL 1: SISTEMUL INFORMATIC

1.1 Ce este un sistem informatic mobil

Putem definii ca fiind un sistem informatic mobil autonom orice robot sau orice operator artificial, fie el mecanic sau virtual, capabil să execute automat o sarcină dată, fără intervenție umană. Acesta este de fapt un sistem compus din mai multe elemente de mecanică, ca: actuator, sistem de mișcare și direcționare , sistem de calcul și decizie, senzori etc. Aspectul robotului și posibilitățile sale de mișcare în plan este reprezentat de sistemul de direcție și mișcare. Mobilitatea robotului este data de acest sistem. Senzorii sunt folosiți pentru acumularea de informații din mediul extern și intern, iar blocul de calcul și decizie este folosit pentru prelucrarea datelor furnizate de senzori și luarea deciziilor bazate pe acestea.

Isaac Asimov, scriitor american de science fiction și profesor la Universitatea din Boston, specifica în opera sa, "*Runaround*" , publicată în octombrie 1942, trei legi pe care un robot trebuie să le respecte. O interpretare a acestora este:

- Un robot, prin acțiunile sau inacțiunile sale, nu poate afecta existența umană sau să îi permită acesteia să meargă către distrugere;
- Un robot trebuie să îndeplinească sarcinile asignate de către un om, atâta timp cât acestea nu intră în conflict cu prima lege;
- Un robot trebuie să își conserve propria existență, atâta timp cât acesta nu intră în conflict cu prima lege și a doua lege;

Dea lungul timpului au fost oferite mai multe definiții ale robotului, de către diferite institute. În standardul ISO 8373 robotul este definit ca, citez, "*manipulator universal, reprogramabil, având control automat și trei sau mai multe axe*".

Institutul de robotică din SUA definesc la rândul lor robotul ca, citez, "*un manipulator universal, reprogramabil, conceput să deplaseze materiale, componente, scule sau sisteme specializate, prin diferite mişcări programate, pentru a îndeplini diferite sarcini* ".

O abordare diferită asupra robotului a avut-o publicația americană Merriam-Webster, care definesc robotul ca, citez, "*o mașină care arată ca și un om, și care realizeaza sarcini complexe (cum ar fi mersul sau vorbirea) caracteristice omului*".

În prezent , termenul de "*robot*" descrie un domeniu destul de vast, fapt ce a dus la categorisirea acestora în funcție de sarcinile îndeplinite.

De asemenea, progresul rapid al dezvoltării roboților a dus la apariția unei noi științe ce se ocupă cu proiectarea și fabricarea acestora: robotica.

1.2 Ce este robotica

Robotica este stiința care se ocupa cu tehnologia, proiectarea și fabricarea roboților. Această știință a apărut de aproximativ șase decenii, deși termenul a fost folosit mai devreme. Robotica este formată din mai multe discipline, principalele fiind informatică, mecanică și electrotehnică iar persoana care lucrează în domeniu se numeste robotician.

Chiar și construcția unui robot ce ar trebui să îndeplinească o misiune simplă, este în sine un proces complicat ce necesită cunoștiințe din mai multe domenii. În mare, robotica se împarte în trei subdiviziuni: percepție, cogniție și acțiune. Cu alte cuvinte, un robot trebuie să fie capabil să ”simtă” pentru a putea primi informații din mediul extern. De asemenea, trebuie să fie capabil să ”înțeleagă” informațiile primite, iar în final trebuie să decidă și să ”facă ceva” în funcție de informațiile primite.

Percepția include tot ceea ce ține de senzorii prin care robotul poate primi informații din mediu în care operează, diversitatea lor fiind foarte vastă. Cogniția are loc la nivelul blocului de calcul și decizie, iar acționarea, în funcție de tipul robotului sau chiar a sarcinii ce trebuie îndeplinită, poate avea loc ori la același nivel, ori la nivelul blocului mecanic.

În ultimii ani, în cadrul roboticii a apărut un nou concept de construcție de robot artificial ce se bazează strict pe programare, definit ca RPA (robot process automation). Aceasta se ocupă în principiu cu automatizarea taskurilor repetitive, ce se realizează pe calculator.

1.3 Primele aplicații robotice

După Al Doilea Război Mondial, America a cunoscut o puternică dezvoltare industrială și o creștere economică semnificativă. Acest fapt a dus la crearea primului robot în acest domeniu, denumit ”*the Unimate*”, care era capabil să mute obiecte în jurul

său. Acesta era de fapt un braț robotic apărut în anul 1961, inventat de George Devol împreună cu Joseph Engelberger și opera sub brevetul ”Programmed Article Transfer” depus de George Devol în anul 1954, tradus ca:

” *Prezenta invenție se referă la funcționarea automată a mașinilor, în special la aparatele de manipulare, și la aparatele automate de comandă potrivite pentru astfel de mașini.*”

În 1962 ”*the Unimate*” a fost cumpărat de General Motors, împreună cu tehnologia de contruire a acestuia, și a fost primul braț robotic folosit pe o linie de asamblare. În aceeași perioada, Engelberger împreună cu Devol au înființat compania ”Unimate Inc.”, care a devenit lider mondial în robotică.

Jorge Angeles definește sistemul robotic ca un sistem compus din mai multe subsisteme, ce comunică între ele prin interfețe ale căror funcții de bază este decodificarea informației de la un subsistem la altul. O diagramă a sistemului robotic, definită în cartea ” , Fundamental of Robotic Mechanical Systems – Theory, Methods and Algorithms”, este:

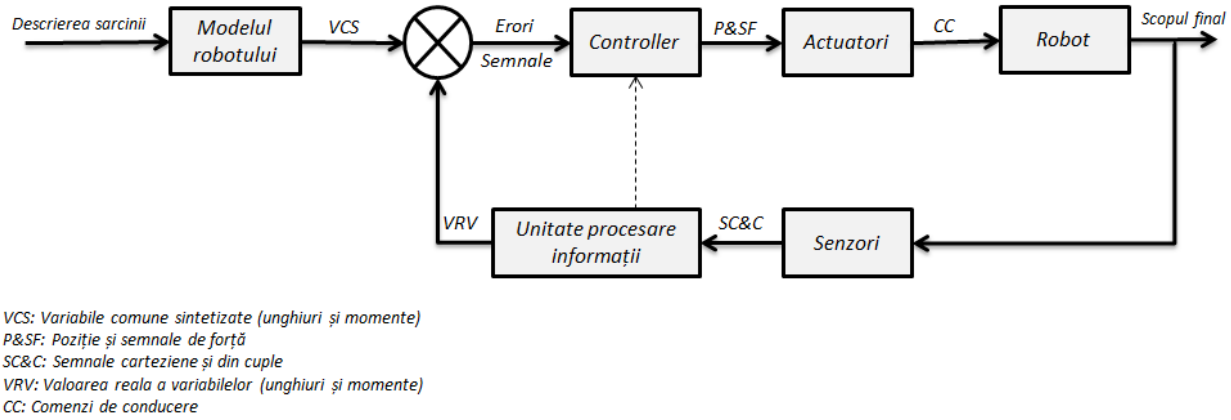


Figura 1.1 Diagrama bloc generală a unui sistem informatic

1.4 Robot process automation

Dezvoltarea industriei IT din ultimul deceniu a adus cu sine un nou mod de muncă unde majoritatea taskurilor sunt făcute pe un sistem digital, iar multe dintre ele sunt repetitive. Acestea necesită mai degrabă atenție din partea operatorului uman, decât gândire. Nevoia de automatizare a acestor taskuri și eliminarea greșelilor umane

a dus la dezvoltarea unui nou concept de robot. Acesta poate fi clasificat ca un robot virtual și este reprezentat de un program software prin care anumite taskuri pot fi automatizate.

Procesul de proiectare și construcție al unui asemenea robot este cunoscut sub numele de ”robot process automation”. Aceasta reprezintă tehnologia care permite oricărui utilizator să configureze un robot care poate emula și integra acțiunile unei persoane care interacționează cu un sistem digital pentru a executa un task. Roboții RPA utilizează interfața cu utilizatorul pentru a capta date și a manipula aplicații la fel ca oamenii. Aceștia interpretează, declanșează răspunsuri și comunică cu alte sisteme pentru a putea realiza o gamă largă de sarcini repetitive.

1.5 Clasificarea roboților

În cartea sa de sinteză, Jorge Angeles definește roboții controlați de o ființă umană, telemanipulatori. În cazul acestora, scopul final ce trebuie îndeplinit de către robot este definit de către om, care devine astfel componenta de control în diagrama robotului. O clasificare a roboților oferită de Jorge Angeles, în funcție de modul lor de operare, este:

- roboți manipulatori seriali;
- roboți manipulatori paraleli;
- brațe robotice;
- roboți mobili pășitori;
- roboți mobili cu roți;

Roboții manipulatori seriali sunt considerați cele mai simple sisteme robotice, ei încercând să reproducă mișcările brațului uman în manipularea obiectelor. Aceștia se confruntă însă cu o gamă largă de constrângeri, fapt ce a dus la separarea lor de manipulatorii paraleli, care sunt formați dintr-o platformă de bază fixă și una mobilă, care acționează mai multe brațe.

Un robot care să imite mersul uman a fost greu de realizat din cauza echilibrului greu de obținut, cauzat de variația parametrilor dinamici de mișcare. Un succes real l-au avut însă roboții cu mai multe picioare.

Varietatea domeniilor în care astăzi sunt folosiți roboții este largă, așadar clasificarea lor după un singur criteriu este irelevantă. O clasificare mai amplă a roboților, ar fi:

În funcție de gradul de mișcare:

- roboți mobili;
- roboți ficși;

În funcție de sistemul de coordonate:

- roboți în sistem de coordonate carteziene;
- roboți în sistem de coordonate cilindrice;
- roboți în sistem de coordonate sferice;

În funcție de sistemul de acționare:

- roboți cu sistem de acționare electric;
- roboți cu sistem de acționare hidraulic;
- roboți cu sistem de acționare pneumatic;
- roboți cu sistem de acționare mixt;

În funcție de generarea traiectoriei:

- roboți cu poziționare continuă;
- roboți cu poziționare secvențială;

În funcție de valoarea capacității portante:

- microroboți (zeci de grame);
- miniroboți (sute de grame);
- roboți mijlocii (de ordinul kilogramelor);
- roboți grei (de ordinul sutelor de kilograme);

În funcție de prelucrarea informației și metoda de control:

- roboți acționați de un om;
- roboți autonomi cu sistem de control cu relee;
- roboți autonomi cu sistem de control secvențial cu program modificabil;
- roboți autonomi repetitori, cu program de instruire;
- roboți autonomi cu învățare automată (inteligenți);

Pentru ca un robot să fie autonom el în primul rând trebuie să dispună de resursele de calcul atât hardware cât și software, altele decât cele de interferență în timp real de la un agent uman, pentru a calcula in mediul fizic incorporat cele mai bune actiuni posibile delimitate de un set de constrangeri in vederea realizarii unui set de obiective. Pe urma, un robot autonom trebuie sa dispuna de resursele adecvate pentru a interprinde un set de actiuni in anumite limite in vederea atingerii unui scop.

În funcție de metoda de programare:

- roboți cu programare rigidă (nu se poate modifica programul);
- roboți cu programare flexibilă (se poate modifica programul);
- roboți cu programare adaptivă (programul se poate adapta în timpul funcționării);

O clasificare a roboților se poate face și în funcție de domeniul în care acesta activează. Se pot identifica astfel următoarele mari categorii de roboți principali:

- roboți industriali – manipulatori concepuți pentru a efectua o serie de sarcini programate în medii de producție și fabricație;
- roboți militari – acționați de un om sau autonomi, cu scopul de căutare, salvare, cartografiere etc.;
- roboți din industria medicală – utilizați în diferite proceduri medicale;
- roboți de serviciu – cu utilitate casnică;
- roboți exploratori – utilizați în explorarea spațiului greu accesibil sau inaccesibil omului;

CAPITOLUL 2: ARHITECTURA SISTEMULUI INFORMATIC

2.1 Structura unui sistem informatic

Structura unui robot reprezintă de fapt un sistem informatic compus din mai multe subsisteme. Un sistem este un ansamblu de părți componente, elemente și legăturile dintre acestea. Elementele în sine sunt un subsisteme care, la rândul său, pot fi compuse din mai multe subsisteme. De aceea există o ierarhizare, iar sistemul

principal poartă denumirea de sistem de rangul I, subsistemele acestuia se numesc sisteme de de rangul II și așa mai departe. Modul de compunere și legăturile dintre subsisteme definesc structura unui sistem. Robotul este un sistem de rangul I, iar o structura orientativă a acestuia se poate observa în următoarea figură:

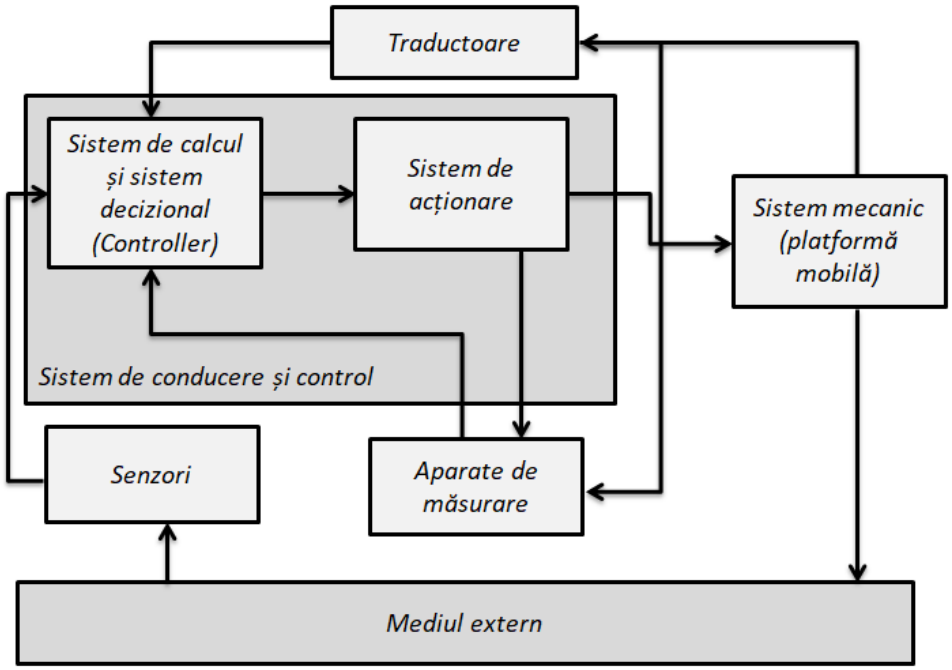


Figura 2.1 Structura unui sistem informatic

Sistemul unui robot comunică cu mediul exterior si este format din următoarele elemente:

- Sistemul mecanic, care reprezintă și ”scheletul” robotului, definește amplitudinea mișcărilor ce pot fi efectuate;
- Sistemul de acționare realizează miscarea efectivă a elementelor din sistemul mecanic. Mobilitatea robotului este dată de acest sistem de rangul II;
- Sistemul de calcul și decizional prelucrează informațiile primite de la senzori , aparatele de măsurare și sistemul mecanic (prin traductoare), iar apoi trimite comenzi către sistemul de acționare;
- Traductorii preiau informații de la sistemul mecanic despre starea internă a robotului, ca de exemplu viteză, accelerație, presiune, temperatură, direcție de deplasare și așa mai departe;
- Senzorii preiau informații de la mediul exterior care sunt trimise către sistemul de comandă;

- Sistemul de conducere este un sistem superior celui mecanic și este compus din sistemul de comandă și sistemul de acționare;
- Mediul extern reprezintă ”spațiul” în care operează robotul. Legăturile dintre robot și mediul extern pot fi directe sau inverse (feedback). Cele directe se fac de obicei cu sistemul mecanic, iar cele indirecte cu senzorii, aparatele de măsurare, traductori;

2.2 Sistemul mecanic – noțiuni generale

Structura robotului este definită de scopul acestuia. De exemplu, un robot industrial trebuie să fie capabil să acționeze asupra mediului înconjurător, să preia informații de la acesta, să decidă în scopul realizării unor sarcini și să acționeze pe baza informațiilor primite. Pentru realizarea acestor funcții, structura robotului industrial este alcătuită din:

- sistem mecanic;
- sistem de acționare;
- sistem programare și comandă;
- sistem senzorial;

Sistemul mecanic al robotului trebuie să aibă capabilitatea și să dețină energia necesară de a deplasa obiecte, în intervalul unei anumite greutăți. Partea din sistemul mecanic care realizează această deplasare se numeşte dispozitiv de ghidare sau manipulator. Subsistemul din cadrul sistemului mecanic dedicat scopului de ghidare a obiectelor se numește dispozitiv de ghidare și are rolul de a da efectua mișcările robotului. Dispozitivele de ghidare pot fi:

- cu topologie serială;
- cu topologie paralelă;
- cu topologie mixtă;

Partea din sistemul mecanic prin care robotul acționează asupra mediului se numește efector final. O diagramă generală a sistemului mecanic al unui robot se poate observa în urmatoarea imagine:

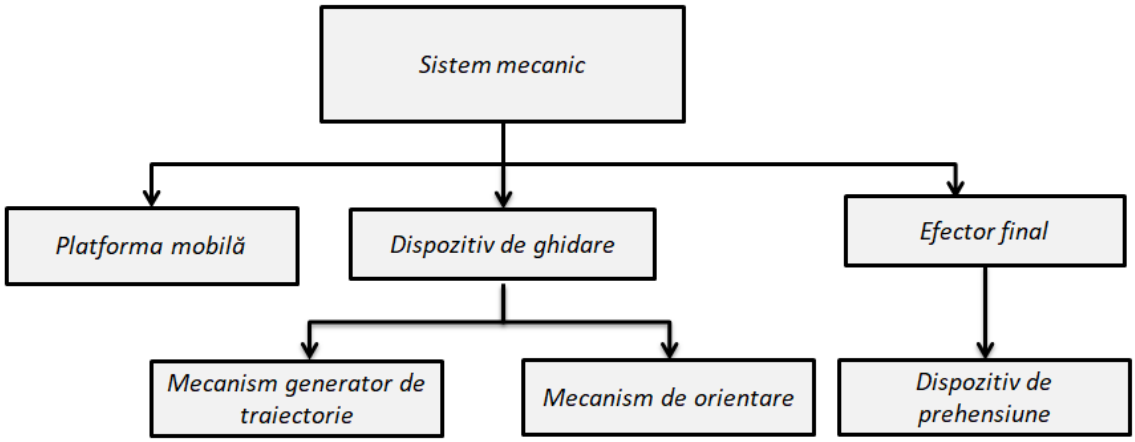


Figura 2.2 Sistemul mecanic al unui sistem informatic

În cadrul sistemului mecanic, componenta al cărui scop o reprezintă orientarea în spațiu și definirea traiectoriei este mecanismul de orientare, iar deplasarea propriu zisă este generată de mecanismul de traiectorie.

Platforma mobilă este componenta a sistemului mecanic care asigură modificarea situării robotului în mediu extern și de asemenea definește tipul robotului:

- Robot staționar (în acest caz platorma mobilă lipsește);
- Robot mobil (în acest caz dispozitivul de ghidare modifică situarea obiectului în raport cu platforma mobilă);

2.3 Sistemul de acționare - noțiuni generale

Sistemul de acționare este de fapt un subsistem ce face parte din sistemul de conducere al unui robot. De asemenea acesta este compus din mai multe componente, fiecare având un scop precis. Acestea sunt:

Actuatorul are scopul de a transforma energia primită de la o sursă primară în energie mecanică. Structura acestuia nu mai poate fi descumpusă în substructuri, deoarece există riscul de a se pierde capacitatea generare a mișcării.

Sistemul de transmisie este un subsistem al sistemului de acționare al cărui scop este transferul energiei mecanice la componentele cu ajutorul cărora mișcarea este posibilă. Acesta conține de asemenea mai multe componente cu scop specific.

Traductorul sau aparatul de măsurare are scopul de a transforma o mărime de natură fizică într-o altă mărime de natură fizică pentru utilizarea acesteia într-un sistem de automatizare, de reglare a mișcării.

Componentele sistemului de acționare sunt de fapt toate motoarele, convertoarele și aparatele de măsurare prin care se obține energia mecanică necesară deplasării robotului și dispozitivele suplimentare ce controlează acest transfer. O schemă generală a unui sistem de acționare se poate observa în următoarea figură:

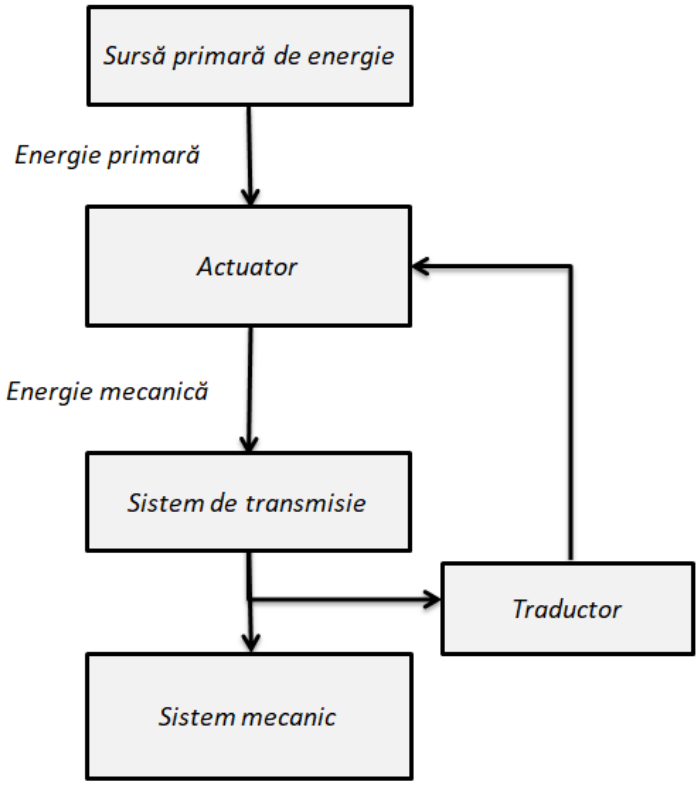


Figura 2.3 Sistemului de acționare al unui sistem informatic

În funcție de energia primară, clasa de operații ce trebuiesc efectuate, viteza de deplasare, spațiul de mișcare, precizia de poziționare și alte proprietăți specifice, sistemele de acționare se împart în următoarele categorii:

- sisteme de acționare electrice (cele mai răspândite la ora actuală);
- sisteme de acționare hidraulice (folosite în general la brațe robotice, unde deplasările în spațiu sunt limitate);
- sisteme de acționare pneumatice;
- sisteme de acționare hibride sau mixte (electropneumatice sau electrohidraulice);

2.3.1 Sisteme de acționare electrice

Sistemele de acționare electrice sunt cele mai răspândite datorită avantajelor ce le oferă, cum ar fi prețul redus al energiei primare și disponibilitatea acesteia, simplitatea racordării elementelor la rețea, fiabilitatea motoarelor electrice, compatibiliatea cu celelalte sisteme și altele. Acesta vine totuși cu dezanvantajul necesității unor mecanisme suplimentare, de exemplu de adaptare a vitezei, de control al inerției și a altora, acestea depinzând de operațiile ce trebuiesc îndeplinite de către robot.

Componenta principală a unui sistem de acționare o reprezintă motorul. Motorul electric, definit și ca electromotor se clasifică în următoarele:

- motoarele electrice de curent continuu;
- motoarele electrice de curent alternativ;

Un motor electric transformă energia electrică în energie mecanică. Pierderile de energie (electrică și mecanică) care au loc în timpul acestui proces se transformă în caldură. Electromotorul este format din *stator* și *rotor*. Statorul reprezintă partea fixă a motorului și este reprezentată de carcasa acestuia, bornele de colectare a curentului, armătura magnetică statorică și înfășurarea statorică. Rotorul reprezintă partea mobilă a motorului, formată dintr-un ax și o armătură magnetică rotorică. Între cele două componente există o porțiune de aer, denumită *intrefier*, ce permite mișcarea rotorului. Dimensiunea intrefierului este un indicator important în stabilirea performanței motorului.

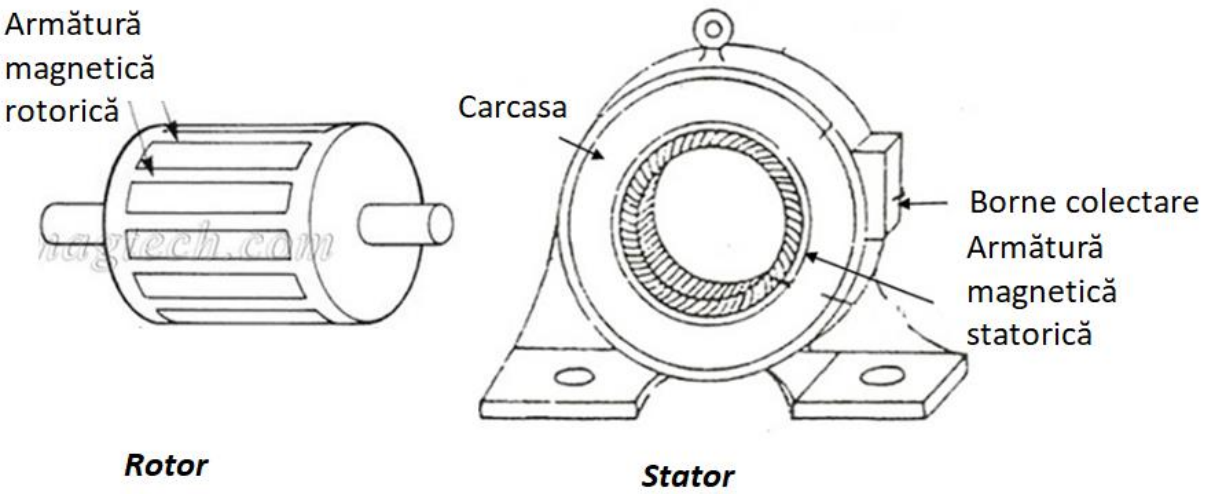


Figura 2.4 Structura motorului electric

Motorul electric de curent continuu a fost descoperit de Zénobe Gramme în anul 1873. Acesta are pe stator poli magnetici și bobine polare concentrate ce crează un câmp magnetic de excitație. Pe axul rotorului este situatut un colector, cu rolul de a schimba sensul curentului în înfășurarea rotorică pentru a exercita constant o forță față de rotor, provocând rotirea acestuia. Turația motorului este invers proporționala cu câmpul magnetic de excitație și direct proporționala cu rotația rotorului. Cuplul dezvoltat de motor este direct proporțional cu turația și curentul electric aplicat.

În funcție de modul înfășurării de excitație, motoarele electrice de curent continuu se clasifică:

- motor cu excitație independentă (înfășurarea statorică și cea rotorică sunt conectate la surse de tensiune separate);
- motor cu excitație paralelă (înfășurarea statorică și cea rotorică sunt conectate în paralel la aceiași sursă de tensiune);
- motor cu excitație serie (înfășurarea statorică și cea rotorică sunt legate în serie la aceeași sursă de tensiune);
- motor cu excitație mixtă;

Motorul electric de curent alternativ a fost descoperit de Nikola Tesla în anul 1882. Acestea funcționează pe baza principiului magnetic învârtitor. Există două tipuri de motoare de curent alternativ: sincrone și asincrone sau cu inducție, fiecare fiind monofazate și trifazate. Datorită eficienței lor, motoarele trifazate cu inducție sunt mai des folosite.

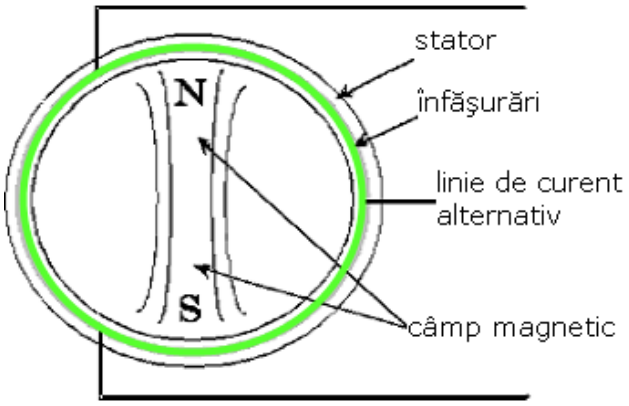


Figura 2.5 Structura motorului electric alternativ

Motorul sincron trifazat se diferențiază prin faptul că turația motorului este egală cu turația câmpului magnetic învârtitor. Acestea sunt folosite pentru îndeplinirea unor obiective pentru care este nevoie de o putere mare acționare.

Un alt tip de motor sincron și răspândit este motorul pas cu pas. Acesta se diferențiază prin faptul că rotirea rotorului se face dintr-un pol al statorului în altul, de aici și denumirea. Acesta ofera o precizie ridicată, prin urmare fiind folosit de exemplu la hard discuri.

În cazul motoarelor asincrone sau cele cu inducție circuitului magnetic îi sunt asociate două mai multe circuite ce se deplasează în raport unul față de celalalt, provocând fenomenul de inducție electromagnetică prin care se face transferul de energie de la partea fixă la cea mobilă. În acest caz, viteza de rotație este diferită de viteza de rotație a rotorului, de aici și numele de asincron. Turația nu depinde de sarcina ce trebuie îndeplinită.

2.3.2 Sisteme de acționare hidraulice

Sistemele de acționare hidraulice au fost primele care au apărut, acestea fiind folosite în general la brațele robotice deoarece acestea permit dezvoltarea unei puteri mari. Dezavantajele unui asemenea sistem constau în spațiul de mișcare restrâns și precizie de poziționare redusă. În cadrul acestor sisteme energia hidraulică este transformată în energie mecanică (de mișcare liniară sau de rotație).

2.3.3 Sisteme de acționare pneumatice

Sistemele de acționare pneumatice la fel ca cele hidraulice transformă energia fluidelor in energie mecanică de rotație. În funcție de cum energia pneumatică este transformată, motoarele pneumatice se împart în:

- motoare pneumostatice (volumice) unde energia mecanică este obținută prin modificarea permanentă a volumului fluidelor;
- motoare pneumodinamice unde energia pneumostatică a fluidului folosit e transformată în energie cinetică, apoi în energie mecanică;

2.4 Sistemul senzorial – noțiuni generale

În funcție de sarcinile pe care trebuie să le îndeplinească, un robot trebuie să fie capabil să execute un număr larg de mișcări și să nu fie limitat de condițiile știute ale mediului extern în care se află. De asemenea trebuie să fie capabil să-și modifice caracteristice funcționale o dată cu modificarea factorilor externi în spațiul în care acționează. Cu alte cuvinte, un robot trebuie să se fie autoadaptiv și să își modifice proprietățile de mișcare (viteză, accelerație, sens de deplasare etc.) în concordanță cu modificarea mediului. Pentru a putea fi adaptativ, un robot trebuie să aibă elemente prin care să ”simtă” mediul extern, să măsoare diferite caracteristici ale acestuia. Totalitatea elementelor prin care robotul e capabil să procure informații din mediul extern formează sistemul senzorial, iar elementul principal al acestui sistem sunt senzorii.

Sistemul senzorial al unui robot depinde în mod direct de sarcinile pe care acesta trebuie să le îndeplinească. Senzorii pot oferi informații despre parametrii intrinseci ai robotului, de exemplu viteza de deplasare, sensul de mers, accelerația acestuia sau pot defini caracteristici ale mediului extern și informații despre alte obiecte din vecinătatea spațiului de funcționare. Astfel, senzorii se pot clasifica din mai multe puncte de vedere, de la obiectul despre care oferă informații până la parametrii măsurați.

O primă clasificare a senzorilor ar fi:

- Senzori interni sau prioceptivi, ce oferă informații despre diferite componente ale robotului;
- Senzori externi sau exteroceptivi, ce furnizează date despre mediul extern și obiectele din mediul extern;

O altă clasificare a acestora se poate face în funcție de distanța robotului față de obiectul măsurat:

- Senzori de contact, ce măsoară de exemplu presiunea dintre obiect și dispozitivul de prehensiune;
- Senzori de proximitate, cum ar fi senzorii optici sau de temperatură;
- Senzorii de distanță, cum ar fi camera video sau senzorii acustici;

Și caracteristicile măsurate:

- Senzori pentru determinarea formelor geometrice;
- Senzori pentru determinarea proprietăților fizice;
- Senzori pentru determinarea proprietăților chimice;

Senzorii dispun de un receptor prin care preiau semnalul și îl transmit mai departe. Există însă și senzori care pe lângă receptor dispun și de emițător, cu ajutorul căruia induc o stare mediului extern sau obiectelor externe și preiau prin receptor răspunsul acestora la semnalul indus. Aceștia se clasifică:

- Senzori activi, cu receptor și emițător;
- Senzori pasivi, doar cu receptor;

O caracteristică importantă al senzorilor este tipul semnalului pe care aceștia îl pot transmite. În ziua de azi, majoritatea senzorilor sunt capabili să transmită semnal atât analogic cât și digital. O schemă orientativă a sistemului senzorial se poate observa în următoarea figură:

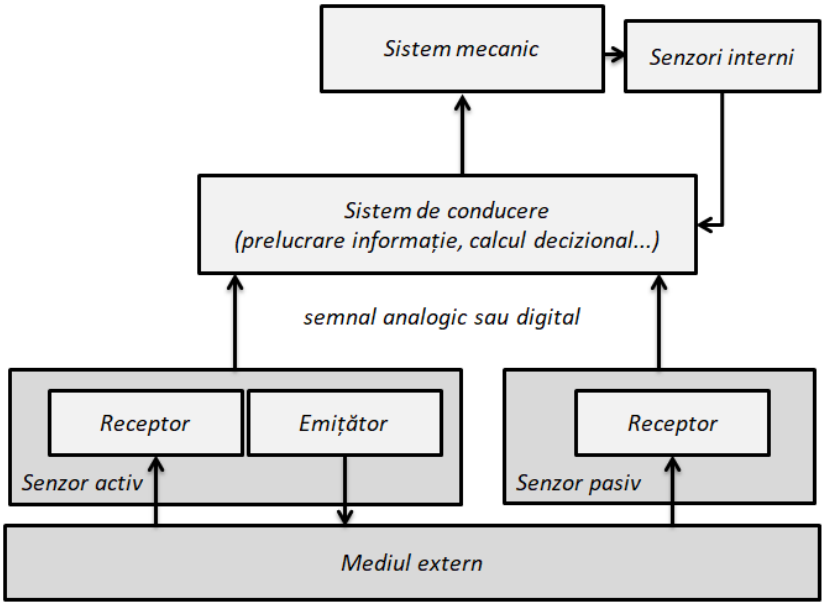


Figura 2.6 Sistemul senzorial al unui sistem informatic

Controlul robotului este direct influențat de informațiile furnizate de către senzori. Astfel, execuția unei operații declanșată de sistemul de calcul și comandă, în urma informațiilor furnizate de către sistemul senzorial, se împarte pe cinci nivele:

1) Nivelul controlului secvențial, ce corespunde controlului secvențial cu senzori tip ”0-1” sau ”DA-NU”;
2) Nivelul controlului înainte, comanda fiind generată de către sistem în urma primirii informației despre sarcina necesară pentru a îndeplini operația;

3) Nivelul controlului senzorial cu buclă închisă, comanda fiind generată continuu de către sistem în urma primirii informației atât despre sarcina necesară, cât și despre modificarea proprietăților robotului și al stării mediului extern;
4) Nivelul controlului adaptiv, unde informația senzorială este folosită pentru reglarea parametrilor sistemului, cu scopul adaptării sau optimizării;
5) Nivelul controlului autonom, unde robotul îndeplinește funcții ce țin de inteligența artificială. Acesta reprezintă scopul final al unui sistem de control bazat pe senzori.

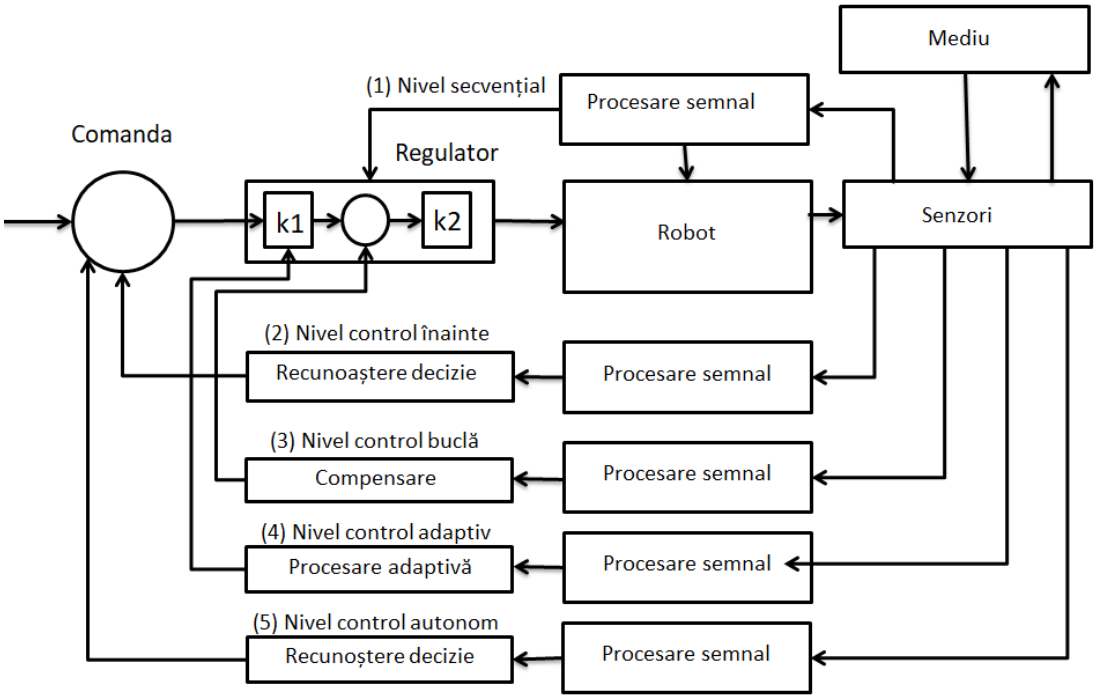


Figura 2.7 Nivelele controlului bazat pe informațiile generate de sistemul senzorial

Scopul sistemului senzorial este acela de a culege informații cât mai precise despre robot și mediul înconjurator și de a le transmite sistemului de calcul și decizional. Acest scop este greu de îndeplinit prin folosirea unui singur senzor sau senzori de acelați tip, așadar astăzi roboții sunt echipați cu sisteme de senzori complexe ce asigură o acuratețe ridicată privind informația transmisă. Integrarea unui sistem senzorial aduce cu sine totodată complexitate mai ridicată în calculul deciziilor ce trebuiesc luate.

2.5 Sistemul de comandă – noțiuni generale

Pentru a putea fi considerat autonom, un robot trebuie să aibă un sistem de comandă, prin intermediul căruia prelucrează datele primite de la sistemul senzorial și în urma acestei acțiuni ia o decizie ce o trasmite către elementele de execuție din cadrul celorlalte sisteme, în general cel mecanic. Aceste funcții sunt îndeplinite în general de un singur element, un calculator minimal integrat, definit microcontroller.

Microcontrollerul este de fapt un microcircuit care are o unitate centrala de calcul (CPU), o memorie(RAM, ROM, EEPROM), blocuri analogice, interfețe de comunicație, porturi de intrare/ieșire, oscilator ce îi permit intercațiunea cu mediul exterior. Fiind stocate în memorie, instrucțiunile se executa la o la o viteză mare.

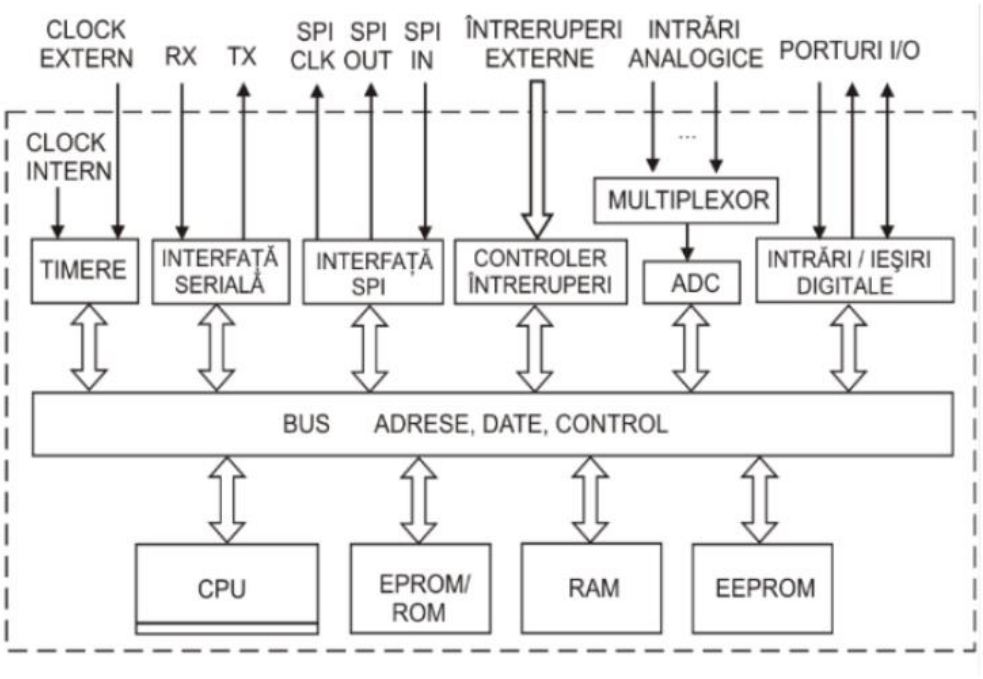


Figura 2.8 Arhitectura microcontroller-ului

2.5.1 Memoria

Informațiile, fie ele instrucțiuni, date de intrare, date de ieșire, rezultatele finale, rezultatele intermediare, variabilele și așa mai departe sunt păstrate în memorie. În

funcție de informațiile stocate, memoria trebuie să fie atât permanentă sau nevolatilă (informațiile sunt păstrate în memorie în cazul opririi și repornirii sistemului) cât și temporare sau volatile. Instrucțiunile care controlează funcționarea sistemului sunt păstrate în memoria nevolatilă, iar variabilele și rezultatele intermediare sunt păstrate în memoria volatilă, unde atât citirea cât și scrierea trebuie să se facă rapid.

Memoria RAM (Random Access Memory) este o memorie volatilă. În general, un microrontroller dispune de o cantitate mică de memorie RAM, datorită costurilor mari de implementare. Ea există sub formă de registre sau este implementată ca atare.

Memoria ROM (Read Only Memory) este o memorie nevolatilă utilizată la stocarea programelor. Unitatea centrală nu poate modifica informațiile din această memorie, doar citi.

Memoria PROM (Programmable Read Only Memory) este o memorie ROM programată de utilizator. Acest tip de memorie se împarte în mai multe categorii, în funcție de posibilitatea de ștergere:

- EPROM (Errasable PROM) – informațiile din acest tip de memorie se pot șterge prin expunere la raze ultraviolete. În acest caz ștergerea este totală, neputând alege fragmente pe care să le ștergi. Numărul de ștergeri este limitat. O variantă la acest tip de memorie o reprezintă memoria Flash EPROM, care pune la dispoziție o cantitate mai mare de memorie nevolatilă și un număr de rescrieri mai ridicat (~10000).
- EEPROM (Electrically EPROM) – este ștearsă selectiv de unitatea centrală de calcul în timpul funcționării. Scrierea este lentă, iar ciclurile de programare și în acest caz sunt limitate. Și aici există varianta Flash EEPROM, care, la fel ca în cazul Flash EPROM, dispune de o cantitate mai mare de memorie și cicluri de rescriere mai mari.
- OTP (One Time Programmable PROM) – este o memorie EPROM de fapt, încapsulată într-un material de plastic fără posibilitate de ștergere. Aplicațiile unde se poate folosi acest tip de memorie sunt limitate.

Există și memorii RAM nevolatile, cunoscute sub denumirea NOVRAM (Non volatile RAM). Implementarea acestora este mai costisitoare, dar timpul de rescriere este mai rapid și nelimitat.

În cadrul microcontroller-lelor, memoria se împarte în două mari categorii: memoria de date și memoria program. Memoria de date este volatilă, de tip RAM. Celulele acestei memorii sunt de fapt regiștrii, care diferă în funcție de numărul de biți ce pot fi stocați pe ele. Memoria program este de tip ROM, nevolatilă deci. Ea poate fi internă care este direct incorporată pe chip, dar poate fi și extinsă extern.

Microcontroller-ele ce nu au memorie program internă îndeplinesc doar funcția CPU-ului, ele lucrând în regim de microprocesor.

2.5.2 Unitatea centrală de procesare (CPU)

Unul din elementele cele mai importante din arhitectura unui microcontroller este unitatea centrala de calcul(CPU). Aceasta este formată dintr-o unitate de comandă(UC) și o unitate aritmetică logică, abreviată UAL. Aceasta primește comenzile de la unitatea de comandă și transformă un șir de date de intrare în alt șir de date de ieșire, fiind stocate în general în regiștrii interni. Prelucrările de date se numesc operații și se fac asupra unor șiruri binare cu lungime varaibilă de biți. Operațiile aplicate asupra datelor sunt operații logice și aritmetice.

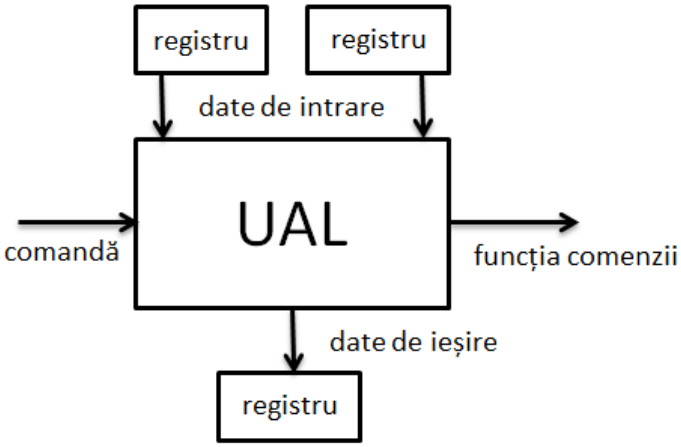


Figura 2.9 Arhitectura unității aritmetice logice

CPU-ul conține un set de regiștrii folosiți pentru informația volatilă ca variabile sau rezultate intermediare. Aceștia diferă În funcție de tipul microcontroller-ului, dar există și regiștrii comuni, ca:

- A - registrul acumulator, folosit la stocarea unui operand sau rezultatul unei operații logice sau aritmetice.
- PC – registrul numărător de program, stochează adresa următoarei instrucțiuni ce urmează a fi executate. Incrementarea se face automat, o dată cu execuția unei instrucțiuni.
- SP – registrul indicator de stivă, stochează adresa unei memorii de date ce conține informații temporare necesare în execuția programului(stivă).

Modificarea acestei memorii se face prin instrucțiuni specifice de PUSH/POP.

Operanzii și rezultatul operațiilor pot fi stocați atât intern în regiștrii cât și extern în memorie. Sincronizarea CPU-ului cu restul elementelor se face prin intrările și ieșirile de sincronizare. Instrucțiunile ce trebuiesc îndeplinite sunt întotdeauna păstrate în memoria program, iar generarea adreselor pentru memoria de date și program, precum și generarea semnalelor de comandă sunt efectuate de unitatea de comandă.

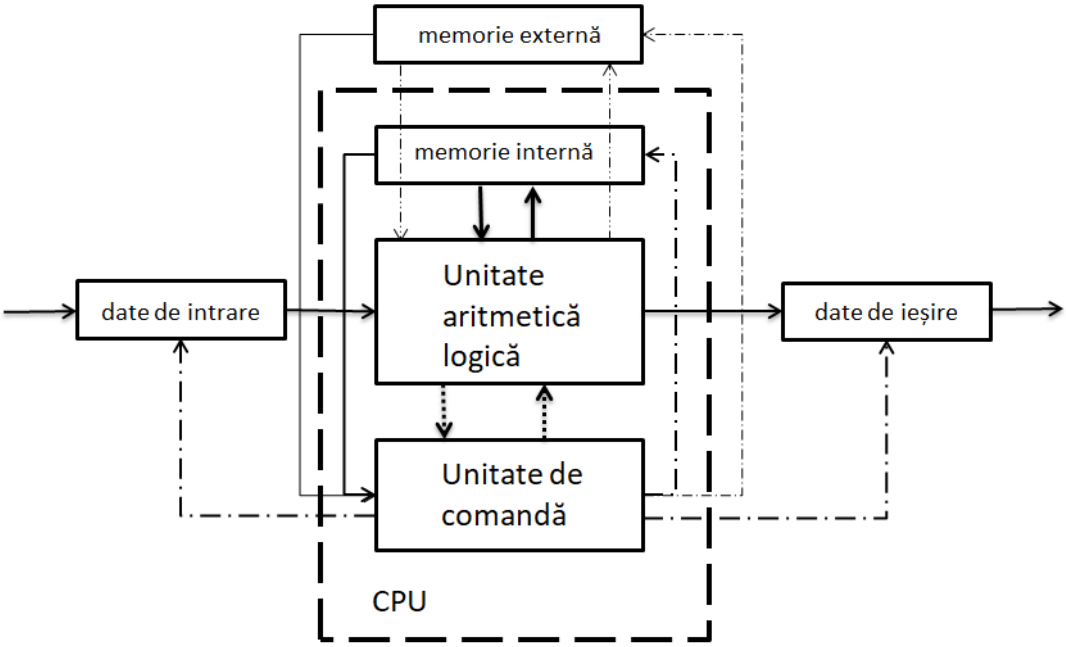


Figura 2.10 Arhitectura CPU

Execuția unei instrucțiuni se desfășoară în general în trei faze, care constituie un ciclu:

1) instrucțiunea este citită de unitatea de comandă din memoria program;
2) instrucțiunea este decodificată de UC, care transmite secvența de comenzi către UAL;
3) UAL-ul execută comenzile, iar UC-ul își modifică în paralel secvența de comenzi în funcție de rezultatul intermediar primit de la UAL. Dacă este necesară citirea datelor din exterior, acestea sunt executate aici;

Procesorul comunică cu celelalte emelente din cadrul microcontroller-ului prin grupuri de conexiuni ce transferă semnale comune, denumite magistrale. În funcție de tipul semnalelor, avem magistrale de date și magistrale de adrese. Magistralele de date sunt bidirecționale și prin ele se citesc sau scriu valori existente la o anumită adresă. Magistralele de adrese transmit semnale într-un singur sens și conțin adresele la care

se găsesc resursele sistemului. Magistralele stabilesc numărul biților ce pot fi transmiși sau recepționați de procesor într-un singur ciclu. Arhitectura procesoarelor diferă în funcție de modul de execuție al unui ciclu, al modului de comunicare sau al modului în care informațiile sunt stocate în memorie.

Arhitectura de tip CISC(Complex Instruction Set Computer) a fost prima dată implementată de IBM în jurul anilor 1970. Procesoarele cu această arhitectură pot include în aceeași instrucțiune mai multe operații cu complexitate redusă, ca de exemplu citirea din memorie a datelor, scrierea în memorie a datelor, modificarea variabilelor. Un set de operații ce însumează o instrucțiune poate ajunge la 80. Această arhitectură era în general folosită pe calculatoarele personale.

Arhitectura de tip RISC(Reduced Inscruction Set Computer) a apărut ca efect al instrucțiunilor complexe ce impactau costul de implementare și performanța procesoarelor cu arhitectura CISC. Aceasta a apărut la sfarșitul anilor 1970 și presupune ca fiecare instrucțiune să fie cât mai mult simplificată. Astfel, performanța procesoarelor crește, deoarece până și operațiile care se referă la citirea sau scrierea în memorie sunt intrucțiuni distincte. Această arhitectură este cunoscută și sub numele de ”arhitectură load/store”, fiind prezentă pe aproximativ 99% din procesoare la începutul anului 2018.

Arhitectura de tip Harvard se caracterizează prin faptul că memoria folosită pentru instrucțiuni și date este separată. Instrucțiunile sunt stocate în memoria program, iar datele sunt stocate în memoria de date. De asemenea, magistralele de date și adrese sunt distincte. Numele provide de la ”Harvard Mark I”, un sistem de calcul ce stoca instrucțiunile pe o bandă perforată de 24 de biți.

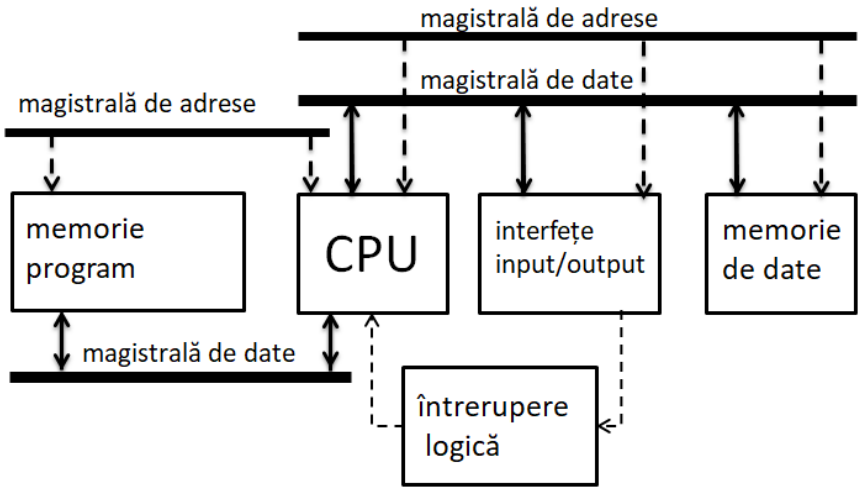


Figura 2.11 Arhitectură CPU tip ”Harvard”

Principalul avantaj al acestei arhitecturi este faptul că permite un anumit grad de paralelism, astfel, în timpul fazei de execuție a unei instrucțiuni, se poate citi următoarea instrucțiune. Faza I și faza III a două cicluri se execută în paralel. Procesoarele cu această arhitectură sunt deci mai rapide. Un alt avantaj, prin prisma paralelismului, este faptul că timpul de execuție al instrucțiunilor este același.

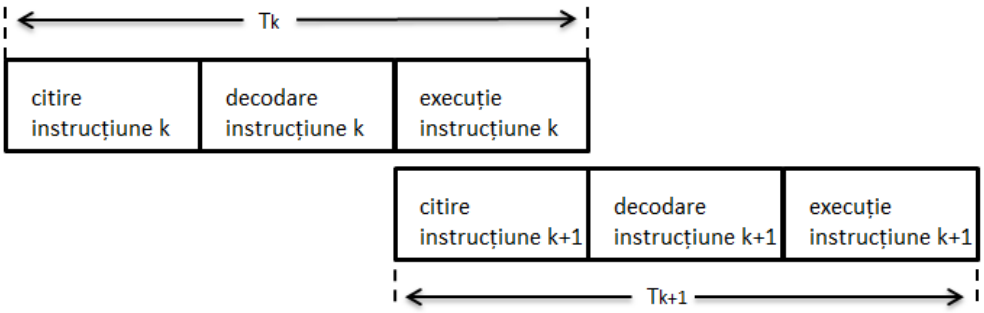


Figura 2.12 Paralelism în arhitectură CPU tip "Harvard"

Arhitectura de tip von Neumann se caracterizează prin faptul că memoria unde sunt stocate instrucțiunile și datele este aceeași. De asemenea, magistralele de date și adrese sunt comune. Denumirea acestei arhitecturi vine de la renumitul John von Neumann, matematician și pionier al informaticii în anii 1940. Deși performanțele procesoarelor s-au îmbunătățit foarte mult comparativ cu anii 1940, principiile arhitecturii von Neumann sunt baza a majorității procesoarelor și în ziua de astăzi, datorită simplității de implementare. Aceasta mai este cunoscută și sub denumirea de arhitectură Princeton.

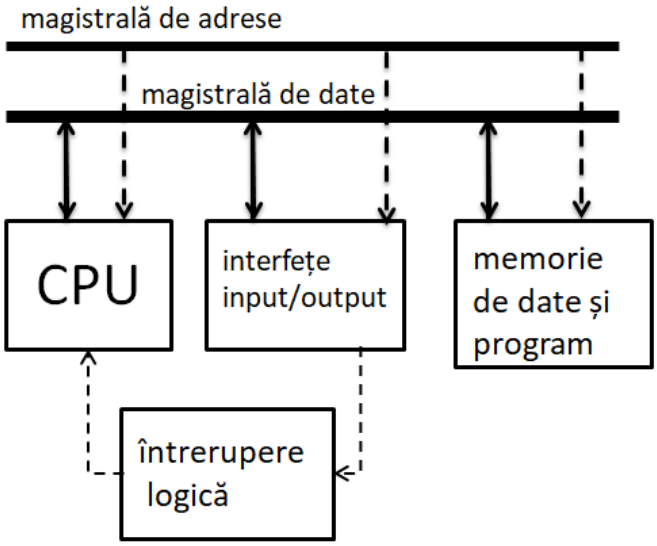


Figura 2.13 Arhitectură CPU tip "von Neumann"

Principalul avantaj al acestei arhitecturi constă în folosirea rezultatelor returnate de UAL ca instrucțiuni, acest fapt fiind posibil prin folosirea comună a memoriei și magistralelor.

2.5.3 Oscilatorul

Oscilatorul este componenta din cadrul microcontroller-ului care furnizează semnalul de sincronizare pentru procesor. Acesta generează ”clock-ul” microcontroller-ului care determină viteza de lucru a acestuia. Principalele tipuri de oscilatoare întalnite sunt oscilatoarele cu cuarț și oscilatoarele RC.

Oscilatoarul cu cuarț se află într-o carcasă metalică cu doi pini, care oscilează la o anumită frecvență. Pinii sunt legați la două condensatoare. Aceste elemente se pot găsii toate încapsulate, iar oscilatorul respectiv mai poartă denumirea și de oscilator piezoceramic.

O variantă mai accesibilă o reprezintă oscilatorul RC. Acesta se diferențiază prin faptul că frecvența de rezonanță depinde de tensiunea alimentării, rezistorul R, condensatorul C și temperatura de lucru. Acest tip de oscilator are dezavantajul că nu oferă o precizie ridicată.

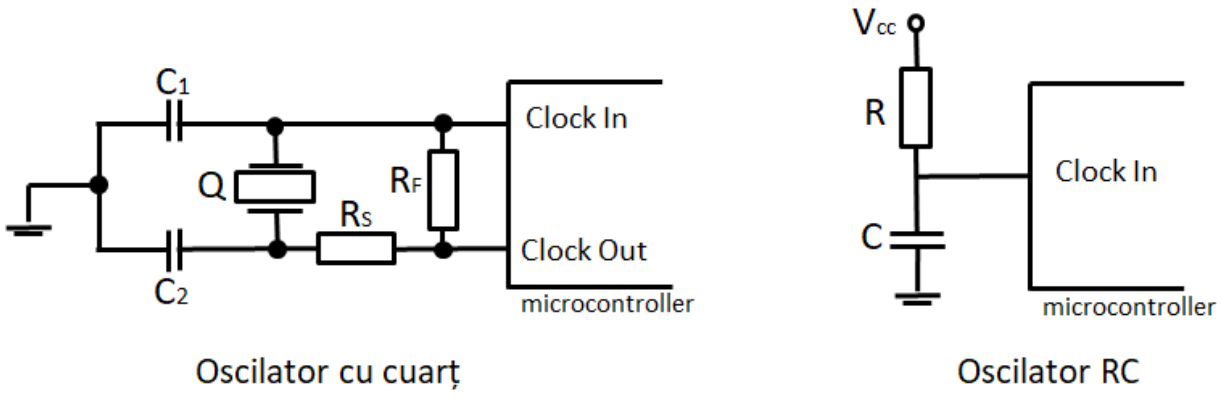


Figura 2.14 Arhitectură oscilator cu cuarț și RC

Un ciclu de instrucțiune este diferit de un ciclu de ceas, fiecare fază din cadrul instrucțiunii putând avea mai multe cicluri de ceas.

2.5.4 Circuitul de reinițializare

Această componentă aduce microcontroller-ul într-o stare definită ca inițială.Este reprezentată în general de un buton (RESET) și induce încărcarea memoriei volatile cu valori predefinite și încărcarea contorului program cu o adresă predefinită.

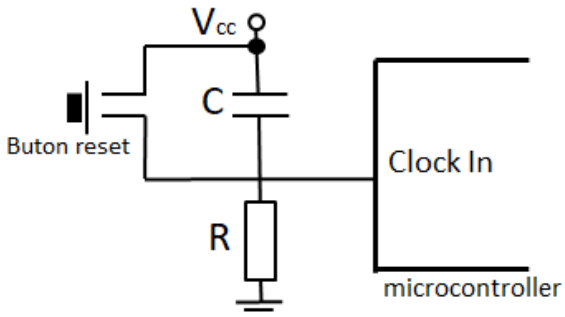


Figura 2.15 Circuit reinițializare în oscilator RC

Inițializarea se poate declanșa prin scăderea tensiunii de alimentare sub o anumită valoare, modificarea stării unui pin de resetare sau prin depășirea unui anumit interval de timp.

2.5.5 Circuitul de monitorizare

Cunoscut sub numele de watch-dog, este un contor care, la depășirea unei valori induce reinițializarea microcntroller-ului. Scopul lui este recuperarea din stări de blocaj.

2.5.6 Controller-ul de întreruperi

Pentru ca microcontroller-ul să fie în sincronizare cu evenimentele produse la nivelul dispozitivelor încorporate sau externe, este nevoie de o întrerupere. La producerea unui asemenea eveniment, un controller-ul de întrerupere generează o cerere către procesor, care execută o serie de instrucțiuni încorporate definite în acest scop. Execuția programului principal este întreruptă, dar registrere interne și externe, precum și memoria volatilă reține informațiile conținute la momentul întreruperii. Astfel, se asigură reluarea și executarea programului în continuare fără modificări de rezultate datorate întreruperii.

2.5.7 Alte periferice din arhitectura microcontroller-ului

Pe lângă cele menționate, mai există componente ce îndeplinesc o funcție specifică în cadrul unui microcontroller. Dintre ele, amintim:

- *dispozitive I/O* se găsesc în arhitectura microcontroller-ului într-o varietate mare, fiecare dintre ele îndeplinind o funcție specifică, ca de exemplu transfer serial sau paralel de date, funcții de numărare, generare de impulsuri și altele. Acestea sunt ”văzute” de microcontroller ca porturi, iar maparea adreselor se face fie în spațiul de memorie, fie într-un spațiu propriu alocat.
- *convertor analogic/digital* are principalul scop de modificare a semnalului analogic de intrare în semnal digital.
- *convertor digital/analogic* se găsesc mai rar în arhitectura unui microcontroller, convertirea semnalului fiind posibilă prin folosirea unui timer în mod PWM și integrarea acestuia în exterior prin folosirea unui circuit RC. Un semnal PWM (Pulse Width Modulation) este în general folosit la motoarele cu curent continuu.
- *controller DMA(Direct Memory Access)* are scopul de a furniza acesul direct la memorie dispozitivelor externe prin porturi I/O, ocolind astfel CPU-ul în cazul operațiilor de memorie, cu scopul creșterii performanței.

CAPITOLUL 3: PROIECTAREA SISTEMULUI INFORMATIC

3.1 Obiectivele sistemului informatic

Acest proiect constă în construirea unui sistem autonom mobil, capabil să urmărească o linie neagră ce descrie un traseu tip labirint și memorează soluția optimă, după parcurgerea inițială a traseului.

Componentele au fost alese astfel încât raportul calitate/preț să fie optim. Principalele elemente din cadrul robotului sunt:

- placă dezvoltare tip Arduino UNO;
- două servomotoare acționate electric;
- driver motoare L298N;
- o bară cu opt senzori infraroși reflectivi;
- sursă de alimentare;
- întrerupător;
- șasiu;
- două roti bidirecționale acționate de cele două motoare;
- o roată multidirecțională tip ball caster;
- elemente pasive: diode,condensatoare, rezistoare, fire;
- fire de conectare;

O descriere generală a sistemului informatic se poate observa în următoarea figură:

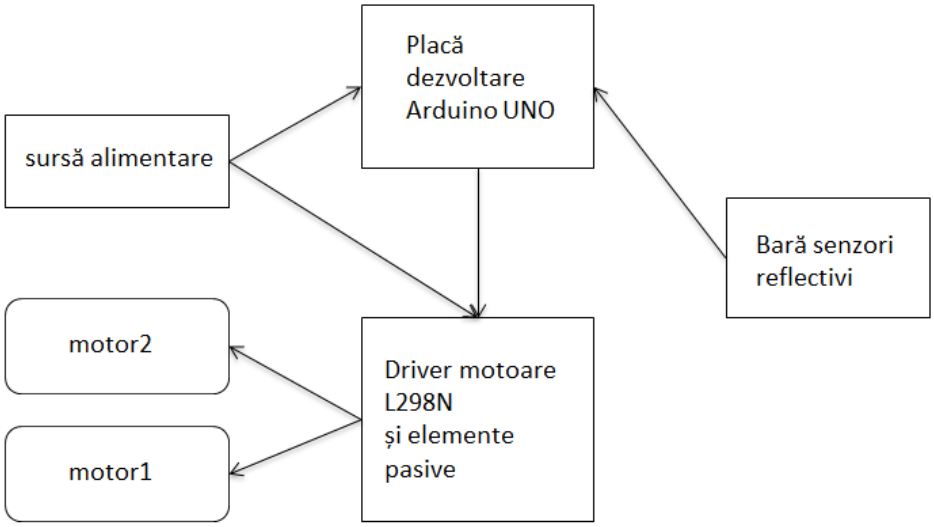


Figura 3.1 Schema generală a sistemului informatic prezentat

Am ales un dispozitiv Arduino UNO cu microcontroller ATMega 328P-PU pentru acest proiect pentru ca oferă destule intrări digitale și analogice pentru senzori și motoare. De asemenea funcționalitățile oferite sunt sufieciente pentru obiectivele ce trebuiesc îndeplinite de către robot. Motoarele în curent continuu în punte H permit robotului să se deplaseze înainte și înapoi, cu o viteză direct proporțională cu tensiunea aplicată asupra lor. Senzorii reflectivi oferă un mod de diferențiere între alb și negru. Ca sursă de alimentare, folosesc patru baterii de 1.5V fiecare și un întrerupător.

Senzorii trimit informații către microcontroller, care la rândul său controlează robotul corectând direcția de mers în funcție de informațiile de la senzori, trimițând semnal către driverul de motoare cu puterea ce trebuie transferată la motoare,

individual. Dacă robotul trebuie să vireze stânga, puterea transferată la motorul din stânga va fi mai mică decât cea transferată la motorul din dreapta și viceversa.

Robotul are următoarele obiective de îndeplinit:

i. să urmărească linia neagră ce delimitează traseul;
ii. să memoreze traseul și să rețină rutele blocate;
iii. să parcurgă a doua oară traseul folosind soluția optimă, unde rutele blocate sunt evitate.

3.2 Schema electrică a sistemului informatic

O schemă electrică generală a sistemului informatic prezentat cu legăturile între componente se poate observa în următoarea figură:

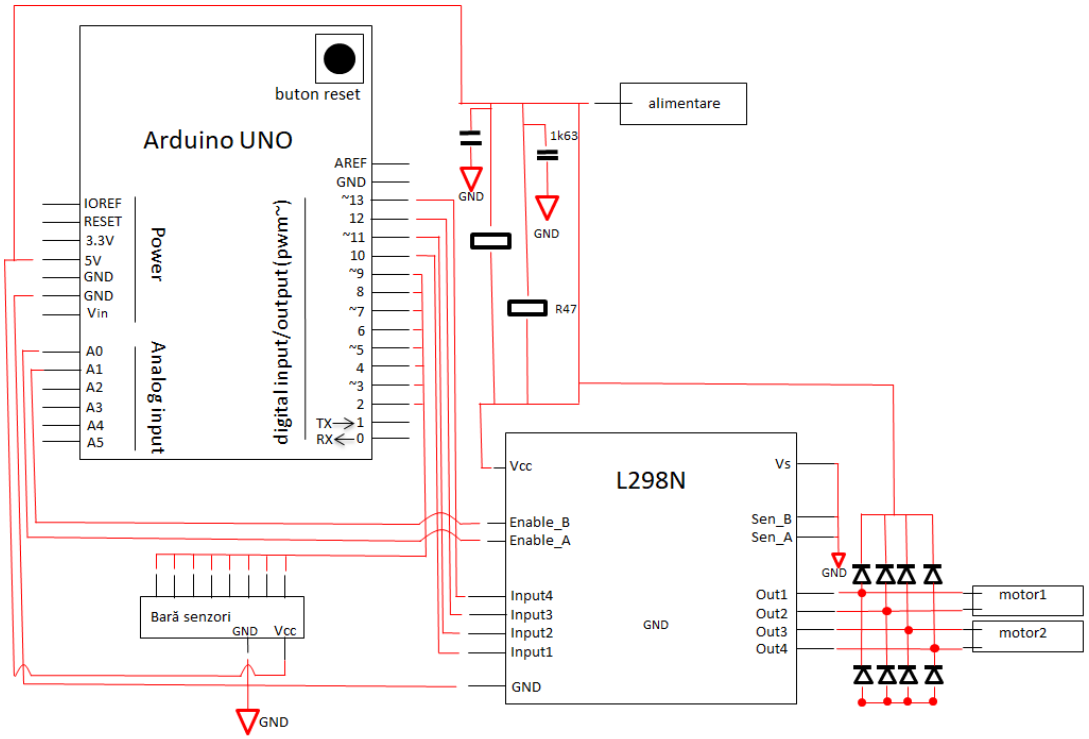


Figura 3.2 Schema electrică

CAPITOLUL 4: IMPLEMENTAREA SOLUȚIEI HARDWARE

4.1 Sistemul mecanic

În alegerea componentelor sistemului mecanic, am avut în vedere scopul principal ce trebuie îndeplinit de sistemul informatic: să se deplaseze înainte și să fie capabil să vireze. Așadar, sistemul mecanic este alcătuit din:

- un șasiu, pe care să poată fi amplasate piesele generatoare de mișcare;
- două roți bidirecționale;
- o roată de sprijin multidirecțională tip ball caster;
- șuruburi și alte elemente folosite pentru fixarea componentelor pe șasiu;

În următoarea figură se observă modul de amplasare a pieselor ce compun sistemul mecanic:

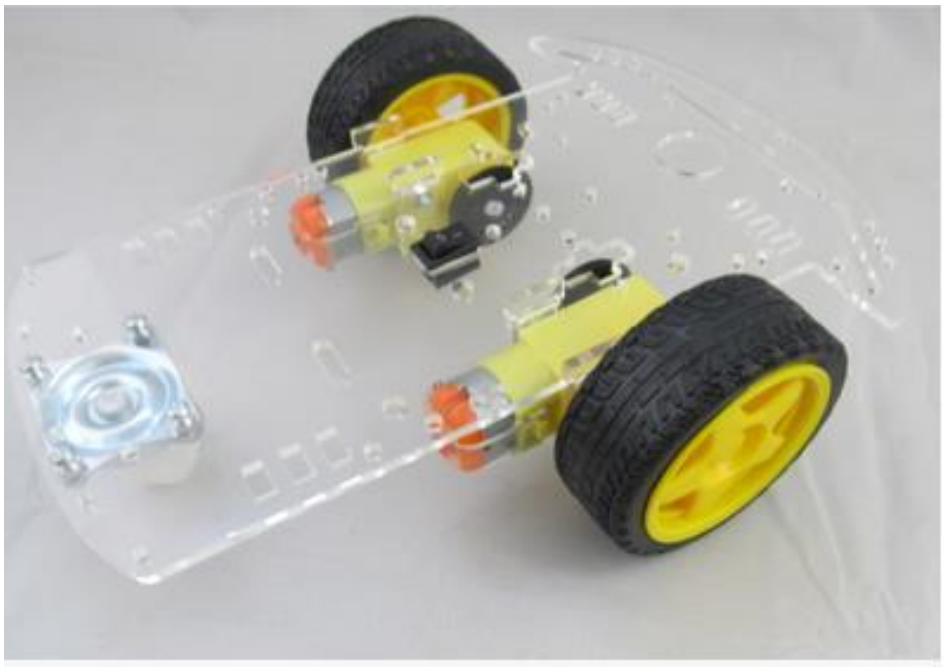


Figura 4.1 Sistemul mecanic

Cele două roți bidirecționale sunt atașate la două motoare, ele fiind elementele asupra cărora se acționează și face posibilă deplasarea. Ball caster-ul are rolul doar de echilibru și nu interferează cu direcționarea robotului, aceasta fiind posibilă prin transferul la roțile bidirecționale o viteză inegală.

4.2 Sistemul de acționare

Sistemul de acționare este format din:

- Sursă de alimentare cu energie electrică, formată din patru baterii de 1.5V și un întrerupător;
- Două motoare de curent continuu cu sistem reductor 1:48 tip Polulu, ce transformă energia electrică în energie mecanică;

Motorul de curent continuu au fost alegerea logică pentru acest proiect, având în vedere avantajele oferite de un astfel de motor. Acest tip de motor este alcătuit din:

- motorul propriu zis unde viteza este dată de tensiunea aplicată;
- un sistem reductor 1:48.

Reductorul este un sistem ce acționează ca o cutie de viteze. Acesta distribuie momentul motor la punțile motoare, oferind în același timp posibilitatea de modificare momentului motor.

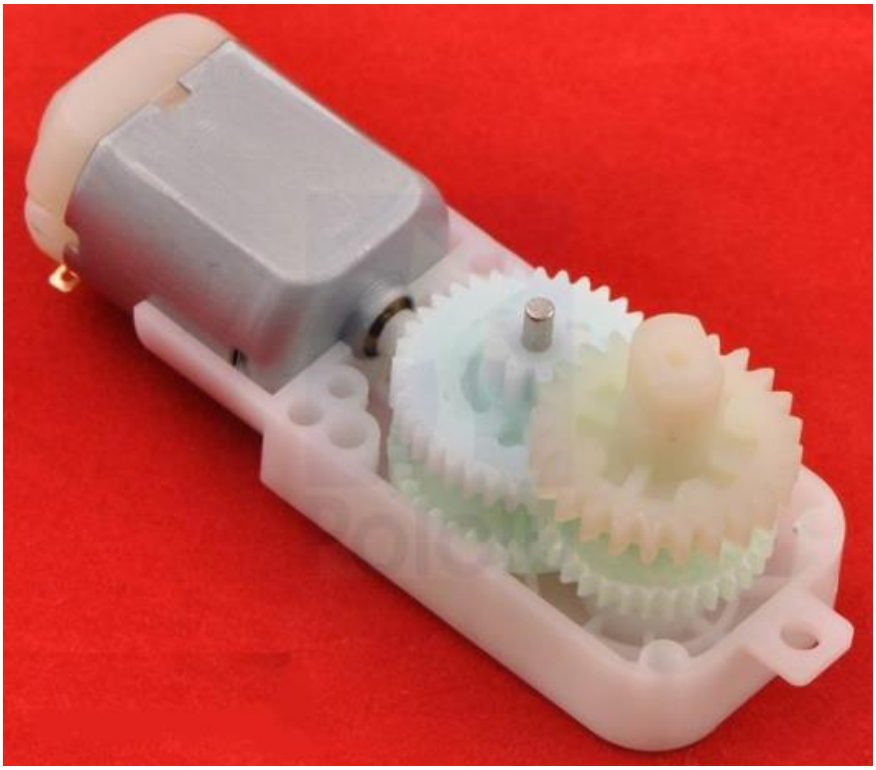


Figura 4.2 Motor cu sistemul reductor 1:48

În acest caz, ansamblul motor folosit este suficient pentru deplasarea unui robot de mici dimensiuni. Tensiunea posibilă aplicată este de 3V - 6V, sistemul motor având un circuit tip punte H. Acesta se diferențiază prin faptul că permite aplicarea tensiunii pe o sarcină în orice sens. Modul de funcționare al unui circuit electric în punte H se poate observa în următoarea figură:

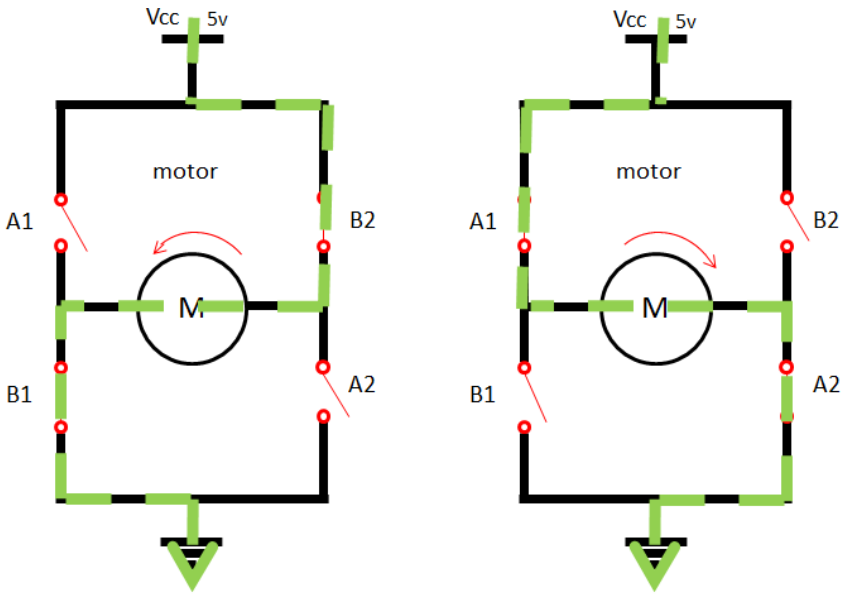


Figura 4.3 Circuit electric punte H

Folosind un circuit în punte H, robotul este capabil să ruleze înainte și înapoi, controlând cele patru întrerupătoare. Dacă întrerupătoarele A1 cu B1 sunt deschise și A2 cu B2 sunt închise și o tensiune este aplicată, se va produce un scurtcircuit. Viceversa este de asemenea valabilă (A2 cu B2 deschise și A1 cu B1 închise). Cele 4 întrerupătoare sunt tranzistoare bipolare. Un tranzistor bipolar este un dispozitiv semiconductor cu scopul de a transmite un semnal electric. Funcțional, un tranzistor într-un circuit electric se comportă ca un robinet într-o rețea sanitară: permite trecerea/stoparea curentului în circuit. Tranzistorul bipolar este format din emițător, bază și colector și funcționează în cusscesiune *npn* sau *pnp:*

Figura 4.4 Tranzistori bipolari

Specificații tehnice:

- tensiune de funcționare: 3V ~ 6V (valoare recomandată – 5V);
- viteză fără sarcină: 90 ± 10 rpm;
- curent încărcare: 190mA (maxim 250mA);
- rată reducere: 1:48

4.3 Sistemul senzorial

Pentru sistemul senzorial am ales să folosesc o bară de senzori tip QTR-8RC, modul senzorial ce poate fi folosit fie ca senzori de proximitate, fie ca senzori reflectivi.

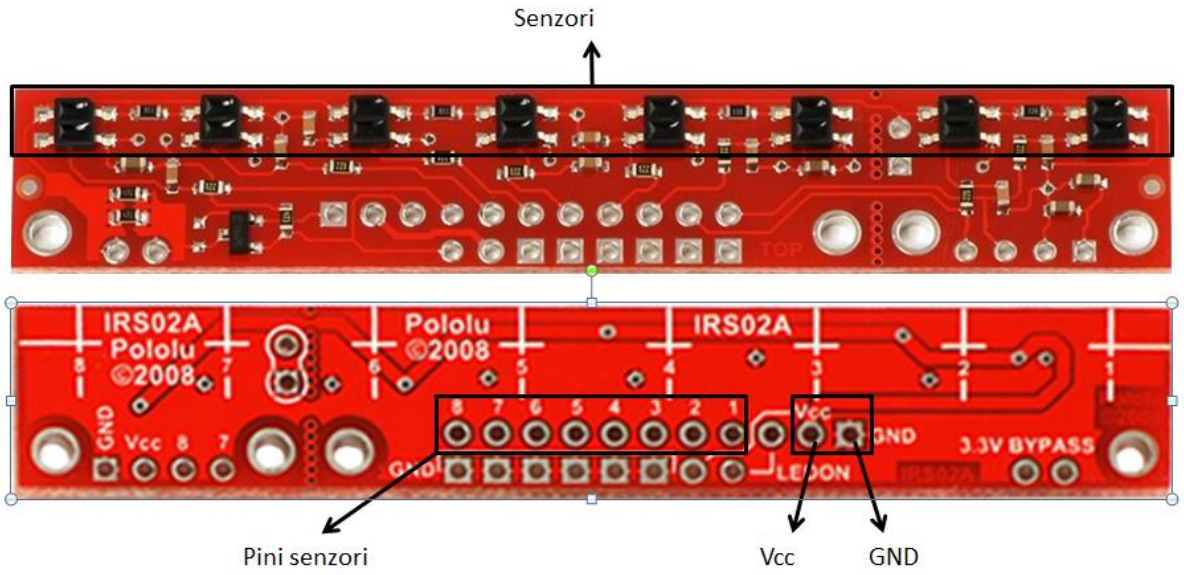


Figura 4.5 Bară senzori QTR-8RC

Modulul are în componență opt fototranzistori (perechi emițător - receptor) repartizați uniform la un anumit interval. Modul de măsurare al reflexiei pe un senzor se face în felul următor:

1) se aplică o tensiune pe pin-ul OUT al senzorului;
2) în momentul retragerii acelei tensiuni, reflexia se obține prin măsurarea timpului de ”descompunere” al acelei tensiuni;

Timpul de descompunere în cazul în care avem reflexie mare este de câteva microsecunde, iar în cazul în care avem reflexie mică, timpul de descompunere de de

ordinul milisecundelor. Timpul exact măsurat depinde de microcontroller. Această modalitate de măsurare a reflexiei are anumite avantaje, ca de exemplu nu e nevoie de un convertor analog-digital. De asemenea citirea senzorilor se poate face în paralel de către microcontroller. Schema electrică se poate observa în următoarea figură:

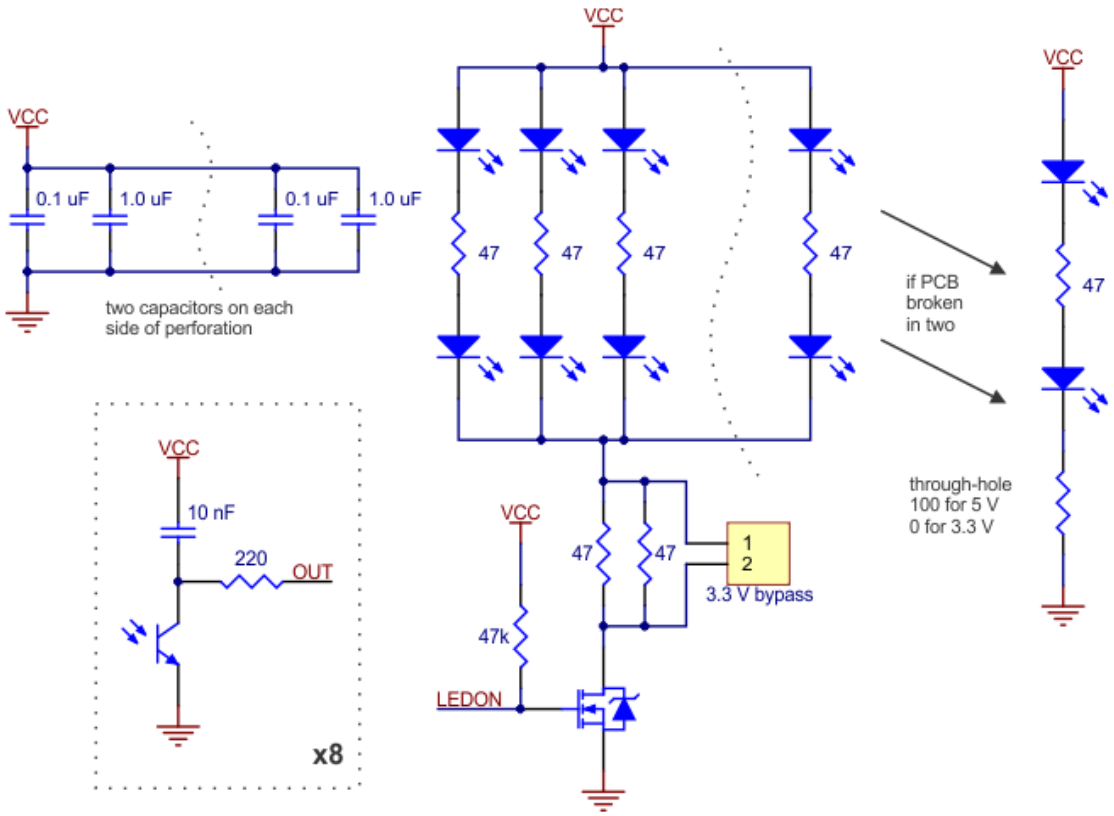


Figura 4.6 Schema electrică bară senzori QTR-8RC

Specificații tehnice:

- tensiune de funcționare: 3.3V ~ 5V;
- dimensiuni: 7.5cm X 1.3cm;
- curent alimentare: 100mA;
- distanță optimă de detectare: ~ 0.3cm;
- distanța maximă recomandată pentru detectare: ~ 0.9cm;
- greutate: ~ 0.3g;

Cei opt senzori sunt conectați la microcontroller prin pinii 2-9, după cum se vede în următoarea imagine, astfel:

- Arduino digital pin2 – QTR senzor pin8(fir culoare albastră)
- Arduino digital pin3 – QTR senzor pin7(fir culoare galbenă)
- Arduino digital pin4 – QTR senzor pin6(fir culoare verde)
- Arduino digital pin5 – QTR senzor pin5(fir culoare gri)
- Arduino digital pin6 – QTR senzor pin4(fir culoare mov)
- Arduino digital pin7 – QTR senzor pin3(fir culoare portocaliu)
- Arduino digital pin8 – QTR senzor pin2(fir culoare maro)
- Arduino digital pin9 – QTR senzor pin1(fir culoare azure)

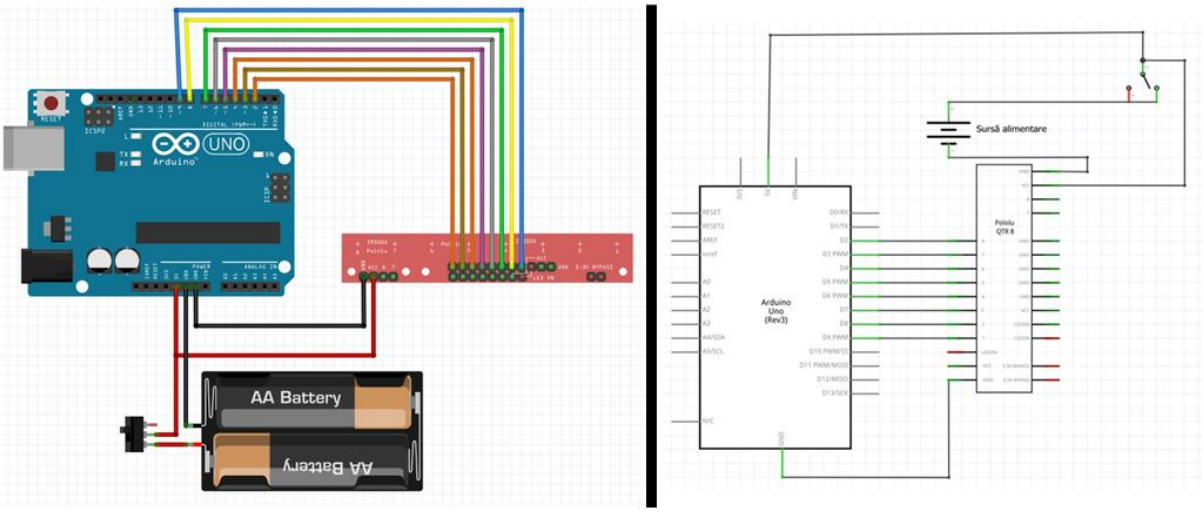


Figura 4.7 Conectare bară senzori QTR-8RC la Arduino UNO

Microcontroller-ul citește infomațiile primite de la fiecare senzor, sub forma:

- 0 – senzorul se află deasupra unei suprafețe albe;
- 2500 – senzorul se află deasupra unei suprafețe negre;

4.4 Sistemul decizional, comandă și control

Acest sistem este format din două componente:

1) Driver motor tip L298N;
2) Un dispozitiv Arduino UNO;

4.4.1 Driverul L298N

Funcția principală a driverului L298N este acela de transfer al unei anumite cantități de energie electrică la motor în funcție de comanda venită de la microcontroller. Cu alte cuvinte, prin driver-ul L298N setez viteza pe cele două motoare. Acest driver mai are totuși o funcție și anume de întrerupător temporar al fluxului de energie electrică. O descriere a acestui dispozitiv, cum sunt fizic conectați pinii în cadrul proiectului, se poate observa mai jos.

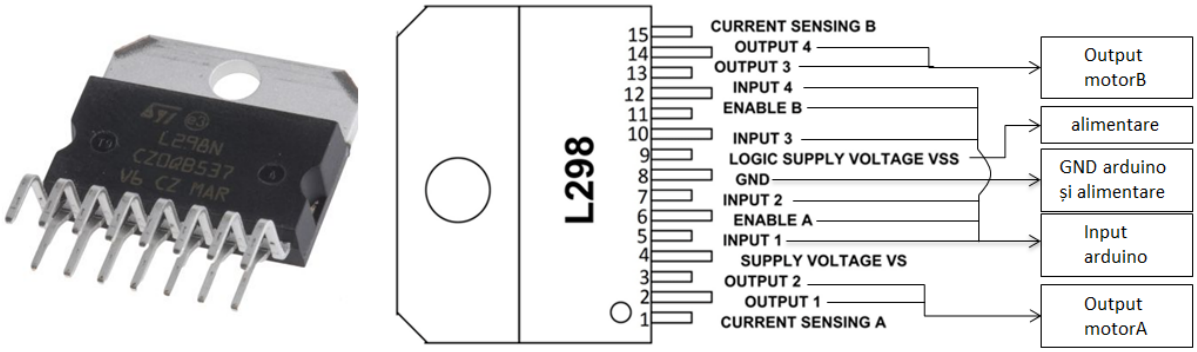


Figura 4.8 Driver L298N

Funcțiile sunt îndeplinite în cadrul driverului prin porți logice, care fizic, sunt tranzistori bipolari de tip NPN. Curentul circulă, în funcție de comanda primită, într-un sens sau altul, sau deloc în cazul în care se transmite comanda de OFF pe pinii EnableA și EnableB. O descriere al modului de funcționare și un exemplu cum direcția curentului este controlată, se poate observa mai jos.

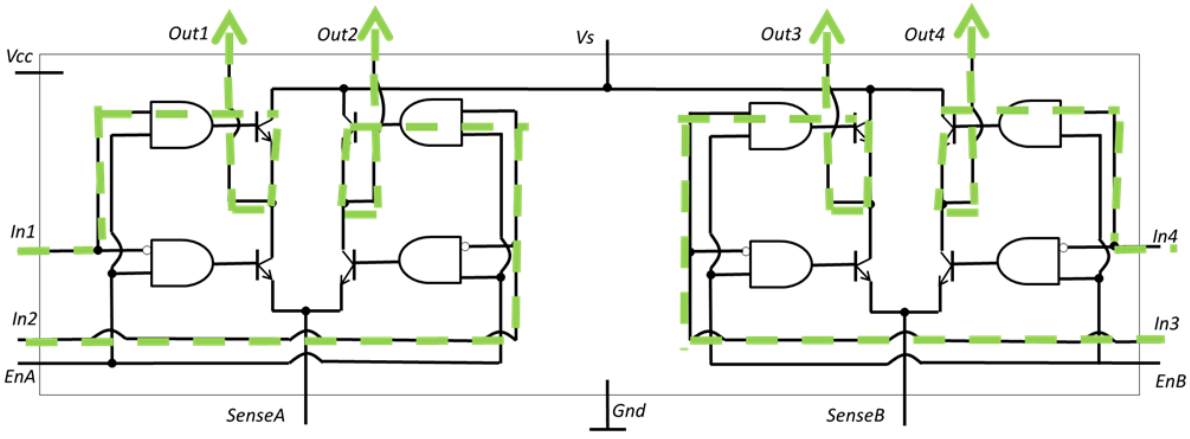


Figura 4.9 Mod funcționare L298N

Driver-ul este conectat cu cele două motoare prin pinii de out și cu microcontroller-ul cu pinii de in și de enable. În următoarea figură se poate observa modul de conectare, astfel:

- Arduino analog pin A0 – enableA (fir culoare verde);
- Arduino analog pin A1 – enableB (fir culoare albastru);
- Arduino digital pin10 – input pin motorA 1 (fir culoare portocaliu);
- Arduino digital pin12 – input pin motorA 2 (fir culoare galben);
- Arduino digital pin11 – input pin motorB 1 (fir culoare mov);
- Arduino digital pin13 – input pin motorB 2 (fir culoare roz);

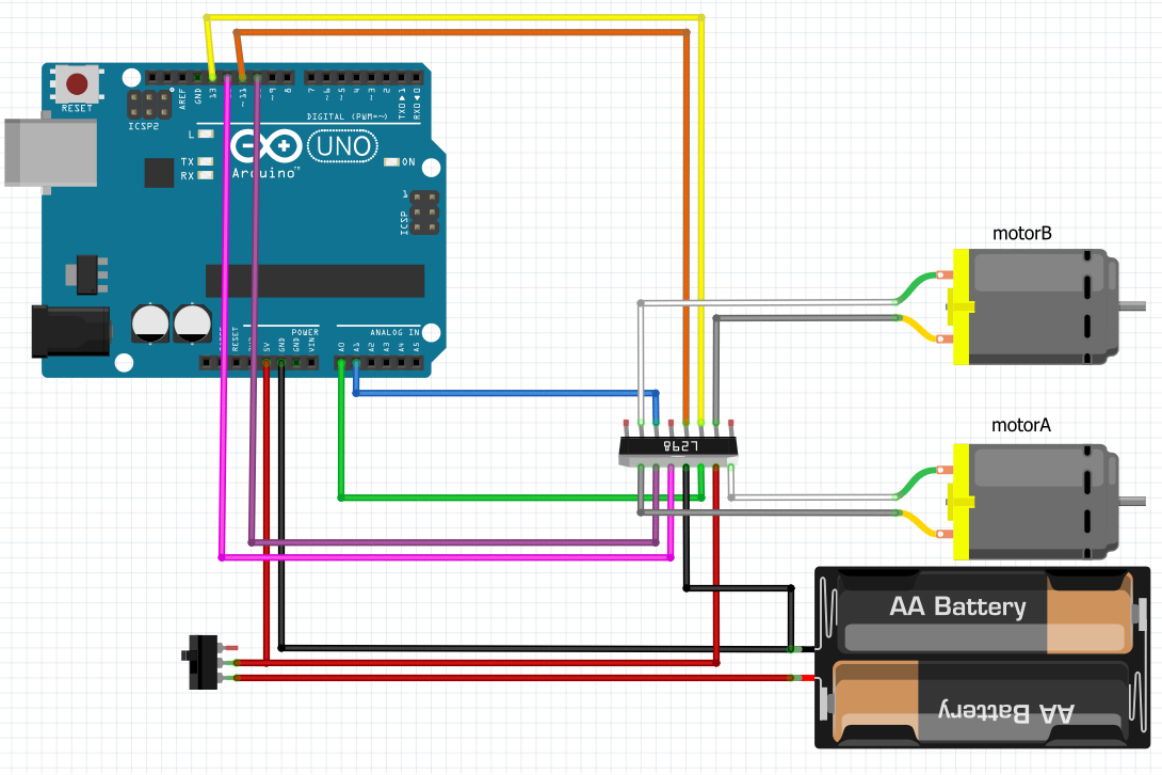


Figura 4.10 Conectare motoare și driver L298N la Arduino UNO

Prin firele de culoare alb, respectiv gri, se face legătura dintre driver-ul L298N și motoarele A și B. Prin pinii analogici (A0 și A1) motoarele se pot opri/porni individual, în funție de nevoi. Prin pinii digitali 10 și 12 se setează viteza motorului A. Diferența dintre cei doi pini o reprezintă direcția de deplasare (pin 10 – față , pin 12 – spate). Același principiu este aplicat și pentru motorul B(pin 11 – deplasare față, pin 13 – deplasare spate).

Având în vedere ca motoarele de curent continuu au și funcție de generator de energie electrică, prin învârtirea manuală a celor două roți legate de ele,există riscul să ajungă la driver curent și astfel să perturbe funcționarea acestuia. Pentru a împiedica acest fapt, pe plăcuța cu driverul de motoare am mai legat in serie câte 4 diode, care au rolul de a stopa curentul generat de motoare. Împreună cu acestea, dar în sens invers, întrea alimentare și driverul de motoare, mai sunt legate în serie doua rezistențe electrice și în paralel două condensatoare, cu rolul de a furniza driverului o tensiune de 5V.

4.4.2 Arduino UNO

Dispozitivul Arduino UNO este dezvoltat de compania Arduino, o companie open source ce produce atât componente hardware, cât și componente software. Dispozitivele Arduino sunt plăcuțe de dezvoltare cu microcontrollere integrate și alte componente, prin care îi asigură utilizatorului suficiente modalități de interfațare și comunicație cu alte dispozitive senzoriale sau de control. În funcție de necesități, plăcuțele de dezvoltare Arduino suportă mai multe tipuri de microcontrollere și alte dispozitive.

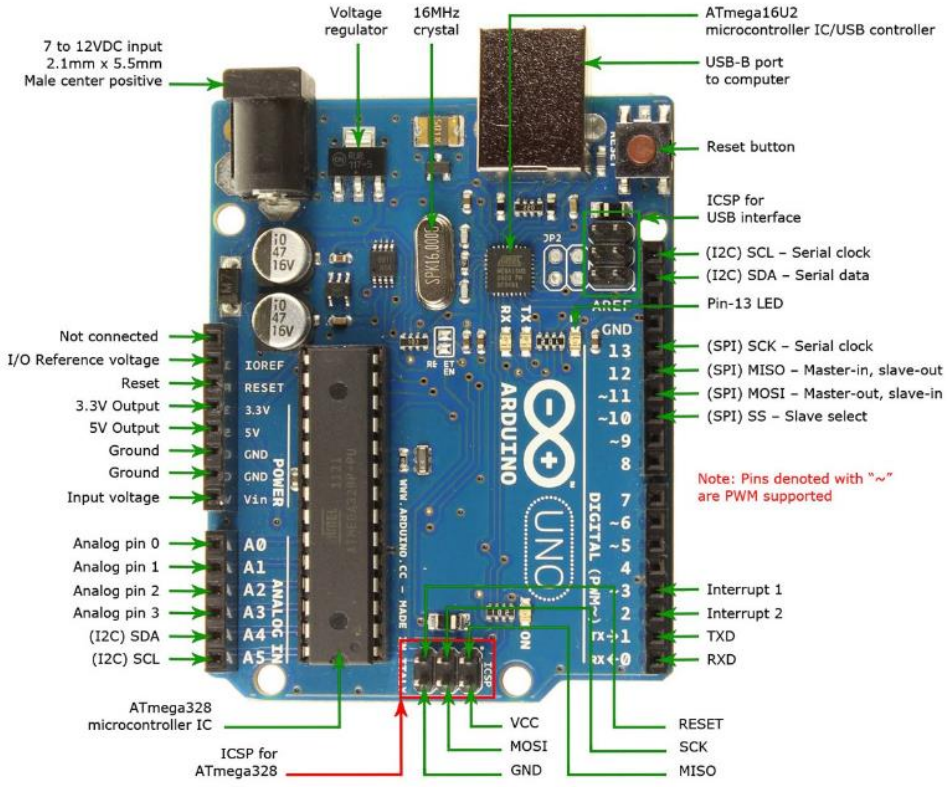


Figura 4.11 Placă dezvoltare Arduino UNO

Plăcuța de dezvoltare Arduino UNO dispune de un microcontroller ATMega328P cu o arhitectură pe 8 biți, porturi intrări și ieșiri, un oscilator cu o frecvență de 16MHz, memorie tip EEPROM de 1Kb, memorie RAM de 2Kb și o memorie flash, pentru program, de 32Kb. De asemenea dispune de mai multe posibilități de alimentare cu tensiune diferită, printre care tensiune de 5V, 3.3V sau mufă de alimentare cu 7V-12V, leduri de activitate, buton RESET, port USB pentru scriere programului pe memoria flash și altele.

Porturile input/output sunt reprezentate de 14 pini digitali, dintre care 6 pot fi ieșiri cu PWM(power width modulation) și 6 pini analogici, care pot fi folosiți ca intrări pentru convertorul încorporat analog-digital, ce funcționează pe 10 biți. Dintre acești pini, pinii digitali 2-9 sunt folosiți ca pini de intrare, pe care se citesc valorile venite de la senzori. Pe baza lor se calculează o valoare generală, pe care o vom folosi ca valoare de referință în urmărirea liniei negre. Pinii digitali 10-13 și analogici A0-A1 sunt folosiți ca pini de ieșire, utilizați în transmiterea vitezei la driver-ul de motoare și la oprirea/pornirea acestora.

Plăcuța dispune și de pini speciali, dintre care:

- Pinii TX/RX utilizați în comunicația seriala USART(Universal Serial Asynchronous Receiver/Transmitter), reprezintă pinii folosiți pentru transmisia, respectiv recepția datelor asincrone prin protocolul TTL(transistor-transistor logic);
- Pinul AREF(Analog Reference) este utilizat de convertorul analog-digital pentru determinarea tensiunii de referință al intrărilor analogice;
- Pinul IOREF(I/O Reference) este utilizat pentru determinarea tensiunii de referință al intrărilor digitale;
- Pinul fara label nu este folosit și nici conectat cu restul componentelor din cadrul plăcuței Arduino UNO.

Pulse width modulation este o tehnică de variație controlată a tensiunii aplicată unui dispozitiv electronic. Semnalul emis este cunoscut sub numele de semnal PWM. Pentru a obține această variație, se schimbă rapid tensiunea aplicată dispozitivului prin opriri și porniri – trecere din ON în OFF și viceversasa sau treceri din HIGH în LOW. În acest caz, aplicarea unei tensiuni de 5V reprezintă HIGH, iar 0V reprezintă LOW. Perioada de timp în care are loc această variație se numește factor de umplere și reprezintă procentul din tensiunea maximă posibilă aplicabilă dispozitivul electronic într-o perioadă de timp. Acest procent se obține astfel:

Procent = timpHIGH(timpHIGH + timpLOW)/100;

Procent=puls/perioadă*100;

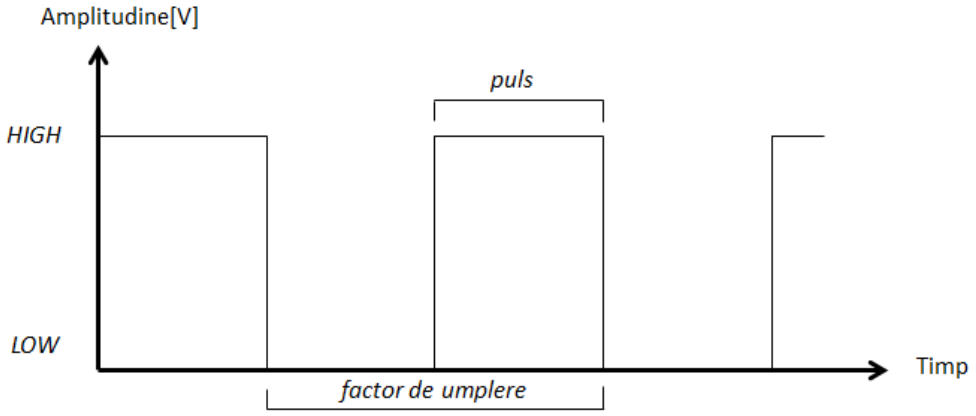


Figura 4.12 Semnal PWM

În această lucrare un astfel de semnal este folosit pentru setarea vitezei, prin controlul turației motorului. Când factorul de umplere este 0%, vom avea turație 0, iar când este 100%, motoarele vor funcționa la capacitate maximă (în acest caz pulsul este egal cu factorul de umplere).

În următoarele figuri se poate observa o schemă generală a sistemului informatic folosit în această lucrare și robotul.

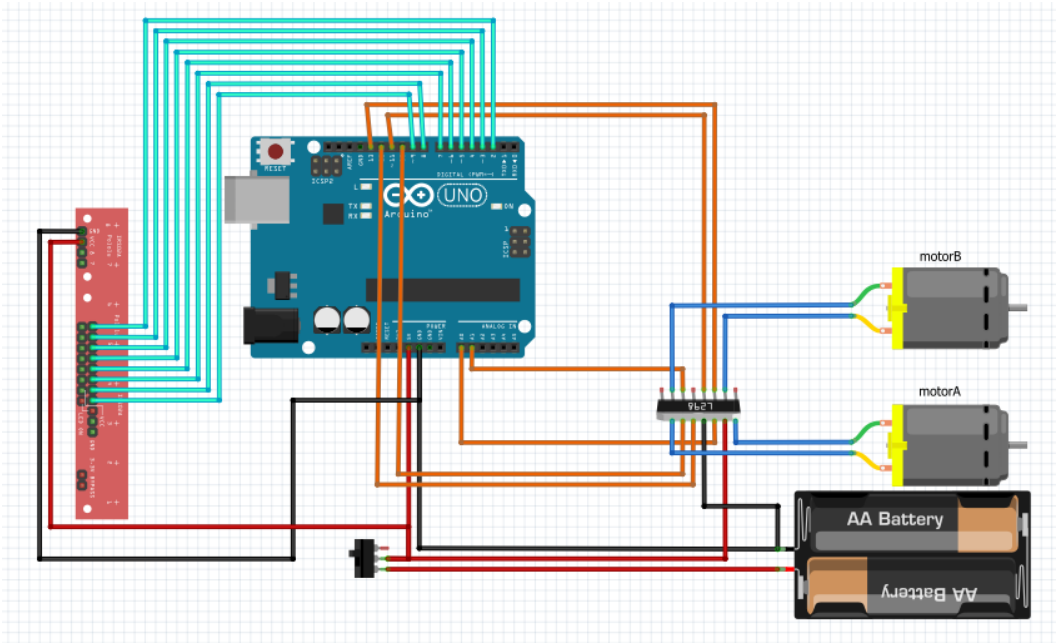


Figura 4.13 Schema generală

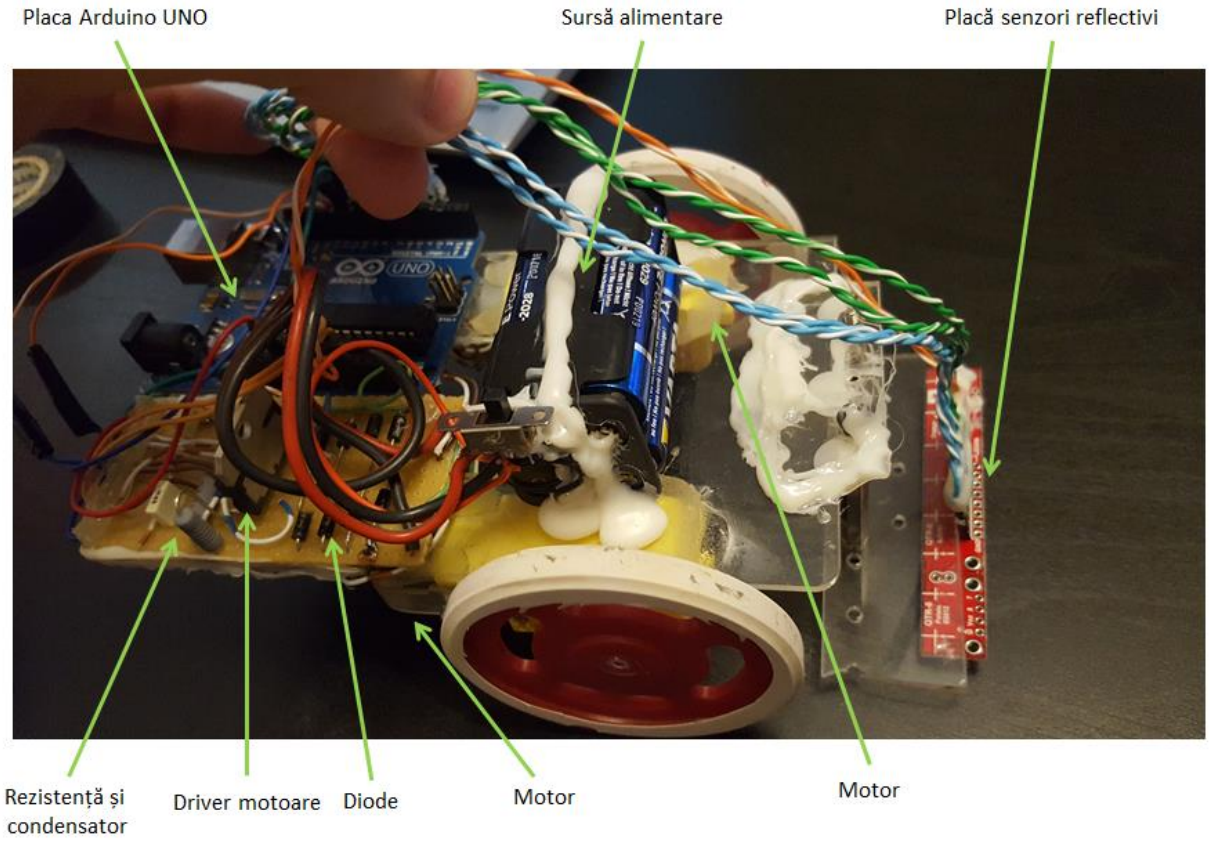


Figura 4.14 Sistemul informatic prezentat

CAPITOLUL 5: IMPLEMENTAREA SOLUȚIEI SOFTWARE

Implementarea soluției software constă în crearea unui program în mediul de dezvoltare arduino care să poată fi încărcat pe plăcuța de dezvoltare, să ”traducă” informațiile preluate de la senzori și să comande mișcările robotului. Prin componenta Arduino UNO se controlează puterea transferată la motoare folosindu-se pini ce transmit semnal PWM, astfel încât robotul să fie capabil să vireze.

5.1 Mediul de dezvoltare Arduino

Arduino pune la dispoziția dezvoltatorilor, pe lângă plăci de dezvoltare fizice, și o platformă de dezvoltare software, Arduino IDE, bazată pe limbajul C și C++. Programele scrise în acest IDE se pot încărca pe plăcuța Arduino printr-o conexiune USB între placă și calculator.

Programele scrise în IDE-ul Arduino poartă numele de ”sketch” și sunt formate din două funcții principale:

```
void setup() {
// instrucțiuni de setare, se rulează o singură dată la pornire
}

void loop() {
// instrucțiuni de execuție în buclă, se rulează în mod repetat până la oprirea  //
alimentării sistemului cu energie
}
```

Înainte de scrierea programului propriu zis, o parte importantă o reprezintă configurarea board-ului și al portului de comunicare. Un exemplu se poate observa în figura următoare:

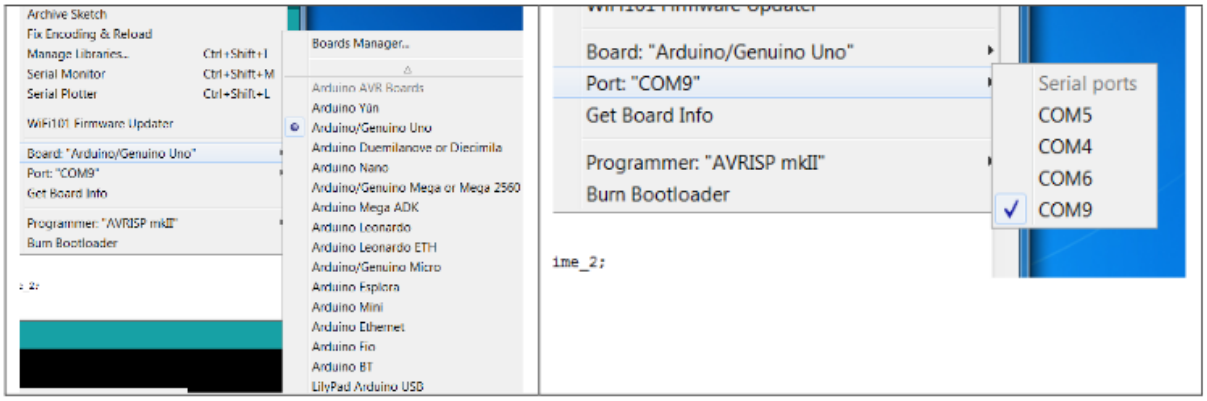


Figura 5.1 Configurare Arduino IDE

După această configurare, se poate trece la scrierea programului propriu zis. Principalele componente al mediului de dezvoltare sunt reprezentate de:

- O bară cu butoane, care ne permite sa compilăm codul, să îl încărcăm pe plăcuță și alte butoane uzuale, de deschidere, salvare și sketch nou;
- O zonă în care se scrie programul propriu zis;

- O zonă cu mesage informative și de eroare;

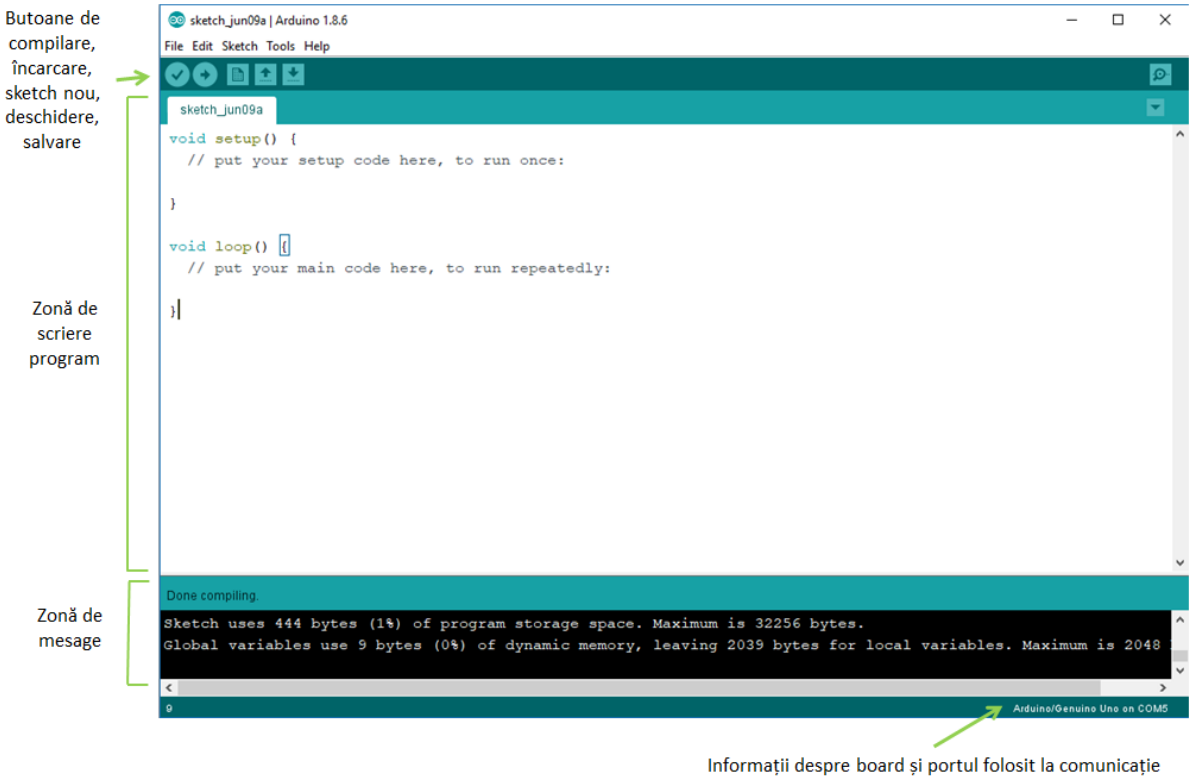


Figura 5.2 Exemplu program Arduino IDE

După încărcarea programului propriu zis, avem posibilitatea să facem debug, în cazul în care este nevoie, prin folosirea utilitarului ”Serial monitor”. Pentru aceasta, este nevoie de o conexiune directă între calculator și plăcuța de dezvoltare.

În exemplul de mai jos se poate observa, din lucrarea curentă, mesaje primite în timpul executării funcției loop:

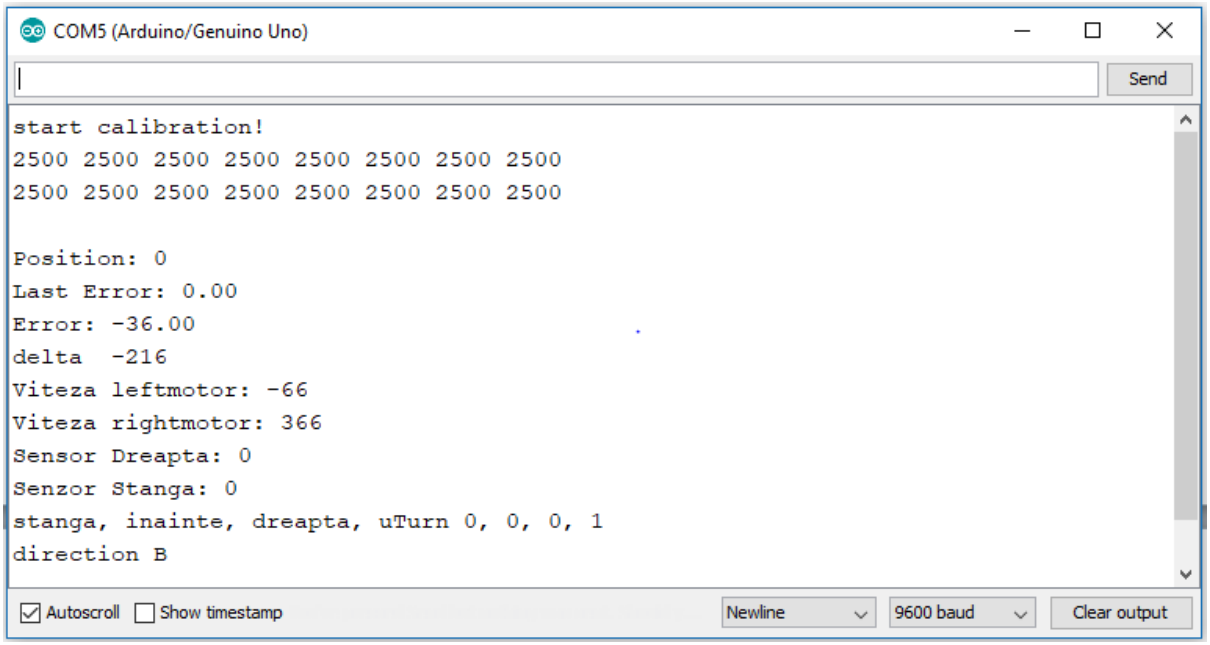


Figura 5.3 Serial monitor

5.2 Librăria QTRSensors.h

O altă utilitate importană în cadrul unui IDE este posibilitatea importării unui cod extern, care să poată fi utilizat în crearea unui program noi. Astfel de programe externe poartă denumirea de librării. În cadrul acestei lucrări am nevoie de o modalitate prin care să citesc valorile primite de la senzori, astfel încât să pot controla robotul. Pentru aceasta am importat o librărie externă, pusă la dispoziție de dezvoltatorul senzorilor reflectivi.

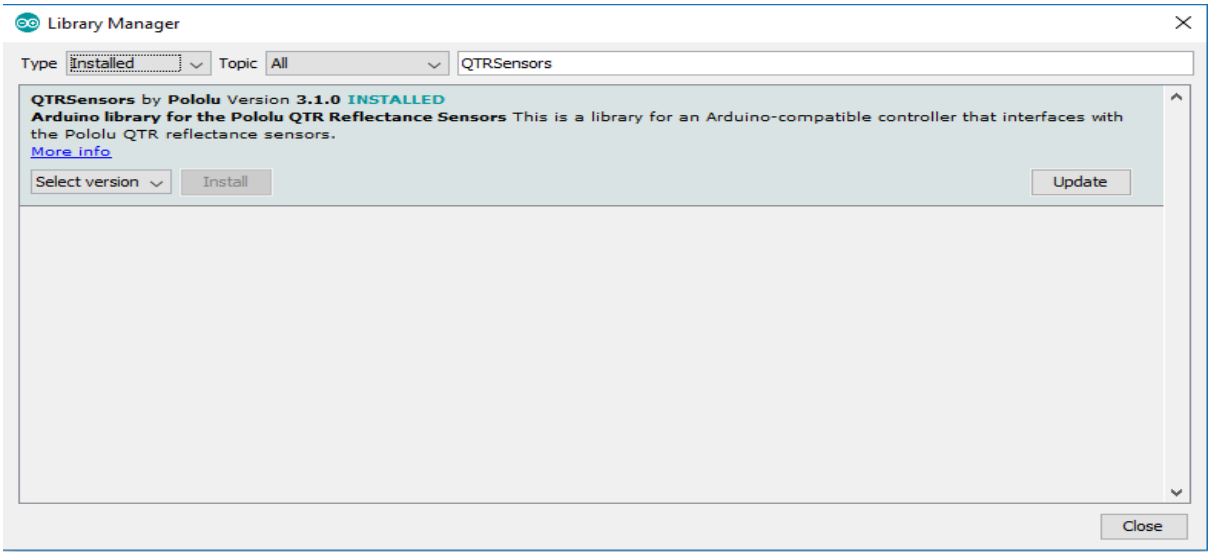


Figura 5.4 Librăria QTRSensors

Codul de inițializare unde este folosită librăria:

```
#define MIDDLE_SENSOR 5      //number of middle sensor used
#define NUM_SENSORS 8        //number of sensors used
#define TIMEOUT 2500         //waits for 2500 us for sensor outputs to go low
#define EMITTER_PIN QTR_NO_EMITTER_PIN     //emitterPin este pinul cu care se controleaza IR LED..nu il folosesc
#define DEBUG 0

QTRSensorsRC qtrrc((unsigned char[]) { 9,8,7,6,5,4,3,2 } ,NUM_SENSORS, TIMEOUT, EMITTER_PIN); // initializare senzori aferenti pinilor
```

De asemenea, librăria este folosită și pentru calibrabrea senzorilor, prin folosirea funcțiilor predefinite *calibrate()*, *calibrateMinimumOn()* și *calibrateMaximumOn().*

5.3 Algoritmul implementat

Algoritmul a fost creat în mediul de dezvoltare Arduino, versiunea 1.8.6. Pentru implementarea lui am folosit o serie de funcții de setare, executate o singură dată la pornire și o serie de funcții repetitive, executate constant până la oprirea alimentării cu energie.

Robotul este programat să urmărească linia neagră, iar dacă ajunge într-o intersecție, am folosit următoarea abordare pentru a lua o decizie:

1. *Dacă poți vira stânga, virează stânga*
2. *Altfel, dacă poți merge în față, mergi în față*
3. *Altfel, dacă poți vira dreapta, virează dreapta*

4. *Altfel întoarce, pentru că ai ajuns într-un punct mort*

De asemenea, robotul este programat să rețină traseul parcurs și să îl reducă, astfel încât, în momentul în care îl parcurge din nou, să ia decizia corectă la fiecare intersecție. În luarea deciziei corecte, bazându-mă pe regulile de selecție de mai sus, am stabilit și regulile de decizie la reparcurgerea traseului, astfel:

LBL = S // este înlocuit in rezolvarea maze-ului cu S
LBR = B // este înlocuit in rezolvarea maze-ului cu B
LBS = R // este înlocuit in rezolvarea maze-ului cu R
SBL = R // este înlocuit in rezolvarea maze-ului cu R
SBS = B // este înlocuit in rezolvarea maze-ului cu B
RBL = B // este înlocuit in rezolvarea maze-ului cu B

De exemplu, dacă a găsit stânga și în față, a virat stânga(L). A ajuns într-un punct mort, deci trebuie să se întoarcă(B). A ajuns în aceeași intersecție, unde a găsit de data aceasta stânga și în dreapta. Conform algoritmului, va face stânga(L). La reparcurgere, LBL este tradus în S, adică se va ști că la stânga este un punct mort și va merge înainte.

Detaliile legate de pinii de intrare și alte constante a căror valoare nu se modifică în timpul execuției, le-am inițializat în startul programului. Pentru setarea variabilelor ce se modifică o dată cu parcurgerea traseului, am folosit funcții de inițializare al căror parametrii se referă la valoarea variabilelor. Unele dintre aceste funcții sunt apelate în funcția principală *setup()*, dar și în *loop()* sau în alte funcții:

- ***manual_calibration()*** – această funcție este apelată o singură dată și scopul ei este calibrarea senzorilor;

- ***set_motors()*** – este folosită la setarea vitezei motoarelor din stânga și dreapta. Această funcție verifică de asemnea ca viteza sa fie încadrată între anumite limite
    1. *Dacă viteza>255, atunci viteza=255*
    2. *Altfel, dacă viteza<150, atunci viteza=150*
    3. *Altfel viteza=parametrul primit*

- ***select_turn()*** – returnează un anumit caracter, în funcție de decizia luată la întâlnirea unei intersecții:

    1. *Dacă am găsit stânga, atunci L*
    2. *Altfel dacă am găsit în față, atunci R*
    3. *Altfel dacă am găsit dreapta, atunci R*

4. *Altfel dacă trebuie să mă întorc, atunci B*
5. *Altfel dacă am ajuns la final, atunci F*

- ***turn()*** – setează viteza motoarelor necesară pentru a vira în funcție de setarea făcută în funcția *select_turn()*;

- ***line_follow()*** – este una dintre cele mai importante funcții din cadrul programului și scopul sau este urmărirea liniei. Pentru a face robotul să urmărească linia, am implementat un regulator liniar de tip PID.
  Regulatorul automat reprezintă principalul element din cadrul unui dispozitiv automat. Acesta îi permite dispozitivului să își regleze modul de funcționare o dată cu modificarea parametrilor de intrare. O schemă reprezentativă pentru un regulator automat se poate vedea în figura de mai jos.

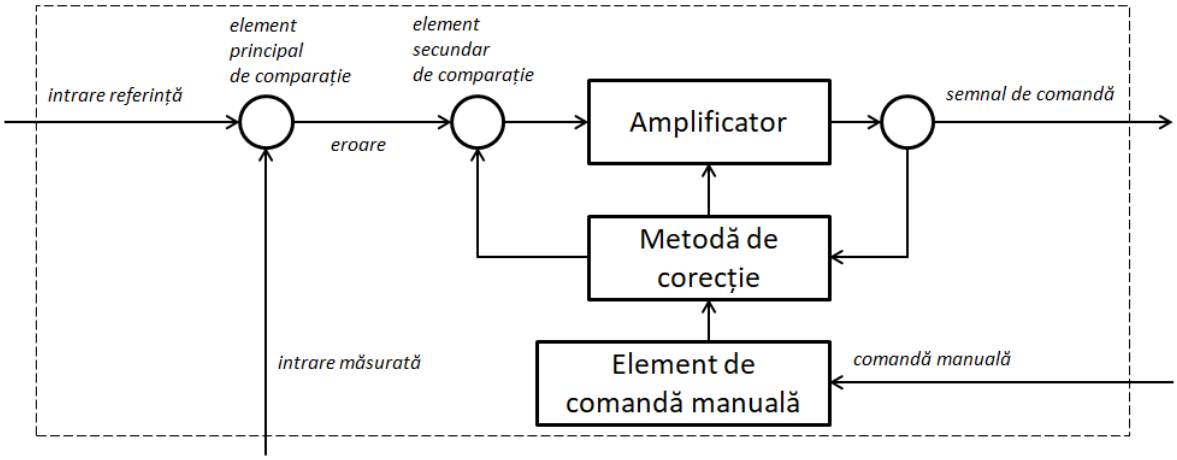


Figura 5.5 Regulator automat

Regulatorul liniar PID(Proporțional-Integral-Derivativ) asigură performațe de reglare ridicate, acesta fiind și motivul pentru care l-am ales, deși pentru realizarea proiectului era de ajuns un regulator PD(Proporțional-Derivativ). Formula de calcul al comenzii aplicate este:

$$u(t) = Kr[e(t) + \frac{1}{T_i}\int e(t) + T_d\frac{de(t)}{dt}]$$

unde:

$u(t)$ – comada aplicată

Kr – factorul de amplificaren al regulatorului

$e(t)$ – eroarea calculată

T_i – factorul de amplificare integral

T_d – factorul de amplificare derivativ

Momentan nu există nicio metodă matematică prin care s-ar putea obține factorii de amplificare, deoarece sistemul este prea complex pentru a putea fi modelat matematic. Pentru obținerea valorilor, se folosesc metode de determinare experimentale. Dintre aceste metode cea mai cunoscută este metoda Ziegler-Nichols, dar și Cohen Coon sau estimarea manuală pur și simplu.

O reprezentare grafică a acestui tip de regulator se poate observa în figura următoare:

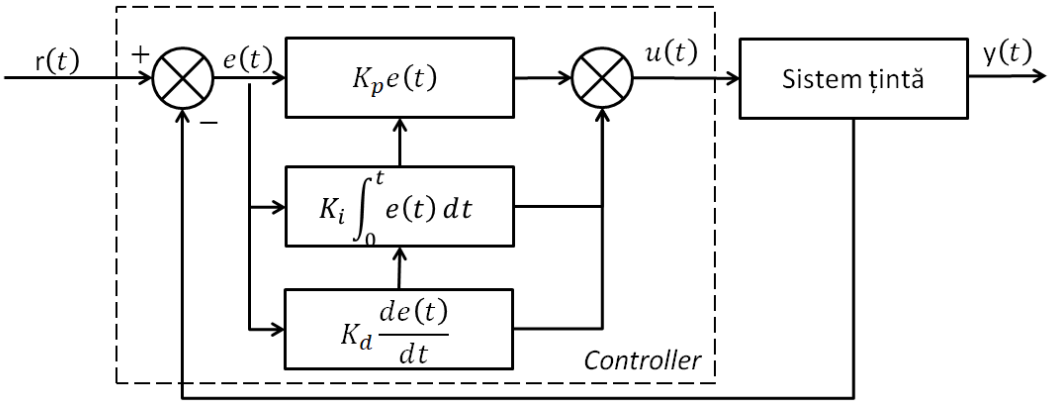


Figura 5.6 Regulator automat de tip PID

În cadrul proiectului, am setat ca referință o valoare primită de la senzori după calibrare, ce indică faptul că robotul are senzorii din mijloc deasupra liniei negre. Aceasta Apoi, am aplicat următoarea formulă de calcul:

$e_r(t) = 0$

$e(t) = e_r(t)$

// buclă continuă până întâlnește o intersecție

$$u(t) = K_i\left[K_p * e(t) + K_d\left(e(t) - e_i(t)\right)\right]$$

$$e_i(t) = e(t)$$

$$vitezăMotorStânga = vitezăInițială + u(t)$$

$$vitezăMotorDreapta = vitezăInițială - u(t)$$

// încheiere buclă

unde:

$e(t)$ – poziția robotului

$e_i(t)$ - valoarea de referință

$e_i(t)$ – ultima poziție a robotului, înainte de aplicarea comenzii

K_i , K_p , K_d – factori de amplificare

Comanda primită de motoare poate fi atât negativă, cât și pozitivă. Astfel se permite virarea în orice direcție.

- ***reduceThePath()*** – în timpul funcționării, de fiecare dată când întâlnește o intersecție, conform algoritmului, robotul va vira stânga(L), va merge în față(S), va vira dreapta(R) sau se va întoarce(B). Aceste mișcări le păstrez într-un array, iar pentru a rezolva maze-ul, în această funcție verific constant ultimele trei elemente din array. Dacă, conform regulilor definite, ultimele trei elemente pot fi modificate, le modific. În acest fel, la reparcurgere, robotul va știi ce să facă în momentul întâlnirii intersecțiilor.
- ***solveMaze()*** – la reparcurgere, robotul va aplica mișcarea definită în array de fiecare dată când va întâlni o intersecție, atâta timp cât nu se află la final.
- ***setup()*** – folosesc această funcție pentru calibrarea robotului doar.
- ***loop()*** – aceasta fiind funcția care se execută constant, încep prin apelarea funcției *line_follow()*. Apoi fac verificările aferente în cazul găsirii unei intersecții, robotul virează conform regulilor definite, se scrie și se reduce array-ul care definiște traseul optim, ca la final, după ce robotul a ajuns la capăt, să apelez funcția *solve_maze()*, ce va face parcurgerea optimă.

Pentru implementarea întregului algoritm, am început prin testarea funcțiilor de setare și prin verificarea că toate elementele elementele funcționează corespunzător. Am continuat apoi prin a determina robotul să urmărească linia neagră. Aici a fost nevoie de implementarea regulatorului. Pentru determinarea valorilor optime al factorilor de amplificare, am folosit metoda Ziegler-Nichols, metodă ce se bazează pe setarea diferitelor valori pentru constante și experimentarea comportamentului, începând cu cea proporțională, apoi derivativă și integrală. Astfel, am început doar prin implementarea

unui regulator proporțional, setând cu 0 factorii integral și derivativ. Am observat următorul comportament:

- Am avut oscilații ample în mișcarea robotului și ieșirea de pe traseu la valori ridicate ale constantei K_p . Am setat într-un final $K_p = 1$, unde oscilațiile erau relativ bune.
- Am continuat cu setarea K_d la valori mici, fără modficări mari în oscilații. Am crescut apoi la valori mai mari, unde oscilațiile au început să fie mai intense. Am setat într-un final $K_d = 2$, unde oscilațiile erau scăzute.
- Deși robotul funcționa cu un regulator PD, am observat o îmbunătățire la aplicarea K_i. Am setat astfel într-un final $K_i = 2$.

Am continuat apoi cu implementarea comportamentului de virare, întoarcere și oprire, scrierea vectorului cu mișcări, iar într-un final prin implementarea algoritmului de urmărire al traseului optim.

După numeroase experimente, robotul a fost capabil să își îndeplinească scopul și a urmat traseul optim la o viteză medie.

CONCLUZII

Robotul reușește să îndeplinească sarcinile atribuite, la o viteză medie. Are o precizie mai ridicată la o viteză mai mică, iar la viteză ridicată are momente când se pierde controlul. Acestea apar din cauza informațiilor primite de la senzori, fiind dependenți de lumină și de distanța dintre ei și mediu, dar și de la timpul de calcul al comenzii ce trebuie aplicată.

Principalele limitări sunt legate de durata de viață a bateriilor care este destul de scurtă, iar pierderea în intensitate se răsfrânge vizibil asupra modului de funcționare a robotului. O altă limitare o reprezintă dependența senzorilor de un factor extern și anume lumina din mediul în care robotul acționează.

Principalele îmbunătățiri ce se pot face sunt atât de natură hardware cât și software. Pentru a limita dependența de lumina din mediul extern, senzorii pot fi aplicați pe cadrul robotului, astfel încât distanța dintre ei și traseu să fie cât mai mică. Din punct de vedere software, se pot modifica factorii de amplificare astfel încât precizia la viteze ridicate să fie mai bună.

Aplicațiile practice în care ansamblul de față poate fi folosit, în urma aplicării unor îmbunătățiri, având în vedere că este capabil să determine un spațiu și să găsească traseul optim între două puncte, sunt în general de natură de explorare. Astfel de roboți pot fi folosiți în aplicații militare sau de explorare al spațiului, dar pot fi folosiți și în aplicații industriale, pentru a căra diferite obiecte pe un traseu stabilit.

ANEXE

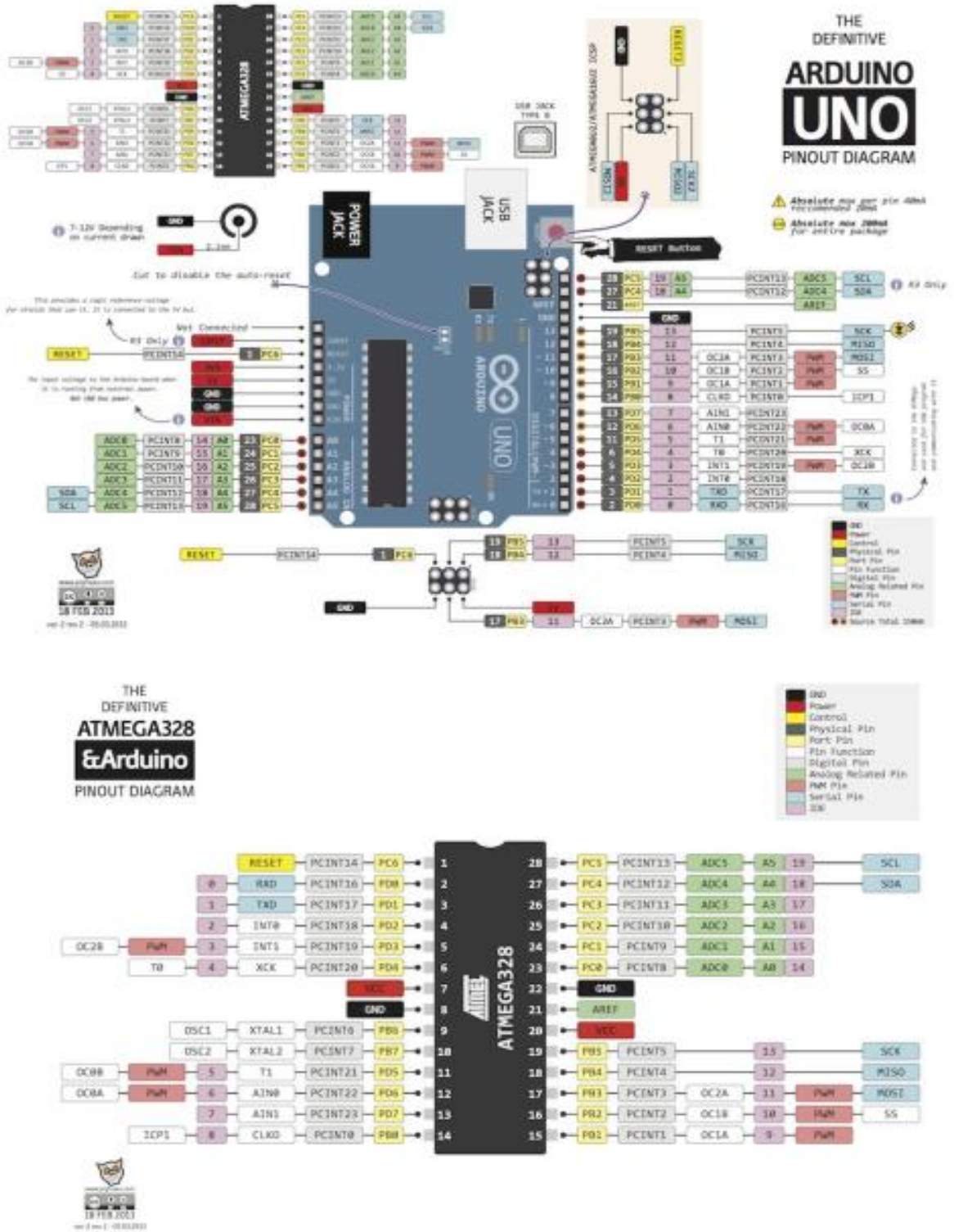
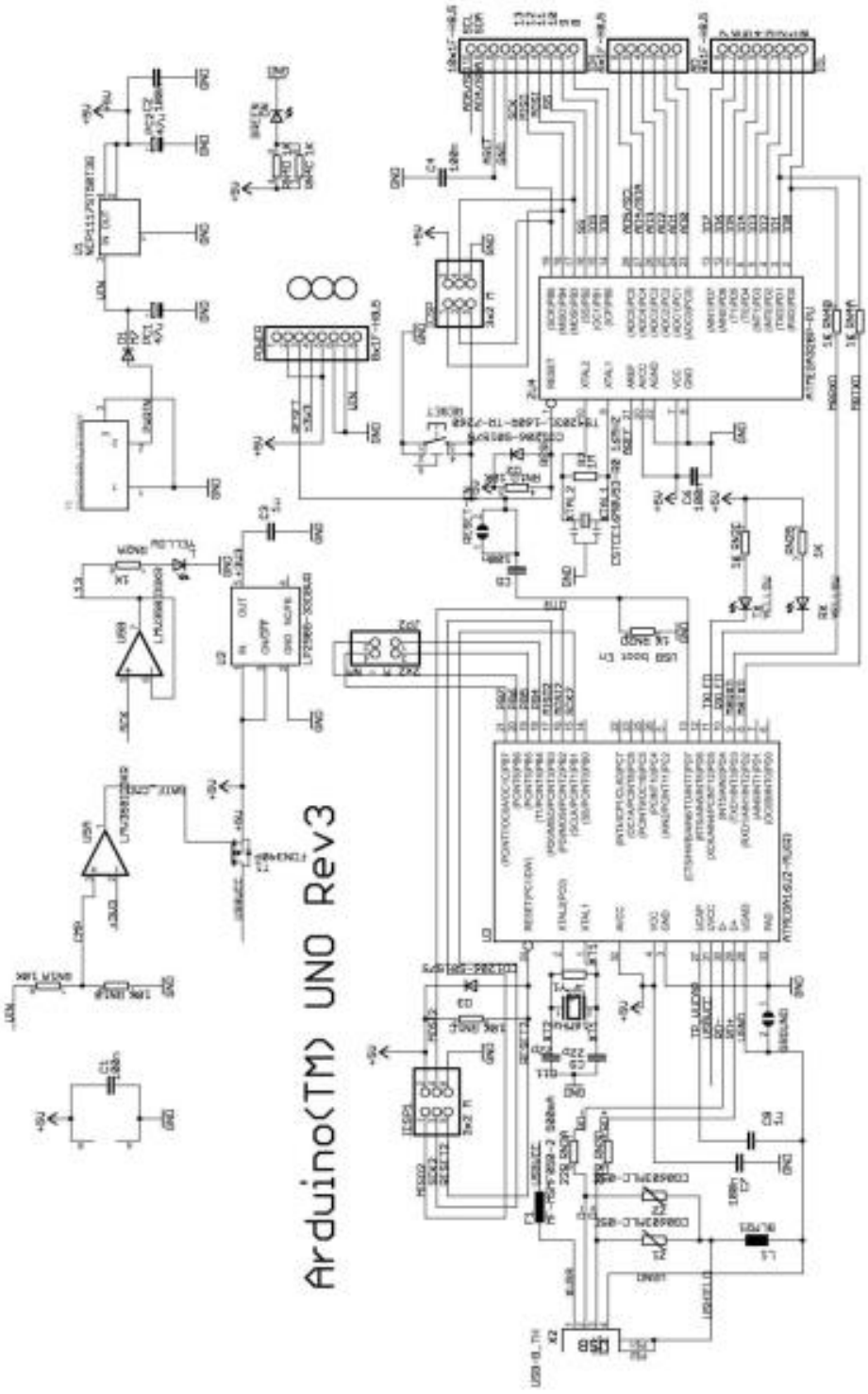


Diagrama pinilor Arduino



Schema electrică a plăcuței Arduino UNO

Cod sursă arduino

```
/*****************************************************************************
*  Author:     Dinca Marius Catalin                                          *
*  Board:      Arduino UNO R3                                                *
*  Platform:   Windows 10                                                    *
*  Year:       2019                                                          *
*  Project:    Licenta - Proiectarea unui robot autonom cu posibilități de recunoaștere a traseului  *
*  Setup:                                                                    *
*    # Driver motoare pinout:                                                *
*    - input1(A) --> pin 11 (galben(in) verde(out))                          *
*    - input2(A) --> pin 13 (galben(in) verde(out))                          *
*    - Enable Dreapta(A)--> pin 12 (verde)                                   *
*    - input3(B) --> pin 10 (verde (in) albastru(out))                       *
*    - input4(B) --> pin 12 (verde (in) albastru(out))                       *
*    - Enable Stanga(B)  --> pin 13 (albastru)                               *
*    # Senzor array 1,2,3,4,5,6,7,8 pinout                                   *
*    - pini arduino 9,8,7,6,5,4,3,2                                          *
*  Descriere:                                                                *
*    Prin codul de mai jos un sistem informatic poate parcuge un traseu, delimitat printr-o linie     *
* neagra, de două ori. Prima parcurgere o reprezintă ”învățarea traseului”, iar cea de-a doua      *
* va fi parcurgerea optimă a aceluiași traseu.                               *
*****************************************************************************/
#include <QTRSensors.h> //Pololu QTR Sensor Library

int dir_a = A0;  //enable A on analog pin A0  (Left motor - roata stanga)
int dir_b = A1;  // enable B on analog pin A1  (Right motor - roata dreapta)
int pwm_a = 10;  //viteza motor stanga  (Left motor - mers inainte)
int pwm_aa = 12;  //viteza motor stanga  (Left motor - mers inapoi)
int pwm_b = 11;  //viteza motor dreapta (Right motor - mers inainte)
int pwm_bb = 13;  //viteza motor dreapta  (Right motor - mers inapoi)
pinMode(A0,OUTPUT); // setare pin analog pentru enable A
pinMode(A1,OUTPUT); // setare pin analog pentru enable B

#define KP 1 //experiment pentru a determina valoarea cea mai buna pentru PID
#define KD 2 // experiment pentru a determina valoarea cea mai buna pentru PID ( Note: Kp < Kd)
#define KI 2 // experiment pentru a determina valoarea cea mai buna pentru PID
#define minim_speed 150  //minimum speed of the motors
#define maxim_speed 255 //max. speed of the motors

#define NUM_SENSORS 8       //number of sensors used
#define TIMEOUT 2500        //waits for 2500 us for sensor outputs to go low
#define EMITTER_PIN QTR_NO_EMITTER_PIN //emitterPin este pinul cu care se controleaza IR LED
#define DEBUG 0

QTRSensorsRC qtrrc((unsigned char[]) { 9,8,7,6,5,4,3,2 } ,NUM_SENSORS, TIMEOUT, EMITTER_PIN);
unsigned int sensorValues[NUM_SENSORS];

int turnSpeed = 140;  // tunare viteza motoare la gasirea unei intersecti (0-255)
int turnSpeedSlow = 140;  // tunare viteza motoare la gasirea unei intersecti (0-255)
unsigned int line_position=0;
float lastError = 0;
char path[] ="";
```

```
/*************** Functia setup *****************/
void setup(){
        Serial.begin(9600);
        Serial.println ("start calibration! ");
        delay(1000);
        manual_calibration();
        set_motors(0,0);
} // end setup

/************Functia de calibrare ********** *****/
void manual_calibration() {
        for (int i = 0; i < 250; i++){
                qtrrc.calibrate();
                delay(20);
        }
        for (int i = 0; i < NUM_SENSORS; i++) {
                Serial.print(qtrrc.calibratedMinimumOn[i]);
                Serial.print(' ');
        }
        Serial.println();

        for (int i = 0; i < NUM_SENSORS; i++) {
                Serial.print(qtrrc.calibratedMaximumOn[i]);
                Serial.print(' ');
        }
        Serial.println();
        Serial.println();
} // end manual_calibration

/******* Functia de setare viteza***********/
void set_motors(int motor1speed, int motor2speed){
 if (motor1speed > 0) {
        if (motor1speed > maxim_speed ) motor1speed = maxim_speed;
        if (motor1speed < minim_speed ) motor1speed = minim_speed;
        digitalWrite(dir_a, HIGH);
        analogWrite(pwm_a, motor1speed);
        analogWrite(pwm_aa, 0);
 }
 else if (motor1speed < 0) {
        if (motor1speed < -(maxim_speed) ) motor1speed = -(maxim_speed);
        digitalWrite(dir_a,HIGH);
        analogWrite(pwm_aa, 0);
        analogWrite(pwm_a, minim_speed);
 }

 if (motor2speed > 0) {
        if (motor2speed > maxim_speed ) motor2speed = maxim_speed;
        if (motor2speed < minim_speed ) motor2speed = minim_speed;
        digitalWrite(dir_b, HIGH);
        analogWrite(pwm_b, motor2speed);
        analogWrite(pwm_bb, 0);
```

```
  }
  else if (motor2speed < 0){
          if (motor2speed < -(maxim_speed) ) motor2speed = -(maxim_speed);
          digitalWrite(dir_b, HIGH);
          analogWrite(pwm_bb, 0);
          analogWrite(pwm_b, minim_speed);
  }

  if (motor1speed == 0 ){
          digitalWrite(dir_a, LOW);
          analogWrite(pwm_a, 0);
          analogWrite(pwm_aa, 0);
  }

  if (motor2speed == 0 ){
          digitalWrite(dir_b, LOW);
          analogWrite(pwm_b, 0);
          analogWrite(pwm_bb, 0);
  }
} // end set_motors

/******* Functia de urmarire linie**********/
void line_follow(){
          unsigned int sensors[8];
          int line_position = qtrrc.readLine(sensors); //line position dupa calibrare senzori
          float error = line_position/100 - 36; // din cauza limitarii vitezelor

          float aux2  = KP * error + KD * (error - lastError);
          int motorSpeed = round (aux2*KI);
          lastError = error;
          int leftMotorSpeed = minim_speed + motorSpeed;
          int rightMotorSpeed = minim_speed - motorSpeed;

          set_motors(leftMotorSpeed, rightMotorSpeed);
          delay(100);
          set_motors(0,0);
} // end line_fllow

/******* Functia de selectare directie in intersectie **********/
// L -> LEFT , intoarcere la stanga 90 grade
// S -> STRAIGHT , nu face nimic, merge inainte
// R -> RIGHT , intoarcere la dreapta 90 grade
// B -> BACK/UTURN , intoarcere 180 grade
// F -> FINISH , a ajuns la final
char select_turn(unsigned char found_left, unsigned char found_straight, unsigned char found_right, unsigned
char uTurn, unsigned char finish){
          if(found_left) return 'L';
          else if(found_straight)  return 'S';
          else if(found_right)  return 'R';
          else if(uTurn)  return 'B';
          else if(finish)  return 'F';

} // end select_turn
```

```
/******* Functia de virare in intersectie ************/
void turn(char directie){
        int i = strlen(path);
        if (directie == "L" || directie == "R" || directie == "B"){
                path[i] = directie;
        }
        switch(directie){
                case 'L':
                        digitalWrite(dir_a, HIGH);
                        digitalWrite(dir_b, LOW);
                        analogWrite(pwm_a, turnSpeedSlow);
                        analogWrite(pwm_aa, 0);
                        line_position = qtrrc.readLine(sensorValues);

                        while (sensorValues[0] > 150)   {
                                line_position = qtrrc.readLine(sensorValues);
                         }
                        while (line_position < 3600) {
                                line_position = qtrrc.readLine(sensorValues);
                        }
                        set_motors(0,0);
                        break;

                case 'R':
                        digitalWrite(dir_a, LOW);
                        digitalWrite(dir_b, HIGH);
                        analogWrite(pwm_b, turnSpeedSlow);
                        analogWrite(pwm_bb, 0);
                        line_position = qtrrc.readLine(sensorValues);

                        while (sensorValues[7] > 150){
                                line_position = qtrrc.readLine(sensorValues);
                        }
                        while (line_position < 3100){
                                line_position = qtrrc.readLine(sensorValues);
                        }

                        set_motors(0,0);
                        break;
                case 'B':
                        digitalWrite(dir_a, HIGH);
                        digitalWrite(dir_b, HIGH);
                        analogWrite(pwm_a, turnSpeedSlow);
                        analogWrite(pwm_aa, 0);
                        analogWrite(pwm_b, 0);
                        analogWrite(pwm_bb, turnSpeedSlow);
                        while (line_position < 3200){
                                line_position = qtrrc.readLine(sensorValues);
                        }
                        set_motors(0,0);
                        break;
```

```
                case 'S': // do nothing, just follow line
                break;

                case 'F':
                        digitalWrite(dir_a, LOW);
                        digitalWrite(dir_b, LOW);
                        analogWrite(pwm_a, 0);
                        analogWrite(pwm_aa, 0);
                        analogWrite(pwm_b, 0);
                        analogWrite(pwm_bb, 0);
                        set_motors(0,0);
                        break;
        }
} // end turn

/******* Functia de reducere traseu la cel optim ************/
char * reduceThePath(char * path){
        /* Reguli reducere path
        LBL = S
        LBR = B
        LBS = R
        SBL = R
        SBS = B
        RBL = B
        */
        int sizePath = strlen(path);
        String checkString;

        for (int i = sizePath-3; i<sizePath; i++ ){
                checkString += path[i];
        }

        if (checkString == "LBL") {
                path[sizePath-1] = '\0';
                path[sizePath-2] = '\0';
                path[sizePath-3] = "S";
        }
        if (checkString == "LBR") {
          path[sizePath-1] = '\0';
          path[sizePath-2] = '\0';
          path[sizePath-3] = "B";
        }
        if (checkString == "LBS") {
          path[sizePath-1] = '\0';
          path[sizePath-2] = '\0';
          path[sizePath-3] = "R";
        }
        if (checkString == "SBL") {
          path[sizePath-1] = '\0';
          path[sizePath-2] = '\0';
          path[sizePath-3] = "R";
        }
        if (checkString == "SBS") {
```

```
            path[sizePath-1] = '\0';
            path[sizePath-2] = '\0';
            path[sizePath-3] = "B";
          }
          if (checkString == "RBL") {
            path[sizePath-1] = '\0';
            path[sizePath-2] = '\0';
            path[sizePath-3] = "B";
          }
          Serial.print("Sunt in reducePath si stringul verificat este ==> ");
          Serial.println(checkString);
          return path;
} // end reduceThePath


/******* Functia reparcurgere traseu optim ************/
void solveMaze( char * path){
          Serial.println( "Am ajuns la rezolvarea maze-ului" );
           int i = 0;
           while (i < strlen(path) && (sensorValues[0] < 2200 && sensorValues[7] < 2200)){
                    line_follow();
                    // daca a gasit o intersectie
                    if (sensorValues[0] >= 100 || sensorValues[7] >= 100)  {
                              turn(path[i]);
                              i++;
                    }
             }
 Serial.println( "Am ajuns la final, am rezolvat maze-ul.." );
 set_motors(0,0);
 delay(100000); // stau 100 secunde
} // end solveMaze


/******* Functia loop ************/
void loop(){
          line_follow();
          unsigned char found_left = 0;
          unsigned char found_straight = 1;
          unsigned char found_right = 0;
          unsigned char uTurn = 0;
          unsigned char finish = 0;

          line_position = qtrrc.readLine(sensorValues);

          // daca senzor dreapta a detectat negru, verificare fata apoi dreapta
          if(sensorValues[0] >= 100){
                    set_motors(150,150);
                    delay(100);
                    set_motors(0,0);
            //verific daca se poate merge in fata, daca da, line_follow()
                    if (sensorValues[1] < 150 &&
                    sensorValues[2] < 150 && sensorValues[3] < 150 &&
                    sensorValues[4] < 150 && sensorValues[5] < 150 &&
```

```
        sensorValues[6] < 150){
                set_motors(-150,-150);
                delay(100);
                set_motors(0,0);
                found_right = 1;
                found_straight=0;
                uTurn=0;
                found_left=0;
                finish = 0;
                delay(1000);
        }
        else line_follow();
 }

// senzor stanga a detectat negru, virare stanga
if(sensorValues[7] >= 100){
        set_motors(0,0);
        found_left = 1;
        found_straight=0;
        found_right=0;
        uTurn=0;
        finish = 0;
        delay(1000);
}
// senzori mijloc au detectat alb, uTurn
if(sensorValues[1] < 150 &&
  sensorValues[2] < 150 && sensorValues[3] < 150 &&
  sensorValues[4] < 150 && sensorValues[5] < 150 &&
  sensorValues[6] < 150){
        set_motors(0,0);
        uTurn = 1;
        found_straight=0;
        found_right=0;
        found_left=0;
        finish = 0;
        delay(1000);
 }

// toti senzorii au detectat negru, verific daca e intersectie, altfel - STOP
if(sensorValues[0] > 2200 &&  sensorValues[7] > 2200){
         set_motors(150,150);
         delay(100);
         set_motors(0,0);

        if(sensorValues[0] > 2200 &&  sensorValues[7] > 2200){
                uTurn = 0;
                found_straight=0;
                found_right=0;
                found_left=0;
                finish = 1;
                delay(1000);
        }
        else{ // este de fapt intersectie, asa ca LEFT
```

```
                        set_motors(-150,-150);
                        delay(100);
                        set_motors(0,0);
                        found_left = 1;
                        found_straight=0;
                        found_right=0;
                        uTurn=0;
                        finish = 0;
                        delay(1000);
                }
        }

        char directie = select_turn(found_left, found_straight, found_right, uTurn, finish);
        turn(directie);

        if (strlen(path) >= 3) {
                reduceThePath(path);
        }

        if (finish){
                Serial.println ("Am ajuns la final. Astept 5 secunde pentru repozitionare..");
                delay(5000);
                solveMaze(path);
        }
       Serial.println("~~~~~~~~~~~~~~~~~~END oF LOOP~~~~~~~~~~~~~~~~~~~~~~~~~");
}
```

BIBLIOGRAFIE

Dumitrache Ioan, 2005 - Implementarea numerică a algoritmilor de reglare. Ingineria Reglării Automate. Editura Politehnica Press. pagina 618- 718

Jorge Angeles - Fundamental of Robotic Mechanical Systems – Theory, Methods and Algorithms

Isaac Asimov, 1942 – Runaround

Isaac Asimov – I, Robot

John Chiasson, 2005 – Modeling and High-Performance Control of Electric Machines , pagina 3-23, 31-44

B.D. Bedford, 1964 – Principles of Inverter Circuits, pagina 325-390

John David Warren, Josh Adams and Harald Molle – Arduino Robotics

J. M. Hughes – Arduino, A Technical Reference

Jeremy Blum – Exploring Arduino

Gunther Gridling, Bettina Weiss, 2007 – Introduction to Microcontrollers

Resurse internet:

https://www.scribd.com/document/136786623/Roboti

http://studii.crifst.ro/doc/2016/2016_05_04.pdf

http://www.robothalloffame.org/inductees/03inductees/unimate.html

https://www.pololu.com/docs/pdf/0J12/QTR-8x.pdf

https://www.arduino.cc/en/guide/introduction

https://www.arduino.cc/reference/en/

https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation