

VEHICUL AUTONOM CU TRACTIUNE INTEGRALĂ

Pătrașcu Valentin

Abstract

Odată cu creșterea tehnologică progresivă, noi domenii și arii din știință și tehnologie au ajuns accesibile pentru a fi dezvoltate de publicul larg. Unul dintre aceste domenii a fost reprezentat de construirea roboților autonomi, care odată cu schimbarea majoră apărută în domeniul inteligenței artificiale la începutul anilor 2010, a avut o și mai mare implicare venită din domeniul de cercetare.

Această lucrare tratează subiectul mașinilor pilotate în mod autonom, subiect ce și-a găsit aplicabilitatea practică în diverse sectoare economice, de la transportul de mărfuri și de persoane până la explorarea suprafețelor planetelor.

Abordarea utilizată folosește fuziunea informațiilor primite de la mai mulți senzori (LiDAR, Cameră RGB și senzorilor de mișcare) pentru a transforma informațiile, stimuli și modificările apărute din mediul exterior într-un comportament specific. Platforma hardware pe care se construiesc funcționalitățile robotului este modulară, scalabilă și permite modificarea ansamblului în funcție de domeniul în care va activa robotul.

Folosind un mecanism precis de cartografiere și localizare simultană pentru generarea traiectoriilor, împreună cu acuratețea introdusă în sistem prin intermediul unei rețele convoluționale, lucrarea prezintă un model robust și scalabil de robot ce poate îndeplini cu ușurință obiective atât în mod autonom, cât și în cel de comandare la distanță.

Abstract

With the progressive technological growth, new fields and areas of science and technology have become accessible to be improved by the general public. One of these areas was the construction of autonomous robots, which, with the major change in the field of artificial intelligence at the beginning of 2010, had an even greater involvement from the field of research.

This paper treats the subject of autonomous cars, a topic that has found its practical applicability in various economic sectors, from the transport of goods and people to the exploration of planetary surfaces.

The approach used in this paper proposes the use of a fusion of the information received from several sensors (LiDAR, RGB Camera and encoders present on the motors) to transform the information, stimuli and changes occurring in the external environment into a specific behavior. The hardware platform on which the robot's functionalities are built is a modular and scalable one that allows the assembly to be modified depending on the target area in which the robot will operate.

Using a precise simultaneous localization and mapping mechanism (SLAM) for generating the trajectories, together with the accuracy introduced into the system by a convolutional network, the paper presents a robust and scalable robot model that can easily meet objectives both autonomously and remotely controlled.

Cuprins

Abstract	2
Cuprins	4
Listă figuri	6
Listă tabele	9
Capitolul 1	10
Introducere	10
Capitolul 2	12
Noțiuni preliminare	12
2.1 Punte H	12
2.2 Sistem de control PID	13
2.3 Filtre Kalman	13
2.4 Filtre de Particule	14
2.5 Localizare și cartografiere simultană	15
2.6 Planificare de traiectorie	16
2.7 Metasistemul ROS	17
2.8 Framework-ul PyTorch	18
Capitolul 3	19
Asamblarea robotului	19
3.1 Componentele fizice ale robotului	19
3.1.1 Microcomputerul Nvidia Jetson Nano	20
3.1.2 Microcontrolerul Arduino Mega	22
3.1.3 Senzorul LiDAR	23
3.1.4 Modulul cameră	25
3.1.5 Driverul de motoare L298N	25
3.1.6 Motoarele DC cu encoder atașat	26
3.1.7 Senzorul IMU 9250	26
3.2 Interconectarea componentelor	27
3.2.1 Nvidia Jetson Nano și Arduino Mega	28
3.2.2 Nvidia Jetson Nano și senzorul LiDAR	29
3.2.3 Nvidia Jetson Nano și modulul cameră	31

3.2.4 Arduino Mega și driverele de motoare	31
3.2.5 Arduino Mega și codurile motoarelor DC	32
3.2.6 Arduino Mega și senzorul IMU	33
3.3 Asamblarea elementelor hardware	35
Capitolul 4	37
Elementele software ale robotului	37
4.1 Sistemul de fisiere ROS	38
4.2 Pachetele ROS ale computerului robotului	38
4.2.1 Diagrama de activități	39
4.2.2 Lansarea minimal.launch	40
4.2.2.1 Nodul rosserial	40
4.2.2.2 Nodul lino_base	40
4.2.2.3 Nodurile IMU	41
4.2.2.4 Nodul de localizare EKF	42
4.2.3 Lansarea bringup.launch	43
4.2.3.1 Nodul laser corespunzător pentru senzorul LiDAR	43
4.2.3.2 Nodul cameră	44
4.2.4 Lansarea slam.launch	45
4.2.5 Lansarea navigate.launch	45
4.2.5.1 Localizarea Monte Carlo adaptivă	46
4.2.5.2 Planificarea traectoriei și stiva de navigație ROS	48
4.2.6 Lansarea lane_detector.launch	52
4.2.6.1 Filtrarea de prag a imaginii	53
4.2.6.2 Modificarea perspectivei	56
4.2.6.3 Găsirea polinoamelor benzilor	56
4.2.6.4 Măsurarea poziției vehiculului relativ la banda pe care este prezent	57
4.2.6.5 Revenirea la perspectiva inițială	58
4.3 Pachetele ROS ale computerului de dezvoltare	59
4.3.1 Sistemul de verificare facială	60
4.3.1.1 Detecția facială	60
4.3.1.2 Verificare facială	63
4.3.2 Sistemul de control la distanță	65
Bibliografie	67
Anexa 1 - Surse imagini	69
Anexa 2 - Listă acronime	71

Listă figuri

Figura 1.1 - Ansamblul Lino-V cu toate componentele hardware conectate în forma finală

Figura 2.1 - Punte H

Figura 2.1 - Sistem de control PID

Figura 2.3 - Filtrul Kalman

Figura 2.4 - Algoritm filtru de particule

Figura 2.5 - Trei pași ai algoritmului de SLAM

Figura 2.6 - Operație de convoluție folosind un filtru de 3x3 pixeli

Figura 2.7 - Operație de max-pooling folosind un filtru de 2x2 pixeli cu pas 2

Figura 2.8 - Exemplu de arhitectură de rețea neuronală convoluțională

Figura 3.1 - Elementele hardware ale Lino-V. Vedere din față

Figura 3.2 - Elementele hardware ale Lino-V. Vedere din lateral

Figura 3.3 - Nvidia Jetson Nano

Figura 3.4 - Pini Arduino Mega

Figura 3.5 - Spectrul radiatiilor solare

Figura 3.6 - Cum funcționează un senzor LiDAR?

Figura 3.7 - Scanse Sweep

Figura 3.8 - YDLIDAR X4

Figura 3.9 - Modulul Pi Camera

Figura 3.10 - Modul driver L298N

Figura 3.11 - a) Senzorul IMU MPU 9250. b) Motor DC cu encoder

Figura 3.12 - Conexiune rosserial Jetson Nano - Arduino Mega

Figura 3.13 - regulă uDev pentru Arduino Mega

Figura 3.14 - regulă uDev pentru Arduino Mega și senzor LiDAR

Figura 3.15 - Comparație performanță cartografiere Scanse Sweep și YDLIDAR X4

Figura 3.16 - Schema Fritzing a conexiunilor dintre Arduino, L298N și motoarele DC

Figura 3.17 - Schema Fritzing a conexiunilor dintre Arduino Mega și encodere

Figura 3.18 - Schema Fritzing a conexiunilor dintre Arduino Mega și senzorul IMU

Figura 3.19 - Schema Fritzing a conexiunilor Arduino Mega cu driverele de motoare, codurile motoarelor și cu senzorul IMU în contextul platformei Lino-V

Figura 3.20 - Ansamblul cu toate componentele hardware conectate în versiunea 1

Figura 3.21 - Ansamblul cu toate componentele hardware conectate în versiunea 2

Figura 4.1 - Rețeaua ROS peste rețea WiFi

Figura 4.2 - Sistemul de fisiere ROS

Figura 4.3 - Structura unui pachet ROS

Figura 4.4.1 - Diagrama de activități a aplicației

Figura 4.4.2 - Diagrama de activități a aplicației

Figura 4.5 - Importanța nodului rosserial în ecosistemul aplicației

Figura 4.6 - Importanța nodului lino_base în ecosistemul aplicației

Figura 4.7 - Importanța nodurilor imu_calib și imu_madgwick_filter în ecosistemul aplicației

Figura 4.8 - Importanța nodului robot_localization în ecosistemul aplicației

Figura 4.9 - Diagrama nodurilor ROS realizată de utilitarul rqt_graph pentru nodurile rulate de minimal.launch

Figura 4.10 - Diagrama nodurilor ROS realizată de utilitarul rqt_graph pentru nodurile rulate de bringup.launch

Figura 4.11 - Diagrama nodurilor ROS realizată de utilitarul rqt_graph pentru nodurile rulate de bringup.launch și slam.launch

Figura 4.12 - Diagrama nodurilor ROS realizată de utilitarul rqt_graph pentru nodurile rulate de bringup.launch și navigate.launch

Figura 4.13 - Importanța nodului amcl în ecosistemul aplicației

Figura 4.14 - Aplicarea algoritmului AMCL într-un context real

Figura 4.15 - O vizinătură high-level asupra nodului move_base și asupra interacțiunilor sale cu alte componente

Figura 4.16 - O vizinătură low-level asupra nodului move_base și asupra interacțiunilor dintre modulele sale interne

Figura 4.17 - Definirea hărților de cost corespunzătoare hărții generate de algoritmul de SLAM

Figura 4.18 - Planificarea celor două traiectorii: globală și locală.

Figura 4.19 - Cele patru comportamente de recuperare în cazul apariției unei erori propuse de nodul move_base

Figura 4.20 - Importanța nodului image_node în ecosistemul aplicației

Figura 4.21 - Imaginea originală transmisă de cameră pe topicul /csi_cam_0/image_raw în cazul unei iluminări bune

Figura 4.22 - Cele 3 metode de filtrare de prag în cazul unei iluminări bune

Figura 4.23 - Filtrare benzii în cazul unei iluminări bune

Figura 4.24 - Comparația imaginii originale cu o iluminare necorespunzătoare cu cea filtrată de metoda prezentată mai sus

Figura 4.25 - Aplicarea transformării de perspectivă într-un scenariu dat de proiectul descris mai sus

Figura 4.26 - Găsirea polinoamelor benzilor folosind cei doi pași. Prima poză prezintă aplicarea “sliding window” în găsirea densităților pixelilor activați. A doua poză prezintă interpolarea centrelor ferestrelor cu un polinom de grad 2.

Figura 4.27 - Evidențierea benzii dintre cele două polinoame găsite

Figura 4.28 - Imaginea rezultată procesată cu benzile găsite și publicată de către nodul image_node pe topicul /lanes_image

Figura 4.29 - Diagrama nodurilor ROS realizată de utilitarul rqt_graph pentru nodurile rulate de bringup.launch și lane_detector.launch

Figura 4.30 - Fluxul decizional al modulului de control la distanță

Figura 4.31 - Fluxul decizional al sistemului de recunoaștere facială

Figura 4.32 - Arhitecturile rețelelor P-Net, R-Net și O-Net

Figura 4.33 - Pipelineul arhitecturii MTCNN

Figura 4.34 - Exemplul inferenței rețelei MTCNN.

Figura 4.35 - Structura modelului FaceNet

Figura 4.36 - Funcția de cost triplet

Figura 4.37 - Rezultatul oferit de sistemul de verificare facială.

Figura 4.38 - Rezultatul oferit de sistemul de verificare facială în cazul unor noi identități

Figura 4.39 - Importanța nodului lino_keyboard în ecosistemul aplicației.

Figura 4.40 - Interfața terminal a nodului lino_keyboard.

Listă tabele

Tabel 1 - Corespondența pinilor Arduino Mega - Drivere L298N

Tabel 2 - Corespondența pinilor Arduino Mega - Encodere motoare DC

Tabel 3 - Corespondența pinilor Arduino Mega - Senzor IMU

Capitolul 1

Introducere

Incepand cu micii roboți industriali de la sfârșitul anilor '50 până la roboții autonomi complecși actuali, domeniul roboticii a reprezentat și reprezintă o manifestație de succes a spiritului practic și totodată plin de imaginea al omenirii. De la introducerea DARPA Challenges în 2004 și până la alăturarea coloșilor din industria automobilelor precum Tesla, BMW, Mercedes și Ford, dar și a unor companii precum Google și Uber, piața industriei automobilelor autonome a cunoscut o reală expansiune. Astăzi, nu doar prognozele inginerilor devin din ce în ce mai îndrăznețe, cât și realitatea atestă acest lucru. Și totuși, mai este nevoie de un efort semnificativ al inginerilor de pretutindeni pentru ca mașinile să-și atribuie caracteristica ideală de complet autonome.

În cele ce urmează voi prezenta rezultatele proprii obținute în urma cercetării și construirii unui prototip compact de vehicul autonom, numit în continuare Lino-V. Capitolul 2 reprezintă o introducere în concepte și noțiunile ce stau la baza proiectării unui vehicul autonom. De la concepte de electronică și teorie a controlului, până la matematica ce stă în spatele algoritmilor fundamentali de planificare și localizare, în acest capitol sunt prezentate succint noțiunile fundamentale din spatele proiectării propriu-zise. Cum în construcția întregului ansamblu funcțional părți foarte însemnante de efort și atenție sunt îndreptate spre hardware, capitolul 3 este cel ce tratează în detaliu construirea fizică a ansamblului. Prima parte a capitolului trei prezintă elementele hardware folosite și considerentele atât subiective cât și obiective ale alegerii lor, iar în a doua parte sunt prezentate modalitățile de

interconectare a acestor componente. Capitolul 4 abordează elementele software ce stau în spatele percepției și controlului mașinii. Aici sunt tratate subiecte precum fluxul de funcționare și sincronizarea proceselor principale, algoritmii folosiți și optimizarea acestora având în vedere sistemul incorporat pe care aceștia rulează. Tot aici se prezintă și rezultatele obținute în urma testelor și se tratează în detaliu anumite studii de caz.

Ultima parte, concluziile, sumarizează toate observațiile realizate și pune în perspectivă întregul efort necesar construirii robotului autonom, Lino-V.

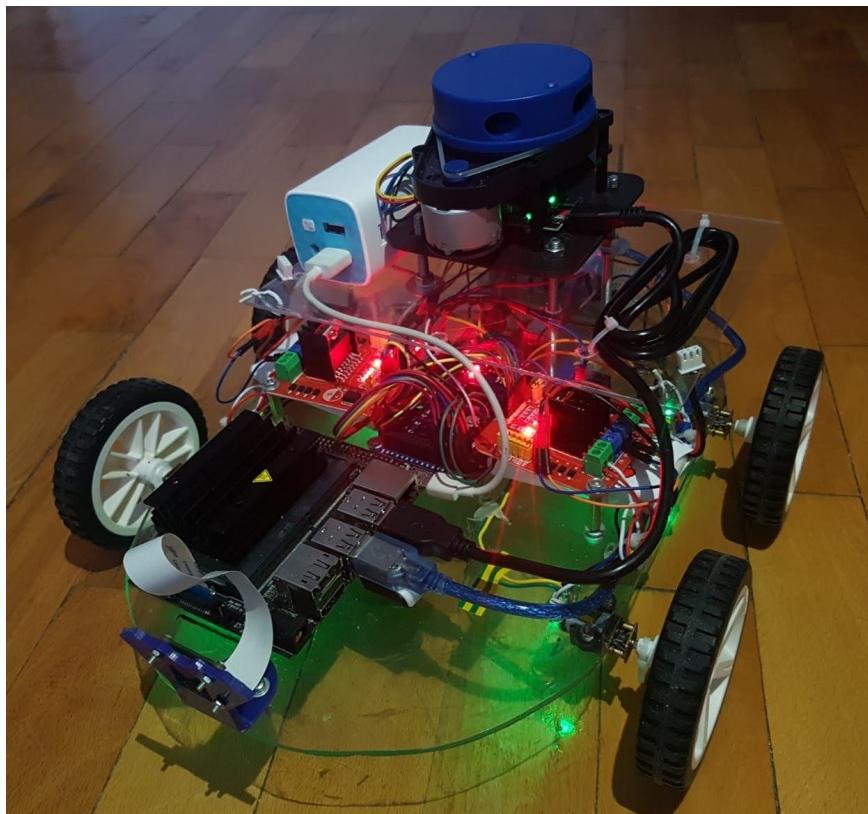


Figura 1.1 - Ansamblul Lino-V cu toate componente hardware conectate în forma finală

Capitolul 2

Noțiuni preliminare

La baza construirii ansamblului Lino-V stau numeroase proceduri și concepte ce au nevoie de o prezentare generală, succintă pentru cititori. Astfel, un cumul de elemente hardware și software ce definesc funcționarea corectă a platformei sunt însemnate în acest capitol.

2.1 Punte H

O punte H este un circuit electric ce conține patru elemente de comutare, cu sarcina în centru (motorul), ce sunt dispuse într-o configurație asemănătoare literei H.

Comutarea pe diagonală a celor patru comutatoare aduce ca output rotația în direcții opuse a motorului. Situațiile posibile sunt:

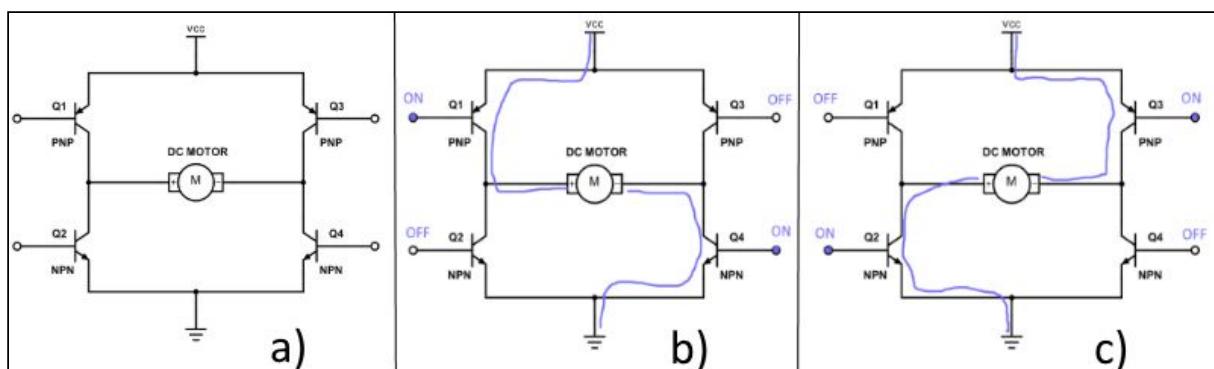


Figura 2.1 - Punte H. a) Schema electronică a unei punți H. b)-c) Căile de scurgere a curentului într-o punte H.

2.2 Sistem de control PID

Controlerul proporțional – integral – derivat (controler PID) este un mecanism de buclă de control ce folosește un sistem de feedback. La fiecare pas se calculează valoarea de eroare ca diferență între valoarea dorită și variabila de proces măsurată, aplicându-se corecții pe baza a trei termeni constanți definiți, proporțional, integral și derivat (respectiv P, I și D).

Fiind ușor de înțeles și implementat, iar rezultatele oferite fiind cele așteptate, un controler PID a fost implementat pentru a controla gradul de rotație al motoarelor.

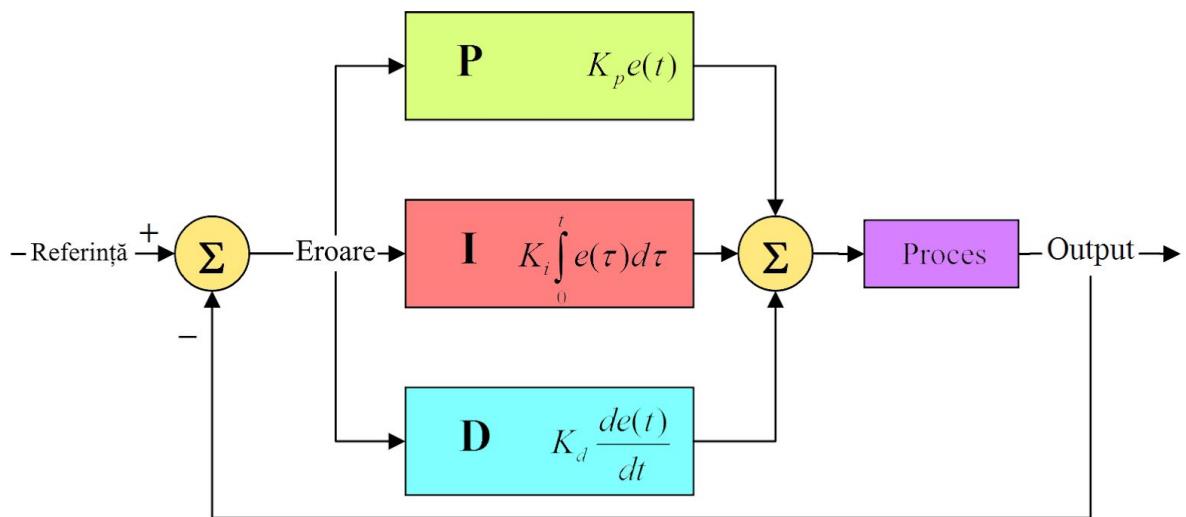


Figura 2.2 - Sistem de control PID. Definirea fluxului informației în mecanismul de control [1]

2.3 Filtre Kalman

În statistică și teoria controlului, filtrarea Kalman este un algoritm care pe baza unor măsurători efectuate de-a lungul timpului, ce conțin zgomot și incertitudine, estimează starea internă a unui sistem liniar dinamic. Filtrele Kalman se pretează perfect pentru situații în care starea sistemului se modifică constant, având avantajul că folosesc o cantitate mică de memorie și sunt foarte rapide, ceea ce le face foarte potrivite pentru sistemele în timp real sau pentru cele integrate.

Algoritmul de filtrare Kalman are doi pași importanți: predicție și actualizare.

În pasul de predicție, algoritmul de filtrare Kalman produce estimări ale variabilelor de sistem și a gradului lor de incertitudine. Luând în considerare noile măsurători venite de la senzori, pasul al doilea are rolul de a actualiza vechile predicții folosind o medie ponderată, acordându-se mai multă pondere estimărilor cu o certitudine mai mare.

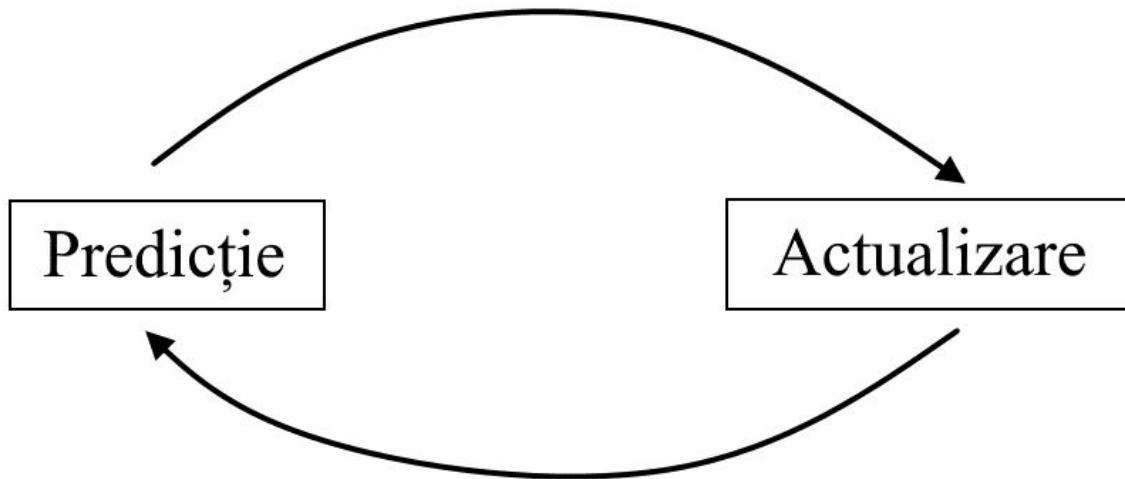


Figura 2.3 - Ciclul discret de filtrare Kalman. Proiecția estimărilor variabilelor de sistem în pasul de predicție și ajustarea estimărilor în funcție de măsurătorile de teren din pasul de actualizare [2]

Filtrele Kalman în forma lor de bază sunt pretabile pentru sisteme liniare cu zgromot sub formă de distribuții gaussiene, însă, având în vedere forma și natura datelor citite de senzori, există posibilitatea ca sistemul să devină nonliniar. Astfel, se introduce conceptul de Filtru Kalman Extins, care are rolul ca, prin intermediul seriilor Taylor, să liniarizeze sistemul.

În lucrarea propusă au fost folosite filtre Kalman extinse pentru a determina estimări cât mai corecte a poziției robotului folosind fuziunea datelor de la mai mulți senzori.

2.4 Filtre de Particule

La fel ca filtrele Kalman, filtrele de particule fac și ele parte din familia estimatorilor recursivi Bayesieni. Diferența majoră dintre cei doi algoritmi, dincolo de implementarea acestora, este că filtrele de particule pot estima valori optime în scenarii non-gaussiene neliniare. Astfel, avantajul principal al filtrelor de particule este că acestea nu se bazează pe nicio tehnică de liniarizare locală sau nicio aproximare funcțională brută. Cu toate acestea, prețul care trebuie plătit pentru această flexibilitate este că aceste metode sunt costisitoare din punct de vedere al calculului. [3]

Algoritmul de filtru de particule se bazează pe 3 pași importanți:

- eșantionarea a N noi particule aleatoare în distribuția inițială
- determinarea unei ponderi pentru fiecare particula în funcție de importanța acesteia (probabilitatea ca punctul estimat să fie cel țintă)
- reeșantionarea prin înlocuirea a M particule cu probabilitate mică cu M particule aleatoare în noua distribuție (“Survival of the fittest” - termen cunoscut în algoritmii evolutivi).

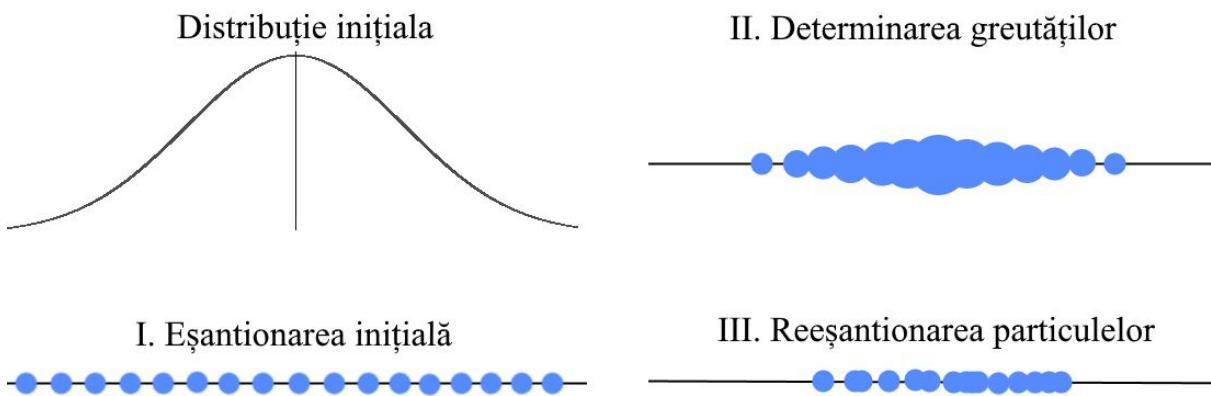


Figura 2.4 - Algoritm filtru de particule. Definirea pașilor algoritmului

Filtrele de particule sunt folosite în lucrarea prezentată pentru localizarea robotului într-un mediu cunoscut (folosind procedura numită “Adaptive Monte-Carlo Localization”)

2.5 Localizare și cartografiere simultană

Cartografierea mediului reprezintă una dintre aplicațiile de bază ale robotului autonom, fiind punctul inițial în fluxul de funcționare pentru navigarea autonomă. Utilizând harta generată de algoritmul de mapare folosind senzorul LiDAR, robotul poate produce traекторii posibile pentru atingerea punctului țintă. Una dintre cele mai populare metode de mapare este localizarea și maparea simultană (SLAM). Totodată, localizarea robotului în mediul deja cartografiat sau care urmează să fie cartografiat este definitorie pentru ca robotul să poată atinge nivelul dorit de autonomie. Astfel, SLAM permite crearea hărții în timp ce, simultan, robotul este localizat în harta deja generată.

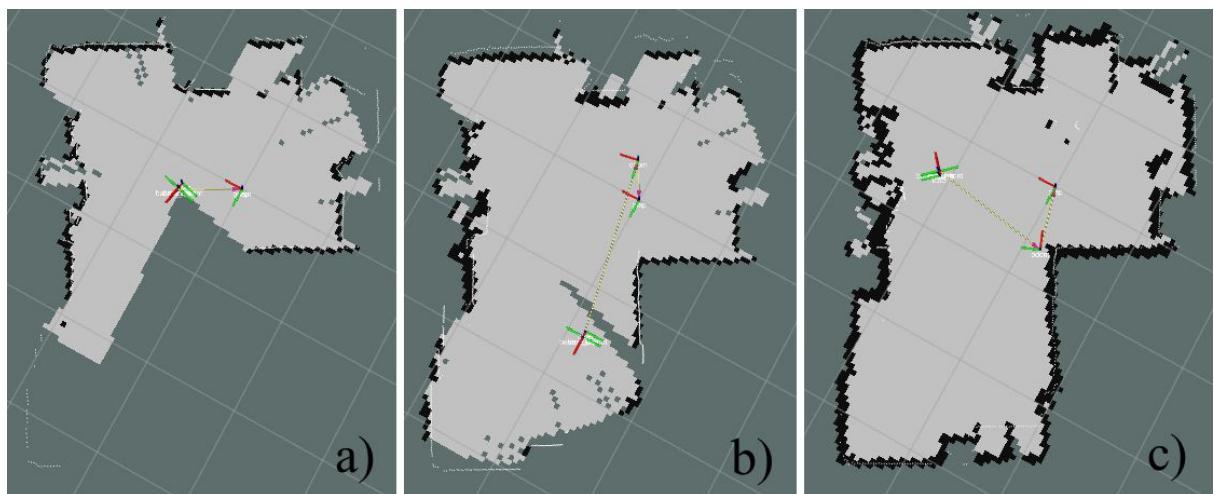


Figura 2.5 - Trei pași ai algoritmului de SLAM. a) Percepție scăzută, inițială a mediului. b) Percepție medie a mediului după două rotații complete și o deplasare liniară. c) Percepție ridicată a mediului după o explorare îndelungată.

2.6 Planificare de traieectorie

Vehiculele autonome își bazează deciziile pe module de planificare care generează traieectorii astfel încât să se poată deplasa fără coliziune pe calea ce ajunge la punctul de destinație. Aceste module sunt capabile de găsirea soluției optime minimizând timpul și distanța parcuse de vehicul, evitând obstacole statice și dinamice. Modulul de navigare este împărțit într-un **planificator global** și **unul local**, unde primul găsește calea optimă cu o cunoștință prealabilă a mediului și obstacolelor statice (folosind harta generată prin intermediul

algoritmilor de SLAM), iar a doua recalculează calea în funcție de obstacolele apărute în mediu ulterior construirii hărții. [4]

Generarea planificatorului global depinde de modul în care este generată harta inițială. Există mai mulți algoritmi ce generează traiectorii folosind o informație a priori a mediului. Unii dintre aceștia sunt cei care rezolvă problema alocând o valoare fiecărei regiuni a foii de parcurs, pentru a identifica drumul cu costul minim. Printre aceștia se numără algoritmii Dijkstra și A* care sunt implementați și în stiva de navigație disponibilă în infrastructura ROS.

Pentru a transforma calea globală în puncte de acces adecvate, planificatorul local creează noi puncte de acces, luând în considerare obstacolele dinamice și constrângerile vehiculului. Deci, pentru a recalculeaza calea cu o rată specifică, harta este redusă la împrejurimile vehiculului și este actualizată pe măsură ce vehiculul se deplasează. Prin urmare planificarea locală generează căi de evitare a obstacolelor dinamice și încearcă în același timp să păstreze referințele planificatorului global.

Modulul de navigație autonomă implementează planificarea de traiectorie în contextul platformei Lino-V.

2.7 Metasistemul ROS

“Robot Operating System” (ROS) este un set de biblioteci software și instrumente open-source pentru construcția de aplicații pentru roboți. Deși ROS nu este un sistem de operare în sine, acesta oferă servicii concepute pentru grupuri de calculatoare, cum ar fi abstractizarea hardware, controlul la nivel scăzut al dispozitivului, implementarea funcționalităților utilizate frecvent, sisteme de trimitere a mesajelor între procese și gestionarea pachetelor.

Arhitectura ROS este reprezentată de o rețea distribuită peer-to-peer cu un nod principal ce gestionează comunicarea între procese: ROS Master, ce are rolul de a configura comunicarea pentru topic-uri și a controla actualizările serverului de parametri. Procesele ROS sunt reprezentate ca **noduri** într-o structură de tip graf, conectate prin arce numite **topic-uri**. Astfel, nodurile, care reprezintă procese singulare, își pot transmite mesaje unul altui prin intermediul topic-urilor, pot cere și oferi **servicii** altor noduri sau pot seta sau

preluă date partajate **din serverul de parametri**, bază de date ce se ocupă cu stocarea variabilelor partajate de procese.

Pentru ca un nod să poată trimite un mesaj spre un alt nod, este nevoie ca primul nod să publice (“**publish**”) date, iar pentru ca acesta să poată primi date este necesar ca aceasta să se asculte (“**subscribe**”) la topicul menționat.

2.8 Framework-ul PyTorch

PyTorch este o bibliotecă de machine learning open source bazată pe Python, utilizată în principal pentru aplicații de vedere artificială și procesarea limbajului natural. PyTorch oferă capabilități de calcul cu tensori (asemănătoare cu biblioteca NumPy din Python) folosind o accelerare puternică prin unități de procesare grafică (GPU). Un tensor este o matrice multidimensională ce conține elemente de același tip (dimensiunea acestei matrice poate varia de la ordin 0 până la n). Cu alte cuvinte, tensorii reprezintă o generalizare a vectorilor și matricelor 2D.

Motivele alegerii framework-ului PyTorch pentru implementări au fost variate, de la participarea la cursul facultativ de Deep Learning ținut de echipa BitDefender până la o curbă de învățare mult mai pronunțată decât în cazul altor framework-uri precum Tensorflow (motivul pentru acest fapt vine în special din multele asemănări dintre PyTorch și pachetele fundamentale Python precum NumPy).

Capitolul 3

Asamblarea robotului

Modelul folosit pentru construirea ansamblului hardware, și nu numai, a fost platforma open-source Linorobot [5]. Linorobot suportă mai multe configurații hardware, fiind astfel o alternativă flexibilă pentru construirea unei mașini autonome.

În cele ce urmează voi prezenta abordarea proprie asupra alegerii componentelor hardware, care în majoritatea cazurilor a fost în funcție de cea mai bună alternativă disponibilă; au existat, însă, situații în care a trebuit să adaptez așteptările în funcție de un minim disponibil rapid.

3.1 Componentele fizice ale robotului

Partea fizică a robotului a fost construită în totalitate individual, fără a exista module deja construite. Astfel, suportul robotului este reprezentat de o placă de policarbonat pe care am atașat fiecare componentă în parte. Adăugarea componentelor noi a trebuit să ia în considerare limitările impuse de un spațiu restrâns și limitat, astfel că fiecare punct al suportului de policarbonat este folosit cu un anume scop.

Calitatea cea mai mare pe care o are această platformă, Linorobot, este scalabilitatea. În funcție de utilitate, modelul poate fi mărit sau micșorat cu ușurință, atât din punct de vedere al dimensiunilor cât și din cel al capacitatei de calcul și percepție.

Astfel, componentele hardware adăugate peste suportul inițial liber sunt următoarele.

În partea frontală:

- Microcomputerul Nvidia Jetson Nano
- Modulul de cameră RGB și suportul acestuia

În partea laterală:

- 4 motoare DC cu encoder atașat și suporturi de prindere pe policarbonat
- 4 roți cu $\varnothing 8$ cm
- 2 drivere de motoare L298N
- 2 baterii cu 7.4V la 1000mA pentru alimentare motoare

În partea centrală:

- Microcontrolerul Arduino Mega

În partea din spate:

- LiDAR-ul YDLIDAR X4
- Senzorul IMU

De menționat este ca microcomputerului Nvidia Jetson Nano i-a fost atașat un adaptor wireless USB 150Mbps 802.11n.

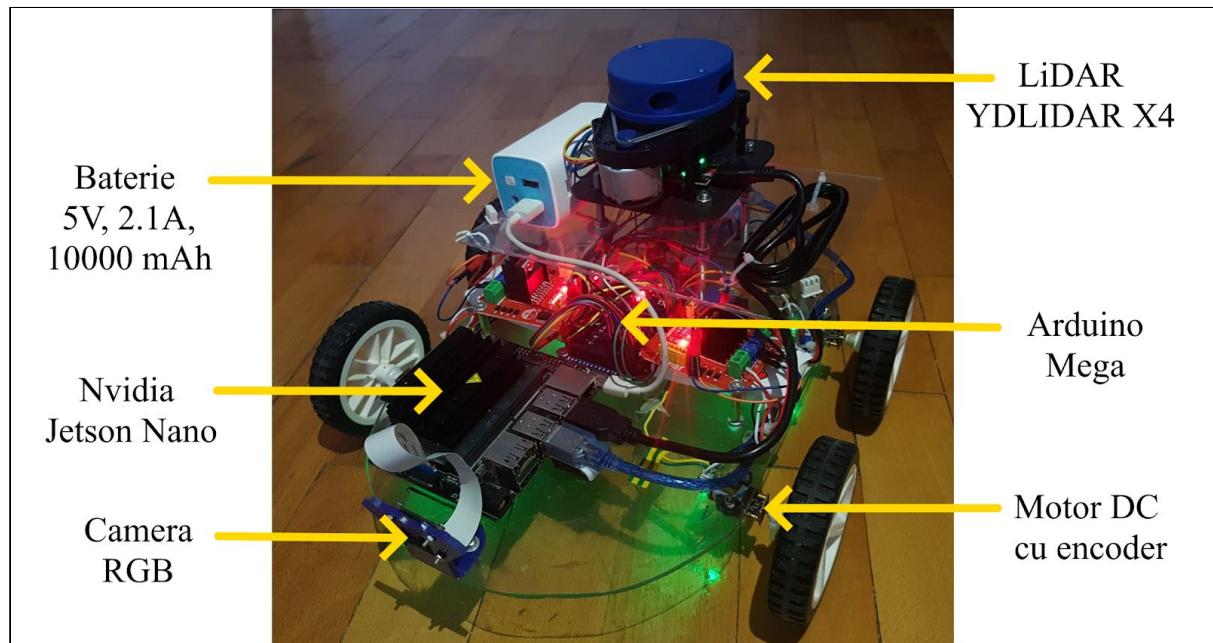


Figura 3.1 - Elementele hardware ale Lino-V. Vedere din față. Corespondențe sunt marcate cu săgețile de culoare galbenă.

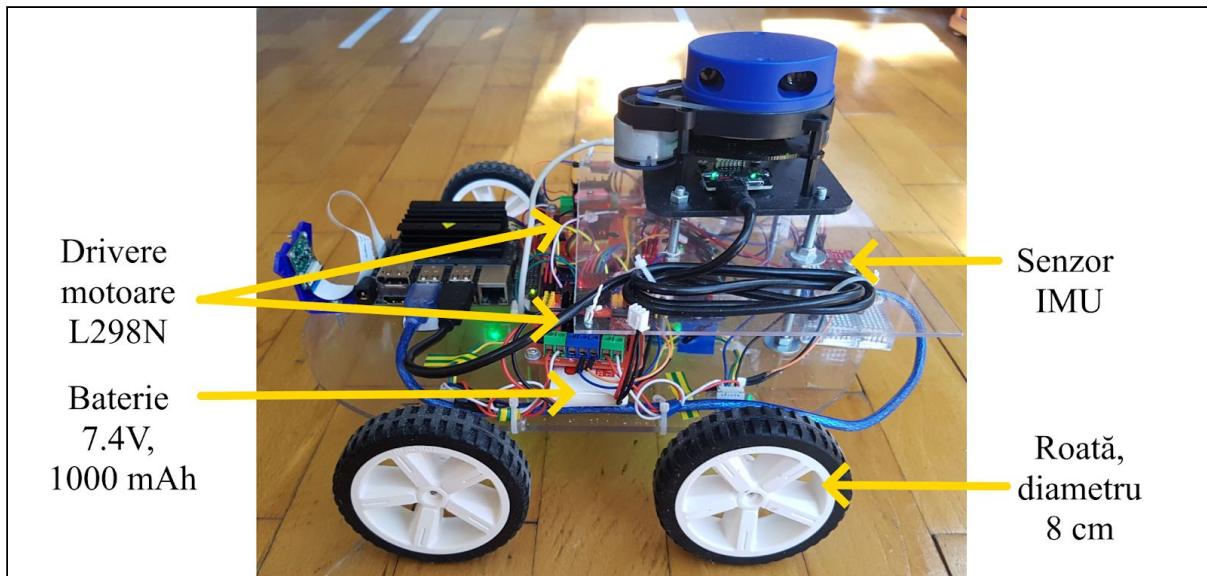


Figura 3.2 - Elementele hardware ale Lino-V. Vedere din lateral. Corespondențe sunt marcate cu săgețile de culoare galbenă.

3.1.1 Microcomputerul Nvidia Jetson Nano

Microcomputerul reprezintă creierul robotului și are ca principale atribuții gestiunea fluxului de informații de la și către microcontroller și senzorul LiDAR și, în plus, oferă toate resursele computaționale necesare pentru desfășurarea într-un mod cât mai fluid a proceselor de calcul interne. Având în vedere nevoia unei puteri de procesare ridicate, alternativele care puteau oferi soluții viabile erau folosirea unui RaspberryPi 4 sau a unui Nvidia Jetson Nano. Motivul alegerii finale a dispozitivului Nvidia a fost că acesta dispune de un GPU Maxwell de 128 de nuclei ce este mult mai potrivit pentru algoritmi de inteligență artificială decât GPU-ul propus de RaspberryPi 4.

În ultimii ani, Nvidia a făcut un progres semnificativ spre a face accesibile procesoarele grafice. Nvidia Jetson Nano este prima lor reușită în această direcție, Nvidia reușind să scoată pe piață acest modul la numai 99 de dolari, făcându-l astfel disponibil unui număr mare de ingineri dormici de a-i testa capabilitățile. Această disponibilitate aduce după sine și un număr mare de articole și un suport semnificativ acordat subiectului acestui microcomputer.

Totuși, dincolo de prețul mic și notorietatea firmei dezvoltatoare, microcomputerul Nvidia Jetson Nano este, prin capacitatele însemnante de procesare, un mediu ideal de

dezvoltare ce permite rularea mai multor rețele neuronale în paralel pentru aplicații precum clasificarea imaginilor, detecția obiectelor, segmentarea și procesarea vorbirii. Astfel, pe kit-ul de dezvoltare găsim:

- GPU 128-core Maxwell
- CPU Quad-core ARM A57 @ 1.43 GHz
- Memorie 4 GB 64-bit LPDDR4 25.6 GB/s
- Camera MIPI CSI-2 DPHY lanes
- Conectivitate Gigabit Ethernet, M.2 Key E
- Display HDMI and display port
- USB 4x USB 3.0, USB 2.0 Micro-B
- Others GPIO, I2C, SPI, UART

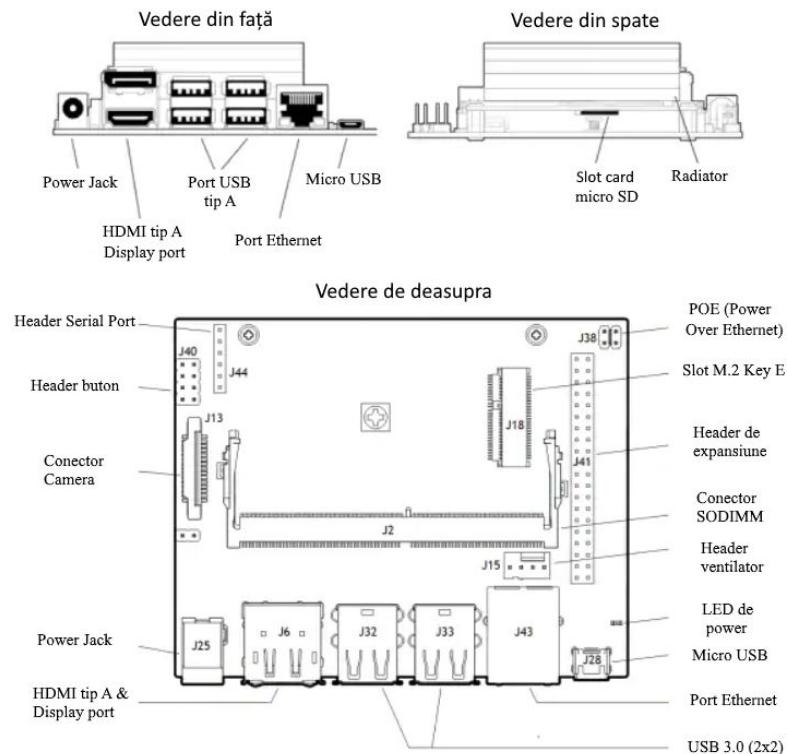


Figura 3.3 - Nvidia Jetson Nano. Aranjarea pinilor pe placa de dezvoltare [6]

3.1.2 Microcontrolerul Arduino Mega

Pentru a funcționa modular, pe lângă microcomputerul central a fost nevoie și de un microcontroler care să se ocupe de controlul complet al motoarelor independent de activitatea microcomputerului.

Pentru acoperirea acestei nevoi, încă de la început au existat două microcontrolere pretendente: Teensy 3.6 și Arduino Mega. Deși inițial am comandat un Teensy 3.6 și eram convins că acesta va fi cel pe care îl voi folosi, din cauza unei neînțelegeri la firma de distribuire, am primit o versiune mai veche de Teensy, și anume 3.2. Fiind nevoie de o aşteptare foarte lungă până când versiunea 3.6 ajungea din nou în stoc și cum la momentul respectiv nu găsisem diferențe explicite raportate la utilizarea microcontrolerelor în contextul mașinii autonome, am decis ca în final să folosesc Arduino Mega pentru această sarcină. Mai târziu, navigând adânc în problematica sincronizării semnalelor am realizat că Arduino Mega dispune de doar 6 pini cu întrerupere, pe când Teensy are atașată capabilitatea de întrerupere pe fiecare dintre pinii digitali.

Întreruperile sunt folosite pentru encoderele motoarelor, iar în total este nevoie de 10 pini de întrerupere într-un scenariu ideal (2x4 pini pentru encodere și încă 2 pentru senzorul IMU). Totuși, deși Arduino Mega dispune doar de 6 pini, în contextul actual de utilizare performanțele encoderelor nu sunt simțitor scăzute (deoarece fiecare encoder are rămas un pin cu întrerupere atașat).

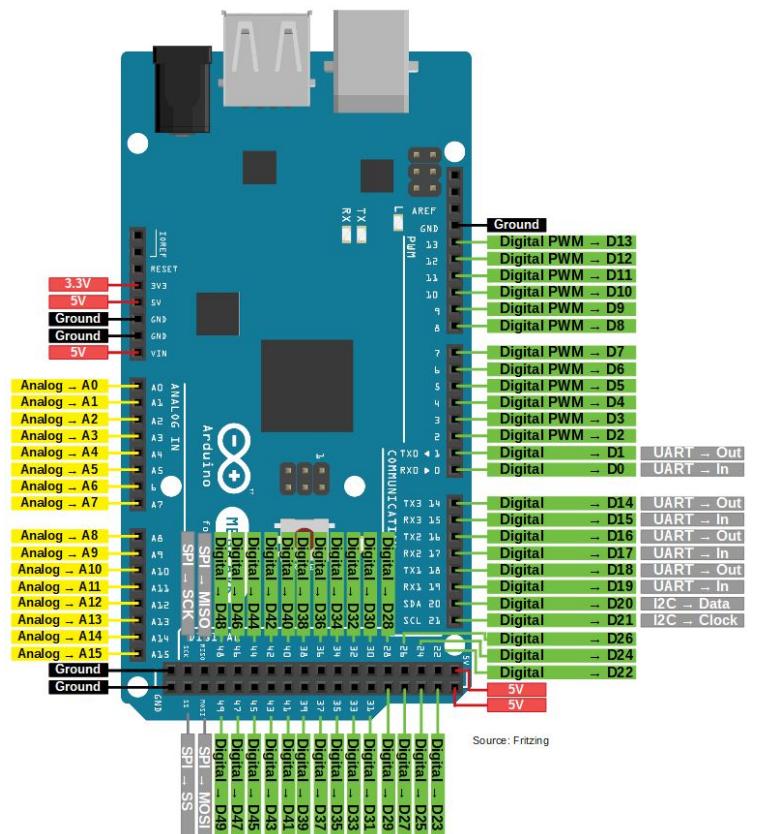


Figura 3.4 - Pini Arduino Mega.

Aranjarea pinilor pe placă Arduino [7]

3.1.3 Senzorul LiDAR

Un LiDAR este o componentă electronică ce face parte din familia senzorilor. Mai precis, face parte din categoria de senzori “Time Of Flight” (ToF).

Tehnologia LiDAR este o tehnologie de teledetectie care măsoară distanța dintre el însuși și o țintă. Lumina este emisă din LiDAR și călătorește către țintă, iar apoi se reflectă din suprafața țintei, revenind spre sursa. Deoarece viteza luminii are o valoare constantă, senzorul LiDAR este capabil să calculeze distanța până la țintă.

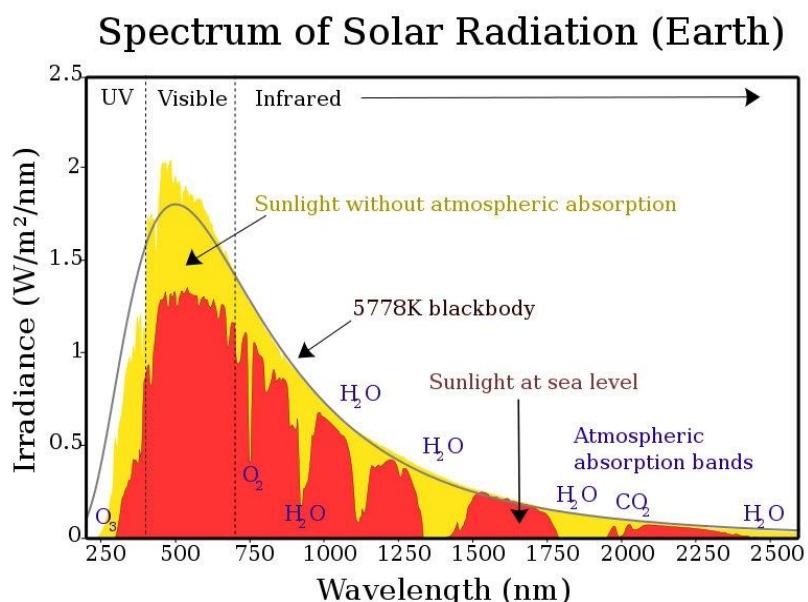


Figura 3.5 - Spectrul radiatiilor solare. Un raport al lungimii de undă a radiațiilor solare raportate la gradul lor de radiație. Zonele cu galben reprezintă lumina solară fără absorbție atmosferică iar cele roșii lumina solară la nivelul mării. [8]

După cum se poate observa în graficul de mai sus, există câteva zone în care radiația undelor de lumină este minimă. Astfel, LiDAR-urile au următoarea aplicabilitate folosind lungimile de undă corespondente:

- Infraroșu (1500 –2000 nm) pentru meteorologie/ LiDAR Doppler - utilizări științifice
- Aproape infraroșu (850 -940 nm) pentru cartografierea terestră
- Albastru-Roșu (500 –750 nm) pentru batimetrie

- Ultraviolet (250 nm) pentru meteorologie

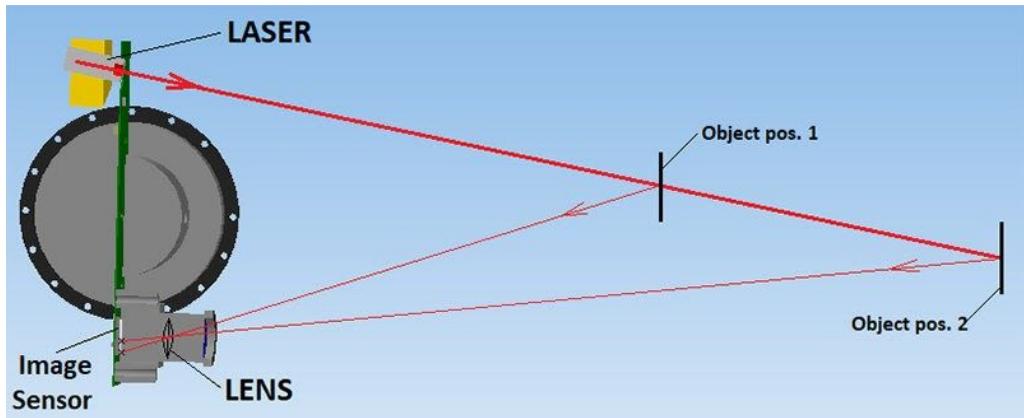


Figura 3.6 - Cum funcționează un senzor LiDAR? Razele sunt emise de către laser și receptate prin lentilă de către senzorul de imagine. Timpul trecut de la emitere până la recepție este transformat în distanța parcursă de lumină. [\[9\]](#)

Fiind una dintre cele mai scumpe și greu de procurat componente din întregul ansamblu prima iterație a proiectului a fost construită cu LiDAR-ul Scanse Sweep, un senzor retras din piață, mai greu de utilizat decât alternativele actuale, dar perfect funcțional. Integrarea dificilă a acestuia m-a convins ca pentru următoarea iterătie să folosesc un LiDAR ce este încă în producție, dispune de suport activ din partea fabricantului și a comunității online. Astfel, a doua iterătie a proiectului, aduce în scenă LiDAR-ul YDLIDAR X4.

Sweep Lidar. Specificațiile tehnice ale Scanse Sweep sunt următoarele:

- Distanță maximă de citire: 40m (discutabilă această distanță)
- Rezoluție: 1cm
- Rată de eșantionare: Până la 1075Hz
- Frecvență de scanare: 1-10Hz
- Power: 5V @ 450mA → 650mA
- Greutate: 120 grame



Figura 3.7 - Scanse Sweep [\[10\]](#)

YDLIDAR X4. Specificațiile tehnice ale YDLIDAR X4 sunt următoarele:

- Distanță maximă de citire: 10 m
- Rezoluție unghiulară: 0.48-0.52°
- Rezoluție: < 0.5 mm (Range < 2 m)
- Rată de eșantionare: 5000 Hz
- Frecvența de scanare: 6-12 Hz
- Power: 5V @ 280mA → 480mA
- Lungimea de undă: 775-795 nm
- Greutate: 189 g



Figura 3.8 - YDLIDAR X4 [11]

3.1.4 Modulul cameră

Pentru gestionarea fluxului video, am ales să folosesc modulul de cameră pus la dispoziție de Raspberry Pi, Raspberry Pi Camera Module V2. Aceasta este o alternativă ieftină și performantă pentru manipularea fluxului video având specificații de performanță, consum și conectivitate potrivite pentru scenariul proiectului de față. Acest modul oferă o cameră de 8 megapixeli capabilă să realizeze fotografii cu rezoluție de până la 3280 x 2464 pixeli.

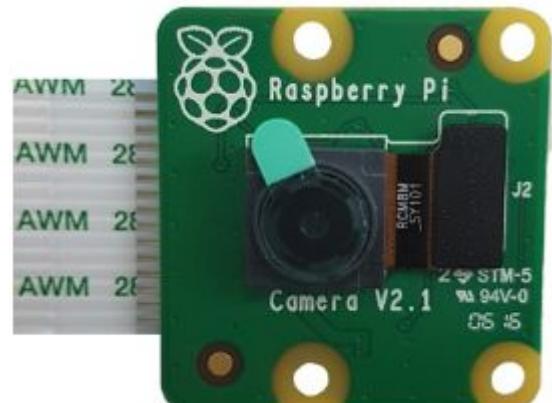


Figura 3.9 - Modulul Pi Camera [12]

3.1.5 Driverul de motoare L298N

Pentru controlul motoarelor am avut de ales între a folosi shield-ul Arduino L293D sau modulul driver L298N. După ce a apărut o defecțiune la shield-ul L293D am fost nevoit să comand două module L298N cu punte H dublă. În ciuda faptului că cele două module ocupă un spațiu mai mare pe baza robotului decât ar fi ocupat shield-ul atașat direct peste Arduino,

acesta are și câteva puncte forte, și anume, curentul pe care îl pot oferi ca output per canal este de 2A (de aici și existența radiatorului extern care disipa căldura), și în plus, am flexibilitate totală în alegerea pinilor folosiți pentru alte sarcini decât controlul motoarelor (dacă as fi folosit shield-ul L293D, adăugarea unor fire exterioare pe pinii digitali 2->13 ar fi fost îngreunată de header-ul shield-ului).

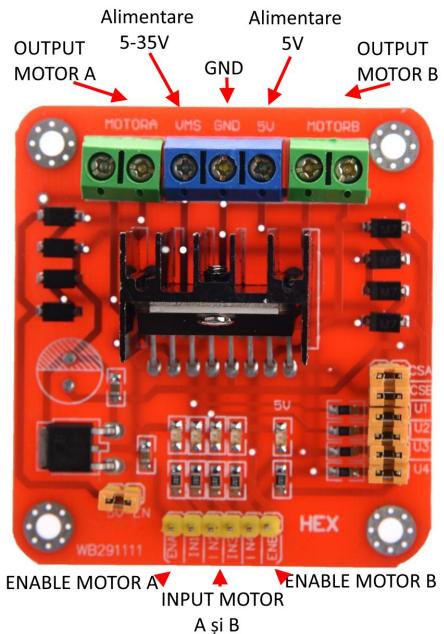


Figura 3.10 - Modul driver L298N.

Aranjarea pinilor pe placă driverului. [13]

3.1.6 Motoarele DC cu encoder ataşat

Pentru alegerea motoarelor a trebuit să iau în considerare natura și ținta proiectului. Cum nu este nevoie de o viteză foarte mare a roților dar de o tracțiune mare datorită greutății ansamblului (aproximativ 1.5 - 2 kg) am folosit patru motoare cu un raport de reducție mare din punctul de vedere al tracțiunii. Astfel, pentru a determina valoarea exactă a reducției am folosit un calculator disponibil online, rezultând astfel un raport ideal de 150:1.

Motorul DC folosit este FIT0484 DFROBOT ce are atașat și un encoder pentru a avea un răspuns din partea motorului la rotație. Motorul funcționează la o tensiune nominală de 6V @ 60mA până la 170mA și cu un număr de 60 de rotații nominale pe minut. Encoderul atașat vine cu un număr de 2100 de impulsuri/minut.

3.1.7 Senzorul IMU 9250

Pentru a avea un mai bun control asupra poziției mașinii în sistemul de coordonate tridimensional, a fost nevoie să adaug un senzor IMU (Inertial Measurement Unit - Unitate de măsură inerțială). IMU 9250 dispune de un giroscop pe 3 axe, un accelerometru pe 3 axe și un magnetometru pe 3 axe.

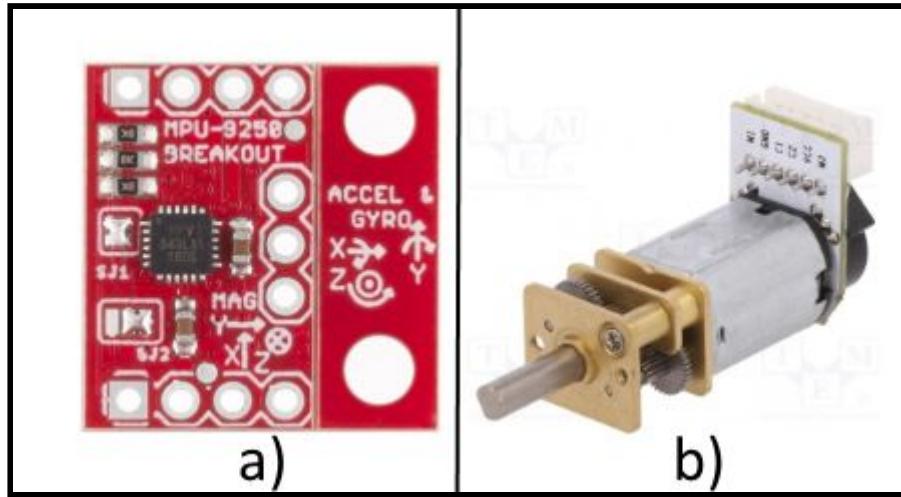


Figura 3.11 - a) Senzorul IMU MPU 9250. b) Motor DC cu encoder [14][15]

3.2 Interconectarea componentelor

În această secțiune sunt prezentate modalitățile de interconectare ale componentelor active prezente pe platforma Lino-V, tratând subiectul din diferite perspective, de la conexiunile fizice, protocole de comunicare până la sincronizarea componentelor.

3.2.1 Nvidia Jetson Nano și Arduino Mega

Comunicarea dintre microcomputerul Nvidia Jetson Nano și microcontrolerul de control motoare Arduino Mega se realizează prin intermediul unui cablu serial USB A - B. Rata de transfer al informației, baud rate, este setată pentru această comunicare la 57.600 bits/sec.

Cum întreg sistemul este bazat pe o integrare în ecosistemul ROS, a fost nevoie ca și microcontrolerul să devină parte integrată în ecosistem, iar cum Arduino Mega nu dispune de o conexiune wireless, transferul informației a fost făcut prin intermediul pachetului rosserial. Rosserial reprezintă un protocol de trimitere a datelor prin intermediul interfeței serial. Arduino a putut deveni atât nod care publică mesaje prin intermediul topicurilor cât și ascultător.

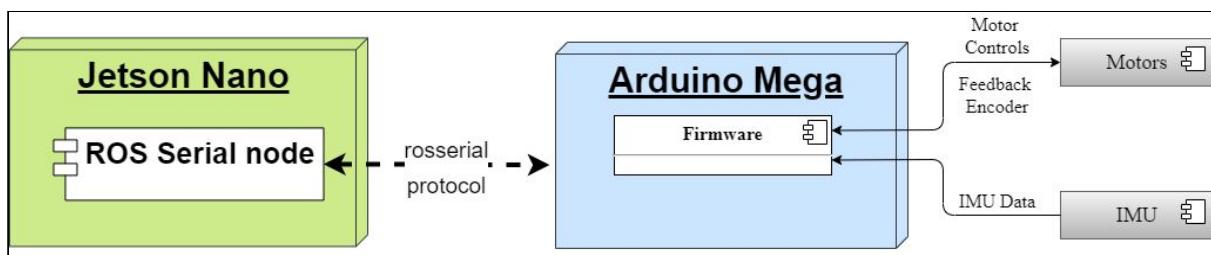


Figura 3.12 - Conexiune rosserial Jetson Nano - Arduino Mega. Informațiile de la senzorii de mișcare (encodere și senzorul IMU) dar și comenziile venite de la nodul ROS Serial circulă prin intermediul protocolului rosserial între Jetson Nano și Arduino Mega.

Pentru ca Arduino Mega să fie recunoscut întotdeauna de Jetson indiferent de portul USB pe care acesta este conectat, am implementat manual o regula uDev, astfel că, de fiecare dată când microcontrolerul este conectat pe oricare dintre cele patru porturi USB ale microcomputerului, Arduino va primi, pe baza identificatorului de produs, un port static numit “/dev/linobase”.

```
jetson@jetson-desktop:~$ ls -l /dev/lino*
lrwxrwxrwx 1 root root 7 mai 29 01:22 /dev/linobase -> ttyUSB0
```

Figura 3.13 - regulă uDev pentru Arduino Mega. Lansarea comenzii de listare a conținutului directorului /dev/lino arată legarea microcontrolerului la portul ttyUSB0*

Ultimul aspect tratat în acest capitol este modul în care codul necesar pentru microcontroler este urcat pe acesta având în vedere natura conexiunilor. De această sarcină se ocupă PlatformIO ce este un utilitar cross-platform și cross-architecture care asigură compilarea, pregătirea și încărcarea codului pe microcontroler.

3.2.2 Nvidia Jetson Nano și senzorul LiDAR

Cât vine vorba de senzorul LiDAR, platforma autonomă se află la iterația a doua, schimbându-se între iterații senzorul LiDAR. În ambele cazuri conexiunea dintre microcomputer și senzorul LiDAR se face prin intermediul unui port USB de partea microcomputerului iar în cazul Scanse Sweep unei legături pe portul serial integrat iar în cazul YDLIDAR X4 pe un port micro-USB ce se ocupă atât de transferul datelor cât și de alimentare.

Asemenea microcontrolerului Arduino Mega, pentru ca senzorul LiDAR să fie recunoscut întotdeauna indiferent de portul USB pe care acesta a fost conectat, atribuindu-i-se un port static, a fost nevoie să se definească o regula uDev. De această dată definirea regulii nu a mai fost o procedură manuală, ocupându-se de aceasta nodul lino_udev pus la dispoziție de dezvoltatorii platformei Linorobot. Astfel, pe baza identificatorului de produs, senzorului LiDAR îi este atribuit un port static numit “/dev/linolidar”.

```
jetson@jetson-desktop:~$ ls -l /dev/lino*
lrwxrwxrwx 1 root root 7 mai 29 16:57 /dev/linobase -> ttyUSB1
lrwxrwxrwx 1 root root 7 mai 29 16:57 /dev/linolidar -> ttyUSB0
```

Figura 3.14 - regulă uDev pentru Arduino Mega și senzor LiDAR. Legatura automată formată de regulile uDev pentru cele două device-uri conectate prin intermediul porturilor USB

După cum se poate observa în [Figura 3.12](#), cele două dispozitive, microcontrolerul și senzorul LiDAR au fost legate la porturile ttyUSB1 respectiv ttyUSB0, deși după cum putem vedea în [Figura 3.11](#), microcontrolerul a fost legat atât pe porturile ttyUSB0 cât și pe ttyUSB1. În ciuda acestui lucru, adresarea Arduino Mega a putut fi făcută fără probleme în ambele cazuri prin intermediul portului static asignat. Aici stă adevărata putere a asignărilor regulilor uDev multiplelor componente conectate via porturi USB.

Studiu de caz: Scanse Sweep sau YDLIDAR X4. Diferențele majore dintre cei doi senzori vin în primul rând din considerente de performanță (rată de eşantionare, rezoluție și frecvență de scanare superioare pentru senzorul YDLIDAR). În plus, la achiziția datelor, Scanse Sweep necesită o convertire a datelor din forma lor brută point-cloud la laser-scan. Ratele de transfer al datelor sunt și ele diferite, Scanse Sweep având un baud rate de 115200, iar YDLIDAR de 128000.

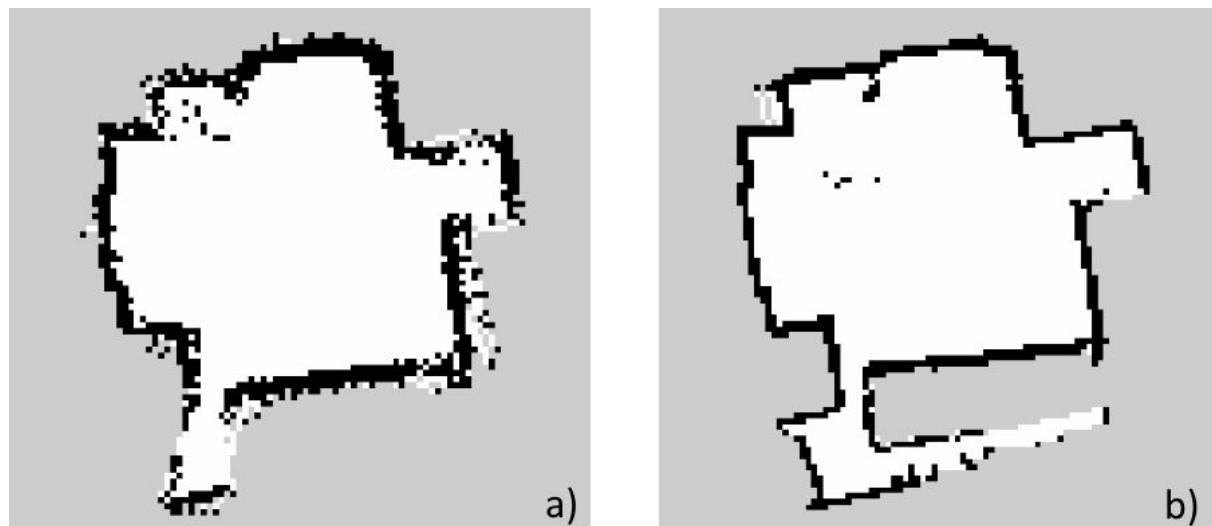


Figura 3.15 - Comparație performanță cartografiere Scanse Sweep și YDLIDAR X4. În b) se poate vedea un grad mai scăzut de zgomot, deci o performanță mai mare

Cea mai dificilă parte a integrării funcționalităților celor două dispozitive conectate via USB a fost sincronizarea lor și asignarea lor pe porturi statice. Neavând cunoștințe prealabile de reguli uDev, a fost o adevărată provocare folosirea funcționalităților oferite de ele în proiectul descris. Apariția senzorului YDLIDAR nu a avut loc doar datorită dorinței de creștere a performanțelor citirii datelor ci și din cauză că înainte să aflu că asignarea

porturilor statice era făcută într-un mod defectuos (problemă care a persistat pentru mai mult de 3 săptămâni până la soluționare) am crezut că bătrânelul senzor Scanse Sweep nu se poate sincroniza cu întregul sistem. Însă, problema a fost găsită chiar înainte ca senzorul YDLIDAR să apară și astfel a ajuns să fie integrat și Scanse Sweep. Apoi, iterația doi a avut loc.

3.2.3 Nvidia Jetson Nano și modulul cameră

Integrarea modulului de cameră pe Jetson Nano prin intermediul interfeței MIPI CSI este un adevărat exercițiu de răbdare dacă nu ai la îndemână suita corespunzătoare de framework-uri și pachete software. Experiența anterioară în lucrul cu modulul de cameră mi-a fost de folos în integrarea ei, dar cum până acum foloseam doar scripturi Python pentru manipularea datelor primite de la ea, de această dată a fost nevoie să o integrez și în ecosistemul ROS. După o lungă luptă cu problemele de incompatibilitate ale Jetson Nano cu versiunea de framework GStreamer și cu a bibliotecii OpenCV, folosind două pachete^{[16][17]} cheie disponibile în comunitate, integrarea camerei ca nod ROS a fost făcută.

Fluxul video este făcut prin intermediul topic-ului `/csi_cam_0/image_raw`, ce propune un flux de imagini pe trei canale RGB cu rezoluția de 320x240 pixeli (rezoluție mică aleasă din considerente de performanță), cu un framerate de 30 FPS.

3.2.4 Arduino Mega și driverele de motoare

Pentru controlul motoarelor sunt folosite două drivere L298N, care să poată disipa, pe baza comutărilor punților H interne, curentul venit din bateriile de 7.4V spre motoarele DC. Cum fiecare dintre cele două drivere au două punți H, la fiecare dintre ele au putut fi conectate câte două motoare, conectând astfel motoarele din partea stângă a mașinii împreună și pe cele din partea dreaptă la fel. Pentru fiecare punct H a fost nevoie de 3 pini, 2 pini digitali pentru comutarea pe diagonală a celor patru elemente comutatoare și un pin digital cu capacitate PWM pentru controlul tensiunii aplicate la intrările motorului.

Tabelul de legături pentru cele 2 drivere este următorul.

Driver 1 - partea stângă			Driver 2 - partea dreaptă		
Pin Driver	Pin Arduino	Motor	Pin Driver	Pin Arduino	Motor
ENA	4	M1	ENA	6	M2
IN1	25		IN1	30	
IN2	24		IN2	31	
IN3	23	M3	IN3	32	M4
IN4	22		IN4	33	
ENB	5		ENB	12	

Tabel 1 - Corespondența pinilor Arduino Mega - Drivere L298N

Spre exemplu, la aplicarea semnalelor HIGH pe pinul 25, LOW pe pinul 24 și a semnalului PWM analogic 128 pe pinul 4, motorul M1 se va răsuci în sensul acelor de ceas la jumătate din viteza maximă. La aplicarea LOW pe pinul 25, HIGH pe pinul 24 și a semnalului PWM analogic 255 pe pinul 4, motorul M1 se va rota în sens trigonometric cu viteză maximă.

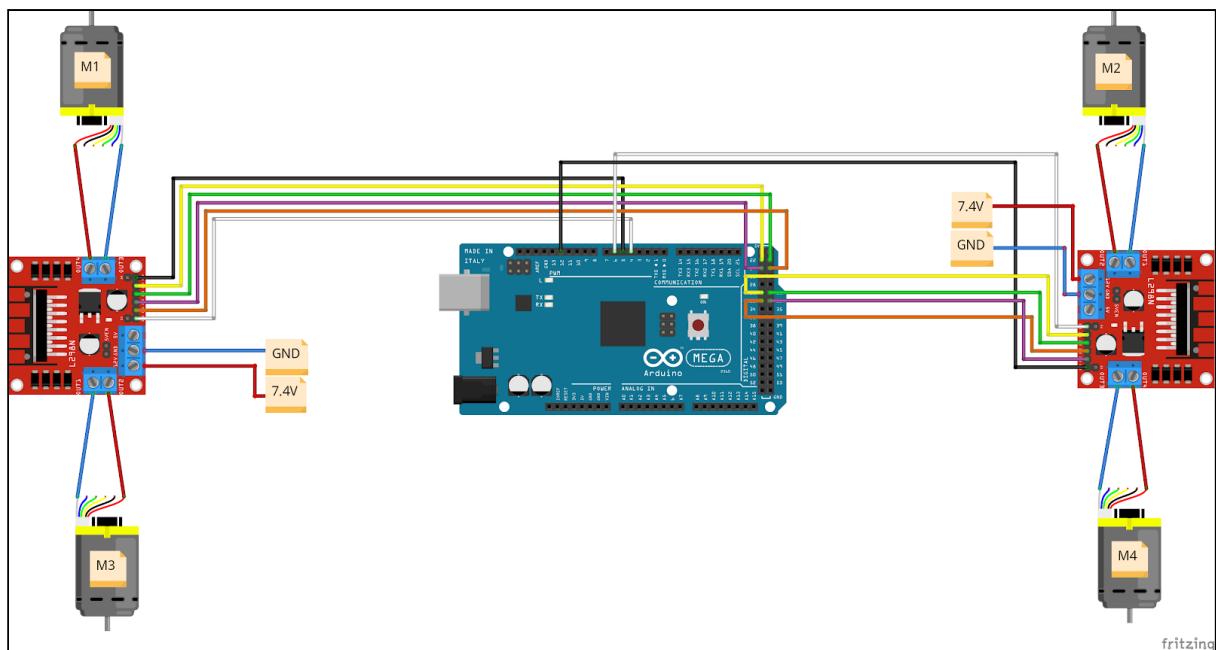


Figura 3.16 - Schema Fritzing a conexiunilor dintre Arduino, L298N și motoarele DC

3.2.5 Arduino Mega și codurile motoarelor DC

Codurile motoarelor sunt un punct esențial pentru navigarea autonomă a robotului. Acestea oferă informații despre nivelul de rotație al motorului din poziția inițială. Pentru a avea informații în timp real de la codere, a fost nevoie ca cel puțin unul dintre cei doi pini ai encoderului să fie atașați la un pin Arduino capabil de a genera o întrerupere. Cum Arduino Mega are doar 6 pini cu întrerupere, iar 2 dintre ei sunt folosiți de senzorul IMU, rămâne ca doar un singur pin cu întrerupere să fie asignat per encoder. (performanță mai scăzută dar insensibilă în contextul actual). Astfel, cei patru pini digitali rămași (2, 3, 18, 19), vor fi folosiți împreună cu patru pini digitali simpli (8, 9, 10, 11).

	Encoder M1	Encoder M2	Encoder M3	Encoder M4
C1 (faza A)	8	9	11	10
C2 (faza B)	19	2	3	18

Tabel 2 - Corespondența pinilor Arduino Mega - Codare motoare DC

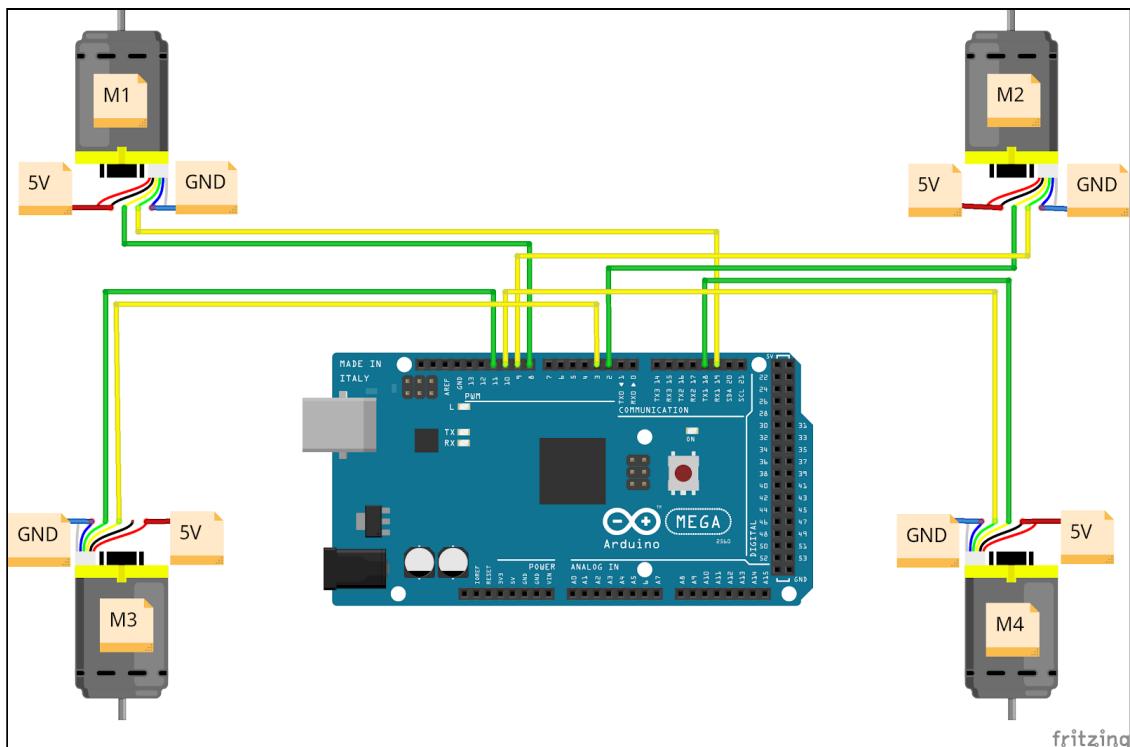


Figura 3.17 - Schema Fritzing a conexiunilor dintre Arduino Mega și codare

3.2.6 Arduino Mega și senzorul IMU

Pentru a oferi și mai multă siguranță și un punct adițional de verificare este nevoie și de integrarea unui senzor IMU (accelerometru, giroscop și magnetometru) în întregul ansamblu. Datele primite de la senzorul IMU au rolul de a fi preluate de filtrul Kalman extins și astfel, se generează estimări cât mai exacte ale poziției mașinii (chiar și în cazul rotirii în gol sau a alunecării roților). Comunicarea cu senzorul IMU se realizează prin intermediul protocolului I2C ce este compus din două semnale: SCL (semnalul de ceas) și SDA (semnalul de date). De remarcat aici este faptul că alimentarea părții logice a senzorului IMU este făcută la 3.3V, diferit față de toate celelalte alimentări (5V).

Pin IMU	Pin Arduino Mega
SDA	20
SCL	21

Tabel 3 - Corespondența pinilor Arduino Mega - Senzor IMU

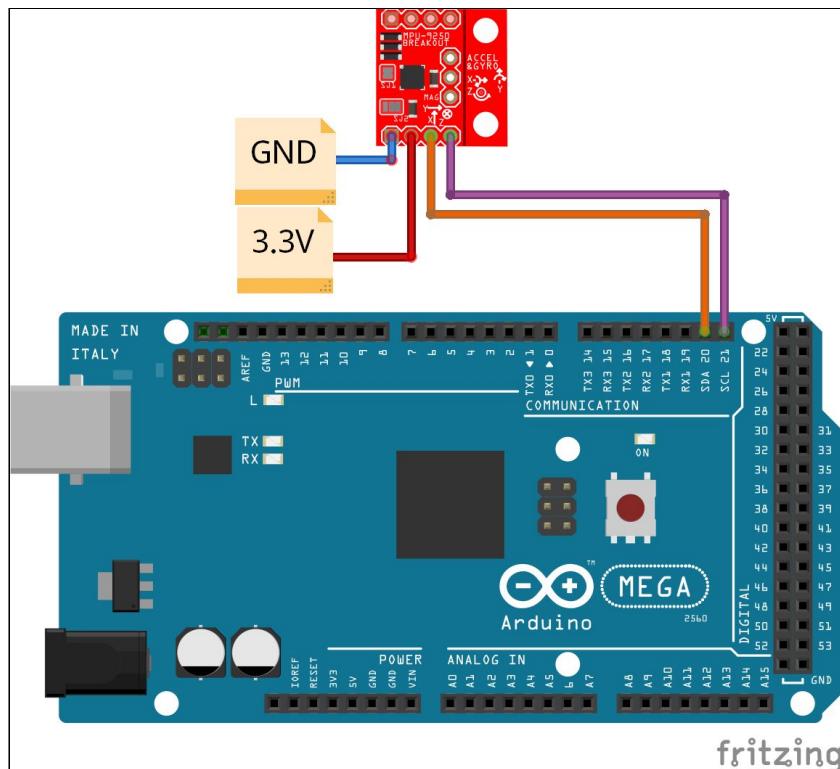


Figura 3.18 - Schema Fritzing a conexiunilor dintre Arduino Mega și senzorul IMU

3.3 Asamblarea elementelor hardware

Asamblarea tuturor elementelor hardware împreună a fost cea mai costisitoare din punct de vedere al timpului, energiei și chiar și resurselor materiale parte a dezvoltării proiectului. Greutățile întâlnite au fost multiple, dar cea mai notabilă a fost dificultatea procurării pieselor necesare asamblării din considerente de izolare socială ce a îngreunat deosebit de mult fiecare operațiune pe care trebuia să o desfășoară. Cea mai frustrantă parte a fost efortul foarte mare pe care l-am depus pentru procurarea șuruburilor și piulițelor necesare prinderii elementelor hardware pe placă de policarbonat. De la prețul de 10 ori mai mare pe care a trebuit să îl plătesc în unele situații până la greșelile de generare a comenzi din partea magazinelor și livrarea unor componente greșite, totul a fost o adevărată aventură. A fost nevoie să găsesc multe soluții ocolitoare pentru ca în final proiectul să arate ca un produs, ținând în același timp cont să nu se compromită nici performanțele robotului.

Cu toate acestea, toate dificultățile întâlnite în calea asamblării robotului au fost în mare parte depășite, propunând astfel un produs funcțional, perfect conectat și robust.

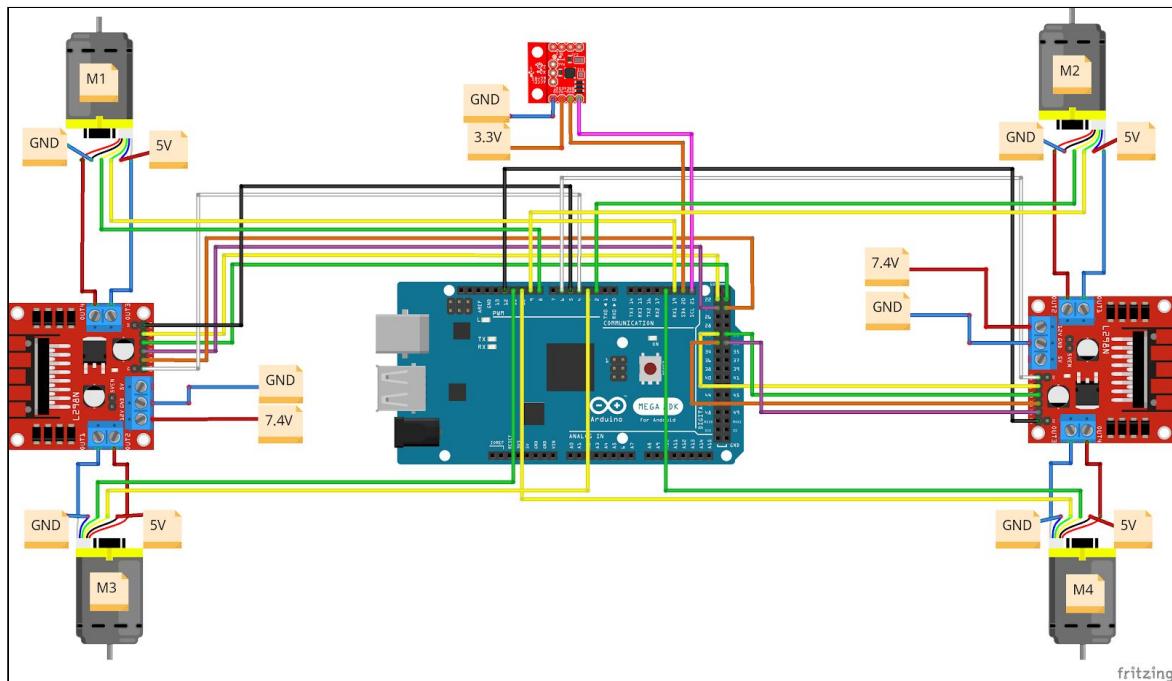


Figura 3.19 - Schema Fritzing a conexiunilor Arduino Mega cu drivele de motoare, encoderele motoarelor și cu senzorul IMU în contextul platformei Lino-V

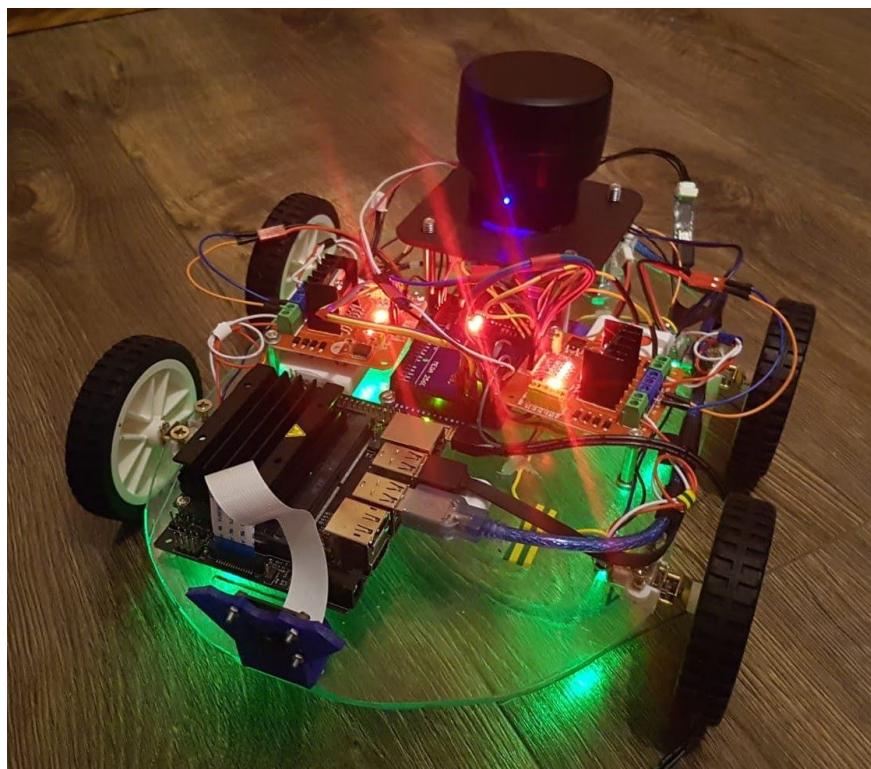


Figura 3.20 - Ansamblul cu toate componentelete conectate în versiunea 1

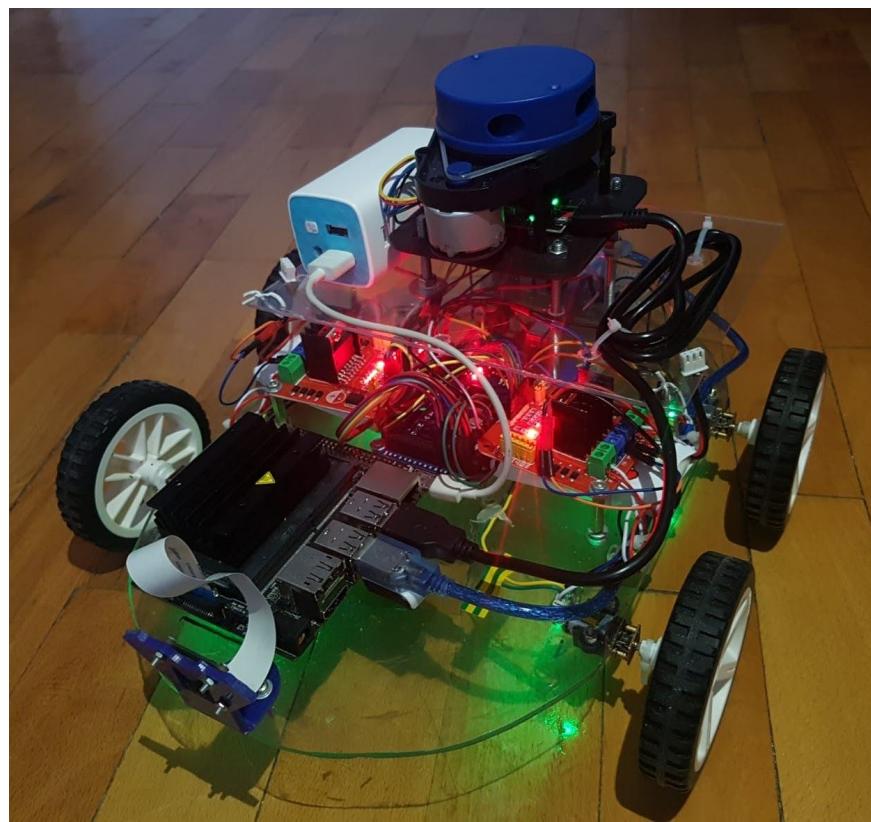


Figura 3.21 - Ansamblul cu toate componentelete hardware conectate în versiunea 2

Capitolul 4

Elementele software ale robotului

Ansamblul software al platformei Lino-V a fost dezvoltată pe scheletul existent al suitei open source, Linorobot. Din punct de vedere software, Linorobot s-a ocupat în întregime de componenta de navigație și integrare a tuturor elementelor hardware, urmând ca efortul propriu, să se concentreze în jurul integrării funcționalităților specifice domeniului de Computer Vision. Configurarea rețelei ROS este realizată astfel încât cele două instanțe, computerul de dezvoltare și computerul robotului, să poată comunica. Protocolul de comunicare folosit este TCP/IP, legătura între noduri facându-se prin socket-uri. Nodul Master al rețelei se află pe computerul robotului, acestuia atribuindu-se adresa IP: **192.168.43.50**, cu specificația că ROS Master folosește portul **:11311** în mod standard. Computerului de dezvoltare i se atribuie adresa IP: **192.168.43.33**.

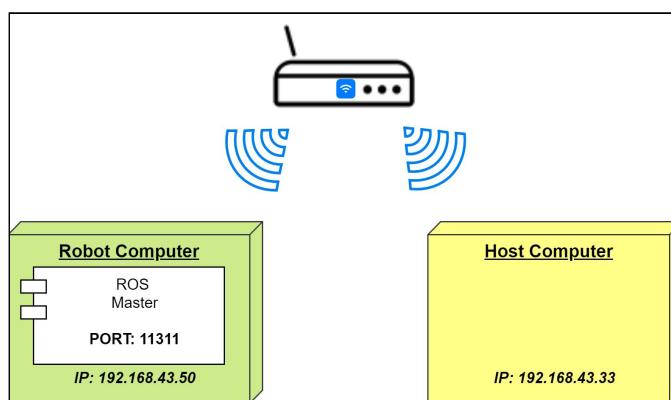


Figura 4.1 - Rețeaua ROS peste rețeaua WI-FI. Conexiunea dintre cele două componente importante ale ansamblului, computerul robotului și computerul de dezvoltare

Cum codul de rulare se află pe computerul robotului procedura de accesare a terminalului robotului este protocolul SSH. De asemenea, anumite operațiuni necesită un control mai mare asupra interfeței robotului și pentru aceasta este folosită o conexiune VNC care este un sistem grafic de partajare a desktop-ului care folosește protocolul Remote Frame Buffer (RFB). În plus, după cum s-a văzut și în [secțiunea 3.2.1](#), încărcarea și compilarea codului pe Arduino Mega este prin cross-compiling folosind platforma PlatformIO.

4.1 Sistemul de fișiere ROS

Platforma Lino-V a fost dezvoltată în mediul ROS ce propune o suită de caracteristici ideale pentru dezvoltarea aplicațiilor robotice. ROS oferă un sistem de fișiere similar cu al unui sistem de operare, fișierele fiind organizate pe disc în modul următor:

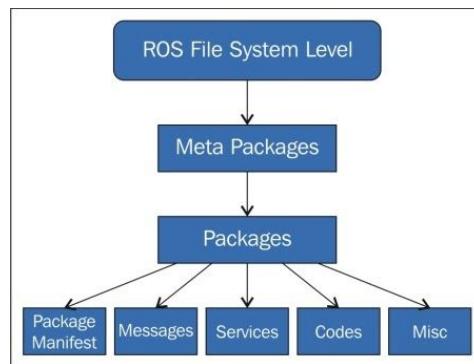


Figura 4.2 - Sistemul de fișiere ROS. Structura ierarhică a sistemului de fișiere ROS. [18]

4.2 Pachetele ROS ale computerului robotului

Deoarece reprezintă unitățile de bază, atomice ale ROS, pachetele folosite vor fi prezentate în secțiunea ce urmează. Ele conțin nodurile, bibliotecile, fișierele de configurație, toate fiind organizate împreună într-o singură unitate. Structura unui pachet ROS este următoarea:

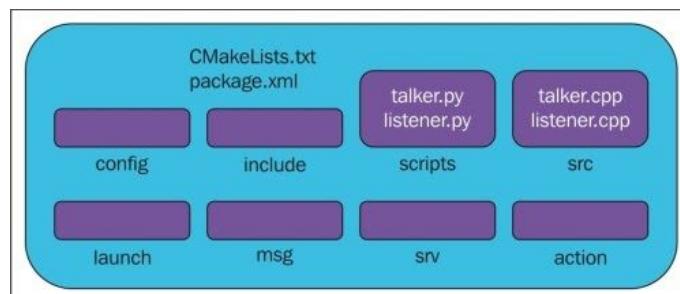


Figura 4.3 - Structura unui pachet ROS. Codul propriu zis se află în /src [18]

4.2.1 Diagrama de activități

Diagrama de activități prezintă totalitatea opțiunilor pe care utilizatorul le are atunci când initializează robotul. Un mod convenabil de rulare a mai multor noduri ROS și setare a unor parametrii de inițializare este folosirea unor fișiere launch bazate pe un format XML caracteristic.

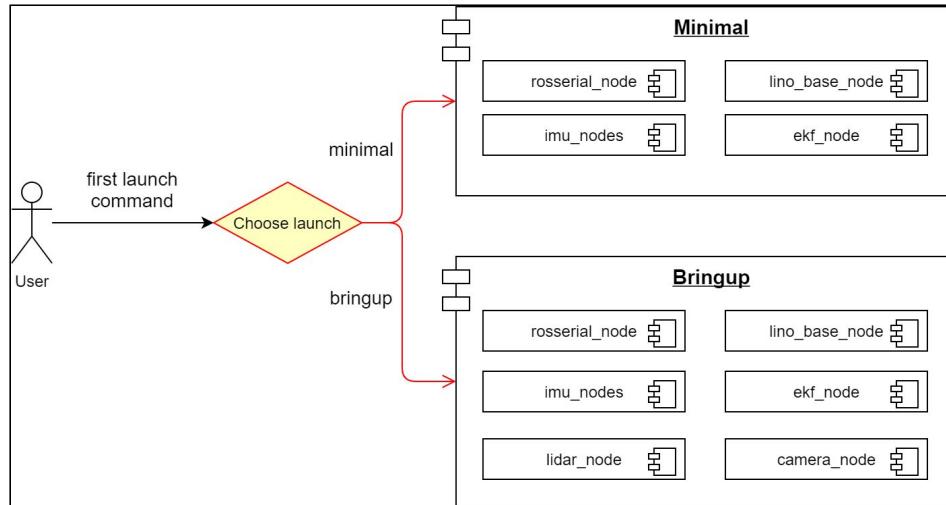


Figura 4.4.1 - Diagrama de activități a aplicației. Prima interacțiune cu platforma când utilizatorul poate lansa minimal.launch cu funcționalități minime și bringup.launch cu noduri adăugate pentru cameră și LiDAR

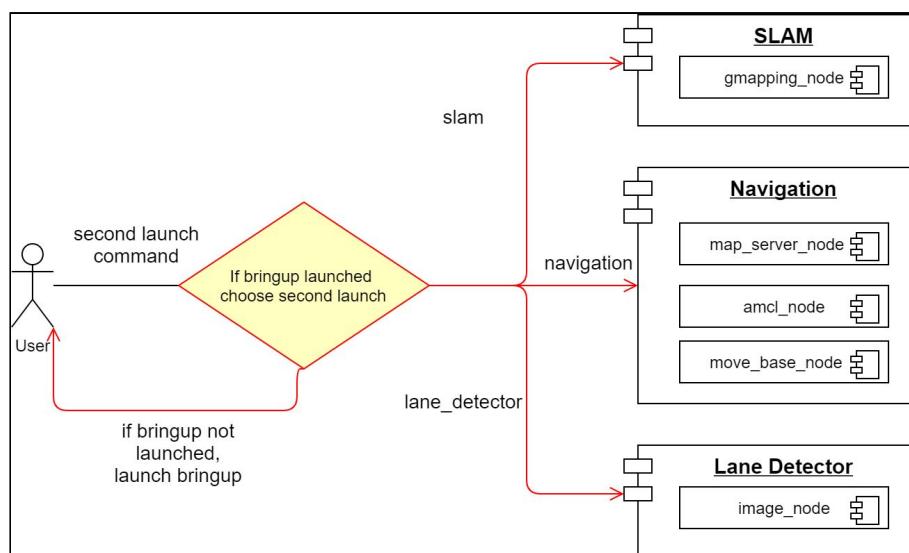


Figura 4.4.2 - Diagrama de activități a aplicației. A doua interacțiune cu platforma când utilizatorul poate lansa slam.launch, navigation.launch sau lane_detector.launch dacă utilizatorul bringup.launch a pornit nodurile de cameră, respectiv LiDAR

4.2.2 Lansarea minimal.launch

Lansarea fișierului minimal.launch propune doar rularea nodurilor necesare pentru calibrarea și verificarea funcționalității motoarelor și a senzorilor de mișcare și a nodului Master ROS. În plus, cum senzorii de mișcare și motoarele sunt conectate direct la controlerul Arduino Mega, verificarea funcționalității lor implică și verificarea conexiunii serial dintre Jetson și Arduino Mega.

4.2.2.1 Nodul rosserial

Stiva de navigație ROS conduce robotul publicând comenzi de viteză la topicul `/cmd_vel` folosind mesaje de tipul `geometry/Twist`. Controlerul Arduino Mega primește mesajele de control motoare prin `rosserial_python` și convertește viteza dorită în comenziile motorului. Un controler PID menține viteza robotului cu un grad minim de eroare calculând cât de mult semnal PWM trebuie generat pentru a roti fiecare motor pe baza erorii totale în timp. Erorile sunt generate prin compararea vitezei reale a motorului cu viteza dorită trimisă de stiva de navigație. De asemenea, tot prin intermediul `rosserial_python`, sunt publicate două topicuri de către Arduino Mega, `/raw_imu` și `/raw_vel` ce conțin datele citite în formă brută de către senzorul IMU respectiv encoderele motoarelor.

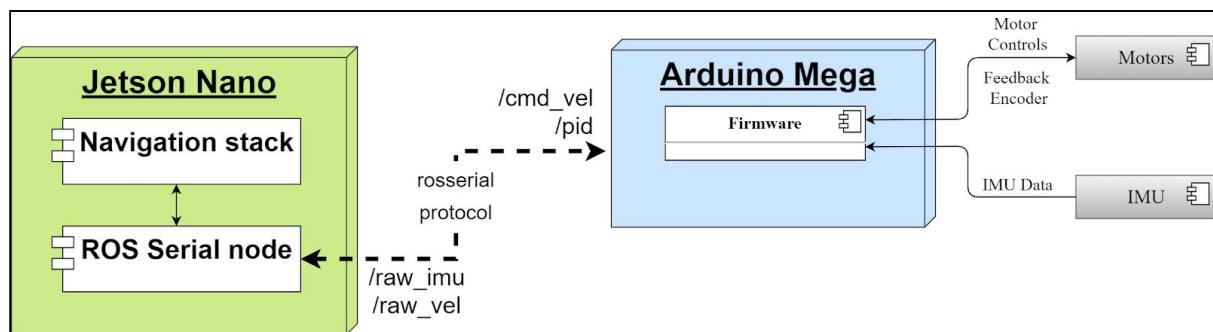


Figura 4.5 - Importanța nodului rosserial în ecosistemul aplicației. Gestionaerea mesajelor realizată de nodul resserial

4.2.2.2 Nodul lino_base

Nodul `lino_base` are rolul de a converti informațiile venite de la encoderele motoarelor prin intermediul topicului `/raw_vel` în informații despre odometrie, publicând aceste informații pe

topicul `/raw_odom`. Odometria reprezintă o estimare a modificărilor poziției robotului în timp.

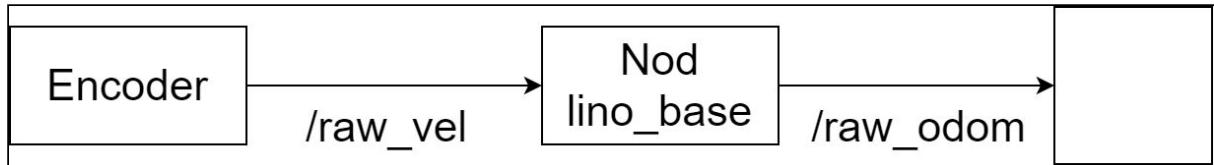


Figura 4.6 - Importanța nodului `lino_base` în ecosistemul aplicației. Transformarea datelor de la encodere în informații de odometrie

4.2.2.3 Nodurile IMU

Nodurile răspunzătoare de procesarea datelor de la senzorul IMU sunt două: un nod de calibrare ce corectează informațiile brute venite de la senzorul IMU pe baza unui fișier de calibrare și un nod de filtrare a datelor corectate de nodul de calibrare.

Calibrarea datelor se realizează pe baza unui fișier de calibrare generat în urma unui proces de calibrare manuală. Calibrarea manuală se realizează cu ajutorul unui utilitar propus de platforma Linorobot, numit `imu_calib`.

După ce fișierul este generat, calibrarea datelor de pe topicul `/raw_imu` este realizată prin intermediul nodului de calibrare, `imu_calib`, ce modifică datele în funcție de fișierul de calibrare și publică datele pe topicul `/imu/raw_data`. Apoi, nodul `imu_madgwick_filter` aplică un algoritm AHRS [19] (Attitude and Heading Reference System) ce este utilizat pentru a filtra și contopi datele brute de pe dispozitivele IMU, fuzionând viteze unghiulare, accelerări și lecturi magnetice dintr-un dispozitiv IMU generic într-o orientare de quaternion. Publicarea datele contopite este făcută pe topicul `/imu/data`.

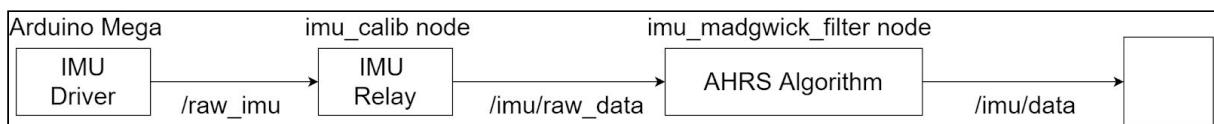


Figura 4.7 - Importanța nodurilor `imu_calib` și `imu_madgwick_filter` în ecosistemul aplicației. Calibrarea și filtrarea datelor brute de la senzorul IMU

4.2.2.4 Nodul de localizare EKF

Nodul de localizare reprezintă implementarea unui estimator de stări neliniar pentru mișările robotului într-un context 3D. Filtrul neliniar folosit în acest proiect este un filtru Kalman Extins ce reprezintă o extindere a filtrului Kalman clasic într-un context neliniar [20]. În consecință, nodul *robot_localization* înglobează informațiile filtrate de la senzorii de mișcare disponibile în rețeaua ROS prin intermediul topicurilor */raw_odom* și */imu/data* și le transformă, în urma aplicării unui filtru Kalman extins, în date corectate de odometrie prezente pe topicul */odom*

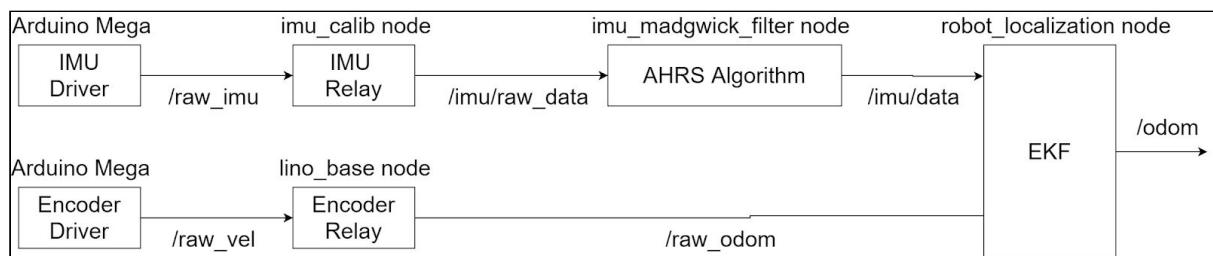


Figura 4.8 - Importanța nodului *robot_localization* în ecosistemul aplicației. Inglobarea datelor filtrate de la senzorii de mișcare în estimări corecte

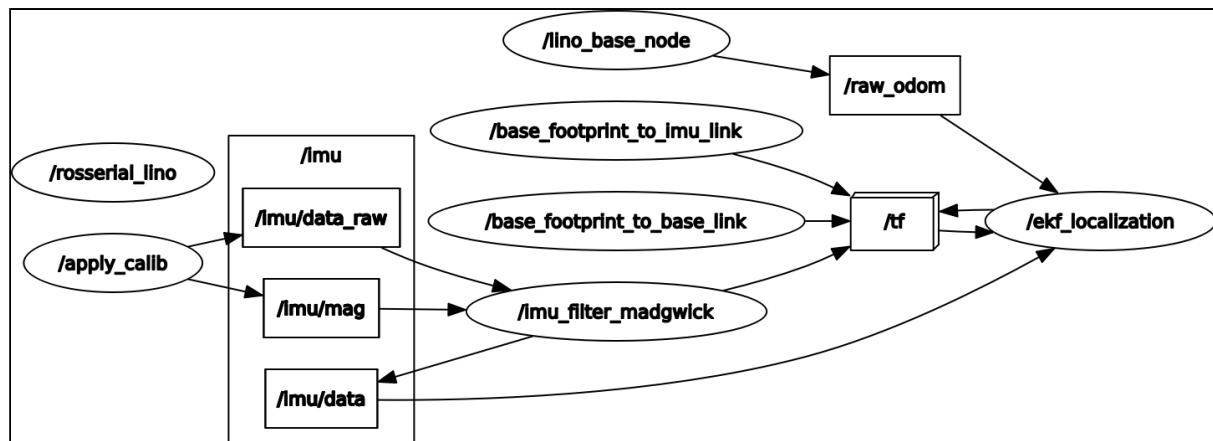


Figura 4.9 - Diagrama nodurilor ROS realizată de utilitarul *rqt_graph* pentru nodurile rulate de *minimal.launch*

4.2.3 Lansarea bringup.launch

Lansarea fișierului bringup.launch propune pe lângă rularea nodurilor specifice funcționalității minime și rularea a două noduri noi corespunzătoare senzorilor LiDAR și camera RGB.

În plus, pe lângă nodurile utilizate în mod direct în funcționalitatea robotului, atât în fișierele de launch bringup cât și în minimal este întâlnit un nod din pachetul *tf2_ros*, anume *static_transform_publisher*. Pachetul *tf2_ros* permite utilizatorului să urmărească mai multe sisteme de coordonate în timp și să transforme puncte, vectori, etc. între două sisteme de coordonate diferite în orice moment de timp.

O transformare are următoarele argumente:

"static_transform_publisher x y z yaw pitch roll frame_id child_frame_id period_in_ms"

Astfel, transformarea din bringup.launch are următoarele argumente:

args="0 0 0.030 0 0 0 /base_footprint /base_link"

, unde */base_footprint* reprezintă coordonata podelei, iar */base_link* reprezintă centrul platformei robotului, spune că centrul robotului se află la 0.03 m în sus pe axa Z față de podea.

4.2.3.1 Nodul laser corespunzător pentru senzorul LiDAR

Pentru a obține o autonomie deplină, stiva de navigație necesită ca robotul să aibă cel puțin o sursă de informații laser pentru a percep mediul robotului și a evita obstacolele pe parcursul rulării. Platforma Linorobot pune la dispoziție o suiată impresionantă de senzori și platforme hardware compatibile. Astfel, integrarea senzorului laser a fost facilă, fiind necesar SDK-ul pus la dispoziție de producător. În fișierul de rulare al nodului LiDAR sunt prezenți parametri de rulare, de la portul utilizat și rata de transfer baud-rate până la frecvența de rotație și rata de achiziție a datelor. Publicarea datelor de la LiDAR se face pe topicul */scan* într-un format specific datelor laser, numit *LaserScan* și pus la dispoziție de pachetul ROS *sensor_msgs*.

Asemenea fișierului părinte de launch, bringup, și fișierul de lansare a nodului senzorului LiDAR propune o transformare statică, cu argumentele:

```
args="-0.060 0 0.050 0 0 0 /base_link/laser"
```

Aceasta spune că frame-ul */laser* se află cu 0.06 m în spatele centrului platformei mașinii, la o înălțime de 0.05 m față de acesta.

4.2.3.2 Nodul cameră

Platforma Linorobot nu pune la dispoziție integrarea unui senzor de camera aşa că Lino-V propune și integrarea acestui senzor. După cum prezentam și în [subcapitolul 3.2.3](#), procesul de integrare a fost unul destul de dificil din cauza compatibilităților specifice Nvidia Jetson Nano. Însă, în final, aceasta a fost făcută prin intermediul pachetelor [\[16\]](#)[\[17\]](#) disponibile open source, rămânând să setez parametri în funcție de cerințele proprii de sistem.

Cum este nevoie de un timp scurt de trimitere și achiziție a datelor pe rețeaua ROS, fluxul video a fost compactat cât mai mult, astfel, parametrii ce țin de dimensiunea imaginii și de rata de transfer a informației au fost, 320 x 240 pixeli fiecare imagine la un framerate de 30 fps. Publicarea datelor este realizată pe topicul */csi_cam_0*.

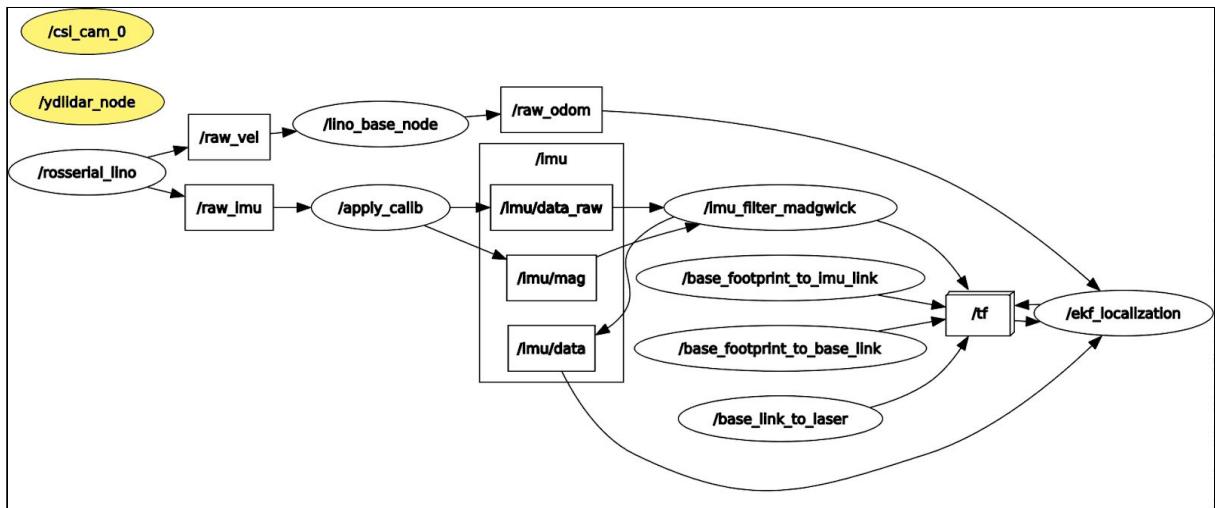


Figura 4.10 - Diagrama nodurilor ROS realizată de utilitarul *rqt_graph* pentru nodurile ruleate de *bringup.launch*

4.2.4 Lansarea slam.launch

Lansarea algoritmului de SLAM este legată în mod direct de lansarea a priori a utilitarului bringup.launch deoarece acesta rulează nodul de scanare a LiDAR-ului. Algoritmul de SLAM folosit în construcția platformei Linorobot este GMapping. Acesta folosește pentru cartografiere atât informații date de LiDAR (scanări) cât și informații de odometrie venite de la senzorii de mișcare. Un filtru de particule este utilizat pentru a estima probabilitățile posterioare, unde fiecare particulă reprezintă o traierorie potențială. O hartă individuală este, de asemenea, calculată pentru fiecare dintre particule, deoarece harta depinde foarte mult de traieroria robotului. În cele din urmă, particula cu cea mai mare probabilitate va fi aleasă și harta asociată va fi transmisă de către algoritm.

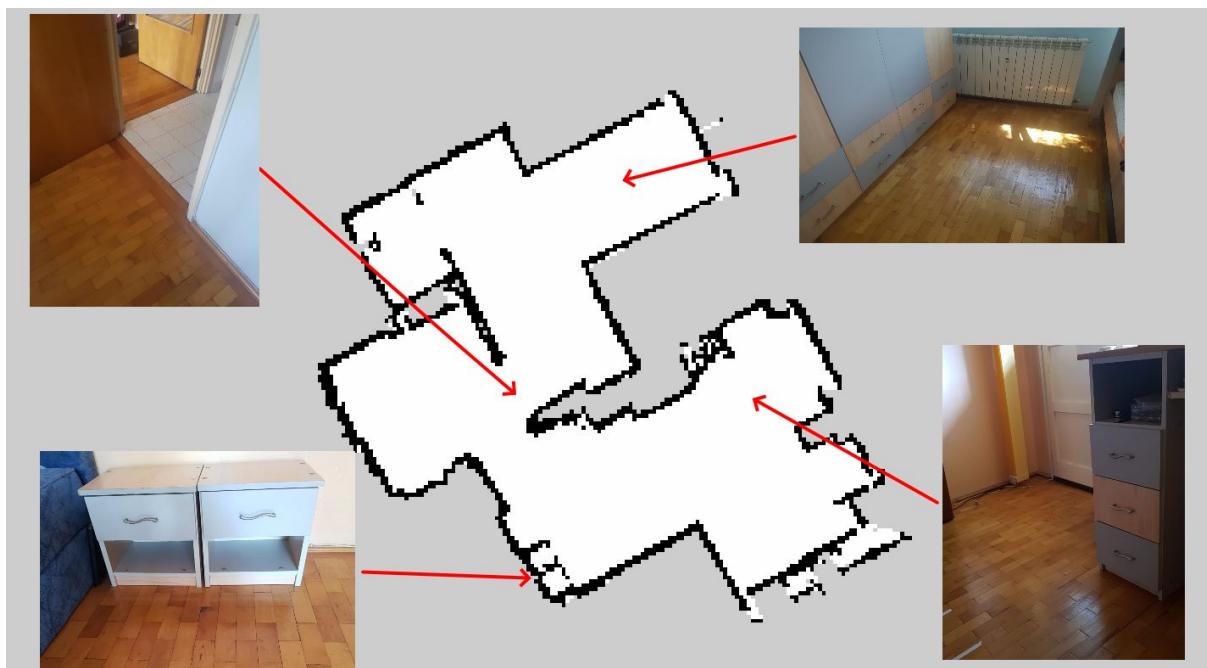


Figura 4.11 - Harta mediului realizată de către nodul gmapping. Săgețile subliniază corespondența dintre harta generată de algoritmul gmapping de SLAM și mediul real. Dispunerea obstacolelor se face pe baza unei grile de celule în care fiecare celule îi corespunde o suprafață de $0.05 \times 0.05 \text{ m}^2$ în realitate .

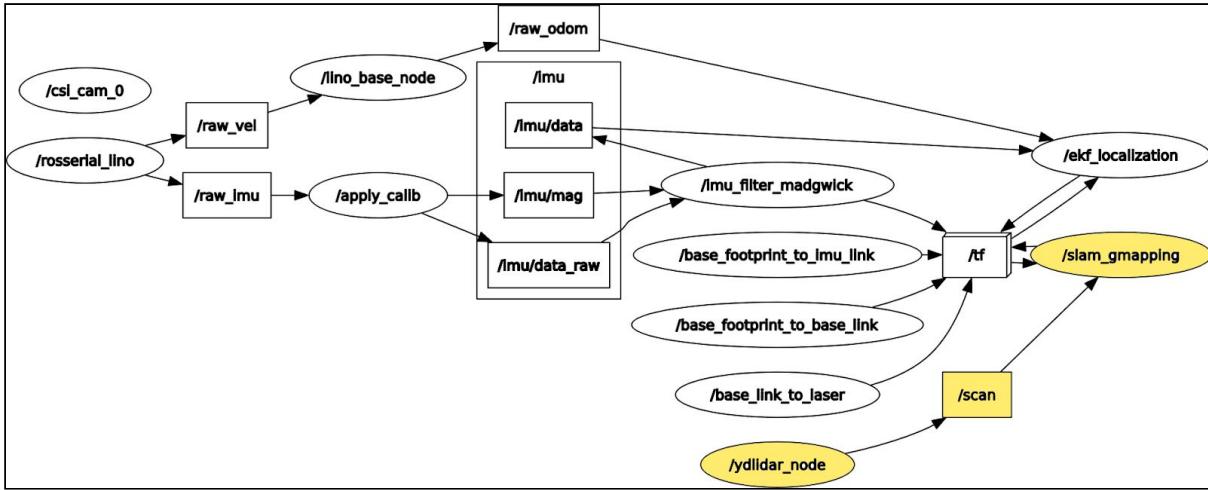


Figura 4.11 - Diagrama nodurilor ROS realizată de utilitarul rqt_graph pentru nodurile rulate de bringup.launch și slam.launch

4.2.5 Lansarea navigate.launch

Sistemul de navigare autonomă reprezintă nucleul de importanță și funcționalitate al platformei Lino-V. Prin intermediul calculelor și estimărilor oferite de acesta, stiva de navigație își coordonează comenziile automat.

Sistemul de navigare autonomă este împărțit în două subsisteme importante: localizarea folosind un filtru de particule (“Adaptive Monte Carlo Localization” - AMCL) și stiva de navigație specifică caracteristicilor robotului (numărul de motoare și sistemul de control), la care se adaugă un modul complementar de map_server care are rolul de a furniza harta generată anterior prin intermediul algoritmului de SLAM.

4.2.5.1 Localizarea Monte Carlo adaptivă

AMCL (“Adaptive Monte Carlo Localization”) folosește un [filtru de particule](#) pentru a reprezenta distribuția stărilor probabile, fiecare particulă reprezentând o stare posibilă, adică o ipoteză despre unde se află robotul. Asemenea oricărui filtru Bayesian estimativ algoritmul de localizare adaptivă se bazează pe doi pași: predicție și actualizare. Principiul de funcționare al algoritmului rămâne același ca cel descris în [secțiunea 2.4](#), se generează un set de N eșantioane ce aproximează poziția robotului în urma pasului de modificare a poziției iar apoi sunt incorporate citirile de la senzori care au rolul de a oferi ponderi estimărilor pe baza cărora se realizează și eșantionarea unui nou set de particule. AMCL propune, însă, și câteva particularități ce țin de robustețea algoritmului și a utilizării eficiente a resurselor

computaționale. Având în vedere că algoritmul generează eșantioane în proximitatea predicției sale cu privire la poziția actuală a robotului, există posibilitatea ca în cazuri restrânsse particulele generate să fi greșit localizarea. Dacă algoritmul ar continua să genereze particule în jurul predicției greșite, particulele nu ar mai putea converge niciodată spre poziția actuală a robotului. În consecință, după fiecare pas de estimare este adăugat un set mic de eșantionare aleatoare, uniform distribuite, care oferă posibilitatea ca în cazul pierderii complete a evidenței poziției robotului, algoritmul să îl poată relocaliza [21].

În practică, numărul de eșantioane necesare pentru localizare diferă în funcție de situație. De exemplu în cazul unei localizări globale, inițiale, un număr mare de particule este necesar pentru a se acoperi cât mai multe posibilități din cele existente, dar în cazul unei evidențe exakte a poziției robotului, cum particulele au convers destul de mult, un număr mic de eșantioane este necesar. Ceea ce aduce în plus AMCL este faptul că un număr mic de eșantioane este ales în cazul în care densitatea particulelor este concentrată într-o suprafață redusă și un număr mare în cazul în care incertitudinea este ridicată. Această abordare aduce îmbunătățiri semnificative din punct de vedere al implementării și din punct de vedere computațional față de metodele ce propun un număr constant de eșantioane. [22]

În cazul platformei Lino-V, algoritmul AMCL este implementat ca nod preexistent în comunitatea open source și pe baza unei hărți generată anterior, a scanărilor actuale ale senzorului LiDAR, a transformărilor corespunzătoare și a unei poziții inițiale predefinite, oferă ca output estimările poziției.

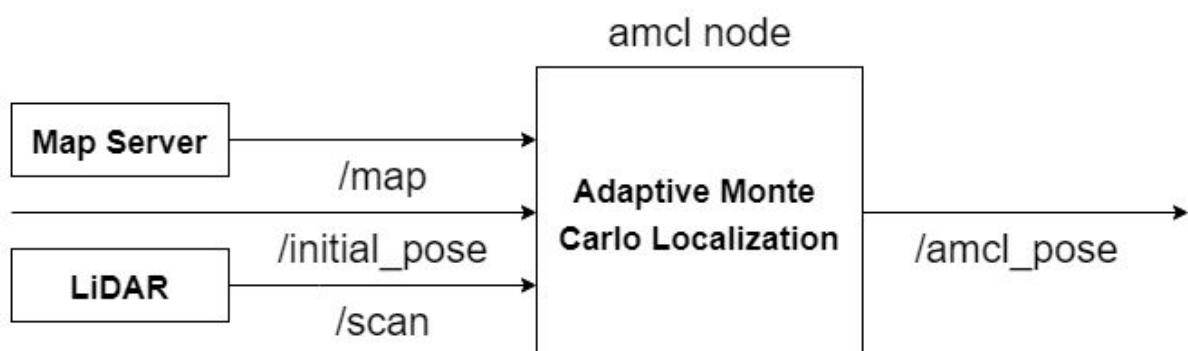


Figura 4.13 - Importanța nodului amcl în ecosistemul aplicației. Transformarea datelor de la LiDAR, și a informațiilor despre hartă în estimări ale poziției robotului.

Ecosistemul ROS pune la dispoziție un instrument puternic de vizualizare 3D a topicurilor și transformărilor prezente în sistem ce se numește RViz. Prin intermediul lui se poate realiza simplu depanarea funcționalității de la inputurile senzorilor până la acțiunile planificate sau neplanificate întreprinse de robot.

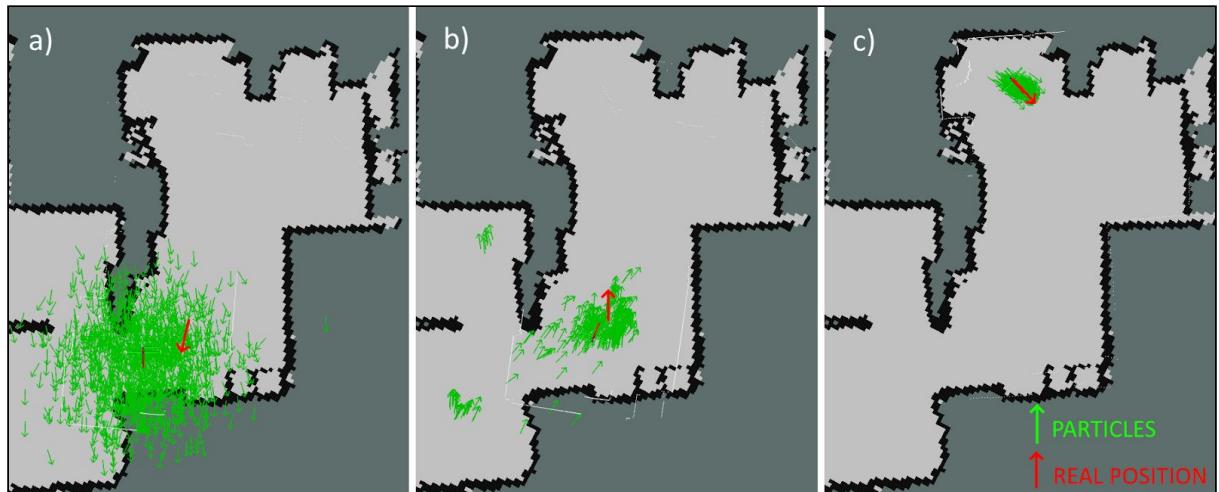


Figura 4.14 - Aplicarea algoritmului AMCL într-un context real. a) Incertitudine mare - Inițializarea filtrului de particule cu aproximativ 2000 de particule. b) Incertitudine medie - Robotul a realizat 2 rotiri complete și și-a schimbat orientarea. c) Incertitudine mică - Robotul a navigat prin hartă ajungând în direcția opusă inițializării. Particulele converg și numărul lor scade ajungând să fie între 500 și 1000.

4.2.5.2 Planificarea traiectoriei și stiva de navigație ROS

Nodul move_base[23] este un nod complex ce gestionează întreaga interacțiune dintre deciziile robotului și punerea în acțiune a acestora. Comunitatea ROS furnizează implementarea acțiunii în care, plecând de la două hărți de cost diferite[24], una locală și una globală și folosind un planificator pentru fiecare dintre cele două hărți[25][26], o țintă posibilă poate fi atinsă prin interacțiunea pachetului cu stiva de navigație a robotului.

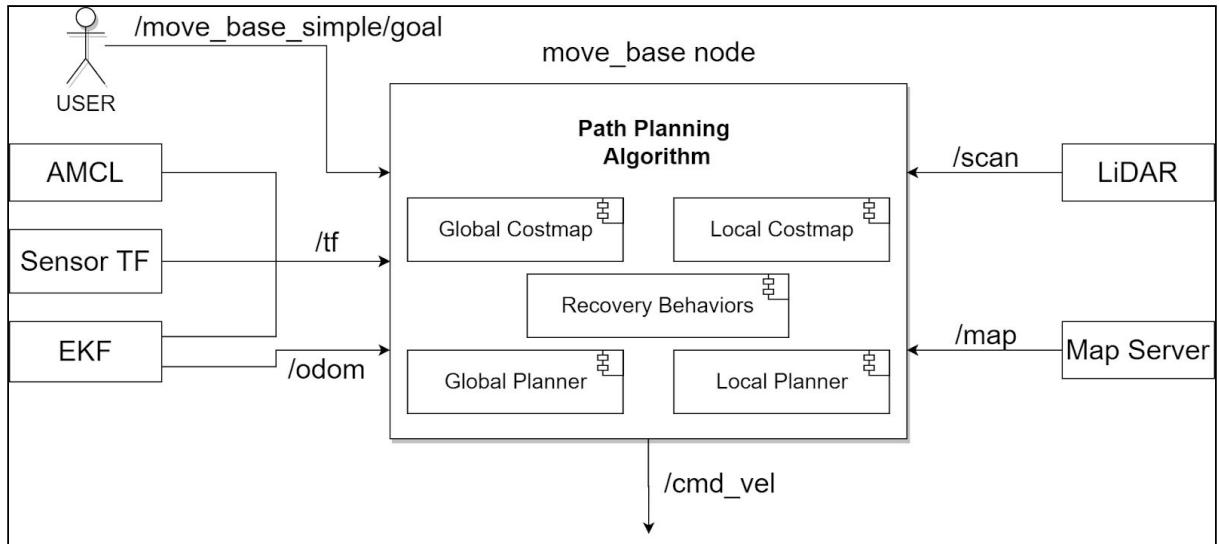


Figura 4.15 - O viziune high-level asupra nodului `move_base` și asupra interacțiunilor sale cu alte componente.

La baza planificatorului de cale stau cinci module, patru module ce definesc funcționalitatea și unul de diagnosticare și remediere a problemelor. Astfel, cele cinci module sunt: o hartă de cost globală și un planificator global, o hartă de cost locală și un planificator local și un modul de diagnosticare și remediere a erorilor.

Datele și informațiile pe care nodul `move_base` le ascultă și le integrează în sistemul intern de flux al proceselor sunt:

- ținta definită în mod dinamic de către utilizator și publicată pe topicul `/move_base_simple/goal`
- transformările propuse de nodul AMCL, de senzori și de nodul EKF
- datele de odometrie publicate de nodul EKF prin intermediul topicului `/odom`
- datele de scanare LiDAR de pe topicul `/scan`
- harta generată anterior de către algoritmul de SLAM publicată de nodul `map_server` pe topicul `/map`

După ce se analizează cele două hărți de cost corespunzătoare mediului robotului și se generează traiectorii pe baza acestora, nodul `move_base` publică prin intermediul topicului `/cmd_vel` către nodul `lino_base` informații pentru controlul motoarelor

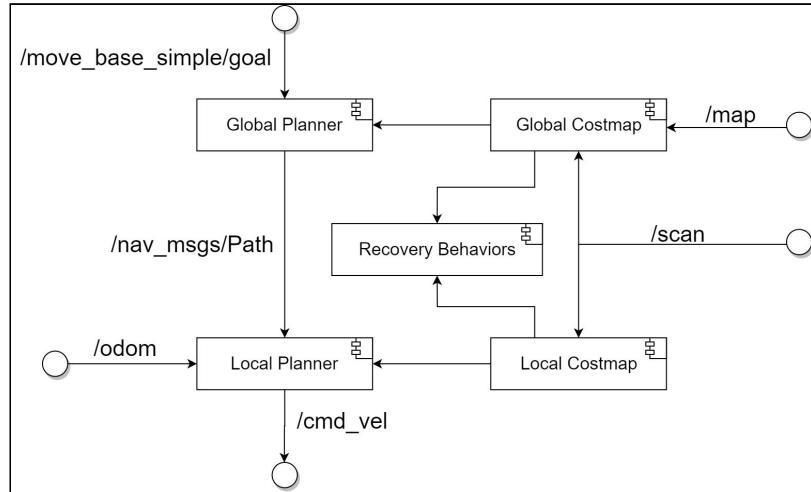


Figura 4.16 - O viziune low-level asupra nodului move_base și asupra interacțiunilor dintre modulele sale interne.

Modulele de creare a hărților de cost au rolul de a crea o grilă de ocupare 2D pornind de la informațiile primite de la senzori și de a “umfla” costurile în harta de cost definită pe baza grilei de ocupare și a razei de inflație definită de utilizator. Procesul de inflație reprezintă propagarea unor valori de cost care descresc direct proporțional cu distanța de la celulele ocupate. Procesul de descreștere are loc până la punctul definit de utilizator ca fiind raza de inflație.

Modulele hărților de cost primesc continuu informații de la senzorul LiDAR și actualizează hărțile în mod constant luând în considerare o frecvență de actualizare. Pe baza informațiilor de la senzor se pot adăuga obstacole în harta de cost, se pot elimina informațiile despre obstacole sau ambele. Pentru fiecare celulă din grilă se pot reprezenta intern trei valori diferite: liber, ocupat sau necunoscut.

Stiva de navigație folosește două hărți de cost pentru a stoca informații despre obstacolele prezente în mediul robotului. O hartă, cea globală, se ocupă de crearea unor traiectorii pe termen lung peste întregul mediu iar cealaltă se ocupă de evitarea obstacolelor imediate. Configurarea acestor hărți se realizează folosind două modalități specifice. Prima este configurarea unui profil comun celor două hărți în care sunt înregistrăți parametri ce țin de dimensiunile robotului, de pragurile pentru citirile obstacolelor, de raza de inflație sau de sursele de observație a robotului. A doua reprezintă configurarea fiecărui set de parametri pentru fiecare tip de hartă. Aici sunt configurați parametri ce țin de frecvențele de actualizare

și publicare ale hărților, a necesității utilizării hărții generate a priori de algoritmul de SLAM, iar în cazul hărții de cost locale a parametrilor ce țin de dimensiunea acesteia.

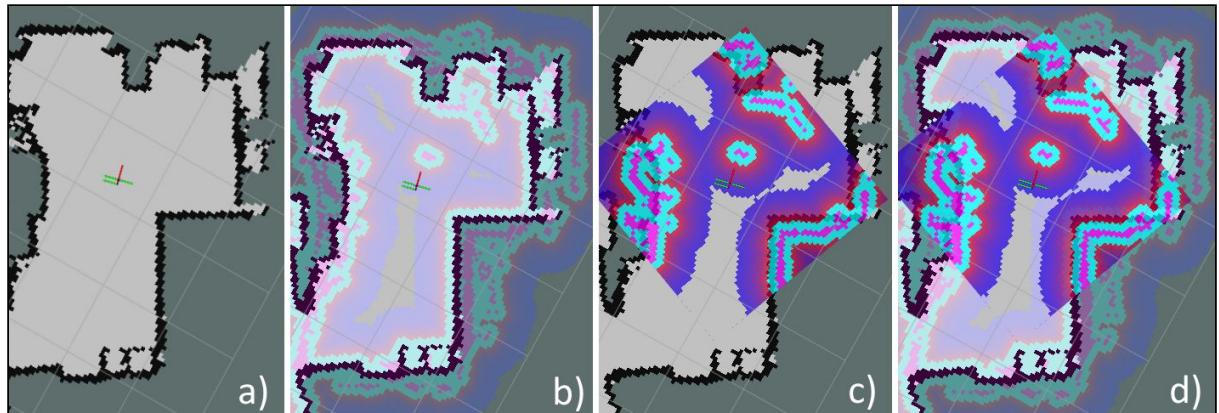


Figura 4.17 - Definirea hărților de cost corespunzătoare hărții generate de algoritmul de SLAM. a) Harta inițială generată de SLAM. b) Harta de cost globală - Harta de cost generată pentru întreg mediul în care se află robotul. c) Harta de cost locală - Harta de cost generată pentru o proximitate de $2.5 m^2$ și cu o rază de inflație de $0.55 m$. Zonele de culoare cyan reprezintă obstacolele iar zona de gradient roșu-albastru raza de inflație de $0.55 m$ cu valori de cost diferite în funcție de proximitatea de zona cyan. d) Harta de cost globală și locală.

Cum hărțile de cost sunt generate, urmează ca un algoritm dinamic de alegere a traiectoriei să fie aplicat pe grilele de cost corespunzătoare hărților. De data aceasta, abordările pentru planificatorul global și local sunt diferite.

Pentru planificatorul global este folosit pachetul ROS *global_planner*. Acesta este folosit în varianta lui standard, având în centru algoritmul Dijkstra de găsire a traiectoriei de cost minim. Un alt argument standard important este cel ce definește modul în care este generată traiectoria, dacă respectă marginile celulelor din grila inițială. Parametrul standard de aici permite folosirea unei metode de “gradient descent” ce oferă o secvențialitate fluidă mișcărilor robotului.

Pentru planificarea locală algoritmul DWA (“Dynamic Window Approach”) este folosit pentru a genera traiectorii în apropierea robotului. Ideea de bază a algoritmului DWA este următoarea:

- Generează o eșantionare discretă a deplasărilor dx, dy, dtheta
- Simulează deplasarea fiecărui eșantion generat anterior
- Oferă un scor fiecărei traectorii pe baza apropiерii de obstacole, de țintă, de traectoria globală și viteza
- Alege traectoria cu cel mai mare scor
- Repetă

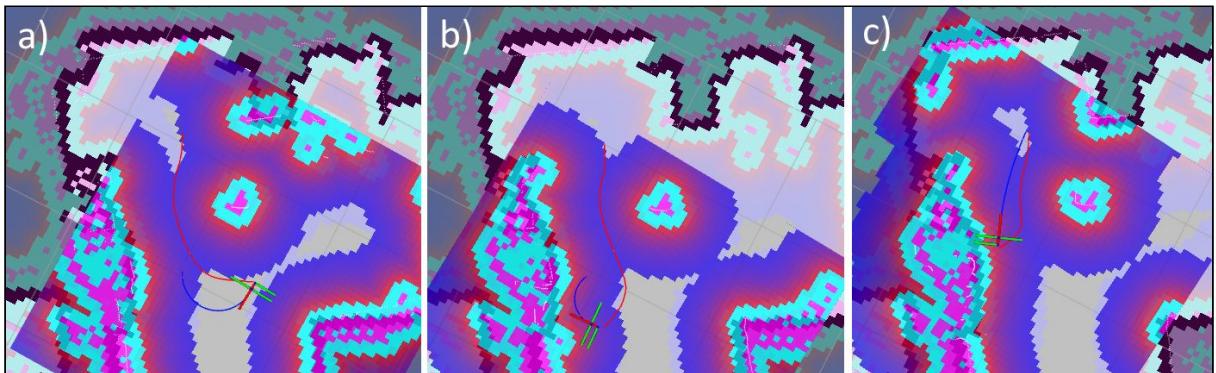


Figura 4.18 - Planificarea celor două traectorii: globală și locală. a)-c) Planificarea globală este reprezentată de curba roșie iar cea locală ce ia în considerare constrângeri de proximitate și orientare este reprezentată de curba albastră.

Ultimul aspect de tratat în acest subcapitol este modul în care robotul remediază erorile apărute în timpul navigării. Patru comportamente diferite sunt invocate în cazul apariției unor erori. Primul, cel mai ușor, este că o mică porțiune din hartă este resetată și se reîncearcă planificarea. Dacă aceasta nu reușește, următorul comportament este o rotație în loc. Dacă nici după aceasta robotul nu poate naviga, o nouă curățare a hărții este realizată, de data aceasta ceva mai agresivă, ștergându-se totul în afara unui dreptunghi în care robotul se poate roti. În final, ultimul comportament este o nouă rotație în loc. Dacă nici după această operațiune robotul nu poate reîncepe navigarea atunci se declară planificarea traectoriei eşuată.

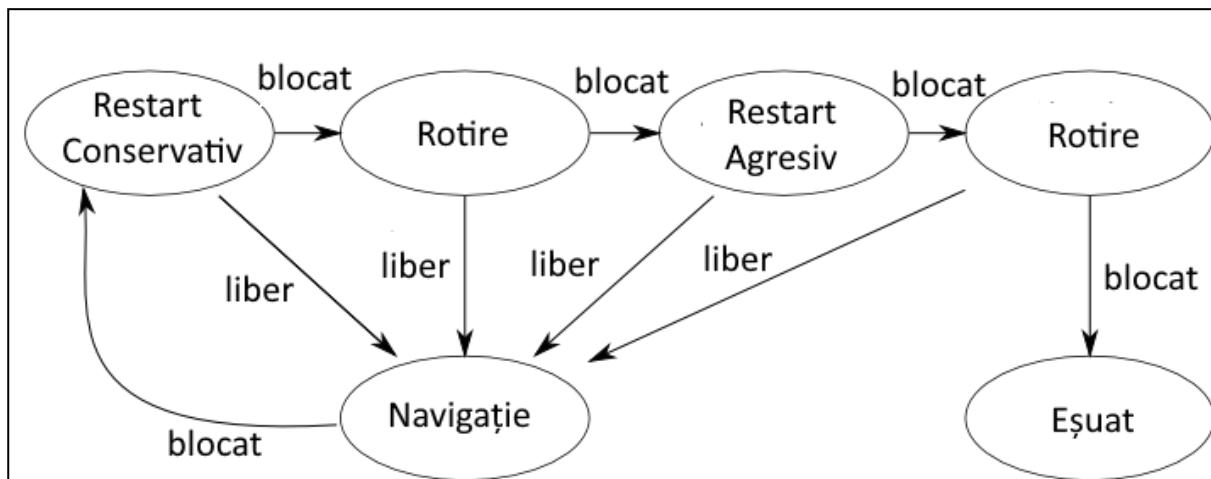


Figura 4.19 - Cele patru comportamente de recuperare în cazul apariției unei erori propuse de nodul move_base. Se testează fiecare modul, dacă eșuează atunci se trece la următorul, dacă nu, reîncepe navigarea. Eșuarea tuturor modulelor duce la terminarea procesului de navigare.

4.2.6 Lansarea lane_detector.launch

Integrarea camerei în platforma Lino-V a adus posibilitatea de a implementa algoritmi de vedere artificială avansați. Pentru că se potrivește în cazul de utilizare actual, am ales să folosesc un algoritm de detecție a benzilor implementat anterior de mine în contextul cursului online de vehicule autonome “Udacity Self Driving Car Nanodegree”.

Algoritmul de detecție a benzilor este reprezentat de cinci pași:

- filtrarea imaginii
- modificarea perspectivei
- găsirea polinoamelor benzilor
- măsurarea poziției vehiculului relativ la banda pe care este prezent
- revenirea la perspectiva inițială.

Integrarea acestui algoritm în ecosistemul ROS a fost realizată prin intermediul nodului *image_node* care are rolul de a prelua imagini de la cameră și de a publica imaginile cu benzile procesate pe topicul */lanes_image*.

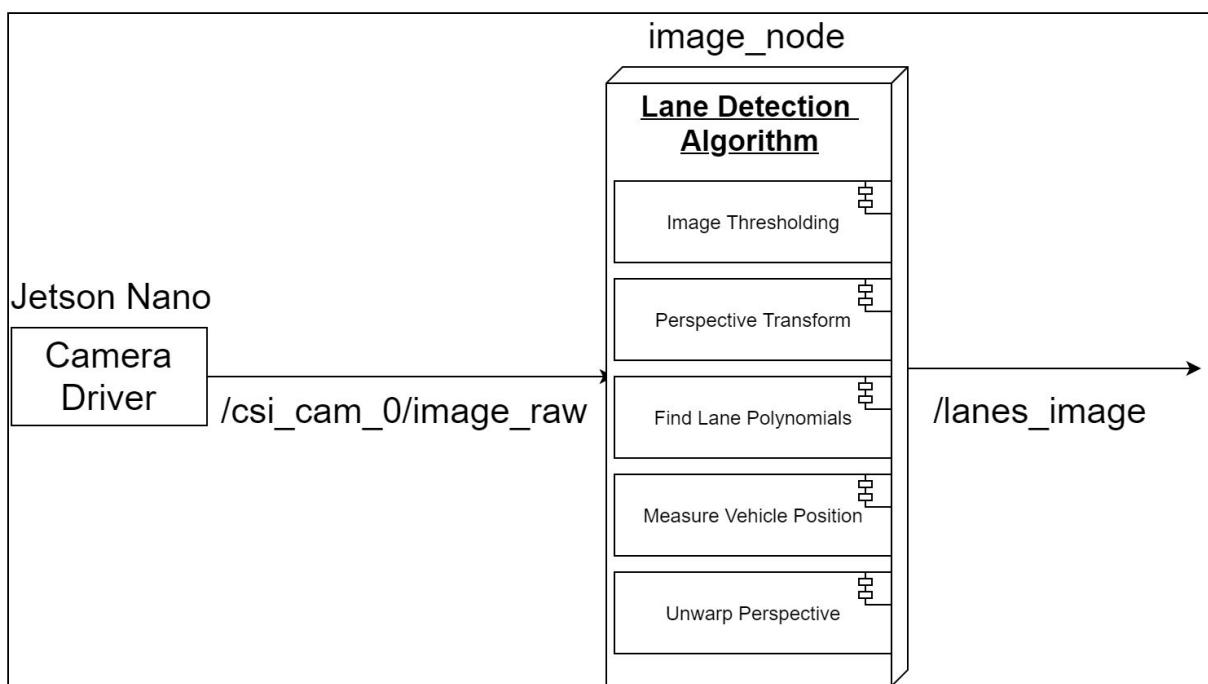


Figura 4.20 - Importanța nodului *image_node* în ecosistemul aplicației. Procesarea imaginilor și găsirea benzilor în ele.

4.2.6.1 Filtrarea de prag a imaginii

Pentru că am realizat în timpul implementării că folosirea unor filtre pe bază de gradient este foarte predispusă la erori din cauza apariției anumitor anomalii la nivelul suprafeței pe care se află mașina (testele actuale se fac pe o suprafață de parchet, nu pe asfalt), am decis că în final metodele de filtrare a imaginii să se realizeze pe baza spațiilor de culori. Au fost realizate 3 filtrări la nivel de spații de culori, toate dintre ele fiind făcute asupra celui de-al treilea canal al spațiului de culori cieLAB, ce exprimă culorile ca trei valori: L * pentru lumină de la negru (0) la alb (100), a * de la verde (-) la roșu (+) și b * de la albastru (-) la galben (+). Acest canal a fost ales deoarece este cel mai puțin sensibil la modificările de luminozitate și umbrări a mediului.

Metodele de definire a pragului de intensitate au fost trei, o filtrare statică (prin intermediul funcției threshold din OpenCV) și două filtrări adaptive (prin intermediul funcției adaptiveThreshold din OpenCV), pentru fiecare dintre cele două folosindu-se metode diferite (*ADAPTIVE_THRESH_MEAN_C* și *ADAPTIVE_THRESH_GAUSSIAN_C*). În plus, cea de-a

treia metoda de filtrare a suferit și o operație morfologică de “opening” (operație de eroziune urmată de una de dilatare pentru a scapa de zgomotul alb; exact ca în cazul unei convoluții 2D, un nucleu (kernel) este trecut peste fiecare pixel al imaginii).

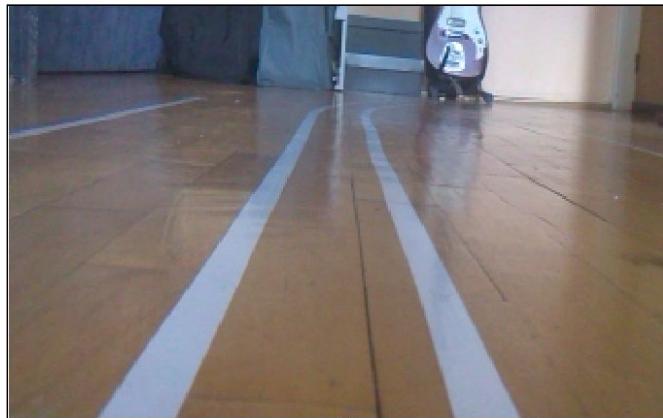


Figura 4.21 - Imaginea originală transmisă de cameră pe topicul /csi_cam_0/image_raw.

Imagine cu două benzi albe în cazul unei iluminări bune.



Figura 4.22 - Cele 3 metode de filtrare de prag în cazul unei iluminări bune. a) Filtrare de prag binară statică. b) Filtrare de prag adaptivă cu ADAPTIVE_THRESH_MEAN_C. c) Filtrare de prag adaptivă cu ADAPTIVE_THRESH_GAUSSIAN_C și operație morfologică de “opening”

În final, cele trei imagini binare rezultate în urma filtrării de prag sunt compactate într-o singură imagine grayscale care suferă și ea o operație morfologică de closing (operație de dilatare urmată de una de eroziune pentru a umple golurile de detecție a albului).

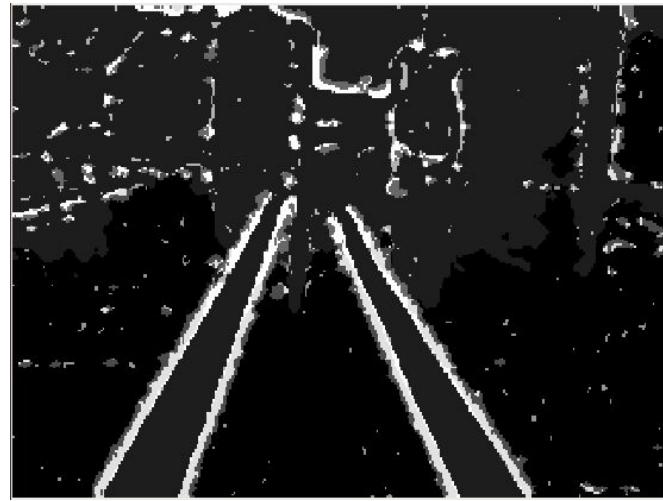


Figura 4.23 - Filtrare benză în cazul unei iluminări bune. Compactarea celor trei metode de filtrare de prag într-o singura imagine

Algoritmul de detecție a benzilor albe se descurcă destul de bine și în cazul unei iluminări dificile a mediului.

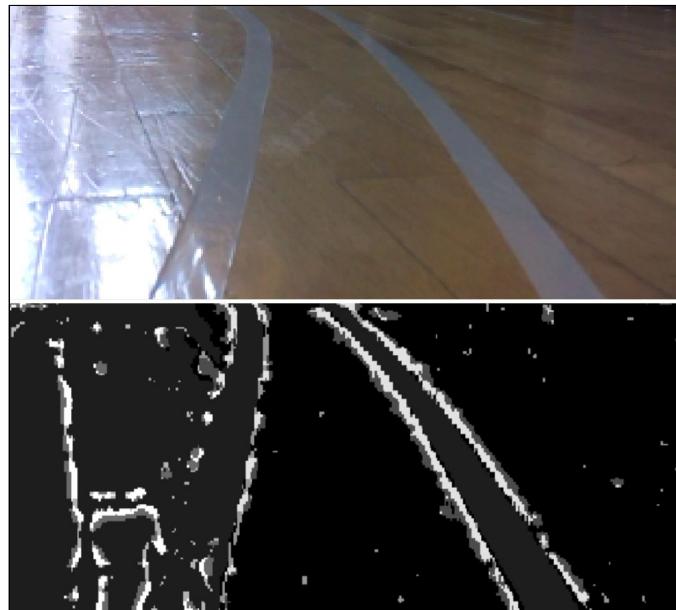


Figura 4.24 - Comparația imaginii originale cu o iluminare necorespunzătoare cu cea filtrată de metoda prezentată mai sus

4.2.6.2 Modificarea perspectivei

Pentru a putea realiza o reprezentare corectă a benzilor din imagini o modificare de perspectivă a fost realizată. Imaginea rezultată, ce poartă și denumirea de “Bird Eye View”, a fost construită prin intermediul unei funcții specifice OpenCV, ce aplică o transformare de perspectivă în funcție de două seturi de puncte, patru puncte sursă și patru puncte destinație. Pe baza celor două seturi de puncte se definește o matrice de transformare ce este aplicată imaginii inițiale.



Figura 4.25 - Aplicarea transformării de perspectivă. Aplicarea unei transformări de perspectivă în proiectul personal ”Self Driving Car Nanodegree”

4.2.6.3 Găsirea polinoamelor benzilor

Pentru identificarea pixelilor de linie de bandă am folosit un algoritm de căutare a benzilor, denumit și “*sliding window*” urmat de o interpolare a punctelor definite pentru fiecare dintre cele N ferestre. Primul algoritm folosește o histogramă inițială a densității pixelilor activați în imaginea dată de algoritmul de filtrare. Această histogramă definește punctul de plecare pentru metoda de “*sliding window*”, care pe baza percepției inițiale asupra poziției celor două benzi, verifică în stânga și dreapta (având în vedere o anumită margine) ferestrei anterioare existența unei alte densități de pixeli activați. Pentru cele N ferestre găsite anterior se interpolează un polinom de gradul doi.

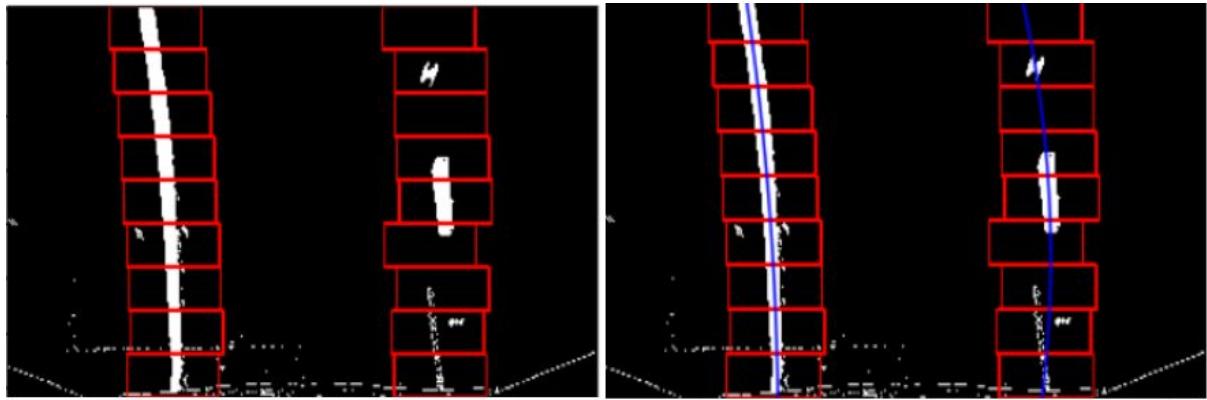


Figura 4.26 - Găsirea polinoamelor benzilor folosind cei doi pași. Prima poză prezintă aplicarea “sliding window” în găsirea densităților pixelilor activați. A doua poză prezintă interpolarea centrelor ferestrelor cu un polinom de grad 2.

După ce sunt găsite cele două linii corespunzătoare benzi, ultimul lucru ce rămâne de făcut este umplerea și evidențierea spațiului dintre cele două polinoame cu o culoare sugestivă.

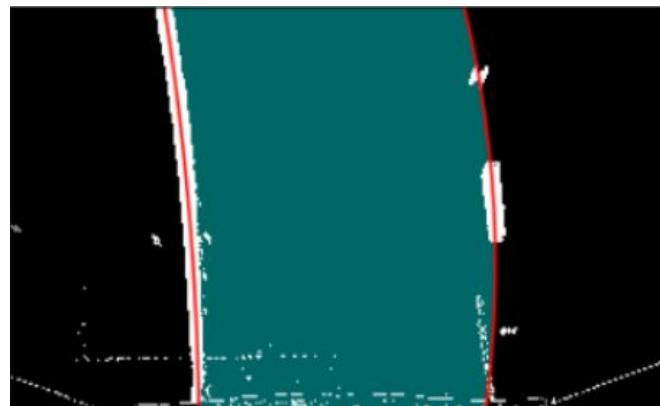


Figura 4.27 - Evidențierea benzii dintre cele două polinoame găsite. Colorarea suprafeței dintre cele două benzi

4.2.6.4 Măsurarea poziției vehiculului relativ la banda pe care este prezent

Pentru a determina poziția vehiculului în interiorul benzii am folosit cele două polinoame determinate anterior. Astfel, definind constantele metrice de conversie a pixelilor în centimetri (constantă atribuită perspectivei montării camerei) și valoarea polinomului în

partea de jos a imaginii, am putut determina valoarea deplasării spre stânga/dreapta față de centru.

4.2.6.5 Revenirea la perspectiva inițială

Revenirea la perspectiva inițială se realizează cu ajutorul inversei matricei de transformare prezentate în [secțiunea 4.2.6.2](#). După cum știm, multiplicarea unei matrice cu inversa ei rezultă matricea identitate, ceea ce în secvențialitatea operațiilor de transformare a perspectivei ne reîntoarce la perspectiva inițială.



Figura 4.28 - Imaginea rezultată procesată cu benzile găsite și publicată de către nodul image_node pe topicul /lanes_image. În partea de sus rezultatul găsirii benzii împreună cu calcularea deplasării față de centru. În partea de jos, stânga, imaginea filtrată de cele trei metode, iar în partea dreaptă transformarea de perspectivă.

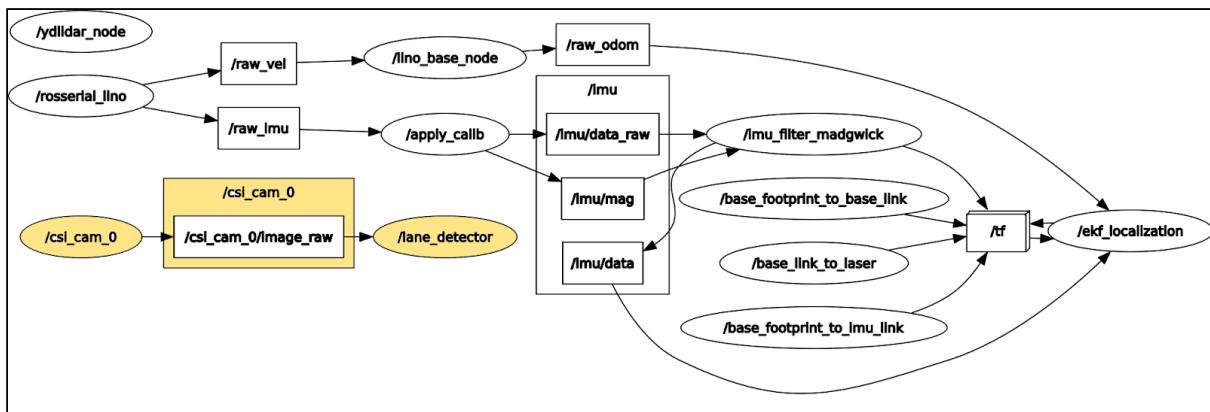


Figura 4.29 - Diagrama nodurilor ROS realizată de utilitarul rqt_graph pentru nodurile ruleate de bringup.launch și lane_detector.launch

4.3 Pachetele ROS ale computerului de dezvoltare

Conecțivitatea dispozitivelor se dezvoltă într-un mod alert, iar cum Lino-V a fost gândit să propună o scalabilitate mare, următorul modul important implementat a fost cel al controlului securizat al vehiculului la distanță. Securitatea vine din implementarea unui cumul de condiții de implementare ce se bazează pe rețelele neuronale.

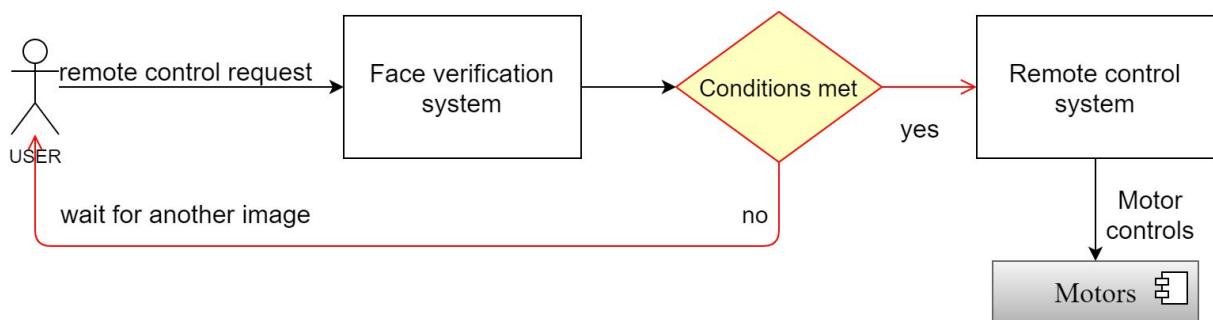


Figura 4.30 - Fluxul decizional al modulului de control la distanță. Controlul motoarelor poate fi obținut doar dacă imagini cu față utilizatorului îndeplinesc anumite condiții descrise de rețelele neuronale din sistemul de verificare facială.

4.3.1 Sistemul de verificare facială

Sistemul de recunoaștere facială, care stă la baza autentificării utilizatorului, este compus în întregime din rețele neuronale. Mai exact, pentru ca acesta să fie unul robust și sigur, două module au fost implementate:

- 1) o rețea neuronală cu rolul de a detecta fețele din imagine și a le alinia
- 2) o rețea neuronală ce verifică dacă una dintre fețele detectate este a persoanei căutate

Pentru ca un utilizator să se poată autentifica, este nevoie ca, pentru fiecare dintre cele trei module, să se îndeplinească anumite condiții.

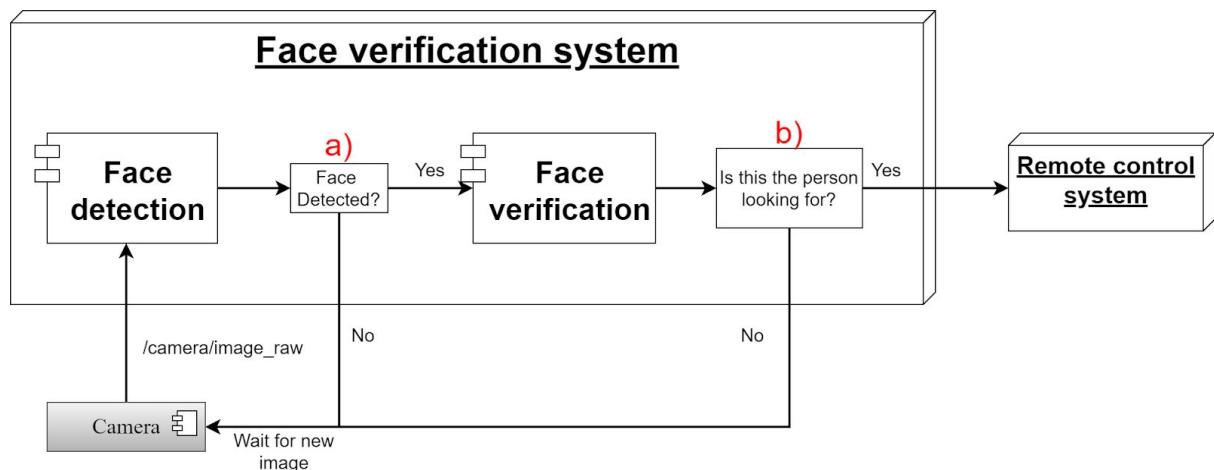


Figura 4.31 - Fluxul decizional al sistemului de verificare facială. a) Condiție continuare detecție facială - A fost detectată cel puțin o față? b) Condiție continuare verificare facială - Este fața detectată a persoanei căutate?

4.3.1.1 Detecția facială

Detecția facială reprezintă procesul prin care sunt identificate fețe umane în imagini. Mai multe abordări ale acestei sarcini sunt prezente, de la algoritmi ce utilizează caracteristici Haar până la abordări folosind rețele neuronale. Având în vedere că tratează cu succes situațiile cu ocluzii, variații mari ale poziționării și iluminări extreme, iar în plus, are un timp de inferență foarte scăzut, am folosit pentru detecția facială un algoritm bazat pe rețele neuronale, anume MTCNN ("Multi-task Cascaded Convolutional Networks") [27].

Metoda MTCNN de detecție facială se bazează pe trei rețele neuronale conectate secvențial care au rolul de a rafina și oferi o detecție cât mai precisă a caracteristicilor faciale. Cele trei rețele neuronale sunt denumite P-Net, R-Net și O-Net

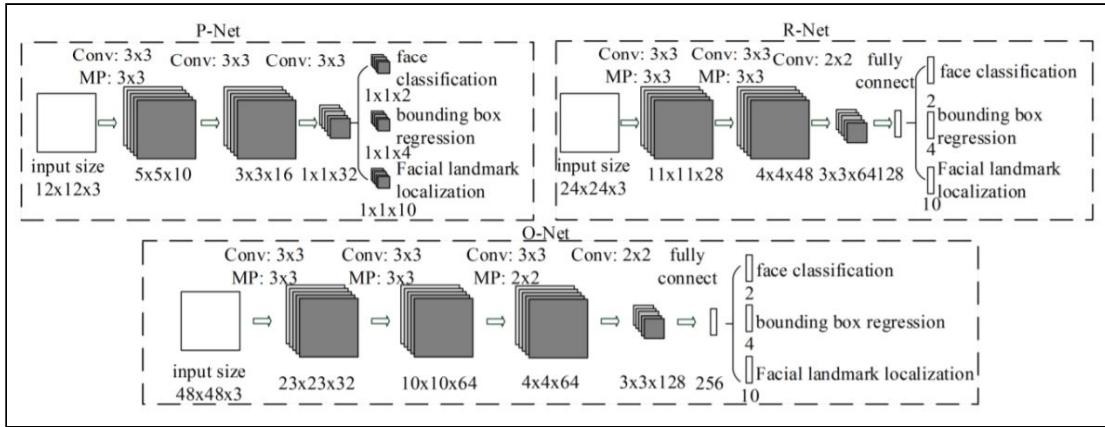


Figura 4.32 - Arhitecturile rețelelor P-Net, R-Net și O-Net. Pasul pentru conoluție este 1 iar pentru stratul de “pooling” este 2. [27]

Având în vedere existența celor trei rețele, sistemul de detecție MTCNN este format din trei pași. Procedura inițială este de a crea o piramidă a imaginii prin construirea unor copii scalate ale imaginii inițiale. Această piramidă a imaginii reprezintă input-ul rețelei P-Net care are rolul de a oferi ca output coordonatele casetelor de încadrare ale fețelor și a gradul de încredere pentru fiecare dintre casete. După ce au fost determinate primele încadrări ale fețelor urmează trecerea acestora printr-o a doua rețea neuronală care are rolul de rafinare și mai multe încadrările având același output ca și P-Net, doar dimensiunea casetelor de încadrare diferă. Ultimul pas este similar cu cel de-al doilea, doar că aici față este descrisă în mai multe detalii, output-ul fiind reprezentat pe lângă coordonatele casetelor de încadrare și gradul de încredere și de coordonatele a cinci repere faciale.

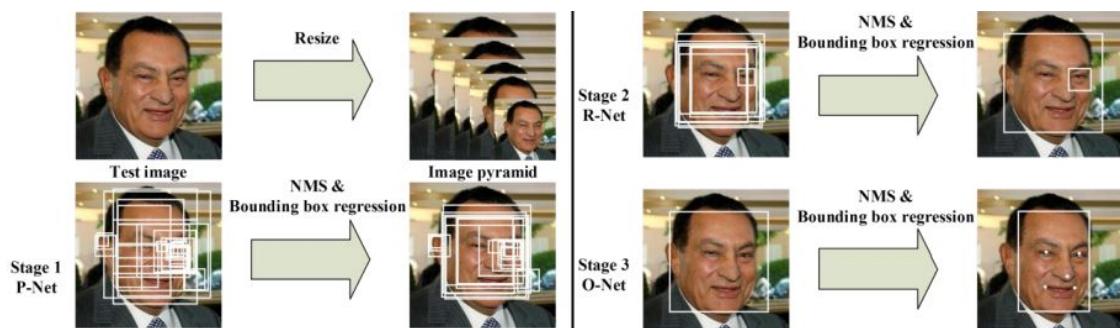


Figura 4.33 - Pipelineul arhitecturii MTCNN. Crearea piramidei imaginii și inferențele celor 3 rețele P-Net, R-Net și O-Net. [27]

Implementarea acestei metode de detecție a fost făcută cu ajutorul unui pachet disponibil open source[28] care implementează pe lângă o rețea Inception Resnet V1 de recunoaștere facială și o rețea eficientă MTCNN. Întregul mediu de inferență a rețelelor a fost implementat pe baza framework-ului PyTorch.



Figura 4.34 - Exemplul inferenței rețelei MTCNN. Detecția facială MTCNN peste imaginea a celor 29 de oameni de știință la conferința de la Solvay. Peste 95% din fețe au fost detectate.

4.3.1.2 Verificare facială

Verificarea facială reprezintă procesul prin care se compară o față candidată cu alta verificând dacă este sau nu o potrivire între cele două. Este o mapare unu la unu (“one to one”), verificând dacă persoana verificată este cea corectă. Acest proces este diferit față de cel de recunoaștere facială care este o problemă de unul la mai mulți (“one to many”) în care, pe baza unor caracteristici se verifică ce persoană dintr-o bază de date de persoane este cea căutată.

Abordarea folosită în această lucrare este cea propusă în anul 2015 de un grup de cercetători de la Google prin lucrarea lor “FaceNet: A Unified Embedding for Face Recognition and Clustering” [29]. FaceNet realizează o mapare a imaginilor cu fețe într-un spațiu euclidian unde distanțele dintre două reprezentări reprezintă o măsură a nivelului lor de similaritate.

Scopul unei rețele FaceNet este de a crea o reprezentare a imaginii inițiale (sub forma unui vector unidimensional) astfel încât distanța dintre două reprezentări construite astfel să

fie diferită în funcție de nivelul de similaritate: reprezentările aceleiași fețe să fie despărțite de o distanță mai mică în spațiul euclidian al reprezentărilor decât reprezentările a două identități diferite. Modul de antrenare al rețelelor FaceNet este puțin diferit față de cel de antrenare al rețelelor clasice. În cazul FaceNet, antrenarea se realizează optimizând direct reprezentările, prin crearea unor triplete imagini în care una dintre imagini este imaginea referință, numită și anora, iar celelalte două, o pereche pozitivă care este o imagine ce reprezintă aceeași identitate ca referință și o pereche negativă ce reprezintă o identitate diferită față de referință.

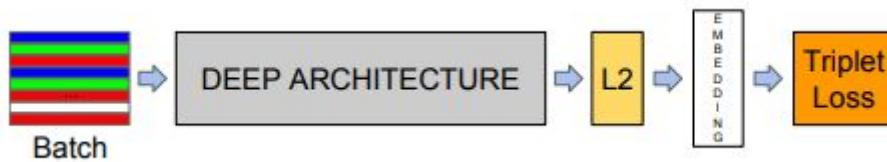


Figura 4.35 - Structura modelului FaceNet. Inputul este trecut printr-o arhitectură de învățare profundă și apoi este normalizat folosind norma L_2 rezultând reprezentările. Apoi, în procesul de antrenare funcția de cost folosită este cea triplet.

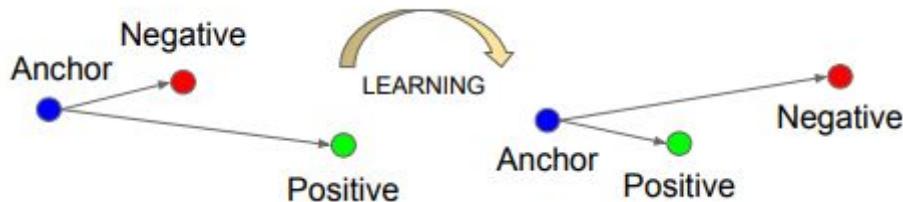


Figura 4.36 - Funcția de cost triplet. Modul de funcționare a funcției de cost specifică FaceNet care minimizează prin antrenare distanța dintre reprezentarea referință/ancoră și perechea pozitivă și maximizează distanța dintre referință/ancoră perechea negativă.

Implementarea metodei FaceNet a fost realizată folosind același pachet open source ca și în cazul implementării [secțiunii 4.3.1.1](#). Pentru modulul de recunoaștere/verificare facială implementarea propune un model Inception Resnet V1 preantrenat pe setul de antrenare VGGFace2. Am ales această implementare în final pentru că, deși am încercat o bună perioadă de timp să dezvolt un model de “transfer learning” pretabil pentru această sarcină nu am reușit să ajung la o performanță bună în timp real. Pentru inferență am ales ca măsură a nivelului de similaritate dintre două reprezentări o funcție de similaritate cosinus

care poate lua valori între -1 și 1, cu -1 reprezentând opoziție totală și 1 simbolizând exact aceeași reprezentare.

Pentru a avea o referință a feței proprii, am trecut 8 poze cu fața mea detectată de MTCNN prin rețeaua Inception Resnet. În plus, am adăugat și câte un set de poze și pentru David Beckham, Barack Obama, Elon Musk și Cristiano Ronaldo. Pozele selectate nu au avut anumite atribute, toate erau realizate într-o iluminare bună, cu un unghi de orientare al feței cât mai perpendicular pe cameră. Outputul oferit de Resnet a fost un vector unidimensional de 512 elemente pentru fiecare imagine. Toate cele 8 reprezentări vectoriale au fost strânse și compactate într-un singur vector referință printr-o procedură de mediere a valorilor. De aici, performanța preantrenării rețelei de FaceNet și-a spus cuvântul, izolând într-o manieră foarte potrivită identitățile diferite. Rata de similaritate folosind similaritatea cosinus este de aproximativ 0.85 pe camera VGA a laptopului de dezvoltare. Aceasta depășește acest prag pe camere mai performante.

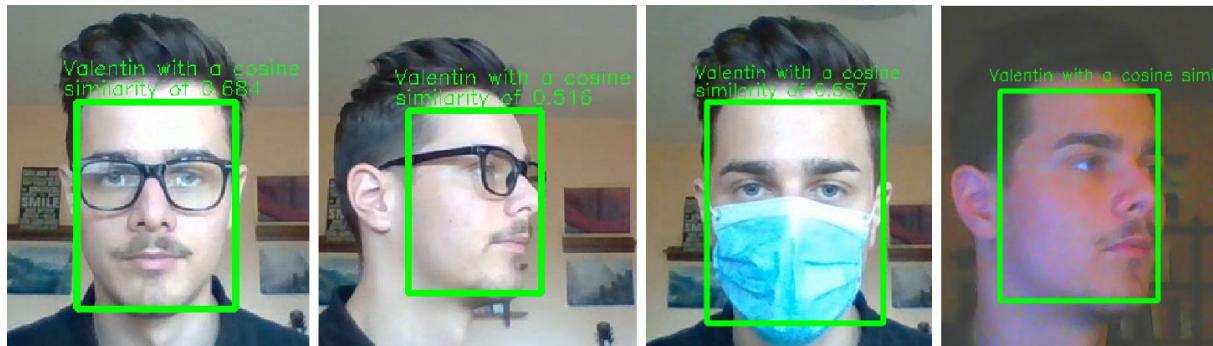


Figura 4.37 - Rezultatul oferit de sistemul de verificare facială. Au fost oferite imagini care au depășit aria în care au fost situate imaginile de referință. Au fost folosiți ochelari, modificări ale poziției față de cameră, acoperirea parțială a feței cu o mască sau iluminări nepotrivite. În toate cele patru cazuri enumerate algoritmul a înregistrat similarități cosinus potrivite după cum urmează, de la stânga la dreapta: 0.68, 0.52, 0.59, 0.63.

Constrângerea cea mai mare de sistem de care m-am lovit în cazul verificării faciale a fost cea a performanței foarte scăzute a camerei web a computerului de dezvoltare, aceasta fiind VGA. Din cauza acestor performanțe scăzute, camera distinge foarte greu fețele celorlalte identități decât a mea, fiind nevoie să ofer imaginile prin intermediul unor poze de

pe telefon. Având această problemă am configurat un alt computer să genereze un stream video de la camera web, cu o calitate mult mai mare a imaginii, însă fluiditatea fluxului video scade din cauza trimiteri lui ulterior peste rețea ROS.



Figura 4.38 - Rezultatul oferit de sistemul de verificare facială în cazul unor noi identități.

Identitatea de verificat este introdusă de la tastatură. Astfel, în imaginea a), algoritmul detectează fața lui David Beckham cu o similaritate cosinus de 0.808. În b), verificând identitatea David Beckham algoritmul detectează fața mea însă refuză autentificarea deoarece nu este fața dorită. c) Este detectată fața lui Barack Obama, iar similaritatea atribuită imaginii este de 0.755 similaritate cosinus. a) și c) sunt poze realizate de camera web a computerului adițional, de aici și posibilitatea detectării fețelor în contextul dat.

4.3.2 Sistemul de control la distanță

Sistemul de control la distanță se bazează pe un nod ROS prezent în pachetele standard ale sistemului. Pentru a servi în totalitate scopurilor platformei Lino-V, acest pachet standard a suferit modificări. Funcționalitatea de nod de control la distanță în ecosistemul aplicației a fost atribuită nodului *lino_keyboard* care are la bază următorul flux de informație: utilizatorul trimit comenzi prin tastatură în terminal, nodul *lino_keyboard* le preia și le interpretează, iar după ce interpretarea lor a luat sfârșit acesta publică informații despre viteza pe topicul */cmd_vel*. Făcând o paralelă cu mecanismul discutat în [secțiunea 4.2.5](#), acum nodul *lino_keyboard* preia rolul pe care, în situația navigării autonome, îl avea nodul *move_base*.

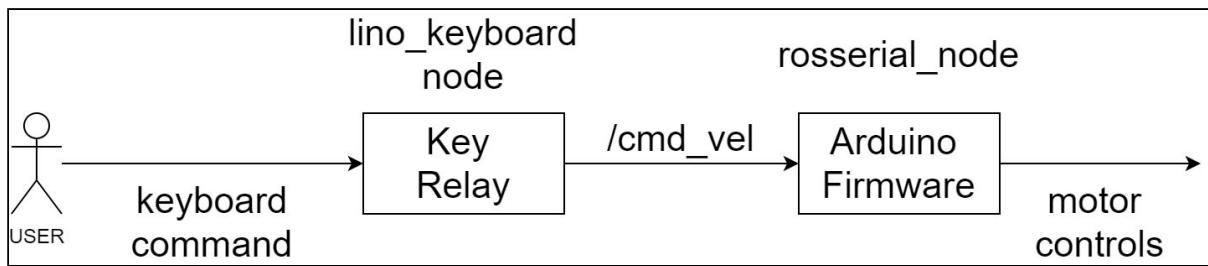


Figura 4.39 - Importanța nodului lino_keyboard în ecosistemul aplicației. Inputul trimis la tastatură de utilizator este interpretat și pe topicul /cmd_vel sunt trimise informații despre controlul motoarelor.

```

Reading from the keyboard and Publishing to Twist!
-----
Moving around:
      w
      a     d
      s

anything else : stop
q/z : increase/decrease max speeds by 10%
r/v : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
CTRL-C to quit

currently:      speed 0.25      turn 1.0

```

Figura 4.40 - Interfața terminal a nodului lino_keyboard. Au fost definite tastele w, s, a, d să direcționeze robotul față, spate, stânga, respectiv dreapta, iar q/z, r/v, e/c au rolul de a modifica vitezele liniare și unghiulare ale robotului. A mai fost adăugată tasta k pentru oprirea instantanea a robotului.

Concluzii

În această lucrare a fost propusă analiza proprie asupra modalităților și implementărilor specifice folosite în industria roboților autonomi. Modulul de navigare autonomă se bazează pe algoritmi eficienți precum SLAM (generează o hartă precisă a mediului), a filtrelor Kalman (estimează deplasările în mediu), a filtrelor de particule (estimează poziția în hartă) și a modalităților de creare a unor hărți de cost pe baza cărora se generează traectorii locale (cu o dinamicitate ridicată) și globale. Integrarea acestor algoritmi este posibilă datorită mecanismului de fuziune a datelor de la mai mulți senzori precum senzorul LiDAR și senzorii de mișcare: encoderele motoarelor și senzorul IMU. În plus, pe lângă modulul de navigare a fost adaugat un detector de benzi de circulație, bazat pe integrarea unui modul de camera RGB, cu o eficiență ridicată și cu o aplicabilitate reală în contextul mașinilor reale. Luând în considerare și dezvoltarea rapidă a conectivității dispozitivelor această lucrare propune și o modalitate de control securizat la distanță, deblocarea acestei funcționalități realizându-se prin intermediul unui algoritm de verificare facială pe bază de rețele neuronale.

Cum platforma propusă este una scalabilă, la nivel hardware, următorul pas natural este integrarea sistemului într-un mecanism mult mai robust, de dimensiuni și performanțe mai mari.

Ca dezvoltare software ulterioară, un prim pas ar fi oferirea unei și mai mari ponderi decizionale a informațiilor de la cameră. Primul pas în această procedură ar fi înlocuirea camerei actuale cu una cu o vedere largă, cu un unghi mai mare de deschidere, dată fiind poziționarea modulului în contextul mașinii autonome. În sistemul de detecție a benzilor de circulație o îmbunătățire considerabilă ar fi adăugarea unui filtru Kalman, modalitate propusă în [30][31], ce luând în considerare curbele interpolate Spline ale benzilor și deplasarea vehiculului în sistemul de coordonate ale drumului, să genereze estimări ale viitoarelor detecții. La nivel de autentificare și control de la distanță a vehiculului, o posibilitate de îmbunătățire stă în adăugarea unui algoritm de detecție a naturii imaginilor faciale, dacă sunt sau nu ale unei persoane sau sunt falsuri cu poze ale persoanei respective. Modalitatea aceasta este propusă în [32][33].

Bibliografie

- [1] C. Foster, Imagine Controler PID, Accesat în 2020 Mar. [Sursă online].
<https://www.controleng.com/articles/modern-updates-in-pid-control-tuning/>
- [2] Welch, Greg, and Gary Bishop. An introduction to the Kalman filter, UNC-Chapel Hill. TR 95-041, 2006.
- [3] Doucet A, Johansen AM. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*. 2009 Dec 12;12(656-704):3.
- [4] Marin-Plaza P, Hussein A, Martin D, Escalera AD. Global and local path planning study in a ROS-based research platform for autonomous vehicles. *Journal of Advanced Transportation*. 2018 Feb 22;2018.
- [5] J. Jimeno. Platforma opensource Linorobot, Accesat în 2020 Ian. [Sursă online]
<https://linorobot.org/>
- [6] A. Kumar, Imagine Nvidia Jetson Nano, Accesat în 2020 Mar. [Sursă online]
<https://maker.pro/nvidia-jetson/tutorial/the-jetson-nano-introduction-and-dev-board-comparisons>
- [7] DIYIOT, Imagine Pini Arduino Mega, Accesat în 2020 Mar. [Sursă online]
<https://diyi0t.com/arduino-mega-tutorial/>
- [8] Mayor LC. The adsorption and charge-transfer dynamics of model dye-sensitised solar cell surfaces (Doctoral dissertation, University of Nottingham).
- [9] V. Mazzari, Imagine Cum funcționează un senzor LiDAR?, Accesat în 2020 Apr. [Sursă online]
<https://blog.generationrobots.com/en/what-is-lidar-technology/>
- [10] Sparkfun, Imagine Scanse Sweep, Accesat în 2020 Apr. [Sursă online]
<https://www.sparkfun.com/products/retired/14117>
- [11] Robotshop, Imagine YDLIDAR X4, Accesat în 2020 Apr. [Sursă online]

<https://www.robotshop.com/uk/ydlidar-x4-360-laser-scanner.html>

[12] Mercadolibre, Imagine Modul Pi Camera, Accesat în 2020 Apr. [Sursă online]

https://articulo.mercadolibre.com.ar/MLA-732920552-raspberry-pi-3-camara-v2-8mpix-1080-emakers-_JM

[13] Mercadolibre, Imagine Modul driver L298N, 2020 Apr. [Sursă online]

https://produto.mercadolivre.com.br/MLB-1310421623-l298-marca-novo-original-l298n-motor-placa-de-unidade-modulo-_JM

[14] TME, Imagine Motor DC cu encoder, Accesat în 2020 Apr. [Sursă online]

<https://www.tme.eu/en/details/df-fit0484/dc-motors/dfrobot/fit0484/>

[15] Sparkfun, Imagine Modul IMU MPU 9250, Accesat în 2020 Apr. [Sursă online]

<https://learn.sparkfun.com/tutorials/mpu-9250-hookup-guide/all>

[16] K. Okanda, Pachet ROS gscam, Accesat în 2020 Apr. [Sursă online]

<https://github.com/ros-drivers/gscam>

[17] P. Moran, Pachet ROS jetson_csi_cam, Accesat în 2020 Apr. [Sursă online]

https://github.com/peter-moran/jetson_csi_cam

[18] Joseph L, Cacace J. Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System. Packt Publishing Ltd; 2018 Feb 26.

[19] Documentație oficială ROS pentru pachetul imu_filter_madgwick, Accesat în 2020 Iun. [Sursă online]

http://wiki.ros.org/imu_filter_madgwick

[20] Documentație oficială ROS pentru pachetul robot_localization, Accesat în 2020 Iun. [Sursă online]

http://docs.ros.org/melodic/api/robot_localization/html/index.html

[21] Fox D, Burgard W, Dellaert F, Thrun S. Monte carlo localization: Efficient position estimation for mobile robots. AAAI/IAAI. 1999 Jul 18;1999(343-349):2-.

[22] Fox D. KLD-sampling: Adaptive particle filters. InAdvances in neural information processing systems 2002 (pp. 713-720).

[23] Documentație oficială ROS pentru pachetul move_base, Accesat în 2020 Iun. [Sursă online]

http://wiki.ros.org/move_base

[24] Documentație oficială ROS pentru pachetul costmap_2d, Accesat în 2020 Iun. [Sursă online]

http://wiki.ros.org/costmap_2d?distro=noetic

[25] Documentație oficială ROS pentru pachetul global_planner, Accesat în 2020 Iun. [Sursă online]

http://wiki.ros.org/global_planner?distro=noetic

[26] Documentație oficială ROS pentru pachetul dwa_local_planner, Accesat în 2020 Iun. [Sursă online]

http://wiki.ros.org/dwa_local_planner?distro=noetic

[27] Zhang K, Zhang Z, Li Z, Qiao Y. Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Processing Letters. 2016 Aug 26;23(10):1499-503.

[28] T. Esler, Implementare PyTorch a MTCNN și FaceNet, Accesat în 2020 Mai. [Sursă online]

<https://github.com/timesler/facenet-pytorch>

[29] Schroff F, Kalenichenko D, Philbin J. Facenet: A unified embedding for face recognition and clustering. InProceedings of the IEEE conference on computer vision and pattern recognition 2015 (pp. 815-823).

[30] Borkar, Amol & Hayes, Monson & Smith, Mark. (2009). Robust lane detection and tracking with Ransac and Kalman filter. 3261-3264. 10.1109/ICIP.2009.5413980.

[31] Mammeri A, Boukerche A, Lu G. Lane detection and tracking system based on the MSER algorithm, hough transform and kalman filter. InProceedings of the 17th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems 2014 Sep 21 (pp. 259-266).

[32] Pan G, Wu Z, Sun L. Liveness detection for face recognition. Recent advances in face recognition. 2008 Dec 1:109-24.

[33] Kollreider K, Fronthaler H, Bigun J. Non-intrusive liveness detection by face images. Image and Vision Computing. 2009 Feb 2;27(3):233-44.

Anexa 1 - Listă acronime

LiDAR - Light Detection and Ranging

RGB - Red Green Blue

SLAM - Simultaneous Localization And Mapping

PID - Proportional Integral Derivative

ROS - Robot Operating System

DC - Direct Current

IMU - Inertial Measurement Unit

GPU - Graphics Processing Unit

CPU - Central Processing Unit

FPS - Frames Per Second

PWM - Pulse Width Modulation

IP - Internet Protocol

TCP/IP - Transmission Control Protocol/Internet Protocol

SSH - Secure Shell

VNC - Virtual Network Computing