

UNIVERSITATEA DIN BUCUREŞTI



FACULTATEA DE
MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA
TEHNOLOGIA INFORMAȚIEI

Lucrare de licență

VEHICUL SEMIAUTONOM CU BRAȚ ROBOTIC

Absolvent
Surdu Bob Alexandru

Coordonatori științifici
Conf. Dr. Bogdan Alexe
Drd. Andrei Dumitriu

București, iunie 2023

Rezumat

Avansul tehnologic a permis omenirii să găsească noi soluții pentru îmbunătățirea calității vieții. O bună parte din aceste soluții se regăsesc în domeniul roboticii, în care cercetătorii au căutat să rezolve problemele populației globului pământesc folosind mecanisme și mașinării complexe, care se adaptează la mediul înconjurător și pot prelua sarcini periculoase sau repetitive de la oameni.

În această lucrare este tratat subiectul robotilor semi-autonomi care au mijloace de interacționare cu mediul înconjurător și pot să execute acțiuni fără o comandă continuă din partea operatorilor, fiind de ajuns ca aceștia să traseze niște sarcini.

Lucrarea, cu caracter demonstrativ, prezintă implementarea unui prototip de robot cu următoarele capabilități: navigație autonomă la punct stabilit prin coordonate, construirea unei hărți a mediului prin care se deplasează, planificarea și urmărirea unei rute de navigație, detectia obiectelor de interes pe baza aspectului acestora și manevrarea lor folosind brațul robotic.

Pentru atingerea acestor obiective, proiectul include o multitudine de tehnologii hardware și software: single-board computer cu performanțe ridicate, cameră video de adâncime, senzor LiDAR, brăț robotic conceput special pentru proiect, meta-sistemul de operare Robot Operating System, algoritmi de cartografiere, inteligență artificială pentru detecție de obiecte, panou de control cu interfață web, precum și elemente de mecatronică și software de control al acestora.

Abstract

Technological advancement has enabled humanity to find new solutions for improving the quality of life. A significant portion of these solutions can be found in the field of robotics, where researchers have sought to address the problems faced by the global population using complex mechanisms and machinery that adapt to the surrounding environment and can take over dangerous or repetitive tasks from humans.

This paper addresses the topic of semi-autonomous robots that have means of interaction with the surrounding environment and can perform actions without continuous supervision from an operator, who can therefore control the robots solely by giving tasks.

The present work introduces as a proof-of-concept the implementation of a robot prototype with the following capabilities: autonomous navigation to a point set by coordinates, building a map of the environment through which it moves, planning and following a navigation route, detecting objects of interest based on their appearance and handling them using the robotic arm.

To achieve these goals, the project includes a multitude of hardware and software technologies: high-performance single-board computer, depth video camera, LiDAR sensor, robotic arm specially designed for the project, the Robot Operating System meta-system, mapping algorithms, artificial intelligence for object detection, control panel with web interface, as well as mechatronics elements and their control software.

Cuprins

1	Introducere	5
2	Noțiuni preliminare	7
2.1	ROS 2 - Robot Operating System v2	7
2.2	Algoritmul de control PID	7
2.3	Cartografiere și localizare simultană - SLAM	8
3	Componenta hardware	9
3.1	Componentele hardware ale robotului	9
3.2	Conexiunile dintre componente	15
3.3	Asamblarea hardware a robotului	16
4	Componenta software	18
4.1	Arhitectura Software	18
4.2	Software-ul pentru Nvidia Jetson Orin Nano	19
4.3	Software-ul care rulează pe microcontrolerul Arduino Mega	30
5	Concluzii	32
6	Anexă - Listă de acronime	34

Listă de figuri

1.1	Robotul asamblat în forma finală	6
2.1	Schema bloc de funcționare a algoritmului PID[1]	8
3.1	Şasiul robotului	10
3.2	Nvidia Jetson Orin Nano montat pe şasiul robotului	11
3.3	Camera Intel Realsense D435	12
3.4	Drivere de motoare Pololu Qik 2s12v10 montate în șasiu	13
3.5	Şasiul robotului în timpul modificării	14
3.6	Brătul robotic montat pe șasiu	14
3.7	Schema conceptuală a conexiunilor dintre componentele electronice ale robotului	16
3.8	Placa de conexiuni a robotului montată pe Arduino	17
3.9	Vedere laterală a robotului	17
4.1	Diagrama arhitecturii software	18
4.2	Modelul URDF al robotului	22
4.3	Exemplu de hartă realizată de robot și poziția acestuia	23
4.4	Comparatie între modelul 3D și obiectul printat	26
4.5	Componentele funcției loss	27
4.6	Augmentarea datelor de antrenare	28
4.7	Evoluția preciziei în timpul antrenării	28
4.8	Obiect detectat folosind YOLO	29
4.9	Interfata web a robotului	30

Capitolul 1

Introducere

Tema acestui proiect de licență este realizarea unui prototip de robot cu scop demonstrativ care implementează capabilitatea de a naviga semi-autonom prin mediul înconjurător și de a manipula obiecte folosind un braț robotic. Controlul sistemului se realizează la distanță, cu ajutorul unui panou de control web. Scopul proiectului este de a demonstra aceste funcționalități de bază care și-ar putea găsi utilitatea într-o multitudine de domenii cum ar fi roboți industriali pentru manipularea coletelor într-un depozit, roboți de explorare a mediilor greu accesibile sau toxice și roboți de căutare și salvare în situații de urgență. Această temă a fost aleasă deoarece face parte dintr-un domeniu de actualitate în continuă dezvoltare, care are potențialul de a revoluționa modul de rezolvare al anumitor probleme din societate prin eficientizarea și creșterea preciziei soluțiilor utilizate în prezent.

În ultimii ani, domeniul roboticii a cunoscut progrese remarcabile și a schimbat profund unele aspecte ale vieții noastre. Roboții au devenit mașini versatile capabile să îndeplinească o gamă largă de sarcini, de la automatizarea industrială la îngrijirea sănătății, de la explorare și până la asistență cotidiană. Cercetătorii au dorit să creeze mașinării capabile să imite și să înlocuiască într-o oarecare măsură abilitățile umane, încercând să confere roboților funcții de navigație autonomă, de detectie și interacțiune cu obiectele din mediul lor înconjurător, reușind astfel să preia sarcini de la oameni și să efectueze acțiuni repetitive sau periculoase fără să necesite intervenție constantă.

Navigația autonomă este o caracteristică esențială pentru roboții care trebuie să se deplaseze în spații necunoscute sau dificil accesibile pentru oameni. Sistemul de tracțiune cu șase roți al robotului prezentat asigură stabilitatea și manevrabilitatea, permitându-i să se deplaseze cu ușurință pe suprafete variate, accidentate sau înclinate. Astfel, robotul este capabil să-și planifice traseul și să evite obstacolele întâlnite în timpul deplasării. Acesta nu este un robot complet autonom, în măsura în care necesită supervizarea unui operator pentru stabilirea coordonatelor de deplasare și a sarcinilor de îndeplinit. Robotul poate construi o hartă a mediului înconjurător și poate planifica rute până la coordonatele introduse de utilizator. Un alt aspect important al acestui robot este brațul robotic, cu ajutorul căruia poate să manipuleze obiecte pe care le de-

tecează în prealabil folosind imagini preluate de la camera video din dotare.

Îmbinarea acestor funcționalități și integrarea lor într-un produs finit care poate fi controlat folosind un panou de comandă ușor de utilizat face ca acest proiect să fie relevant pentru domeniul din care face parte.

În cadrul acestei lucrări sunt explorate designul și funcționalitățile robotului și sunt prezentate tehnologiile, componentele și algoritmii folosiți pentru dezvoltarea acestor funcționalități.

Structura lucrării de licență

Capitolul al doilea introduce câteva noțiuni necesare pentru înțelegerea tehnologiilor și conceptelor care stau la baza proiectului. Al treilea capitol prezintă proiectul din punct de vedere hardware, componentele utilizate pentru platforma hardware și conexiunile dintre acestea. Arhitectura software, tehnologiile și pachetele software utilizate, precum și modul de implementare a funcționalităților proiectului sunt prezentate în capitolul al patrulea. Ultimul capitol prezintă concluziile și observațiile la care s-a ajuns după cercetarea, construirea și implementarea acestui proiect.



Figura 1.1: Robotul asamblat în forma finală

Capitolul 2

Noțiuni preliminare

2.1 ROS 2 - Robot Operating System v2

Robot Operating System (prescurtat ROS) este un framework software open-source care oferă o suită de unelte, librării și convenții pentru dezvoltarea de sisteme robotice complexe. ROS a fost proiectat să fie un sistem flexibil, modular și distribuit, fiind ideal pentru dezvoltarea și integrarea componentelor software pentru aplicații în domeniul roboticii.

Chiar dacă ROS nu este un sistem de operare propriu-zis, el oferă un set standard de servicii asemănătoare celor oferite de sistemele de operare, cum ar fi abstractizarea hardware, controlul dispozitivelor la nivel scăzut, transmiterea de mesaje între procese și un sistem pentru gestionarea pachetelor software. Din acest motiv, ROS este considerat un meta-sistem de operare.

ROS folosește o arhitectură distribuită de tip peer-to-peer bazată pe comunicarea între procese printr-o rețea. Procesele sunt numite noduri, iar acestea comunică între ele printr-un sistem de mesagerie de tip publisher-subscriber, unde nodurile se pot abona la mesajele publicate de alte noduri pe canale numite subiecte (din engleză "topics").

Pentru dezvoltarea de software pentru ROS, acesta pune la dispoziție librării pentru o multitudine de limbaje de programare, cele mai populare fiind C/C++ și Python. ROS poate să ruleze pe mai multe sisteme de operare (Linux, Windows și macOS), cel mai utilizat fiind Linux, cu o preferință deosebită pentru distribuția Ubuntu deoarece este distribuția suportată oficial.

În acest proiect este folosit ROS 2, o versiune nouă de ROS introdusă pentru a adresa limitările primei versiuni și pentru a extinde funcționalitățile și platformele suportate.

2.2 Algoritmul de control PID

Algoritmul PID (proportional-integral-derivative) este un algoritm de control folosit pentru reglarea automată a unui proces astfel încât acesta să atingă o valoare de referință setată. Aceasta face parte din categoria algoritmilor de control cu buclă închisă deoarece reglarea procesului

controlat este realizată numeric folosind valori măsurate ale rezultatelor procesului. Acesta are o multitudine de aplicații practice cum ar fi controlul temperaturii, al presiunii, al poziției sau al vitezei. Pentru calculul valorilor de ieșire, algoritmul se folosește de trei componente: proporțională (P), integrală (I) și derivativă (D).

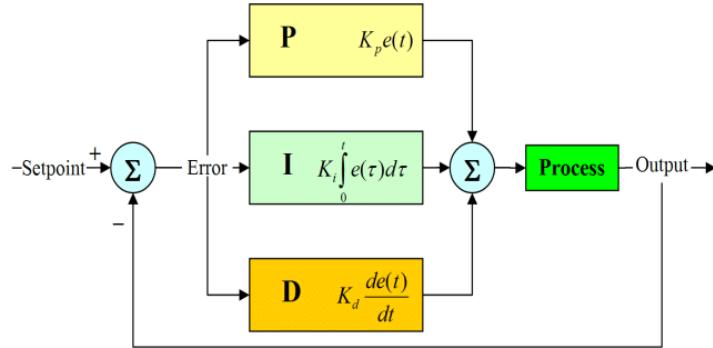


Figura 2.1: Schema bloc de funcționare a algoritmului PID[1]

2.3 Cartografiere și localizare simultană - SLAM

Prescurtarea SLAM provine din limba engleză de la "Simultaneous Localization and Mapping", în traducere "Cartografiere și Localizare Simultană". SLAM este o tehnică folosită în robotică, bazată pe construirea unei hărți a unui mediu necunoscut în timp ce se determină locația robotului. Aceasta este o funcție de mare importanță pentru roboții care operează în medii necunoscute și schimbătoare în timp, unde hărțile construite în avans nu sunt disponibile sau nu sunt îndeajuns de precise.

Harta rezultată în urma folosirii tehnicii SLAM poate fi folosită ulterior pentru diverse sarcini cum ar fi navigația autonomă și localizarea și evitarea obstacolelor.

SLAM este o problemă dificilă din punct de vedere hardware și computațional deoarece presupune ca robotul să își folosească senzorii (camera, LiDAR, sonar etc) pentru a colecta date despre mediu și să le proceseze în timp real pentru a construi harta în timp ce estimează locația proprie în raport cu harta.

Capitolul 3

Componenta hardware

Pentru a putea îndeplini sarcinile propuse, robotul are nevoie de un hardware puternic și eficient care să fie capabil să percepă mediul înconjurător într-un mod cât mai precis, să proceseze datele cu o viteză cât mai mare și să poată fi controlat cu o acuratețe cât mai bună. Astfel, în cele ce urmează vor fi prezentate componentele fizice din care este construit robotul.

3.1 Componentele hardware ale robotului

3.1.1 Șasiul robotului

Șasiul este componenta fizică structurală pe care sunt montate toate celelalte piese, care asigură mobilitatea și adaptabilitatea robotului la mediul înconjurător. Pentru acest proiect am folosit un șasiu prefabricat denumit "Wild Thumper", dezvoltat de compania Dagu Robot.

Caracteristicile deosebite ale șasiului Wild Thumper au fost elementul determinant pentru alegerea acestuia, având un sistem de tracțiune cu șase roți cu motoare și suspensii independente. Dimensiunile relativ mari ale șasiului au fost de asemenea un punct forte deoarece am avut nevoie de un spațiu suficient pentru montarea tuturor componentelor.

Mișcarea acestui șasiu se bazează pe principiul de direcție diferențială. Cele șase roți ale robotului nu au posibilitatea să își modifice unghiul pentru a executa un viraj, direcția în care se deplasează robotul fiind controlată de diferența de viteză a roților de pe fiecare parte. Acest principiu este folosit, spre exemplu, și în mișcarea tancurilor, în care senilele nu își pot schimba direcția dar diferența de viteză dintre cele două senile determină direcția de mișcare.

Folosind acest principiu de mișcare, vehiculele cu direcție diferențială sunt capabile inclusiv să se rotească pe loc prin rotirea roților/senilelor în direcții opuse.

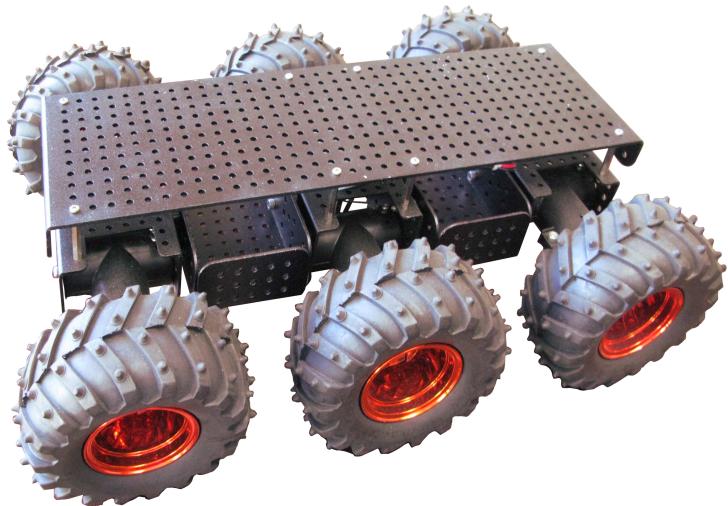


Figura 3.1: řasiul robotului

3.1.2 Calculatorul Nvidia Jetson Nano

Una dintre cele mai importante componente care fac parte dintr-un robot este unitatea de procesare. Robotii cu un grad scăzut de complexitate folosesc de obicei un microcontroller pe post de unitate de procesare, dar pentru a putea crește gradul de complexitate, robotii au nevoie de un procesor mai puternic. Cele mai folosite unităti de procesare în robotică sunt calculatoarele compuse dintr-o singură placă ("single board computer" - SBC) care rulează Linux. Cele mai populare SBC-uri sunt "Raspberry PI", calculatoare cu dimensiuni foarte reduse, eficiente din punct de vedere energetic și capabilități de procesare moderate.

Pentru acest proiect, în fazele de început ale dezvoltării am folosit un SBC Raspberry PI 4, dar după ce complexitatea proiectului a atins un anumit nivel am realizat că acesta nu reușește să îndeplinească standardele de viteză de procesare necesare pentru buna funcționare a robotului. Astfel am optat pentru un alt SBC produs de compania americană Nvidia, cunoscută pentru producția de plăci grafice pentru PC-uri. Aceasta este "Nvidia Jetson Orin Nano", din gama Jetson, o familie de SBC-uri speciale pentru aplicații cu inteligență artificială și calcul complex. Caracteristica deosebită a familiei de dispozitive Jetson este utilizarea unui procesor grafic puternic care are capacitatea de a accelera mult execuția modelelor de inteligență artificială.

Orin Nano este cel mai nou membru din familia Jetson și are performanțe foarte ridicate față de majoritatea SBC-urilor. Este foarte potrivit pentru acest proiect deoarece, pe lângă performanțele computaționale prezintă un număr mare de porturi de conectivitate, consum mic de curent, posibilitatea stocării datelor pe SSD M.2 și dimensiuni totale reduse.

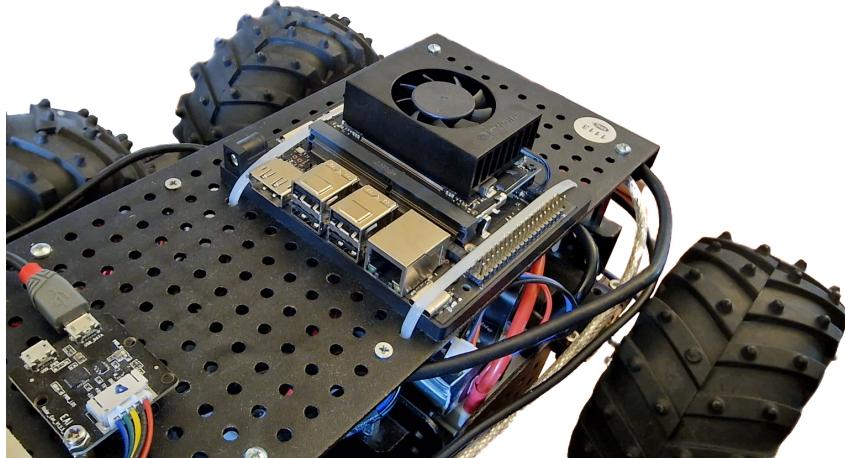


Figura 3.2: Nvidia Jetson Orin Nano montat pe șasiul robotului

3.1.3 Microcontrolerul Arduino Mega

Pentru ca software-ul de nivel înalt care rulează pe calculatorul Nvidia Jetson să poată comunica cu hardware-ul de nivel scăzut (cum ar fi motoarele, encoderele, servomotoarele), este necesar un microcontroller care să joace rolul de interfață hardware. Astfel, am ales să folosesc placă de dezvoltare cu microcontroller Arduino Mega, un produs bazat pe procesorul ATmega2560. Acesta se distinge prin numărul mare de pini de intrare/ieșire, memoria extinsă și capacitatea de comunicare pe patru interfețe seriale (UART) în același timp.

3.1.4 Senzorul LiDAR

Pentru a putea măsura mediul înconjurător și a construi harta acestuia, robotul are nevoie de un senzor de distanță multidirectional de precizie. Aceste cerințe sunt îndeplinite de senzorul LiDAR, un senzor de distanță pe bază de laser rotativ, astfel având un grad de libertate de 360 de grade. Prințipiu pe baza căruia funcționează LiDAR-ul este asemănător cu cel al unui RADAR, folosind pulsuri de lumină în locul undelor radio.

Modelul folosit pentru acest proiect este YDLidar X4, un senzor LiDAR fabricat de compania YDLIDAR, care se remarcă prin dimensiunea sa compactă, precizia ridicată și prețul accesibil.

Senzorul poate detecta obiecte la o distanță de până la 10 metri și are o precizie de până la 1 cm.

3.1.5 Camera Intel Realsense D435

Pentru recunoaștere/detectie de obiecte și calcularea poziției obiectelor detectate, robotul are nevoie de o cameră video și de o modalitate de a măsura distanța până la obiectele detectate. Componenta ideală pentru rezolvarea acestor probleme este o cameră video de adâncime.

Camerele de adâncime sunt denumite și RGB-D, unde RGB provine de la culorile de bază din care se compune o imagine color (Red-Green-Blue, adică Roșu-Verde-Albastru) iar D provine de la cuvântul din limba engleză "depth", adică "adâncime".

Camera Realsense D435

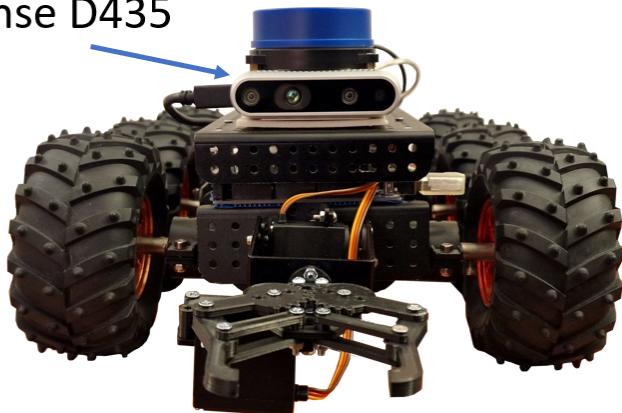


Figura 3.3: Camera Intel Realsense D435

Acum robot utilizează camera Intel RealSense D435, care folosește o combinație de senzori și tehnologie de procesare de imagini pentru a crea modele color 3D precise ale obiectelor și ale mediului înconjurător[2].

Principiul de funcționare al camerei RealSense D435 se bazează pe combinația dintre vederea stereo și un model de lumină infra-roșie proiectat pe obiectele din câmpul vizual al camerei. Astfel, sistemul de percepție al unității D435 este compus din:

- Două camere cu tonuri de gri cu lentilă fish-eye, poziționate la o distanță de 50mm una față de cealaltă, folosite pentru a calcula imaginile de adâncime
- Un proiectoare de raze infra-roșii folosit pentru a ilumina scena pentru îmbunătățirea percepției de adâncime
- O cameră color folosită pentru percepția culorilor scenei

3.1.6 Driverele de motoare Pololu Qik 2s12v10

Pentru a putea controla cele șase motoare de putere ale robotului este nevoie de un modul "motor driver". Deoarece curentul consumat de motoare este mare, un singur modul driver nu este suficient pentru a alimenta toate motoarele. Sunt utilizate două module identice, de tip Pololu Qik 2s12v10. Acestea sunt controlate prin interfață serială și au posibilitatea să controleze două canale simultan, având două circuite VNH2SP30-E de tipul "punte H" (H bridge). Puntea H este un circuit electronic format din patru tranzistoare, folosit în special pentru controlul direcției motoarelor de curent continuu prin inversarea polarității curentului.

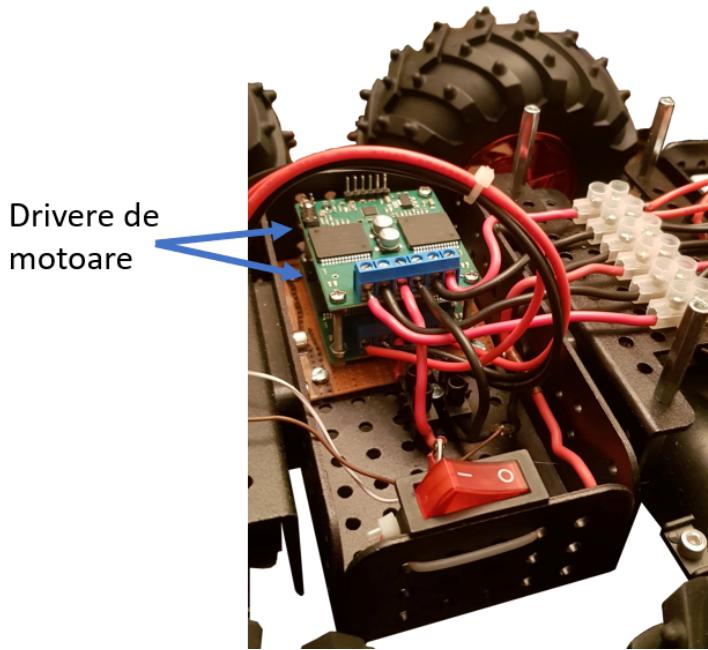


Figura 3.4: Drivere de motoare Pololu Qik 2s12v10 montate în șasiu

3.1.7 Regulator de curent Pololu S7V8A

Pinii microcontrolerului Arduino suportă un curent maxim de 40 mA. Acesta poate să controleze servomotoarele brațului robotic dar nu poate să le și alimenteze, deoarece servomotoarele consumă câte 500 mA fiecare. Astfel, servomotoarele au nevoie de o sursă de curent care să poată oferi minim 1A. Sursa reglabilă S7V8A este ideală pentru această sarcină deoarece la o tensiune de intrare de 7.4V provenită de la una dintre bateriile robotului, sursa poate asigura până la 2A cu o tensiune de ieșire de 5V.

3.1.8 Motoarele DC cu encodere

Şasiul robotului foloseşte şase motoare de curent continuu, cu cutie de demultiplicare cu raport 1:34. Pentru a putea măsura cu acurateţe viteza de deplasare a robotului şi distanţa parcursă, sunt utilizate codere la două dintre cele şase motoare. Şasiul robotului nu este echipat cu codere la roţi din fabrică, fiind necesară înlocuirea a cel puţin două motoare cu variante de motor cu encoder inclus.

Coderele motoarelor funcţionează pe baza efectului Hall, având câte un senzor Hall cu două canale fiecare. Senzorul detectează mişcarea unui magnet ataşat pe axul motorului şi trimite impulsuri electrice către microcontroller. Motoarele folosite în acest proiect au codere de 48 CPR (counts per rotation), ceea ce înseamnă că trimit 48 de impulsuri electrice la fiecare rotaţie completă a axului motorului. Având în vedere că rotaţia axului motorului este apoi demultiplicată cu un raport de 1:34, rezultă că la fiecare rotaţie finală a roţii robotului vor fi emise $48 * 34 = 1632$ de impulsuri electrice, oferind măsurători suficient de precise.

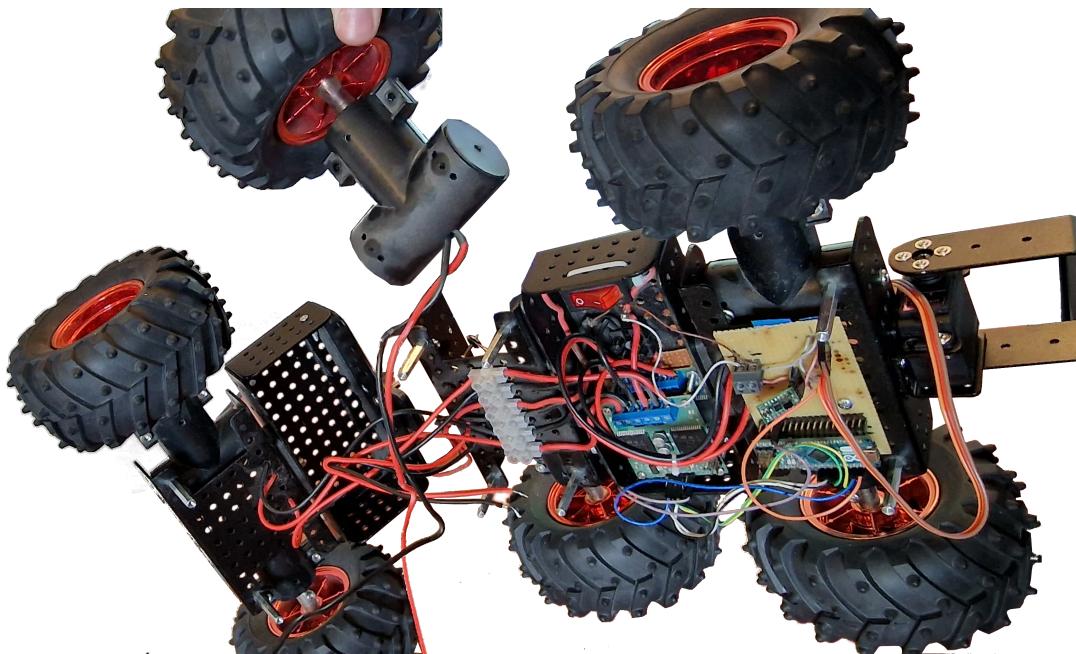


Figura 3.5: Șasiul robotului în timpul modificării

3.1.9 Bratul robotic și servomotoarele lui

Pentru manipularea obiectelor, robotul este dotat cu un braț robotic cu două articulații, controlate cu ajutorul a două servomotoare MG996R. Brațul robotic este format din două părți principale, una metalică și una realizată din plastic cu ajutorul unei imprimante 3D.

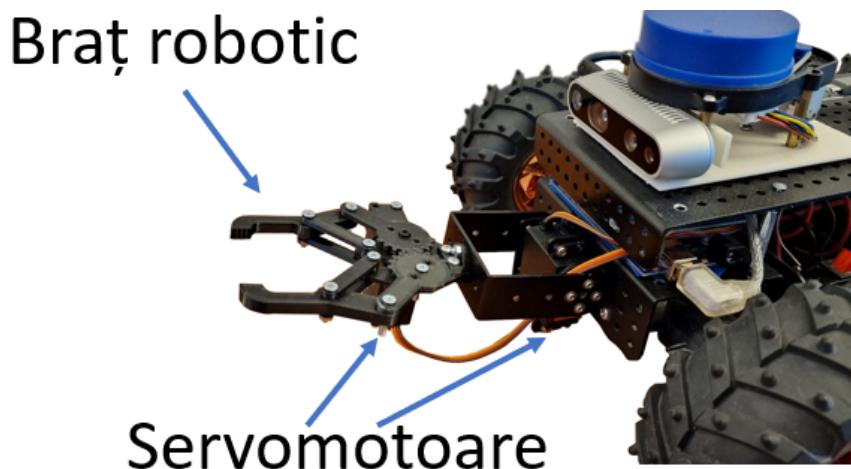


Figura 3.6: Brațul robotic montat pe șasiu

Componentele brațului robotic au fost realizate și asamblate special pentru proiect.

3.1.10 Bateriile

Pentru ca robotul să aibă o autonomie cât mai mare și o fiabilitate cât mai bună, alimentarea cu curent electric a sistemului de comandă se face separat față de sistemul de mișcare. Astfel, robotul are trei baterii la bord, fiecare alimentând un subansamblu diferit.

Bateriile folosite pentru alimentarea robotului și utilizările lor sunt:

- O baterie cu tehnologie litiu-polimer (LiPO) de 6500 mAh cu două celule, cu o tensiune nominală de 7.4V, este folosită pentru a alimenta cele șase motoare și servomotoarele
- O baterie LiPO de 3300 mAh cu trei celule, cu tensiune nominală de 11.1V alimentează micro-calculatorul Nvidia Jetson Orin Nano, care distribuie prin USB curentul și către camera de adâncime și microcontrolerul Arduino Mega
- Un power bank USB de 10000 mAh alimentează senzorul LiDAR care are nevoie de alimentare separată deoarece curentul care vine prin USB de la Jetson nu este îndeajuns pentru o funcționare normală

3.2 Conexiunile dintre componente

Componentele electronice care alcătuiesc acest robot pot fi împărțite la nivel conceptual în mai multe subansamble, după cum urmează:

- Ansamblul de comandă și control de nivel înalt, format din micro-calculator, cameră de adâncime și senzor LiDAR. Aceste componente sunt interconectate folosind interfețele USB pe care le pune la dispoziție micro-calculatorul.
- Ansamblul de comandă și control de nivel scăzut, care este format din microcontroler Arduino Mega, drivere de motoare Pololu Qik și encodere care măsoară mișcarea roților.
- Ansamblul de componente auxiliare, de alimentare și de mișcare, format din baterii, regulator de tensiune, motoare, servomotoare și întrerupător de curent continuu

În figura 3.7 este prezentată schema de conexiuni dintre componente, protocolele de comunicare folosite și împărțirea componentelor robotului pe subansamble.

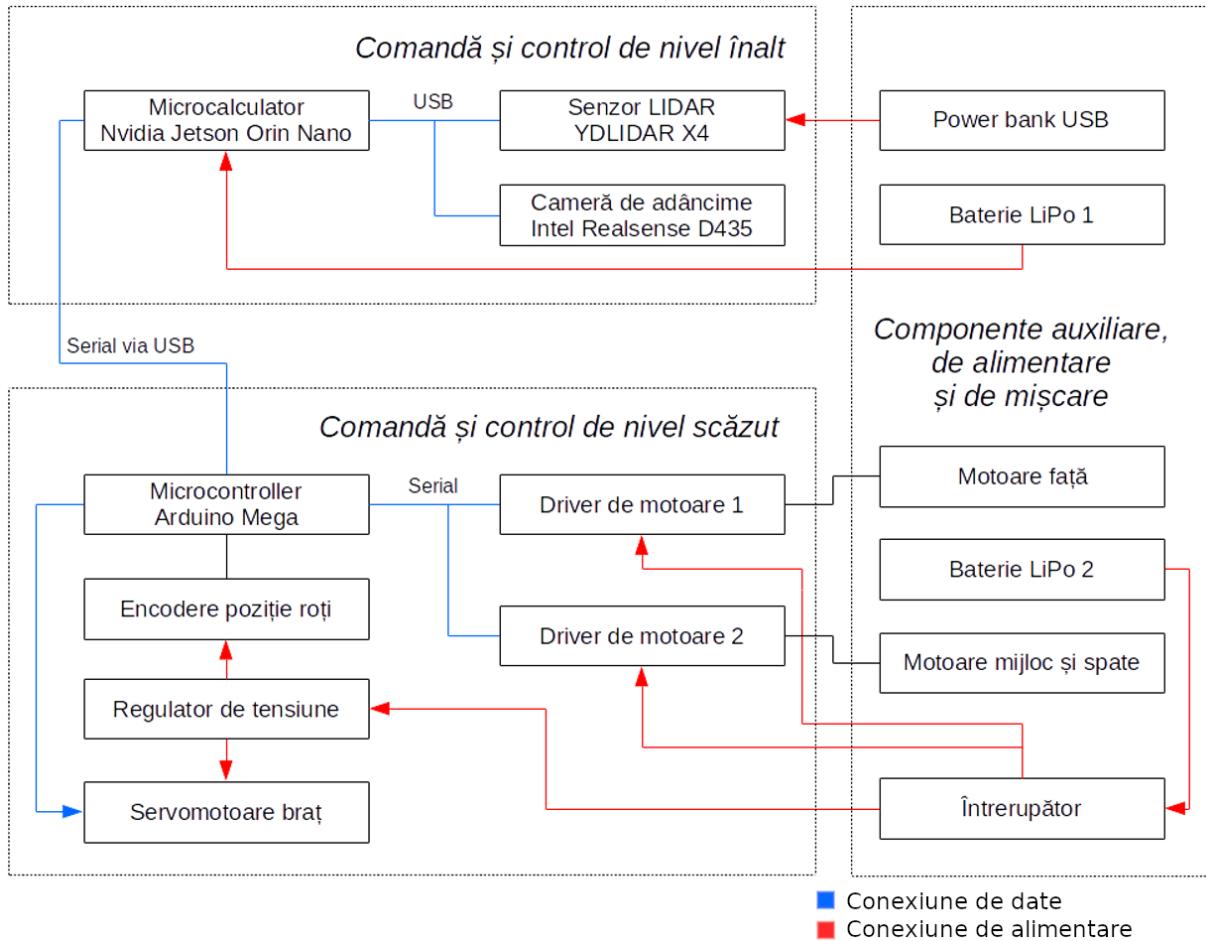


Figura 3.7: Schema conceptuală a conexiunilor dintre componentele electronice ale robotului

3.3 Asamblarea hardware a robotului

Elementele hardware au fost asamblate în mai multe etape, rezultând mai multe versiuni diferite ale robotului. De asemenea, în timpul dezvoltării componente hardware au fost întâmpinate probleme complexe care au trebuit soluționate pentru a putea continua dezvoltarea. De la probleme de aliniere ale componentelor pe șasiu și până la optimizarea spațiului astfel încât să încapă toate componente, soluțiile nu au fost niciodată triviale. Alte probleme semnificative întâmpinate au fost necesitatea modificării șasiului pentru acomodarea encoderelor la motoarele roților și necesitatea alimentării separate cu curent electric subansamblurilor, fapt care a condus la existența celor trei baterii diferențiale pe robot.

Pentru reducerea necesarului de spațiu pentru componente și conexiunile electronice a fost necesară dezvoltarea unei plăci de conexiune care să fie așezată direct în pinii plăcii de dezvoltare Arduino Mega. Această placă a fost realizată dintr-un cablaj de prototipare cu găuri, bare de pini, conectori cu șurub și conductori izolați. Pe această placă este apoi conectat regulatorul de tensiune cu ajutorul pinilor mamă. În figura ce urmează poate fi observată placa rezul-

tată montată pe microcontrolerul Arduino, care după construire au fost instalate între plăcile metalice ale șasiului, în partea frontală a robotului.

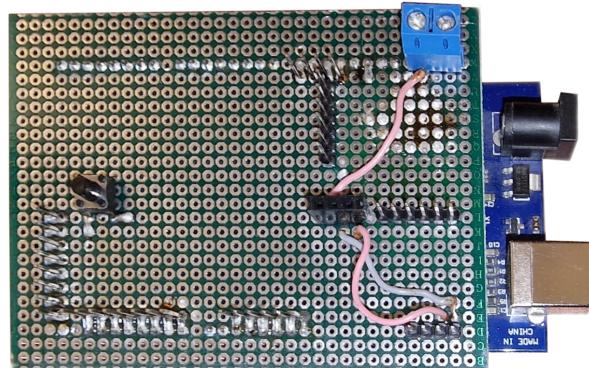


Figura 3.8: Placa de conexiuni a robotului montată pe Arduino

După rezolvarea tuturor problemelor apărute a rezultat produsul finit care se poate observa în figura 3.9.

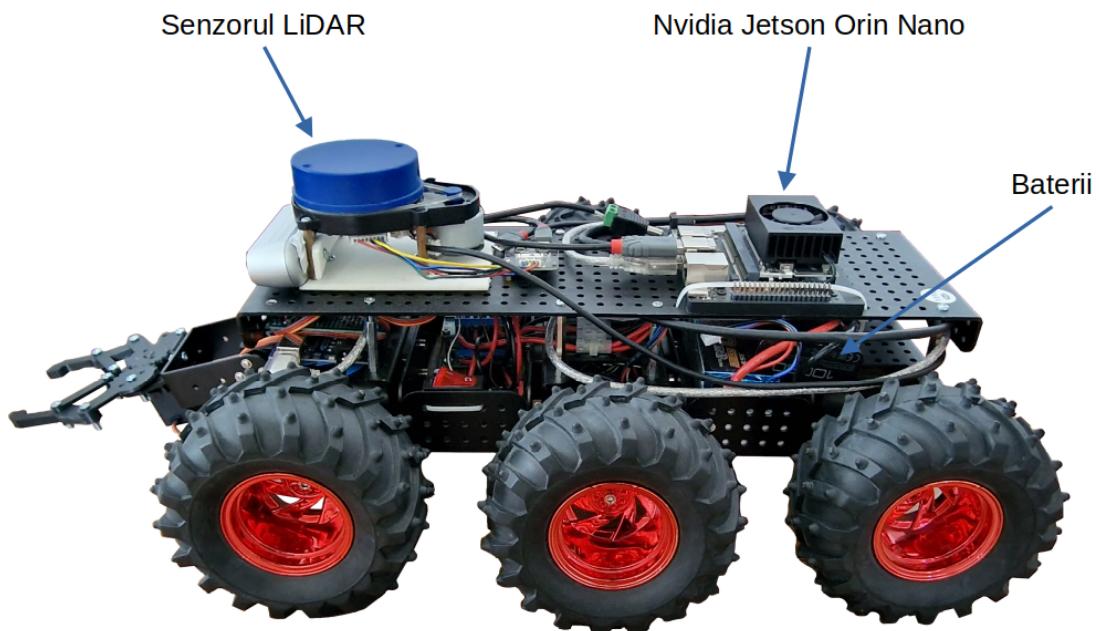


Figura 3.9: Vedere laterală a robotului

Capitolul 4

Componenta software

Acet capitol prezintă componentele software ale acestui proiect și argumentarea alegerilor făcute în privința limbajelor de programare folosite și ale librăriilor utilizate.

4.1 Arhitectura Software

Componenta software este împărțită în mai multe procese și noduri ROS care comunică între ele prin diverse modalități, formând la nivel conceptual un graf, prezentat în figura ce urmează.

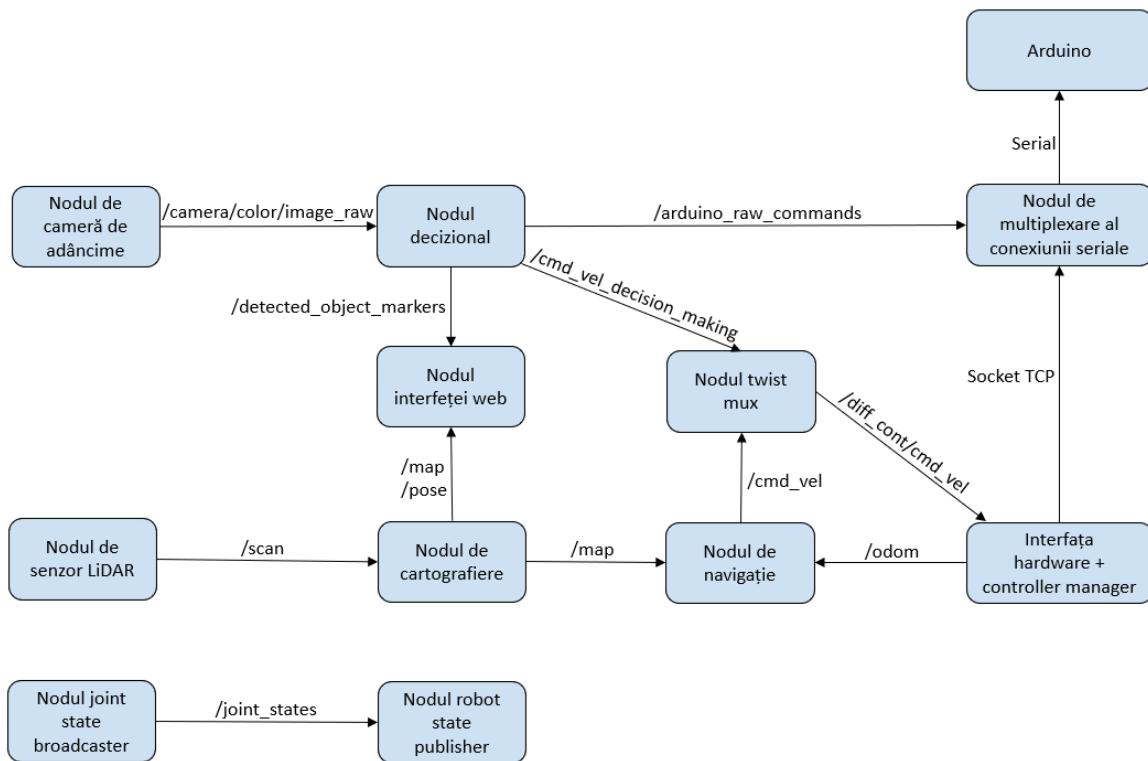


Figura 4.1: Diagrama arhitecturii software

4.2 Software-ul pentru Nvidia Jetson Orin Nano

4.2.1 Jetson Linux

Nvidia oferă pentru platforma Jetson propriul sistem de operare bazat pe distribuția de Linux Ubuntu, cuprins într-un kit de dezvoltare software (SDK) care include programe și drivere necesare pentru folosirea hardware-ului într-un mod cât mai eficient. Acest SDK este denumit JetPack iar sistemul de operare distribuit împreună cu acesta se numește "Jetson Linux".

Versiunea folosită în acest proiect este Jetson Linux 35.3.1, care face parte din JetPack SDK 5.1.1, bazată pe Ubuntu 20.04 LTS, o versiune de Ubuntu lansată în aprilie 2020 care beneficiază de suport pe o perioadă prelungită din partea dezvoltatorului. Am ales această variantă deoarece este versiunea suportată oficial de către Nvidia și de versiunea denumită "Foxy Fitzroy" a meta-sistemului Robot Operating System 2.

4.2.2 ROS 2

Robot Operating System (ROS) este un meta-sistem de operare folosit pe scară largă pentru dezvoltarea de sisteme robotice. Software-ul care alcătuiește ROS este grupat sub formă de pachete care pot fi instalate după nevoie fiecărui proiect. În general, dezvoltatorii de sisteme robotice scriu unul sau mai multe pachete ROS care conțin scripturile și programele necesare obținerii funcționalităților necesare în proiect. Una dintre cele mai importante caracteristici pe care le pune la dispoziție ROS este arhitectura distribuită și facilitarea comunicării între procese. Comunicând între ele, procesele alcătuiesc o structură de graf orientat, unde muchiile sunt căile de comunicare (denumite subiecte, en. "topic") iar procesele sunt nodurile grafului[3].

ROS pune la dispoziție o multitudine de pachete scrise de dezvoltatorul principal, dar un rol important îl are și comunitatea creată în jurul lui ROS, care pune la dispoziție alte pachete utile.

4.2.3 Nodurile și pachetele ROS folosite

În continuare sunt enumerate și explicate pachetele și nodurile folosite în dezvoltarea componentei software a acestui proiect:

- Nodul de LiDAR face parte din categoria nodurilor driver pentru hardware. Aceasta este pus la dispoziție de producător și folosește SDK-ul oferit pentru a comunica prin interfață serială USB cu hardware-ul propriu zis. Nodul ROS are rolul de a prelua datele citite de la senzor și de a le transmite pe topicul "/scan" pentru a fi recepționate de nodul de cartografiere.
- Nodul de cameră de adâncime este tot un nod de tip driver pentru hardware și este pus la dispoziție de producătorul camerei. Acesta face parte din pachetul `realsense_ros`

și are rolul de a receptiona stream-ul de imagini color și de adâncime de la camera Intel Realsense D435 și de a le publica apoi pe topicuri diferite.

- Nodul ”joint state broadcaster” citește periodic pozițiile articulațiilor robotului (în cazul acestui robot doar pozițiile roțiilor folosind informația de la encodere) și le publică pentru a fi citite de nodul ”robot state publisher”
- Nodul ”robot state publisher” este responsabil cu publicarea stării robotului în rețeaua ROS. Acesta oferă o interfață standardizată pentru reprezentarea stării, inclusiv pozițiile articulațiilor, vitezele lor și alte informații relevante. Acest nod este folosit de obicei împreună cu o descriere a robotului care este citită dintr-un fișier de tip URDF (”Unified Robot Description Format”). Fișierele URDF descriu structura robotului, inclusiv articulațiile, legăturile dintre articulații și relațiile dintre ele. Pe lângă fișierul URDF, acest nod citește și datele neprocesate de la nodul ”joint state broadcaster”, iar apoi folosind toate aceste date calculează transformările de coordonate pe care le va publica.
- ”Controller manager” (Managerul de control) este o componentă software care furnizează o interfață pentru gestionarea și coordonarea controlului senzorilor și actuatorilor unui robot. Este conceput pentru a gestiona ciclul de viață, configurația și execuția controlerelor responsabile cu comanda și cu citirea datelor din interfețele hardware ale robotului. Managerul de Control acționează ca un intermediar între hardware-ul robotului și componente de control de nivel superior, cum ar fi planificatorii de mișcare sau sistemele de percepție. Aceasta abstractizează detaliile specifice ale interfețelor hardware și furnizează o interfață unificată pentru controlul și interacțiunea cu robotul.
- Nodul ”Twist Mux” este un utilitar care oferă o modalitate convenabilă de a combina și multiplexa mai multe comenzi de viteză, cunoscute sub numele de ”twists” (acest termen provine de la faptul că în ROS mișcarea se controlează de obicei printr-o combinație de comenzi de viteză unghiulară și liniară), într-un singur twist de ieșire. Pachetul Twist Mux primește mai multe mesaje twist de la diferite surse, cum ar fi intrările de la un joystick sau tastatură, planificatoarele de traseu sau alte module de control, și le combină într-un singur mesaj twist care poate fi folosit pentru a controla mișcarea robotului.
- Nodul de cartografiere face parte din pachetul `slam_toolbox` și este responsabil cu recepționarea datelor de odometrie de la nodul ”robot state publisher”, a măsurătorilor laser de la nodul de LiDAR și construirea unei hărți 2D pe care o publică apoi pe topicul ”/map”. Aceasta rulează o serie de algoritmi complexi pentru ca în final să poată oferi o hartă cât mai precisă folosind resursele disponibile într-un mod cât mai eficient [4].
- Nodul de navigație autonomă și planificare de traseu face parte din pachetul ”Nav2” sau ”Navigation 2”, care oferă un set cuprinzător de instrumente și funcții pentru navigația autonomă a robotilor în medii complexe. Nav2 este conceput pentru a fi modular, flexibil

și scalabil, permitând dezvoltatorilor să creeze soluții de navigație personalizate, adaptate platformelor și cerințelor specifice ale robotilor [5].

Nav2 integrează o varietate de algoritmi și componente pentru a oferi un sistem robust de navigație a robotilor, inclusiv planificarea traseelor, evitarea obstacolelor, localizarea, cartografierea și controlul.

Fișierele URDF

În sistemul ROS, mai multe pachete au nevoie să primească informații despre configurația fizică a robotului. Pentru a răspunde la această necesitate a fost creat standardul URDF (Universal Robot Description Format). Acesta folosește fișiere cu un format bazat pe XML pentru a reprezenta robotii și componentele acestora. Fișierul URDF cuprinde o reprezentare ierarhică a structurii cinematice a robotului, inclusiv informații despre articulații, "legături" (en. "link") și senzori. Principalele componente ale unui fișier URDF sunt:

- Robot: elementul de bază al fișierului URDF este eticheta `<robot>`. Acesta conține descrierea generală a robotului, inclusiv numele său și legătura de bază.
- Legături: componente fizice ale robotului, cum ar fi corpurile rigide, sunt reprezentate utilizând etichetele `<link>`. Fiecare legătură primește un nume unic și poate avea proprietăți vizuale și de coliziune asociate. Legăturile pot fi conectate cu alte legături folosind articulații.
- Articulații: definesc conexiunile cinematice între legăturile robotului. Ele specifică mișcarea relativă între două legături și pot fi de diverse tipuri, precum de revoluție, prismatice, continue, etc. Articulațiile sunt definite utilizând eticheta `<joint>` și includ proprietăți precum nume, tip, conexiuni de legătură părinte și copil, și limite ale articulației.

Fișierele URDF mai pot să conțină și alte tipuri de elemente care sunt utile în special în simulările sistemelor robotice. Aceste caracteristici includ geometria utilizată în scopul de a detecta coliziuni, proprietățile inerțiale ale componentelor robotului și date despre senzori.

În acest proiect specificația URDF este necesară în procesul de cartografiere și navigație autonomă. Procesul "robot state publisher" utilizează fișierele URDF pentru a publica descrierea fizică a robotului, pe care mai apoi nodul de cartografiere o folosește pentru o localizare mai bună a robotului în relație cu harta și nodul de navigație o folosește pentru cunoașterea dimensiunilor robotului pentru evitarea coliziunilor.

Robotul prezentat în această lucrare este descris în fișierul URDF ca fiind compus dintr-un paralelipiped care modelează șasiul, două roți (au fost reprezentate doar rotile despre care avem informații de poziție de la encodere) și un cilindru care reprezintă senzorul LiDAR. Brațul robotic nu este inclus în descriere deoarece nu are senzori de poziție care să poată citi coordinatele exacte ale articulațiilor.

Cea mai importantă funcție pe care o pune la dispoziție folosirea standardului URDF este sistemul de transformări de coordonate. Știind configurația fizică a robotului, ROS poate să calculeze coordonate relative la orice componentă a robotului. Spre exemplu, dacă senzorul LiDAR este montat în partea din față a robotului, măsurătorile de distanță vor fi relative la poziția senzorului. Construcția hărții trebuie să aibă originea în centrul amprentei la sol a robotului. Datorită descrierii fizice URDF, sistemul ROS este capabil să transforme coordonatele și valourile măsurătorilor pentru a obține măsurători relative la centrul robotului.

În figura de mai jos este vizualizat modelul robotului împreună cu originile articulațiilor fiecărei legături folosind utilitarul `rviz2` pus la dispoziție de ROS. Pentru o vizualizare mai bună a articulațiilor, transparența modelului a fost redusă la 0.5.

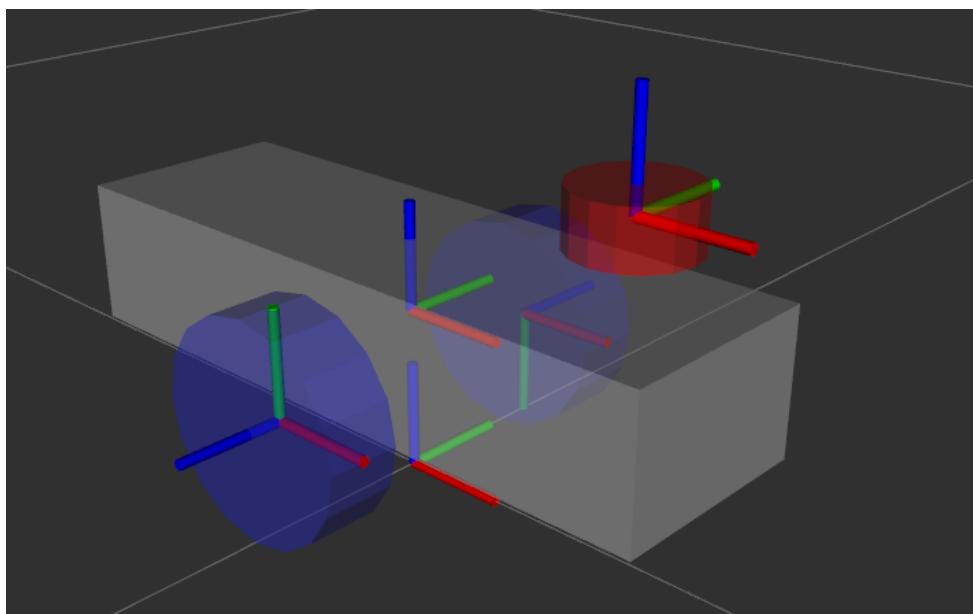


Figura 4.2: Modelul URDF al robotului

4.2.4 Nodul de cartografiere

În figura 4.3 este prezentată ca exemplu o hartă realizată de robot, vizualizată cu ajutorul utilitarului ROS `rviz`. De asemenea, se poate observa că poziția robotului în spațiu este marcată folosind modelul robotului preluat din reprezentarea în formatul URDF.

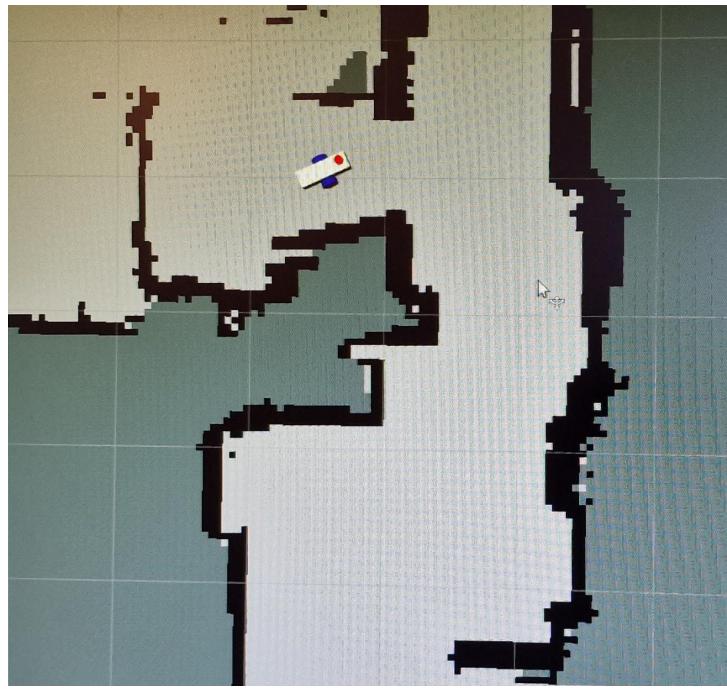


Figura 4.3: Exemplu de hartă realizată de robot și poziția acestuia

4.2.5 Nodurile și pachetele ROS dezvoltate în proiect

Interfața ROS cu hardware-ul proprietar

ROS este un sistem versatil și portabil care trebuie să se adapteze în funcție de hardware-ul pe care îl controlează. Din cauza variabilității foarte mari a componentelor hardware și a arhitecturii alese de către fiecare dezvoltator, hardware-ul nu poate avea o interfață de control standardizată. În aceste condiții, dezvoltatorii ROS au ales să ofere posibilitatea scrierii unei interfețe software care să faciliteze comunicarea dintre sistemul ROS și hardware-ul divers al fiecărui robot.

În versiunea curentă de ROS, implementarea acestei interfețe este posibilă doar în limbajul C++ și este realizată prin folosirea mecanismului de moștenire din programarea orientată pe obiecte. Astfel, programatorul trebuie să creeze o clasă care să implementeze o interfață de tipul `hardware_interface::BaseInterface<SystemInterface>`

Interfața implementată conține mai multe metode care trebuie suprascrise și configurață interfețele de stare și de comandă de implementat. Interfețele de stare sunt responsabile cu citirea valorilor de stare ale robotului (cum ar fi poziția roților) iar interfețele de comandă sunt cele prin care se pot transmite comenzi către hardware (spre exemplu comenzi de mișcarea motoarelor)

În acest proiect, implementarea acestor interfețe este atipică, deoarece în locul comunicării directe cu hardware-ul, din interfețe se trimit/primește fluxul de date prin intermediul unui socket TCP conectat la un alt nod ROS care se ocupă cu trimiterea efectivă a datelor prin conexiunea serială către microcontrolerul Arduino. Motivele alegerii acestei soluții tehnice sunt detaliate în

descrierea următorului nod ROS, cel de multiplexare a conexiunii.

Nodul pentru multiplexarea conexiunii seriale cu Arduino Mega

Conexiunea serială prin care comunică calculatorul Jetson cu microcontrolerul Arduino poate fi folosită de către un singur proces la un anumit moment de timp. O problemă întâmpinată în timpul dezvoltării componentei software a fost transmiterea comenziilor de mișcare de la mai multe procese către Arduino. Spre exemplu, nodul de navigație autonomă trimite și primește mesaje cu o viteză de aproximativ 20 Hz, trimițând comenzi legate de controlul motoarelor robotului. Atunci când nodul decizional are nevoie să trimită comenzi către servomotoarele brățului robotic apare problema comunicării simultane a mai multor procese printr-o singură interfață serială.

Această problemă a fost rezolvată prin dezvoltarea în limbajul Python a unui nod ROS care să multiplexeze comenziile venite de la mai multe procese și să le trimită mai departe către conexiunea serială. Acest nod de multiplexare este singurul care poate comunica prin interfață serială, iar celelalte noduri care au nevoie să comunice trebuie să trimită mesajele către acest nod.

Transmiterea mesajelor de la alte noduri către acest nod de comunicare se realizează în mai multe moduri, alese în funcție de natura fiecărui nod:

- Nodul "hardware interface" nu este un nod propriu zis, el este implementarea interfeței hardware necesară pentru ca alte noduri ROS să poată comunica cu hardware-ul. Deoarece nu are statut de nod, acesta nu poate să trimită mesaje pe topicuri ROS. Din acest motiv am ales ca mediul de comunicare dintre "hardware interface" și nodul responsabil cu conexiunea serială să fie un socket TCP.
- Nodul decizional, fiind un nod ROS propriu zis, comunică cu nodul de multiplexare al conexiunii seriale prin topicul "/arduino_raw_commands", prin mesaje ROS de tip std_msgs/String .

Nodul decizional

Nodul decizional este unul dintre cele mai importante noduri din arhitectura acestui proiect. Aceasta se prezintă sub forma unui nod ROS implementat în limbajul Python care are rolul de a stabili modurile de funcționare și comportament ale robotului. În timpul funcționării, robotul trece prin mai multe stări care determină comportamentul robotului. Totodată, indiferent de starea în care se află, nodul decizional are unele atribuții de bază permanente, rulate pe fire de execuție (thread-uri) diferite pentru a putea menține performanța de care este nevoie în procesul decizional fără a fi încetinit de sistemul de primire de date de la alte noduri și procesarea lor.

La inițializarea nodului, acesta se abonează la topicul de imagini al camerei de adâncime. La

fiecare imagine receptionată este rulată rutina de recunoaștere de obiecte care după detecție salvează rezultatele într-o variabilă la care are acces și firul de execuție decizional și calculează distanța până la obiectele detectate. Imaginele sunt apoi re-publicate pe alt topic după ce sunt desenate rezultatele detecției pe ele. De asemenea, funcția de initializare pornește și firul de execuție dedicat buclei de control.

Atunci când sunt detectate, coordonatele obiectelor de interes sunt salvate într-o variabilă de tip listă din nodul decizional și sunt publicate periodic pe topicul ROS

/detected_object_markers sub formă de mesaje de tipul visualization_msgs/Marker cu scopul de a fi afișate pe hartă în interfața web. Deoarece detectia obiectelor este realizată în fiecare frame individual, există problema tratării unui obiect ca mai multe obiecte diferite dacă este detectat în mai multe frame-uri consecutive. Astfel apare necesitatea filtrării detecțiilor după coordonatele estimate ale obiectului. În momentul în care se detectează un obiect trebuie estimate coordonatele sale relativ la harta construită în funcție de poziția robotului în hartă. După acest pas este necesară parcurgerea listei de obiecte detectate și compararea coordonatelor lor. Dacă două detectii sunt foarte apropiate există o probabilitate foarte mare ca acestea să se refere la același obiect, și deci una dintre ele trebuie eliminată.

Modurile de rulare ale robotului (stările) sunt următoarele:

- Modul de navigație semi-autonomă - În acest mod, nodul decizional este în starea de căutare a obiectelor de interes. Firul de execuție al buclei de control folosește detecțiile realizate la primirea imaginilor și calculează coordonatele obiectelor în raport cu harta, pe care le publică pentru a putea fi afișate în interfața web. Totodată, în acest mod mișcarea robotului este controlată de nodul de navigație, controlat de utilizator prin setarea de coordonate întărite pe hartă.
- Modul de apropiere de obiect - Când utilizatorul selectează în interfața web un obiect afișat pe hartă pentru a fi apucat cu brațul robotic, nodul decizional trece în modul de apropiere. Aceasta preia comanda de la nodul de navigație și trimită comenzi către motoare în mod direct astfel încât să se poată apropiă de obiect până la o distanță convenabilă de la care poate începe procedura de apucare, după care trece automat în modul de corectarea unghiului.
- Modul de corectarea unghiului de apropiere de obiect este modul de funcționare în care robotul se întoarce în loc cu mișcări fine pentru a potrivi unghiul de apropiere de obiectul care trebuie apucat cu scopul de a alinia brațul robotic deschis cu obiectul. După ce unghiul devine corect, robotul trece automat în modul de apucare obiect.
- Modul de apucare obiect - Robotul se apropii încet în linie dreaptă până când obiectul care trebuie apucat intră în interiorul brațului robotic, după care strâng și ridică brațul pentru a apuca obiectul, apoi trece în modul de obiect apucat.

- Modul ”obiect apucat” este un indicator că robotul a parcurs toată procedura de apucare și acum ține un obiect în brațul robotic. În acest mod, nodul decizional oferă din nou controlul motoarelor către nodul de navigație și deci către utilizator, care poate selecta un loc pe hartă unde să fie lăsat obiectul. După ce robotul navighează la locul selectat de utilizator, dă drumul la obiect și nodul decizional trece înapoi în modul de navigație.

Pentru detecția de obiecte de interes în imagini este utilizat YOLOv5, un framework bazat pe deep learning, implementat folosind librăria PyTorch. Nodul decizional folosește un model de detecție YOLOv5 pre-antrenat, în care a fost reantrenat ultimul strat de neuroni pentru adaptarea la obiectele de interes ale acestui proiect [6]. În scop demonstrativ, pe post de obiecte de interes am folosit niște ciuperci albe din plastic realizate cu ajutorul unei imprimante 3D. În figura următoare se poate observa modelul 3D după care a fost printat obiectul și obiectul în sine rezultat după printare.



Figura 4.4: Comparație între modelul 3D și obiectul printat

Motivul pentru care a fost ales acest tip de obiect este absența posibilității de măsurare a forței de strângere a brațului robotic. În aceste condiții, brațul funcționează cu distanțe prestatibile, adaptate la obiectul țintă.

Antrenarea rețelei YOLOv5 a fost realizată pe un set de 331 de imagini, din care 34 au fost folosite ca date de test iar 33 au fost folosite ca date de validare. Imaginele au fost adnotate manual folosind programul open-source LabelImg, iar rețeaua a parcurs algoritmul de antrenare timp de 100 de iterări. În timpul antrenării, algoritmul calculează după fiecare pas valoarea funcției denumită ”loss”. Aceasta este compusă din trei componente majore:

- Loss-ul de localizare (box loss / localization loss) este componenta care măsoară acuratețea localizării predicțiilor în imagine. YOLO prezice localizarea casetelor în care se încadrează obiectele detectate folosind o regresie, unde modelul răspunde cu patru valori (x, y, lățime, înălțime) pentru fiecare casetă de încadrare (en. ”bounding box”). Loss-ul de localizare cuantifică discrepanța dintre bounding box-urile prezise și cele adevărate din datele de antrenare.

- Loss-ul de încredere (confidence loss): YOLO prezice scorul de încredere, care reprezintă probabilitatea ca bounding box-ul să conțină un anumit obiect.
- Loss-ul de clasificare (class loss) este componenta care măsoară discrepanța dintre probabilitățile claselor prezise și clasele adevărate prezente în datele de antrenare.

În acest proiect, obiectele de interes fac parte dintr-o singură clasă, astfel loss-ul de clasificare este întotdeauna nul, pentru că rețeaua clasifică întotdeauna corect, având de făcut o alegere care are întotdeauna o singură opțiune. În graficele ce urmează se pot observa cele trei componente ale funcției loss și evoluția lor în funcție de progresul iteratiilor antrenării. Pe abscisa graficelor se află numărul epocii de antrenare iar pe ordonată valoarea loss-ului corespunzător.

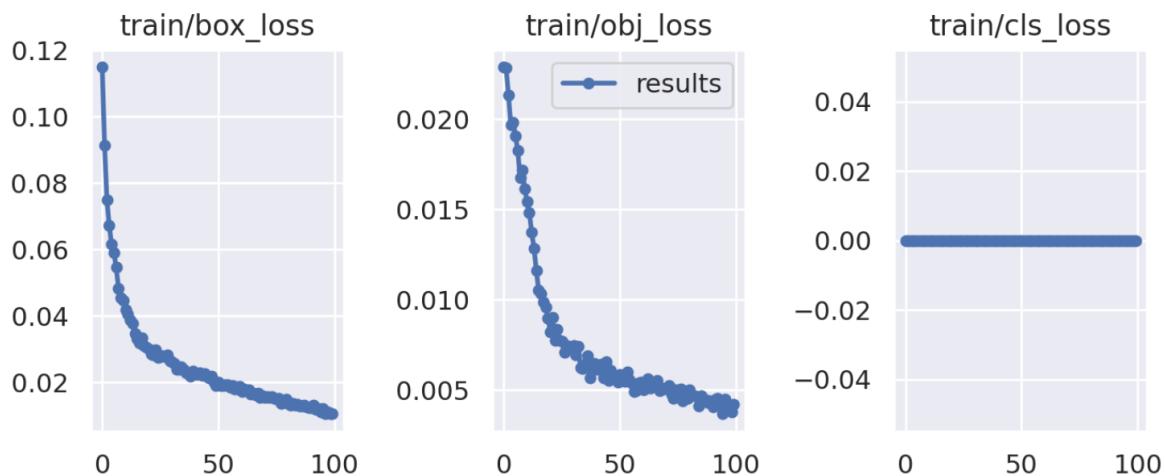


Figura 4.5: Componentele funcției loss

Pentru rezultate cât mai bune, înainte de procesul propriu-zis, sistemul YOLO parurge și niște pași de augmentare a datelor, în care imaginile de antrenare sunt preprocesate, combinate, suprapuse, rotite și scalate pentru a avea o diversitate cât mai bună a datelor de antrenare, pentru ca rețeaua să aibă capabilități de generalizare cât mai bune. Un exemplu de imagine cu date augmentate produsă de rețea în timpul antrenării este prezentă în figura 4.6.

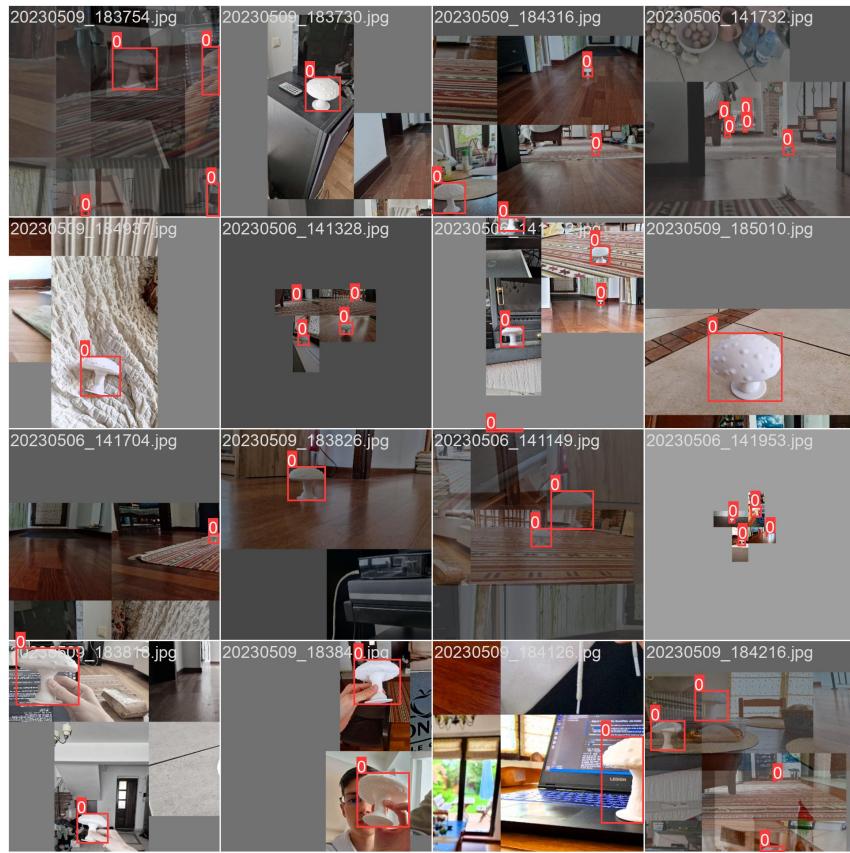


Figura 4.6: Augmentarea datelor de antrenare

Pentru evaluarea finală a preciziei modelului este folosită și metrica de precizie calculată automat de sistemul YOLO. În figura 4.7 se poate observa evoluția metricii de precizie în funcție de numărul de iterații de antrenare (epoci).

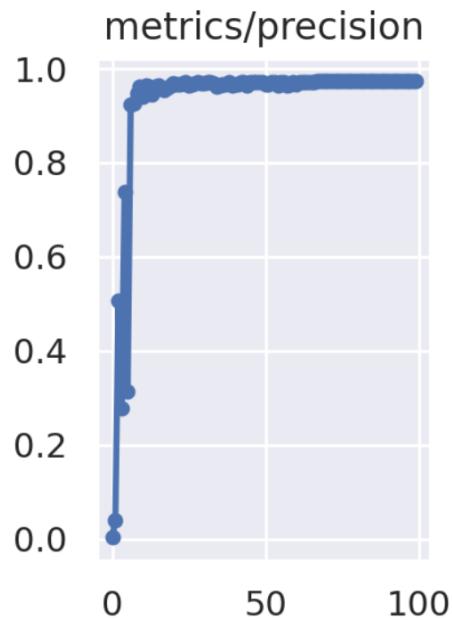


Figura 4.7: Evoluția preciziei în timpul antrenării

Modelul antrenat reușește să detecteze obiectele cu o precizie bună și o viteză îndeajuns de bună încât să poată fi folosit pe un flux de imagini în timp real. O imagine în care este detectat și evidențiat un obiect este prezentă în figura 4.8.

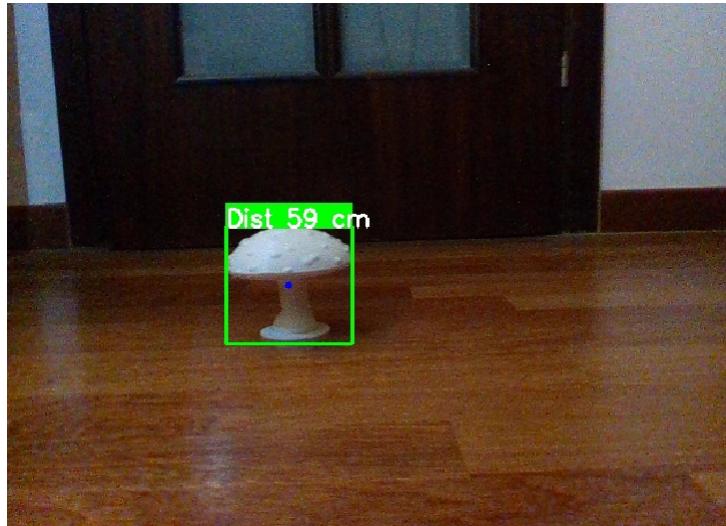


Figura 4.8: Obiect detectat folosind YOLO

Nodul pentru interfața web

Controlul robotului se realizează prin interfața web, aceasta fiind o variantă comodă și foarte flexibilă pentru crearea unui panou de control. În interfața web sunt prezente elemente precum harta mediului construită de robot, fluxul de imagini de la cameră în care sunt evidențiate și obiectele de interes și butoane pentru controlul parametrilor nodului decizional.

Interfața web este dezvoltată folosind Node.js împreună cu librăria client ROS pentru acesta, numită `rclnode.js`. Astfel, nodul ROS corespunzător interfeței web servește pagina de interfață folosind librăria `express` și, atunci când este inițializat, se abonează la topicurile ROS de hartă (`/map`), de poziție a robotului în mediu (`/pose`), de poziție a obiectelor detectate și de imagini (`/object_detection_image`). O dată ce este accesată dintr-un browser web, pagina de control pornește niște funcții cu apel periodic, care au rolul de a reîmprospăta datele afișate în pagină. Aceste funcții trimit o serie de cereri HTTP care cer date de la serverul Node.js. În pagina de control sunt folosite librăriile `bootstrap`, `CSS` și `jQuery` pentru a îmbunătăți aspectul și funcționalitățile paginii.

Harta afișată în pagină este desenată cu ajutorul unui obiect HTML5 de tip `canvas` și este actualizată de fiecare dată când sunt disponibile date noi. Mesajele ROS care conțin datele hărții sunt primite sub formă de JSON de la nodul de cartografiere și au tipul de date `nav_msgs/OccupancyGrid`. Acestea conțin o matrice bidimensională în care valoarea fiecărei celule reprezintă probabilitatea ca acea celulă să fie ocupată. Pe lângă matricea propriu-zisă, fiecare mesaj conține și date despre mărimea și precizia hărții (exprimată în metri/celulă), coordonatele în care se află originea hărții și orientarea ei.

Pe hartă este desenată și poziția și orientarea actuală a robotului. Această funcționalitate este realizată prin abonarea la topicul ROS /pose care servește mesaje de tipul geometry_msgs/Pose, care sunt cerute periodic de pagina web folosind o cerere HTTP GET. Aceste mesaje conțin poziția și orientarea robotului relative la hartă. De asemenea, coordonatele obiectelor detectate sunt marcate pe hartă folosind datele primite prin abonarea la topicul ROS /detected_object_markers.

Figura 4.9 reprezintă o captură de ecran preluată din interfața web de control a robotului.

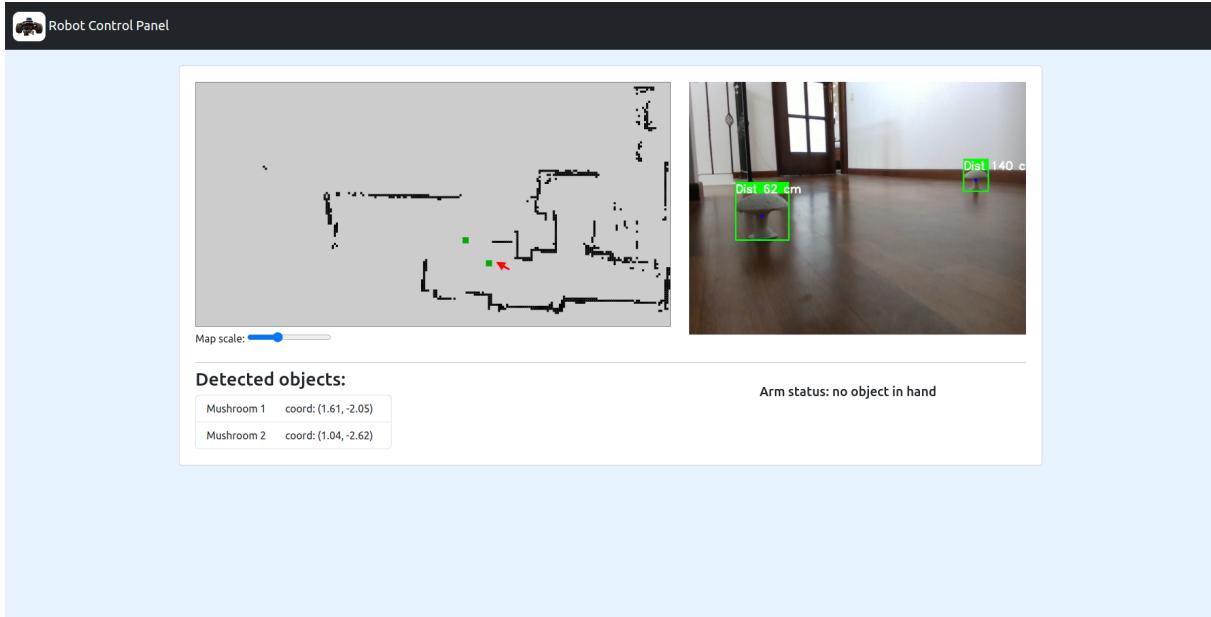


Figura 4.9: Interfata web a robotului

4.3 Software-ul care rulează pe microcontrolerul Arduino Mega

Microcontrolerul Arduino Mega rulează cod implementat cu ajutorul mediului de dezvoltare Arduino IDE. Rolul principal al microcontrolerului este de a executa comenzi primite prin interfața serială și de a răspunde cu date legate de poziția roțiilor atunci când sunt cerute. Comenziile au o structură simplă, majoritatea fiind formate dintr-o singură literă și parametri separați prin caracterul spațiu. Finalul unei comenzi este marcat de caracterul "carriage return" care are codul ASCII 13. Comenziile folosite sunt formatare în felul următor:

- m viteza_stanga viteza_dreapta - Această comandă este folosită pentru controlul vitezei motoarelor robotului. Atunci când este primită, parametrii de viteză sunt trimiși mai departe către controlerul PID. Valorile viteza_stanga și viteza_dreapta sunt exprimate în rotații pe minut ale roțiilor. Algoritmul PID se ocupă de controlul vitezei motoarelor astfel încât mișcarea rezultată să fie cât mai apropiată de cea dorită.

- e - Această comandă este o cerere pentru poziția absolută a roților măsurată cu ajutorul encoderelor. Când este primită această cerere, microcontrolerul răspunde cu valorile citite de la encodere în formatul `pozitie_stanga pozitie_dreapta`.
- s id_servomotor valoare - Această comandă este folosită pentru controlul servomotoarelor care acționează brațul robotic. Parametrul `id_servomotor` este numărul corespunzător servomotorului care se vrea a fi acționat, iar parametrul `valoare` reprezintă un unghi exprimat în grade, cuprins între 0 și 180. Servomotorul cu id-ul 0 este cel care mișcă brațul robotic pe verticală, ajutând la ridicarea obiectelor iar cel cu id 1 este cel care acționează strângerea brațului.

Viteza motoarelor este controlată folosind câte un algoritm PID pentru fiecare parte a robotului. Astfel, se rulează două instanțe ale algoritmului, una pentru controlul vitezei roților de pe partea stângă și una pentru roțile de pe partea dreaptă. Ca date de intrare, algoritmul acceptă o viteză exprimată în rotații pe minut (RPM). După fiecare comandă de viteză trimisă către motoare, algoritmul măsoară viteza roților folosind codurile motoarelor și ajustează comanda următoare pentru a se apropiă cât mai mult de valoarea primită prin interfața serială.

Capitolul 5

Concluzii

Proiectul prezentat în această lucrare este rezultatul unei evoluții a platformei hardware și a soluției software pe baza unor etape. Într-o primă fază, arhitectura generală a sistemului s-a bazat pe achiziția datelor pe SBC-ul robotului (a fost utilizat un Raspberry Pi 4), cu procesarea acestora pe un calculator extern. Concluzia acestei abordări a fost că resursele de comunicație necesare (cu limitările aferente unei rețele Wi-Fi) sunt prea mari, în multe situații aducând întârzieri semnificative care au afectat semnificativ performanțele sistemului. Pe baza acestei concluzii, arhitectura a fost modificată astfel încât procesarea datelor achiziționate să aibă loc pe hardware-ul robotului, utilizând un SBC mai puternic (Nvidia Jetson Orin Nano). În acest fel comunicația prin canalul Wi-Fi se limitează la comenzi de control al robotului și respectiv transmisia imaginilor și reprezentarea datelor de cartografie către utilizator. Noua abordare a eliminat întârzierile apărute în îndeplinirea sarcinilor robotului, eventualele întârzieri care ar mai putea apărea astfel fiind limitate strict la reprezentarea datelor în interfață.

Ca proiect demonstrativ, robotul ar putea beneficia de generalizarea topologiei obiectelor țintă, obiectele pe care le poate recunoaște și manipula. În acest scop, ar fi necesare următoarele îmbunătățiri:

- adăugarea unor senzori de presiune/forță de apucare la nivelul brațului robotic
- reantrenarea modulului YOLO pentru detectarea unei varietăți de clase de obiecte
- extinderea modelului URDF al robotului pentru a include noi seturi de date provenite de la senzori

Consider că arhitectura actuală a robotului, realizată cu scop demonstrativ, este un punct bun de plecare pentru eventuale proiecte cu aplicabilitate practică: în industrie, îmbunătățirea calității vieții (de exemplu pentru asistență cotidiană) etc. Cu adaptările necesare, platforma este suficient de flexibilă încât să poată fi extinsă cu funcționalități noi și hardware specializat pe aplicație. Software-ul este scris utilizând tehnologii care asigură portabilitatea pe platforme hardware superioare (sau restrânse, în funcție de aplicație).

Bibliografie

- [1] Nikunj Mehta et al. “Design of HMI Based on PID Control of Temperature”. In: *International Journal of Engineering Research and V6* (May 2017). DOI: [10.17577/IJERTV6IS050074](https://doi.org/10.17577/IJERTV6IS050074).
- [2] Leonid Keselman et al. *Intel RealSense Stereoscopic Depth Cameras*. 2017. arXiv: [1705.05548 \[cs.CV\]](https://arxiv.org/abs/1705.05548).
- [3] Steven Macenski et al. “Robot Operating System 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: [10.1126/scirobotics.abm6074](https://doi.org/10.1126/scirobotics.abm6074). URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>.
- [4] Steve Macenski and Ivona Jambrecic. “SLAM Toolbox: SLAM for the dynamic world”. In: *Journal of Open Source Software* 6.61 (2021), p. 2783. DOI: [10.21105/joss.02783](https://doi.org/10.21105/joss.02783). URL: <https://doi.org/10.21105/joss.02783>.
- [5] Steven Macenski et al. “The Marathon 2: A Navigation System”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.
- [6] Glenn Jocher et. al. *ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support*. Version v6.0. Oct. 2021. DOI: [10.5281/zenodo.5563715](https://doi.org/10.5281/zenodo.5563715). URL: <https://doi.org/10.5281/zenodo.5563715>.

Anexă - Listă de acronime

ROS - Robot Operating System

SLAM - Simultaneous Localization and Mapping (Localizare și Cartografiere Simultană)

SDK - Software Development Kit

SBC - Single Board Computer

LiDAR - Laser imaging, Detection, And Ranging

URDF - Unified Robotics Description Format

TCP - Transmission Control Protocol

YOLO - You Only Look Once

Wi-Fi - Wireless Fidelity - Protocol pentru internet fără fir