Universitatea “Politehnica” din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

SISTEM DE EVITARE AUTOMATĂ A OBSTACOLELOR CU ROBOTUL JAGUAR 4X4 PE BAZA PROCESĂRII UNOR SECVENȚE VIDEO

Proiect de diplomă

prezentat ca cerință parțială pentru obținerea titlului de
Inginer în domeniul Electronică și Telecomunicații
Program de studii Microelectronică, Optoelectronică și Nanotehnologii

Conducători științifici:
Prof. Dr. Ing. Corneliu Burileanu
Ing. Ana-Antonia Neacșu
Ing. George Cioroiu

Absolvent:
Tudoroiu Mihai-Cristian

București
2019

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul **DCAE**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **TUDOROIU C.L. Mihai-Cristian , 442E**

1. Titlul temei: Sistem de evitare automată a obstacolelor cu robotul Jaguar 4x4 pe baza procesării unor secvențe video

2. Descrierea contribuţiei originale a studentului (în afara părţii de documentare) și specificații de proiectare:
Lucrarea de față își propune proiectarea și dezvoltarea unui sistem de evitare automată a obstacolelor pe baza unor imagini. Platforma robotică trebuie să ajungă la destinația indicată de către utilizator în deplină siguranță și într-un timp cât mai scurt.
Achiziția de imagini se va realiza cu ajutorul unui sistem Kinect, echipat cu o camera video RGB și senzori 3D ce va fi poziționată astfel încât să poată cuprindă punctele de plecare și sosire ale robotului. Algoritmul pentru recunoașterea de obstacole va fi implementat cu ajutorul librăriei OpenCV care facilitează prelucrarea de imagini sub formă matriceală. Totodată algoritmul va fi optimizat pentru diferite situații de luminozitate și forme ale obstacolelor.
Evitarea obiectelor se va realiza prin intermediul unor algoritmi decizionali in funcție de obstacolele identificate în imagini astfel încât robotul să aibă o mișcare uniformă.
Conectarea cu Jaguar 4x4 se realizează wireless, prin intermediul unui protocol de control al transmisiunii. După estimarea celui mai scurt traseu de către computerul gazdă, acesta va trimite comenzi către controller-ul robotului unde vor fi interpretate și va rezulta acționarea corespunzătoare a motoarelor. Testarea sistemului se va realiza într-un mediu controlat modificând atât poziția cât și forma obstacolelor.

3. Resurse folosite la dezvoltarea proiectului:
• Limbaje de programare: Phyton – OpenCV • Medii de dezvoltare: PyCharm • Cunoștințe de bază: Prelucrarea imaginilor, Algoritmi decizionali

4. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline:
Microcontrolere, Programare Obiect-Orientata, Proiect 2, Decizie si Estimare in Prelucrarea Informat

5. Proprietatea intelectuală asupra proiectului aparţine: U.P.B.

6. Data înregistrării temei: 2018-11-28 22:05:53

Conducător(i) lucrare,
Prof. dr. ing. Corneliu BURILEANU
semnătura: .....

Student,
semnătura: .....

Ing. Ana-Antonia NEACȘU, Ing. George CIOROIU
semnătura: .....

Director departament,
Prof. dr. ing Claudius DAN
semnătura: .....

Decan,
Prof. dr. ing. Cristian NEGRESCU
semnătura: .....

Cod Validare: **8b3ed7ba08**

DECLARAȚIE DE ONESTITATE ACADEMICĂ

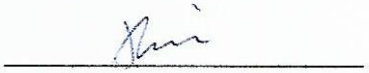
Prin prezenta declar că lucrarea cu titlul "*Sistem de evitare automată a obstacolelor cu robotul Jaguar 4X4 pe baza procesării unor secvențe video*", prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Electronică, Telecomunicații și Tehnologia Informației*, programul de studii *Microelectronică, Optoelectronică și Nanotehnologii* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, *2019*

Mihai - Cristian Tudoroiu



(semnătura în original)

CUPRINS

CAPITOLUL 1 INTRODUCERE 15
1.1 MOTIVAȚIA LUCRĂRII 15
1.2 OBIECTIVUL PRINCIPAL 16
1.3 OBIECTIVE SPECIFICE 16
CAPITOLUL 2 TEHNOLOGII ALE PROIECTULUI 17
2.1 PLATFORMA MOBILĂ JAGUAR 4X4 17
2.1.1 Componente hardware 19
2.1.1.2 Controlerul PMS5006 21
2.1.1.3 Decodarea mesajelor de la senzori 21
2.1.1.3.1 Mesaje de la modulul GPS 21
2.1.1.3.2 Mesaje de la modulul IMU 22
2.1.1.3.3 Mesaje de la senzorii și driverele motoarelor 23
2.1.1.4 Comenzi de control 23
2.1.1.5 Controlul motoarelor 25
2.1.1.6 Senzori externi 25
2.1.1.6.1 Camerele video integrate 26
2.1.1.6.2 Scanerul Laser 26
2.1.2 Componente software 27
2.1.2.1 Clase și *namespace-uri* 28
2.1.2.2 Constructori, metode și atribute 28
2.1.2.3 Clasificarea tipurilor C# 28
2.1.2.4 Cuvinte cheie 29
2.1.2.5 Biblioteca OpenCV 29
2.1.3 Aplicații client-server 30
2.1.3.1 Porturi și clasa Socket 30
2.1.4 Aplicația demonstrativă pentru robotul Jaguar 4X4 31
2.1.4.1 Mod de utilizare al aplicației demonstrative 32
2.2 CAMERA KINECT 35
2.2.1 Camera video 35
2.2.2 Biblioteca PyKinect2 35

CAPITOLUL 3 PRELUCRAEA ȘI ANALIZA IMAGINILOR 37
3.1 INTRODUCERE 37
3.2 METODE DE DETECȚIE 38
3.2.1 Binarizare 38
3.2.2 Detecția de contur 41
3.3 ALGORITM PENTRU GĂSIREA CELUI MAI SCURT DRUM ÎNTR-O IMAGINE 44
3.3.1 Implementarea unui algoritm de tip Dijkstra 45
3.3.3 Metode de optimizare al algoritmului 49
CAPITOLUL 4
CONTROLUL PLATFORMEI MOBILE ROBOTICE JAGUAR 51
4.1 DESCRIERE APLICAȚIE 51
4.2 ELEMENTE COMPONENTE 52
4.3 ARHITECTURA CLIENT-SERVER 55
4.4 TRATAREA ERORILOR DE POZIȚIONARE 57
CAPITOLUL 5
SISTEM DE EVITARE AUTOMATĂ A OBSTACOLELOR 59
5.1 INTRODUCERE 59
5.2 ALGORITM DE FUNCȚIONARE AL SISTEMULUI 60
5.3 DATE EXPERIMENTALE 61
CONCLUZII ȘI POSIBILITĂȚI DE DEZVOLTARE 65
CONCLUZII GENERALE 65
CONTRIBUȚII PERSONALE 66
POSIBILITĂȚI DE DEZVOLTARE 66

LISTĂ DE FIGURI

Figura 1.1 Pași parcurși în implementare....16
Figura 2.1 Platforma robotică Jaguar 4X4 a)....18
Figura 2.2 Platforma robotică Jaguar 4X4 b)....18
Figura 2.3 Diagramă interconexiuni componente de bază....19
Figura 2.4 Semnal de la ieșirea codificatorului....20
Figura 2.5 Raza de acoperire a senzorului laser....26
Figura 2.6 Schema funcțională a platformei mobile Jaguar 4X4....27
Figura 2.7 Conexiunea client-server utilizată....31
Figura 2.8 Fereastra de autentificare în aplicația demonstrativă....32
Figura 2.9 Fereastra de dialog pentru încărcarea hărților....33
Figura 2.10 Aplicația demonstrativă Jaguar....33
Figura 2.11 Datele de la motoare afișate în timp real....34
Figura 2.12 Instrucțiuni utilizare Gamepad a)....34
Figura 2.13 Instrucțiuni utilizare Gamepad b)....35
Figura 3.1 Imagine preluată de la camera Kinect....39
Figura 3.2 Imaginea după procesul de binarizare....40
Figura 3.3 Pragul nivelului de culoare....40
Figura 3.4 Detecția de contur....41
Figura 3.5 Transformări morfologice....42
Figura 3.6 Nucleu folosit în procesul de eroziune....42
Figura 3.7 Imagine după procesul de eroziune....43
Figura 3.8 Drumul echivalent între punctele A și B....44
Figura 3.9 Model de graf eligibil pentru algoritmul Dijkstra....44
Figura 3.10 Simularea algoritmului de căutare....45
Figura 3.11 Caz imposibil de găsire al traseului....47
Figura 3.12 Timpul de procesare în funcție de complexitatea algoritmului....49
Figura 4.1 Axe de rotație ale platformei mobile....52
Figura 4.2 Legăturile între componentele sistemului....56
Figura 4.3 Mediul de operare al robotului Jaguar....57
Figura 4.4 Eroarea pentru o mișcare de deplasare....58
Figura 4.5 Eroarea pentru o mișcare de rotație....58
Figura 5.1 Schema de funcționare a sistemului automat de evitare a obstacolelor....60
Figura 5.2 Simulare scenariu 1....61
Figura 5.3 Traseu calculat de către sistem....62
Figura 5.4 Punctul final atins de robotul Jaguar....62

Figura 5.5 Simulare scenariu 2 63
Figura 5.6 Traseu calculat 63
Figura 5.3 Traseul calculat de către sistem 67

LISTĂ DE TABELE

Tabel 2.1 Componente platformă robotică Jaguar 4X4 [1] 19
Tabel 2.2 Modul de conectare al motoarelor la drivere[2] 21
Tabel 2.3 Date de la senzorul GPS [2] 22
Tabel 2.4 Date de la senzorul IMU [2] 22
Tabel 2.5 Comenzi pentru motoarele robotului Jaguar 24

LISTĂ DE ACRONIME

A
ADC – Analog to Digital Converter

C
CLRF – Carriage Return Line Feed
CPR – Count Per Revolution

D
DCM – Direction Cosine Matrix
DOF – Direction of Field

F
FTP – File Transfer Protocol

G
GPS – Global Positioning System
GPU – Graphics processing unit

H
HTTP – Hypertext Transfer Protocol

I
I/O – Input/Output
I2C – Inter-Integrated Circuit
IMU – Inertial Measurement Unit
IP – Internet Protocol

L
LAN – Local Area Network
LiPo – Lithium Polymer

R
RGB – Red Green Blue

S
SDK – Software Development Kit
SRM – Scan Response Message

T
TCP – Transmission Control Protocol

U
UART – Universal Asynchronous Receiver Transmitter

X
XML – Extensible Markup Language

CAPITOLUL 1
INTRODUCERE

1.1 MOTIVAȚIA LUCRĂRII

Transportul persoanelor în zonele formate de punctele de frontieră dintre statele cu conflicte politice puternice nu se desfășoară în deplină siguranță. În prezent sunt mai multe astfel de puncte de frontieră și chiar zone mai extinse în țari care au fost afectate recent sau pe timp îndelungat de războaie în care au rămas plantate mine explozive. Semnele existenței unei astfel de zone sunt formele neregulate ale obiectelor sau culori ale acestora care se deosebesc de mediul natural în care se găsesc.

Componenta principală a acestui proiect este robotul Jaguar 4X4 creat de Dr Robot Inc. capabil să funcționeze atât în operațiuni interioare cât și în spațiu liber, operațiuni care necesită gardă la sol ridicată și manevrabilitate rapidă.

Acest sistem își propune să ajute cadrele militare să parcurgă zonele cu risc de explozie. Robotul Jaguar 4X4 poate parcurge orice teren accidentat și va avea rolul de explorator, fiind ajutat în acest proces de un dispozitiv zburător.

1.2 OBIECTIVUL PRINCIPAL

În acest context, lucrarea își propune crearea unui sistem de evitare automată a obstacolelor, capabil să găsească și să parcurgă un drum caracterizat de două puncte fixe fără intervenție umană.
Pentru a demonstra acest lucru, acest proiect implică un sistem care găsește o legătură între două puncte din spațiu astfel încât distanța parcursă să fie minimă. Sistemul este compus din două părți importante: prima parte este de a capta imagini în timp real de la senzorul Kinect și de a prelucra aceste date pentru al doilea modul. A doua parte interpretează datele primite și construiește comenzile pentru robotul Jaguar.
Pașii necesari pentru realizarea acestor obiective sunt enumerați mai jos:

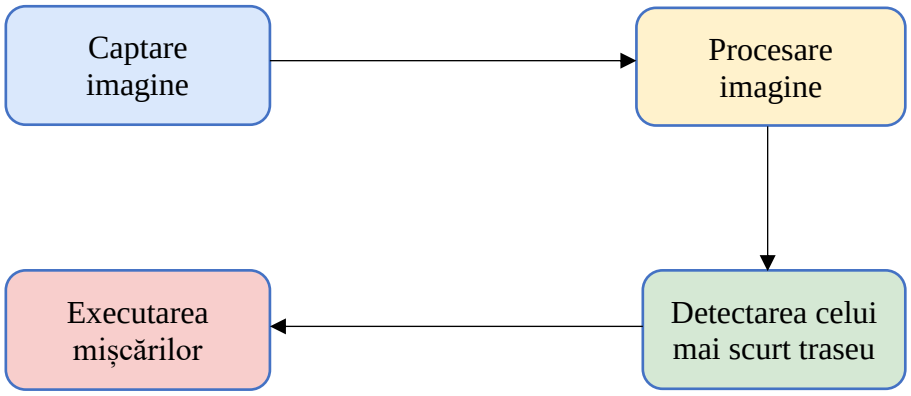


Figura 1.1 Pași parcurși în implementare

1.3 OBIECTIVE SPECIFICE

- Captarea imaginilor de la camera Kinect.
- Implementarea unei soluții pentru a detecta obstacolele din imagine.
- Dezvoltarea unui algoritm pentru detectarea unei rute cât mai scurte între punctul în care se află robotul și punctul destinație.
- Dezvoltarea unui program pentru controlul robotului Jaguar
- Implementarea comunicațiilor dintre robot, computerul de control al acestuia și computerul care efectuează procesarea imaginilor de la camera Kinect și calculează ruta optimă.
- Programarea robotului pentru a executa mișcări precise de rotație și deplasare.

Această lucrare conține cinci capitole, după cum urmează:
Capitolul 1 prezintă motivația, obiectivele și schița acestei lucrări. În *Capitolul 2* sunt detaliate tehnologiile hardware și software utilizate pentru a dezvolta acest proiect. *Capitolul 3* este primul capitol care ilustrează contribuțiile autorului lucrării. Acesta descrie metodele de procesare a imaginilor pe care le-am abordat și metoda de calcul a distanței dintre două puncte evitând locurile inaccesibile. *Capitolul 4* se referă la dezvoltarea aplicației pentru controlul robotului Jaguar 4X4. În *Capitolul 5* este prezentat modul de funcționare al aplicației finale. În cele din urmă, *Capitolul 6* rezumă principalele concluzii ale lucrării și subliniază contribuțiile autorului.

CAPITOLUL 2
TEHNOLOGII ALE PROIECTULUI

2.1 PLATFORMA MOBILĂ JAGUAR 4X4

Platforma Jaguar 4X4 este proiectată pentru a fi folosită pentru operațiuni care se desfășoară atât în interior, în spații închise, cât și în exterior, pentru operațiuni ce necesită manevrabilitate rapidă și gardă la sol ridicată. Această platformă dispune de un braț cu patru articulații și trei grade de libertate. Printre avantajele brațului se numără greutatea redusă, consumul mic de putere și dimensiunile compacte. În capătul acestuia este montat un clește care poate prinde obiecte cu dimensiuni de până la 710 mm și greutate maximă de 4 kg. Pe brațul robotului este atașată o cameră video de rezoluție 640X480. Platforma robotică este propulsată de 4 motoare de putere 105W, are o greutate totală de 20 kg și atinge o viteză maximă de 11 km/h. Acest robot este proiectat pentru a face față modificărilor atmosferice fiind rezistent la apă. Este ideal pentru operațiuni pe terenuri accidentate fiind capabil să urce trepte de până la 110 mm înălțime. Comandarea robotului se face printr-o conexiune fără fir, pentru orientarea în spațiu se folosește de un GPS pentru exterior, de un senzor IMU cu 9 grade de libertate, scaner laser cu raza de acoperire de 30 m cu un unghi de vizibilitate de 240 grade. Este integrată și o a doua cameră în partea din față a structurii robotului ceea ce facilitează accesarea informației referitoare la mediul înconjurător.[1]

Caracteristici cheie:

- Platformă mobilă utilă atât pentru aplicații de exterior cât și pentru interior
- Manevrabilitate ridicată
- Rezistent la apă si la modificările atmosferice
- Poate trece peste scări cu o înălțime maximă de 110 mm
- Ușor si compact
- Senzori pentru orientare GPS și IMU
- Rezistent la o cădere de la 1.2 m pe beton
- Scaner Laser integrat
- Camere video cu captare audio
- Conexiune fără fir
- Aplicație demonstrativă

Pentru o vizualizare cât mai bună a acestei platforme am identificat principalele componente:

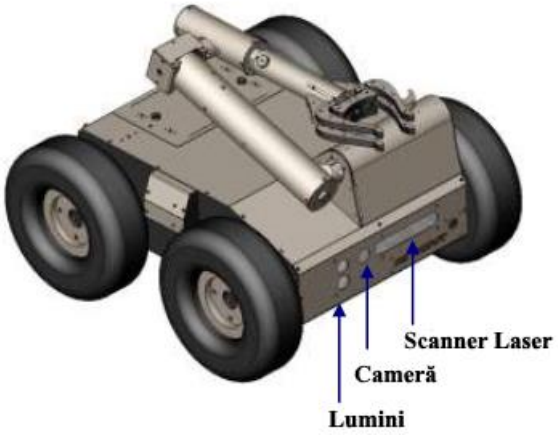


Figura 2.1 Platforma robotică Jaguar 4X4 a) [1]

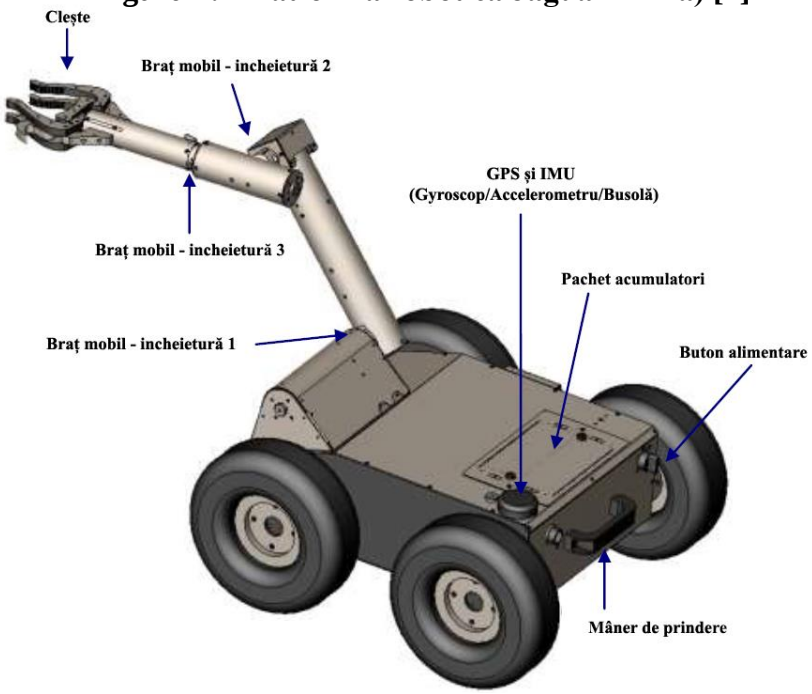


Figura 2.2 Platforma robotică Jaguar 4X4 b) [1]

2.1.1 Componente hardware

Platforma robotică Jaguar oferă o gamă largă de componente hardware care ajută la dezvoltarea multor tipuri de aplicații. Structura hardware a robotului conectează senzori (Scaner Laser, GPS, IMU), actuatoare și module de rețea pentru a facilita capabilitățile de comunicare.

În următorul tabel sunt menționate principalele componente de bază ale robotului.

Tabel 2.1 Componente platformă robotică Jaguar 4X4 [1]

JGUAR 4x4W-ME	Carcasă Jaguar 4X4 (include motoare și codificatoare)	1
JAGUAR ARM	Braț mobil	1
PMS5006	Controler de mișcare și senzori	1
WFS802	Modul de rețea	2
SDC2130	Driver de motoare DC cu canal dual 10A	2
OGPS501	GPS de exterior cu rată de update de 5Hz	1
IMU9002	Senzor IMU (Accelerometru/Giroscop/Compas)	1
WRT802G	Router wireless 802.11N	1
BPN-LP-10	Acumulator LiPo de 22.2V	1
LPBC5000	Încărcător LiPo de 2A	1
GPC0010	Controler (Nvidia Shield)	1
PMCHR12	Placă de alimentare DC-DC	1

Arhitectura hardware a sistemului și inter-conexiunea între componentele de bază ale platformei robotice Jaguar 4X4 este ilustrată în Figura 2.3.

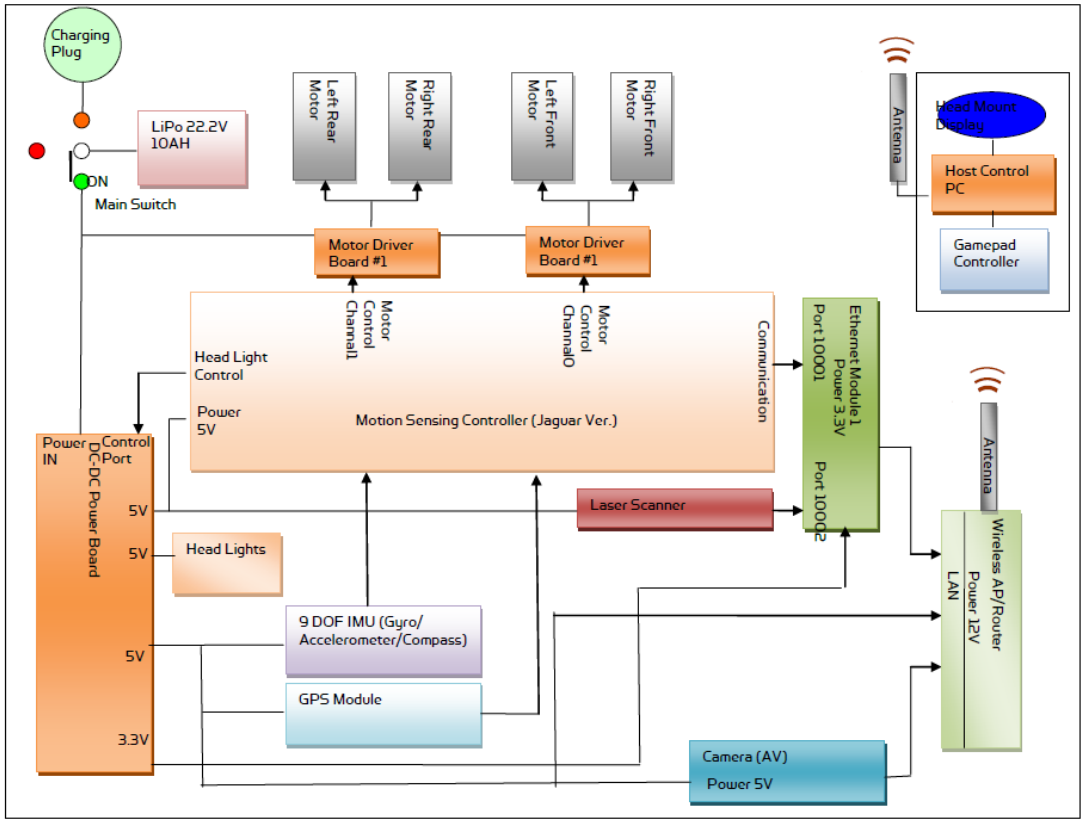


Figura 2.3 Diagramă interconexiuni componente de bază [1]

La o primă analiză a diagramei se observă modulele de funcționare și componentele principale ale robotului: controlerul de mișcare reprezintă principala componentă a acestei platforme, este poziționat în mijloul diagramei, acesta primește informații de la senzorul IMU și senzorul GPS pe care le prelucrează și le transmite mai departe prin intermediul Modulului Ethernet.
Modulul Ethernet dispune de 2 porturi, cel de-al doilea este ocupat de Scanerul Laser, informațiile de la controler și scaner sunt transmise către router prin interfață LAN. Acest router primește informație și de la cele două camere video disponibile pe platforma robotică, toate aceste date sunt transpuse direct cu utilizatorul prin rețeaua proprie.
Cea care alimentează toate componentele este placa de alimentare DC-DC care primește la intrare 22.2 V de la baterie atunci când comutatorul se află în poziția ON și oferă la ieșire o tensiune de 5V de la care se alimentează controlerul, scanerul laser, farurile, camerele video, router-ul și senzorii IMU și GPS. De asemenea placa de alimentare oferă la ieșire și o tensiune de 3.3V pentru alimentarea modulului Ethernet.
Cele doua drivere de motoare au caracter dual ceea ce rezultă o manipulare independentă a fiecărui motor. Primul driver controlează motoarele stânga-spate și dreapta-spate, iar cel de-al doilea controlează motoarele stânga-față și dreapta-față.
O altă componentă hardware importantă este codificatorul optic cu incrementare care se află integrat în caracterul constructiv al motorului. Acestea sunt module care ajută la măsurarea distanței parcurse pentru un motor și a vitezei de rotație a acestuia. Codificatoarele absolute returnează un număr pe mai mulți biți (în funcție de rezoluție), pe când cele cu incrementare trimit pulsuri când se rotesc. Numărul de revoluții care au avut loc la mișcarea motoarelor se pot transmite numărând aceste pulsuri. După numărul de pulsuri dintr-un interval de timp scurs poare fi determinată viteza de rotație. Deoarece codificatoarele cu incrementare sunt dispozitive digitale, ele vor măsura viteza și distanța cu acuratețe mare. Întrucât motoarele se pot mișca atât în spate cât și în față, este necesar să diferențiem modul de numărare al pulsurilor astfel că acestea vor decrementa sau incrementa un număr.
Codificatoarele în cuadratură au două canale, A și B, ca în figura de mai jos. Aceste canale sunt defazate electric cu 90^o. În acest mod direcția de rotație se poate afla prin monitorizarea relației de fază dintre canalele A și B. [3]

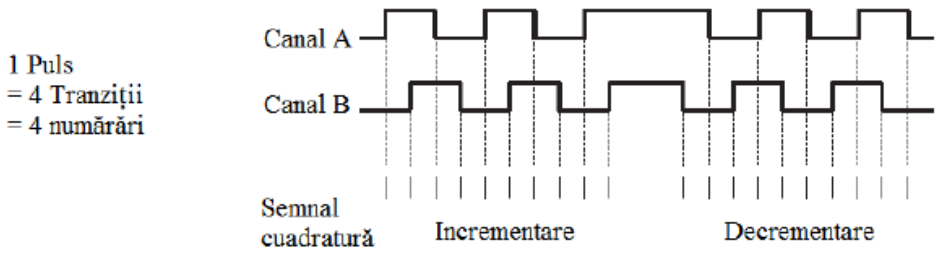


Figura 2.4 Semnal de la ieșirea codificatorului [3]

2.1.1.2 Controlerul PMS5006

Controlerul PMS5006 este componenta principală a platformei robotice Jaguar 4X4. Acesta comunică prin intermediul unui port serial UART1 cu computerul gazdă. Portul serial se va conecta la portul 1 al modulului de rețea ethernet. De la computerul gazdă se poate stabili un socket TCP pentru a comunica cu acest controler folosind IP-ul: 192.168.0.XX și portul 10001. (XX – variază în funcție de platforma robotică, în acest caz s-a folosit ip-ul 192.168.0.60).

PMS5006 poate controla până la 4 drivere de motoare SDC2130 de la RoboteQ prin porturile seriale UART 2, 3, 4, 5. Controlerul poate citi curentul motoarelor, puterea consumată, tensiunea, temperatura motoarelor și a driverelor de motoare, datele de la codificatoare și starea în care se află driverul de motoare. Acesta poate suporta controlul în buclă deschisă, controlul vitezei encoderelor în buclă închisă și controlul poziției encoderelor în buclă închisă.[2]

Modul predefinit pentru driverul de motoare este controlul în buclă deschisă, driverul 1 va comanda motoarele din față iar driverul 2 pe cele din spate.

Controlerul PMS5006 citește toate datele de la senzorul 9DOF IMU prin portul I2C. Orientarea robotului este estimată cu ajutorul algoritmului DCM utilizând date de la IMU și GPS. În același timp va citi mesajele de la modulul GPS prin portul UART.

Tabel 2.2 Modul de conectare al motoarelor la drivere[2]

Driver 1	Canal 0	Motor stânga față	Encoder 1 și intrarea analogică 3 ca senzor de temperatură
	Canal 1	Motor dreapta față	Encoder 2 și intrarea analogică 4 ca senzor de temperatură
Driver 2	Canal 0	Motor stânga spate	Encoder 1 și intrarea analogică 3 ca senzor de temperatură
	Canal 1	Motor dreapta spate	Encoder 2 și intrarea analogică 4 ca senzor de temperatură

2.1.1.3 Decodarea mesajelor de la senzori

Controlerul PMS5006 va trimite toate datele de la senzori către utilizator. Există 4 tipuri principale de mesaje cu date de la senzori și toate aceste mesaje se încheie cu terminația CRLF.

2.1.1.3.1 Mesaje de la modulul GPS

Controlerul PMS5006 va transmite mesajele senzorului GPS către computerul de control. Pentru a reduce volumul de date setăm GPS-ul să transmită doar poziții GPRMC. Orice pachet de date care va începe cu “$GPRMC”, va fi asociat unui mesaj transmis de senzorul GPS.[2]

Formatul datelor GPRMC este prezentat mai jos. Exemplu de mesaj transmis de senzorul GPS:

$GPRMC,120619,A,44.42676,N,26.102537,E,120.5,221.4,130619,004.2,W*70

1 2 3 4 5 6 7 8 9 10 11 12

Tabel 2.3 Date de la senzorul GPS [2]

1	120619	Amprenta temporală
2	A	Validitatea: A-Valid, V-Invalid
3	44.4267	Latitudine curentă
4	N	Nord/Sud
5	26.10253	Longitudine curentă
6	W	Est/Vest
7	120.5	Viteză în Noduri
8	221.4	Cursul real
9	130619	Data
10	004.2	Variația
11	W	Est/Vest
12	*70	Verificare sumă

Rata de reîmprospătare a acestui pachet de date este de 5 Hz.

2.1.1.3.2 Mesaje de la modulul IMU

Controlerul va transmite pachetul de date de a modulul IMU și valoarea estimată a orientării către computerul gazdă. Orice pachet de date care începe cu simbolul “#” va fi asociat unui mesaj de date provenit de la modulul IMU. Formatul acestui pachet de date este prezentat mai jos.

Următoarea secvență reprezintă un exemplu de mesaj transmis de senzorul IMU:

#Num,Yaw,xxx,Gyro,gyroX,gyroY,gyroZ,Accel,accelX,accelY,accelZ,Comp,compX,compY, compZ,ADC,xxx,xxx,xxx,xxx

Tabel 2.4 Date de la senzorul IMU [2]

1	Numărul de secvență al pachetului de date 0~255
2	Yaw - fixat
3	Valoarea orientării estimată în radiani de către algoritmul DCM
4	Gyro - fixat
5	Valoarea citită de la giroscop pe axa X
6	Valoarea citită de la giroscop pe axa Y
7	Valoarea citită de la giroscop pe axa Z
8	Accel - fixat
9	Valoarea citită de la accelerometru pe axa X
10	Valoarea citită de la accelerometru pe axa Y
11	Valoarea citită de la accelerometru pe axa Z
12	Comp - fixat
13	Valoarea citită de la compasul digital pe axa X
14	Valoarea citită de la compasul digital pe axa Y
15	Valoarea citită de la compasul digital pe axa Z
16	ADC - fixat

Valorile pentru convertorul analog digital nu sunt folosite de către această platformă.
Rata de reîmprospătare a acestui pachet de date este de 50Hz.

2.1.1.3.3 Mesaje de la senzorii și driverele motoarelor

Controlerul PMS5006 colectează datele de la driverul de motoare și le transmite computerului gazdă. Din moment ce PMS5006 poate controla până la 4 drivere de motoare SDC2130, aceste mesaje vor începe cu:

- MM0 – datele sunt primite de la driverul de motoare 1
- MM1 – datele sunt primite de la driverul de motoare 2
- MM2 – datele sunt primite de la driverul de motoare 3
- MM3 – datele sunt primite de la driverul de motoare 4

Controlerul va trimite comenzi către driverul de motoare pentru a accesa datele de la senzorii acestora.
Un exemplu de secvență provenită de la senzorii motoarelor: "MM0 V=125,246,4750".
Mesajul se poate decoda astfel: tensiunea pentru driverul 1 este de 12.5V, tensiunea totală este de 24.6 V și ieșirea regulatorului de 5V de pe placă este de 4.750V.
Controlerul PMS5006 va trimite 10 comenzi în buclă pentru fiecare driver de motoare la o frecvență de 50Hz, deci rata de reîmprospătare pentru fiecare comandă de la senzor este de aproximativ 5Hz.

2.1.1.4 Comenzi de control

Toate comenzile de sistem sau de control al motoarelor sunt delimitate prin terminația CRLF.
În următoarele rânduri vor fi detaliate comenzile de control pentru platforma mobilă Jaguar 4X4:

- Comanda PING
  Format: PING
  Descriere: Computerul gazdă trebuie sa trimită acest mesaj către PMS5006 la o rată de aproximativ 4Hz, altfel PMS5006 va opri toate motoarele – consideră ca a pierdut comunicarea cu computerul gazdă.
- Comanda de control GPIO
  Format: SYS GPO n
  Descriere: Această comandă va controla un port IO, unde "n" este un număr pe 8 biți. Este rezervat pe robotul Jaguar.
- Comanda de control de putere
  Format: SYS MMC n
  Descriere: Această comandă va controla un canal de putere pe controlerul PMS5006, unde "n" este un număr pe 8 biți. Doar bitul 7 este disponibil pentru robotul Jaguar, bit7 = 1 va aprinde farurile, bit7 = 0 va stinge farurile. Restul biților sunt rezervați.
- Comanda de control pentru senzorul IMU
  Format: SYS CAL
  Descriere: Această comandă va face ca PMS5006 să recalculeze ofsetul senzorului giroscop. Dacă se găsește o valoare a senzorului giroscop mai mare ca 10 unități atunci când robotul stă nemișcat, trebuie să trimitem această comandă către PMS5006.

- Comanda de control GPS
  Format: EGPS n
  Descriere: Această comandă îi va spune controlerului PMS5006 dacă să folosească GPS sau nu. Pentru n = 1 controlerul va folosi datele de la GPS în algoritmul DCM iar dacă n = 0 va ignora aceste date.
- Setare valoare inițială DCM
  Format: DCM n
  Descriere: Această comandă va seta valoarea inițială pentru DCM. Parametrul "n" este valoarea direcției setată între -180 ~ +180
- Comenzi de control pentru motoare
  Pentru ca PMS5006 poate controla până la 4 drivere de motoare SDC2130, există 5 tipuri de comenzi.[2]

Tabel 2.5 Comenzi pentru motoarele robotului Jaguar

MM0 (comandă motor)	PMS5006 va trimite o comandă către driverul de motoare 1
MM1 (comandă motor)	PMS5006 va trimite o comandă către driverul de motoare 2
MM2 (comandă motor)	PMS5006 va trimite o comandă către driverul de motoare 3 (nu este folosit la robotul actual)
MM3 (comandă motor)	PMS5006 va trimite o comandă către driverul de motoare 4 (nu este folosit la robotul actual)
MMW (comandă motor)	PMS5006 va trimite o comandă către driverele de motoare 1 și 2 în același timp

Comanda MMW va fi folosită pentru robotul Jaguar 4X4 dacă este nevoie de control pentru toate motoarele simultan. Se pot trimite comenzile de control și configurare listate în manualul SDC2130. Pentru o mai bună înțelegere vom considera un exemplu concludent:
"MMW !M 200 -200" va controla toate cele 4 motoare în același timp și va determina robotul să se miște în față.
"MM0 !G 0 200" va controla doar motorul stânga față.
"MM0 !G 1 -200" va controla doar motorul dreapta față.
"MM1 !G 0 200" va controla doar motorul stânga spate.
"MM1 !G 1 -200" va controla doar motorul dreapta spate.
"MMW !EX" va opri de urgență toate motoarele platformei Jaguar 4X4.
"MMW !MG" va rezuma accesul la controlul tuturor motoarelor.
Inițial motoarele vor funcționa în modul de control în buclă deschisă. Dacă este nevoie de control de viteză sau poziție trebuie schimbat modul de control al motoarelor în funcție de necesitatea aplicației implementate. [2]

Intervalul de unități de putere pe care îl putem folosi în comandarea motoarelor este (-1000,1000), capetele reprezentând puterile maxime pentru fiecare sens de rotație.

2.1.1.5 Controlul motoarelor

Pentru fiecare motor, controlerul suportă mai multe moduri de operare. În configurația inițială a controlerului este setat modul de control în buclă deschisă pentru fiecare motor (mai puțin motoarele brațului care sunt setate în modul de control al poziției).

- Modul de control în buclă deschisă

  În acest mod, controlerul acționează motoarele folosind o putere proporțională cu informația din comandă, viteza motorului nefiind măsurată. Astfel motorul va încetini dacă există o schimbare a sarcinii cum ar fi întâlnirea unui obstacol sau modificarea unghiului unei rampe. Acest mod este adecvat pentru majoritatea aplicațiilor unde operatorul menține contact vizual cu robotul.

- Modul de control în buclă închisă

  În acest mod un codificator optic este folosit pentru a măsura viteza actuală a motorului. Dacă viteza se modifică datorită unei schimbări în sarcină, controlerul va compensa automat puterea de ieșire. Acest mod este de preferat în modul de control de precizie sau în aplicații pentru roboți autonomi.

- Modul de control în buclă închisă cu poziție relativă

  În acest mod, ilustrat în figura de mai jos, axul unui motor este cuplat la un senzor de poziție care este folosit pentru a compara poziția unghiulară a axului cu poziția dorită. Motorul se va mișca urmând o accelerație controlată până la o viteză definită de utilizator și încetinește pentru a ajunge ușor la destinația dorită.

- Modul de control în buclă închisă cu numărătoarea poziției

  În acest mod, un codificator este atașat la motor după care controlerul poate fi comandat să miște motorul la o valoare specifică de numărători, folosind accelerația, viteza și decelerația definite de utilizator.

- Modul de tracțiune

  Acest mod de tip buclă închisă, face ca motorul să fie mișcat în sensul producerii unei anumite valori a tracțiunii, neconsiderând viteza. Acest lucru se realizează folosind curentul din motor ca feedback. [3]

2.1.1.6 Senzori externi

Senzorii externi sunt senzori care ajută la obținerea informațiilor din mediul care înconjoară robotul (de exemplu distanțele pana la diferite obiecte).

Platforma robotică Jaguar are 2 tipuri de astfel de senzori:

- Scaner Laser (Hokuyo Laser Range Finder UTM-30LX) [4]
- Două camere video (Axis IP Camera)

2.1.1.6.1 Camerele video integrate

Platforma robotică Jaguar dispune de 2 camere video color amplasate în puncte special alese pentru a facilita utilizarea cât mai sigură a robotului. Una dintre camere este amplasată la baza robotului, iar cealaltă este amplasată în interiorul cleștelui.

Camera de la baza robotului are o rezoluție de 640X480 pixeli, captează 30 cadre pe secundă și este capabilă de înregistrare audio. Camera din interiorul cleștelui are rezoluția de 640X480 pixeli care captează 30 cadre pe secundă.

Camerele sunt de tip Axis IP, ceea ce înseamnă că transmisia video poate fi accesată via IP.

Din această cauză accesul la camere este protejat cu nume și parole.

IP-urile pentru a accesa aceste camera sunt:

- Pentru camera de la bază: 192.168.0.75 cu portul 8081, ID: *root,* parola: *drrobot*
- Pentru camera de pe braț: 192.168.0.74 cu portul 8082, ID: *root,* parola *drrobot*

2.1.1.6.2 Scanerul Laser

Robotul Jaguar 4x4 este dotat cu un senzor laser de distanță Hokuyo Laser Scanner UTM-30LX. Acest senzor are o rază de acoperire de 30 m cu un unghi de 270 de grade (după cum este reprezentat în figura următoare).

Protocolul de comunicare cu acest dispozitiv este prezentat în manualul său de utilizare, vom prezenta aici doar cele mai importante comenzi.

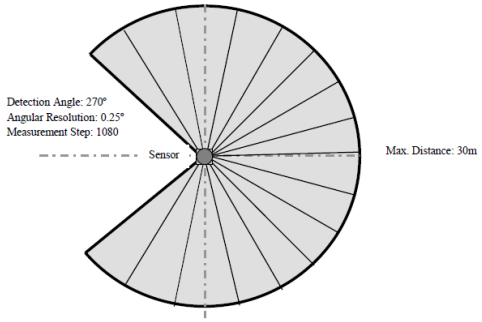


Figura 2.5 Raza de acoperire a senzorului laser [4]

În comunicarea uzuală, se trimite un mesaj de cerere de la computerul gazdă către senzor și apoi este returnat mesajul răspuns către computer. Mesajul trimis de senzor pentru fiecare măsurătoare se numește SRM. Un SRM este trimis de la senzor la computerul gazdă până la terminarea unui mesaj de cerere sau până când este trimis mesajul de cerere de oprire.

Un mesaj de cerere de la computerul gazdă include un cod de comandă. Răspunsul și SRM sunt definite pentru fiecare cod de comandă.

Există două tipuri de comenzi:

- Comenzi de măsurare
- Comenzi fără măsurare

Exemplu comenzi de măsurare:

- GD sau GS (codul de comandă) – realizează funcția de achiziție a distanței, va returna starea, timpul și distanța.

- GE – realizează funcția de achiziție a distanței și a intensității, va returna starea, timpul, distanța și intensitatea.

Exemplu de comenzi fără măsurare:

- BM – comandă de tipul tranziție de stare, realizează tranziția în modul de măsurare (nu se pot realiza măsurători fără tranziția în această stare). Va trimite către computerul gazdă un mesaj cu starea dispozitivului.
- RS – comandă de resetare, realizează resetarea senzorului. Va returna un mesaj cu starea
- RB – comandă de resetare, realizează operația de repornire. Va returna un mesaj cu starea dispozitivului.

Codurile de comandă prezentate sunt printre cele mai utilizate, cu ajutorul acestora putem realiza o rutină de execuție pentru obținerea măsurătorilor de distanță.

2.1.2 Componente software

Aplicația de control a robotului rulează pe un computer gazdă, trimite comenzi prin rețeaua fără fir către controlerul PMS5006 al robotului, acesta le interpretează și acționează motoarele corespunzător. Aplicația inițială demonstrativă de control creată de Dr. Robot Inc. este o aplicație de tip Windows, scrisă folosind Microsoft Visual Studio 2010 și limbajul de programare C#. Astfel am ales ca dezvoltarea aplicației finale pentru acest proiect să aibă la bază același limbaj și mediu de programare ca și aplicația demonstrativă combinat cu limbajul de programare Python care facilitează procesarea imaginilor.

Schema funcțională a sistemului este prezentată în figura de mai jos.

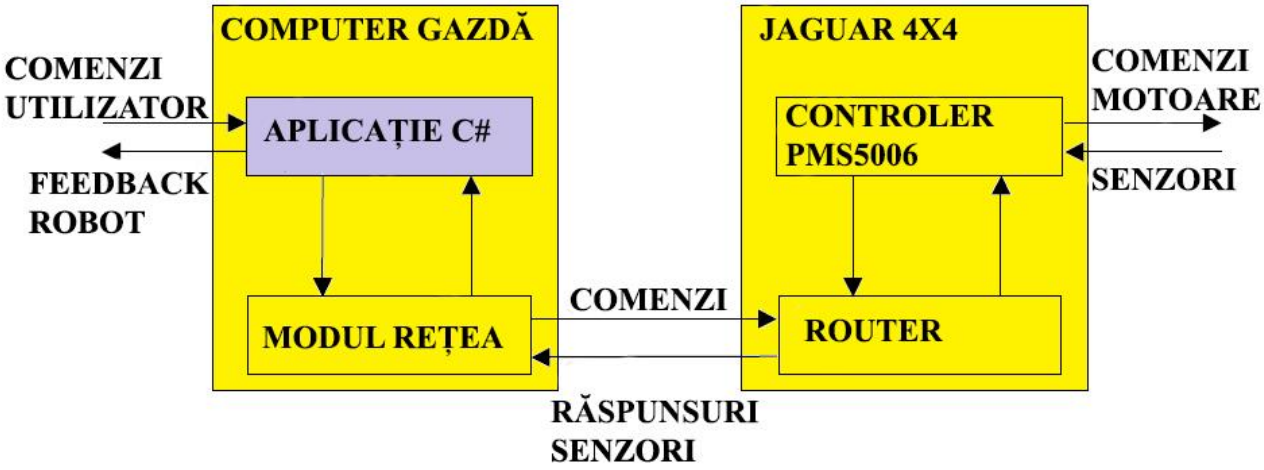


Figura 2.6 Schema funcțională a platformei mobile Jaguar 4X4

Pe calculatorul gazdă rulează o aplicație C# care transmite semnale de comandă către controlerul robotului folosind o conexiune fără fir (între computer și modulul de rețea al robotului). Recepționând aceste semnale, controlerul va acționa motoarele conform comenzilor primite. De asemenea, controlerul primește și date de la senzori, pe care le transmite mai departe către aplicația C# (prin aceeași conexiune fără fir). Aplicația decodează mesajele primite și folosește datele de la senzori fie pentru a le afișa, fie pentru a lua decizii ce influențează controlul actuatoarelor.

2.1.2.1 Clase și *namespace-uri*

Namespace-urile sunt utilizate pentru a defini un scop pentru un set de clase. În program folosim cuvântul cheie *namespace* pentru a declara un nou spațiu, spre exemplu *namespace JaguarControl*. Putem să privim conceptul de *namespace* ca o categorie din care fac parte mai multe clase și care comunică între ele. Un *namespace* conține unul sau mai multe tipuri, cum ar fi , o clasă din program. O clasă definește o nouă abstractizare de date, astfel ea are ca și caracteristici un nume de clasă și o colecție de membri. O clasă este unul din tipurile posibile într-un *namespace*. Clasele în C# suportă moștenirea singulară, astfel clasa moștenește de la o altă clasă atributele și metodele. Membri unei clase pot fi constante, câmpuri, metode, proprietăți, evenimente, operatori și constructori.

Principalul avantaj al *namespace-urilor* este că oferă o modalitate de a separa seturi de clase cu același nume, numele din cadrul unui spațiu nu va intra în conflict cu același nume declarat în alt spațiu.

2.1.2.2 Constructori, metode și atribute

Elementele principale cu care operează o clasă sunt atributele sale. Acestea reprezintă însușirile unei clase, ce informații poate să o definească.

Pentru o anumită clasă putem defini constructori și metode. Acestea pot fi declarate cu un anumit tip de accesibilitate, C# aduce mai multe tipuri de nivele de accesibilitate ca public, protejat și privat.

Primul membru se numește constructor și funcționează ca un constructor în C++. Dacă constructorul inițializează o nouă instanță a unei clase se numește constructor de instanță. Un constructor de instanță fără parametri se numește constructor implicit. C# suportă de asemenea constructori statici pentru a inițializa clasa. În cazul moștenirii, clasa copil nu moștenește și constructorul, acesta trebuie implementată separat.

Un alt membru al clasei poate fi o metodă. O metodă este un membru care face o operație în clasa respectivă. O metodă de instanță operează pe o instanță a clasei cât timp o metodă statică operează pe tipul însuși. Metodele din C# funcționează în mare parte ca în C++.

Un constructor de instanță este invocat când un obiect al clasei respective este creat. În mod normal, obiectele de orice tip sunt inițializate folosind cuvântul cheie *new*. O metodă trebuie invocată explicit în program.

2.1.2.3 Clasificarea tipurilor C#

Pentru a inițializa orice tip în C# este utilizat cuvântul cheie *new*. Acesta include atât clase și structuri cât și tipuri simple ca întregi sau enumerări.

Există două clasificări pentru tipurile de date în C#, fiecare având un anumit comportament de inițializare.

- Tipuri de valori, conțin respectivele date pentru tip. Acesta include tipuri integrate cum ar fi întreg, caractere, boolean dar și structuri create folosind cuvântul cheie *struct*. Tipurile de valori sunt de obicei mici, ceea ce face ușoară stocarea lor în stivă sau în obiectul care le conține.
- Tipurile de referință conțin o referință la date. Toate clasele din C# sunt tipuri de referință, ca și tipurile integrate *object* și s*tring*. Compilatorul transformă automat tipurile de valori în tipuri de referință folosind un proces numit *boxing*.

Aria de memorie rezervată pentru referințe de date se numește *heap*. Memoria alocată în *heap*, este recăpătată folosind *garbage collection*. Fără a intra prea mult în detalii despre acest subiect, vom considera acest *garbage collector* ca o metodă prin care scăpăm de pointeri și utilizări ineficiente de memorie [5].

2.1.2.4 Cuvinte cheie

Prima schimbare care poate fi observată într-un cod C# este folosirea cuvântului cheie *using* la începutul unui program.

Exemplu:

- *using System;*
- *using System.Windows.Forms;*

Cuvântul *using* joacă de fapt două roluri în C#. Primul rol este de a specifica o scurtătură, al doilea este pentru a ne asigura că resursele ce nu țin de memorie sunt înlăturate.

Ca directivă, *using* declară un *namespace* sau alias, care va fi folosit în fișierul curent. A nu se confunda cu fișierele *include* din C / C++. Fișierele *include* nu sunt necesare în C# deoarece asamblarea încorporează toate aceste informații.

În mod normal directiva *using* pur și simplu indică *namespace-ul* folosit de program, pentru a simplifica scrierea codului.

2.1.2.5 Biblioteca OpenCV

Open Source Computer Vision este o librărie *open-source*, specializată pe vedere computerizată în timp-real, care include sute de algoritmi din această categorie.

OpenCV are o structură modulară, acest lucru înseamnă că pachetul include atât librării statice cât și distribuite. Următoarele module sunt disponibile [6]:

- *Core* – un modul compact care definește structuri de date de bază. Incluzând șirurile multidimensionale și funcțiile de bază folosite de toate celelalte module.
- *Imgproc* – un modul de procesare de imagine care include atât filtre liniare cât și neliniare, transformări geometrice (transformări afine, scalare, remapare), conversie în diferite spații de culoare, histograme, etc.
- *Video* – un modul de analiză video care include estimări de mișcare, extrageri de fundal, și algoritmi de urmărire pentru diferite obiecte.
- *Calib3d* – algoritmi de bază geometrici, calibrare pentru camere (single și stereo), estimarea posturii unui obiect, elemente de reconstrucție 3D.
- *Features2d* – detector de caracteristici, descriptori.
- *Objdetect* – detecție de obiecte și de clase predefinite (spre exemplu, fețe, ochi, oameni, etc.)
- *Highui* – o interfață ușor de folosit pentru captură video, *codecuri* pentru imagini și video
- *Gpu* – algorimti de accelerare GPU.
- Alte module ca wrappere pentru Python sau C#.

În acest proiect librăria *OpenCV* a fost folosită pentru a detecta obstacolele pe care platforma robotică trebuie sa le ocolească. Deoarece toată informația despre mediul înconjurător este obținută de la camera video, biblioteca *OpenCV* s-a potrivit perfect la procesările de imagini și extragerea de informație utilă robotului Jaguar 4X4.
OpenCv este scris în limbajul C++ și are interfața principală în C++. Există translații în Python, Java și MATLAB/OCTAVE. Noile dezvoltări și algoritmii în *OpenCV* se realizează acum pentru interfața C++.
Biblioteca *OpenCV* rulează pe cele mai întâlnite sisteme de operare cum ar fi: Windows, Linux, macOS, FreeBSD, NetBSD, OpenBSD; dar această librărie poate să ruleze și pe platforme mobile cum ar fi: Android și iOS.

2.1.3 Aplicații client-server

Internetul funcționează pe un sistem de protocoale numit TCP/IP. Acestea stabilesc regulile cu ajutorul cărora două calculatoare comunică între ele. Java implementează protocoalele de nivel superior al stivei de protocoale TCP/IP, acest lucru facilitând utilizarea protocoalelor HTTP și FTP. Astfel, programatorul va utiliza niște clase și interfețe predefinite, fără a fi necesar să cunoască detaliile de implementare a acestora.
Serverul este o aplicație ce oferă servicii clienților sosiți prin rețea. Serverele oferă o gamă variată de servicii. Cel mai cunoscut server este serverul *Web*, care furnizează documentele cerute de către clienți. Arhitectura client-server este instrumentul de bază în dezvoltarea aplicațiilor de rețea.
Clientul este o aplicație care utilizează serviciile oferite de către un server. Pentru a putea realiza acest lucru, clientul trebuie să cunoască unde se află serverul în rețea, cum trebuie comunicat cu acesta și ce servicii oferă. Cu alte cuvinte, dacă un client dorește o comunicare cu serverul, trebuie să cunoască trei lucruri:

- adresa server
- portul server utilizat pentru comunicare
- protocolul de comunicație utilizat de server

Dacă aceste date sunt disponibile, se poate realiza comunicația cu serverul.

2.1.3.1 Porturi și clasa Socket

Porturile și soclurile reprezintă mecanismul prin care se realizează legătura cu un server. La aceeași adresă se pot oferi diferite servicii, acestea fiind oferite la porturi diferite. Același calculator (cu o singură adresă IP) poate să ofere oricâte servicii dorește. Clienții care apelează la serviciile acestui calculator vor utiliza aceeași adresă, indiferent la care serviciu apelează, și toți clienții care doresc utilizarea unui anumit serviciu vor utiliza același port. Un număr de port este un număr întreg din intervalul (1,9999).
Un soclu este de fapt un nod abstract de comunicație. Soclurile reprezintă o interfață de nivel scăzut pentru comunicarea în rețea. Acestea permit comunicarea între procese aflate pe același calculator sau pe calculatoare diferite din rețea.

Mecanismul de socluri a fost definit prima dată în Berkeley Software Distribution Unix. Java suportă trei tipuri de socluri. În lucrarea de față am utilizat clasa *Socket* care folosește un protocol orientat pe conexiune TCP. Soclurile utilizează fluxuri de date pentru a trimite și a recepționa mesaje.
Modul de comunicație între client și server are la bază protocolul TCP. Într-o aplicație de rețea întotdeauna avem două părți: partea client care inițializează conversația și trimite cereri, și partea server care primește cererile și răspunde la acestea. Clientul întotdeauna creează un soclu pentru a iniția conversația și trebuie să cunoască serverul căruia adresează cererea, iar serverul trebuie să fie pregătit pentru a recepționa aceste cereri. În momentul recepționării mesajului creează un soclu pe partea serverului, soclu care va facilita deservirea clientului. Atât pe partea de client cât și pe cea de server se utilizează câte un obiect de tip *Socket* pentru comunicare. Pe partea de server mai trebuie să creăm un obiect de tip *ServerSocket*, care are sarcina primirii conexiunilor și acceptarea acestora.
Schema bloc a acestei conexiuni folosită în lucrarea de față este exemplificată în Figura 2.7.

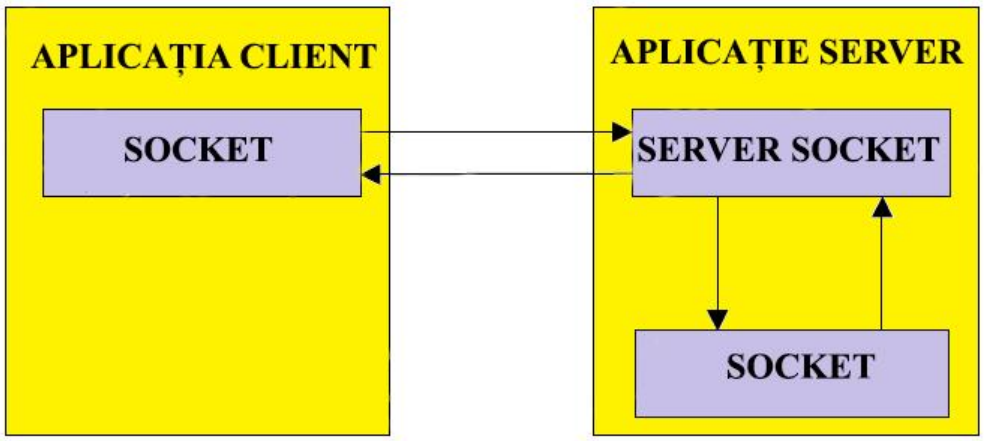


Figura 2.7 Conexiunea client-server utilizată

2.1.4 Aplicația demonstrativă pentru robotul Jaguar 4X4
Manualul de utilizare pentru platforma robotică mobilă Jaguar 4x4 a fost însoțit de un DVD care conține două programe demonstrative pentru utilizarea și controlarea robotului.
Programele au fost scrise în totalitate în C# folosind platforma Microsoft Visual Studio 2010. De asemenea există și o librărie de dezvoltare folosită pentru diferitele funcții de control și de interconectare.
În scenariul clasic de utilizare, aplicația demonstrativă trebuie să ruleze pe un computer gazdă, acesta fiind conectat la platforma robotică prin intermediul rețelei fără fir.
Astfel, pe computerul gazdă trebuie instalate următoarele programe [1]:

- Programul *Jaguar Control*
- Programul *Google Earth*
- SDK pentru camerele Axis

Programul va demonstra controlul și navigarea robotului Jaguar, mișcarea brațului și cum trebuie interpretate, procesate și afișate datele de la senzori.

Principalele acțiuni și informații disponibile în aplicația demonstrativă:

- Reîmprospătează datele citite de la codificatoarele motoarelor, de la senzorii de temperatură ai motoarelor, citește tensiunea de pe placa de alimentare, tensiunea disponibilă de la baterii, cu o frecvență de 10Hz.
- Citește și afișează date de la senzorul IMU și scanerul Laser.
- Afișează datele GPS folosind *Google Earth*.
- Afișează secvența video de la camerele Axis.

2.1.4.1 Mod de utilizare al aplicației demonstrative

Primul pas pentru conectarea la robot a fost impunerea computerului gazdă de a utiliza un IP static din gama de IP-uri corectă, pentru a mă conecta eu am folosit IP-urile 192.168.0.101 și 192.168.0.104 (server și client). IP-ul modulului de rețea asociat platformei robotice a fost 192.168.0.60 și restul modulelor din care este alcătuită platforma primesc IP-uri până la 192.168.0.65.

După accesarea executabilului aplicației demonstrative a platformei mobile ne întâmpină fereastra de autentificare, Figura 2.8, în care trebuie sa introducem IP-urile asociate de către rețeaua robotului modulelor sale [1].

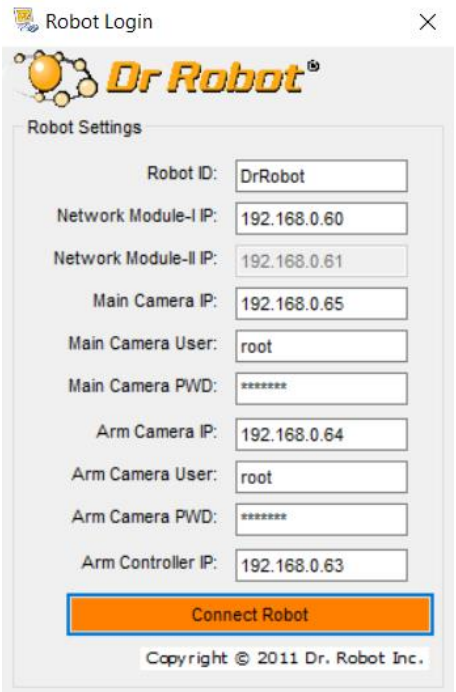


Figura 2.8 Fereastra de autentificare în aplicația demonstrativă [1]

În această fereastră se pot observa datele de conectare, și anume, IP-urile pentru fiecare din modulele robotului. După cum se poate observa în diagrama de conectare din capitolul 2.2, interconectarea tuturor modulelor se realizează prin intermediul unui modul de rețea, astfel fiecare componentă conectată la acest modul are asociată o adresă IP.

Clasa folosită pentru realizarea inițializării câmpurilor de IP-uri ale robotului este *DrRobotRobotConnection.cs*, în această clasă se realizează scrierea și citirea dintr-un fișier XML salvat în memoria internă a computerului gazdă, fișier din care se preiau IP-urile modulelor introduse anterior în aplicația demonstrativă pentru a ușura cât mai mult munca utilizatorului și de a parcurge rapid această etapă.

După apăsarea butonului *Connect Robot* aplicația realizează conexiunea fără fir cu robotul prin intermediul *socketelor* TCP, odată conectați se pot transmite și primi date de la fiecare din componentele prezentate în capitolul 2.2.

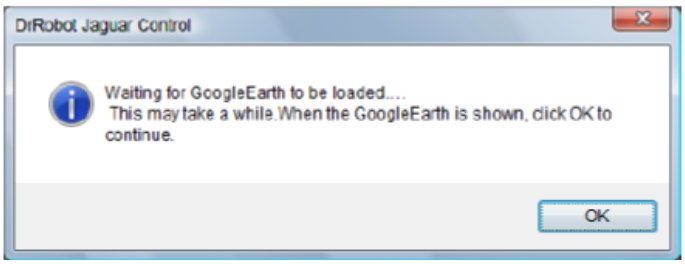


Figura 2.9 Fereastra de dialog pentru încărcarea hărților [1]

După această etapă se așteaptă verificarea conexiunii la internet pentru încărcarea hărților *Google Earth*. *Google Earth* suportă de asemenea și utilizarea fără internet, dar harta trebuie obținută în prealabil. Pe parcursul încărcării hărților va fi afișată următoarea fereastră de dialog.

După încărcare va apărea interfața completă a aplicației demonstrative.

Figura 2.10 Aplicația demonstrativă Jaguar [1]

În această interfață se poate observa imaginea în timp real primită de la camere, ferestrele pentru transmisia video sunt poziționate în partea dreaptă a interfeței. Se mai pot observa datele de la senzori în partea stângă, de asemenea, se pot observa și *trackbar-urile* care sunt folosite pentru controlul robotului (față/spate, stânga/dreapta).

Cele mai importante informații primite în această fereastră sunt datele de la senzori și de la driverele motoarelor afișate în partea din stânga jos. Deși aceste date nu sunt utilizate în controlul robotului, modul în care acestea sunt accesate este foarte important. Date precum poziția codificatoarelor sau temperatura motoarelor pot fi foarte folositoare în dezvoltarea și realizarea de aplicații cu roboți autonomi.

Figura 2.11 Datele de la motoare afișate în timp real

Dacă platforma robotică folosește bateriile LiPo incluse, acesta trebuie oprit daca tensiunea măsurată a pachetului de baterii coboară sub 22.2 V, așadar afișarea tensiunii bateriei în aplicației fiind foarte importantă din acest motiv. (Figura 2.11)

După cum se poate observa în această interfață nu avem acces la butoane de control pentru mișcarea brațului robotic. Butoanele de control pentru mișcările brațului au fost setate și implementate pe un controler de tip *Gamepad*.

În Figura 2.12 și Figura 2.13 avem detaliate mișcările robotice atribuite *Gamepad-ului*, prin această metodă de control avem acces la toate caracteristicile robotului cum ar fi operațiuni de prindere cu ajutorul brațului și al cleștelui, accesul la cele 2 camere video, acces la ventilatorul de pe brațul mobil.

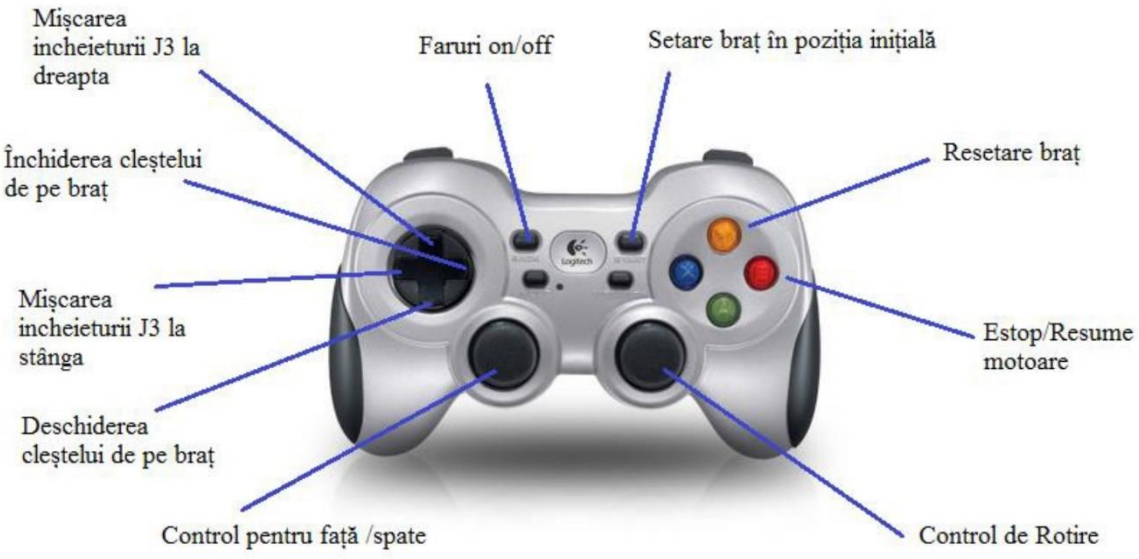


Figura 2.12 Instrucțiuni utilizare *Gamepad* a) [1]

Figura 2.13 Instrucțiuni utilizare *Gamepad* b) [1]

2.2 CAMERA KINECT

Modulul Kinect reprezintă un dispozitiv de detectare a mișcării, care a fost inițial dezvoltat pentru consola de jocuri Xbox 360. Unul dintre factorii distinctivi care fac ca acest dispozitiv să stea în evidență, printre alții în acest gen, este că nu este un dispozitiv simplu de control al mâinilor, ci poate detecta poziția corpului, mișcarea și vocea. Kinect oferă o interfață naturală de utilizare pentru interacțiune utilizând mișcarea și gestul corpului, precum și comenzile vorbite. Controlerul care a fost odată inima unui dispozitiv de jocuri, se găsește redundant în această epocă Kinect. Pentru cele mai recente versiuni ale Kinect, controlerul este reprezentat de utilizatorul însuși. Acest modul a inaugurat o nouă revoluție în lumea jocurilor și a schimbat complet percepția unui dispozitiv de jocuri [7].

2.2.1 Camera video

Camera video este utilizată pentru captarea și transmiterea datelor video color. Funcția sa este de a detecta culorile primare din sursă (RGB). Aceasta returnează fluxul de date constând într-o succesiune de cadre de imagine. Fluxul de culori Kinect suportă o viteză de 30 cadre pe secundă la o rezoluție de 1920X1080 pixeli. Unghiul de vizualizare pentru camera color este de 135^o [7].

2.2.2 Biblioteca PyKinect2

Această bibliotecă este destinată conectării camerei video Kinect cu unitate care se ocupă de prelucrarea imaginilor. Biblioteca PyKinect2 este scrisă în limbajul de programare Pyhon astfel încât este compatibilă cu limbajul de programare al sistemului utilizat.
Cu ajutorul metodelor prezente în această bibliotecă putem extrage ultimul cadru captat de camera Kinect, această informație este recepționată sub forma unei matrice cu 4 dimensiuni asociate culorilor roșu, verde, albastru și detecției în adâncime. Aceasta este forma standard pe care o transmite camera Kinect.

CAPITOLUL 3
PRELUCRAREA ȘI ANALIZA IMAGINILOR

3.1 INTRODUCERE

Prelucrarea și analiza imaginilor a apărut datorită necesității de a înlocui observatorul uman print-un robot mașină. Această necesitate a apărut pentru a ușura munca observatorului sau pentru a îl proteja de eventuale pericole aflate în afara zonei de siguranță. Analiza imaginilor a continuat să se dezvolte și au apărut prelucrări de imagini non-vizibile cum ar fi imaginile acustice (radar, ultrasonore) care duc la dezvoltarea unor sisteme robotice autonome fără a fi nevoie de intervenție umană.

Prelucrarea digitală constă în manipularea imaginilor (date bidimensionale) pe un calculator. Orice imagine are o structură bidimensională, matrice de date. Fiecare element din această matrice se numește pixel. În cazul nostru, imaginea primită de la camera Kinect este compusă dintr-o matrice tridimensională, cele 3 matrice din componență corespund unei culori din gama RGB, iar fiecare element din aceste matrice reprezintă intensitatea culorii respective pentru pixelul selectat. Amplitudinile unei imagini vor fi, în majoritatea cazurilor, fie numere reale, fie numere întregi. Numerele întregi sunt de obicei rezultatul unui proces de cuantizare care transformă o gamă continuă într-un număr discret de nivele.

Scopul prelucrării și analizării de imagini poate fi împărțit în trei categorii:

- Analiză de imagini (sunt extrase măsurători din imagine)
- Procesare de imagini (imaginea este prelucrată după nevoie rezultând la ieșire tot o imagine)
- Înțelegere de imagini (sunt detectate elemente reale din imagine, exemplul cel mai bun este o rețea neurală care detectează obiecte dintr-o imagine)

3.2 METODE DE DETECȚIE

Cele mai multe aplicații care a nevoie de informație de la o cameră digitală folosesc pentru procesare imagini binare. Acestea sunt simplu de generat, fie direct prin cameră digitală ori fie prin transformarea imaginilor color în imagini alb-negru. Un alt avantaj îl reprezintă memoria redusă pe care o ocupa o astfel de imagine, fiecare pixel reprezentând un bit sau mai puțin daca se aplică compresie.

Am ales sa folosesc o imagine binară deoarece algoritmii de prelucrare sunt mult mai rapizi atunci când parcurg o astfel de imagine. Un alt criteriu a fost cantitatea de informație extrasă din imaginea binară care este mai mult decât suficienta pentru aplicația dezvoltată.

Cum pentru orice alegere există un compromis, imaginile binare au și ele dezavantaje. Cel mai important este că acestea nu pot sugera natura tridimensională a obiectelor imortalizate. De asemenea pentru a obține o imagine binară suficient de sugestivă pentru această aplicație s-a presupus un nivel de iluminare mediu al spațiului în care se află robotul Jaguar, fără surse de iluminare directe cum ar fi razele soarelui sau surse artificiale.

3.2.1 Binarizare

Un mediu ideal în care să putem prelucra o imagine prin metoda de binarizare astfel încât sa obținem un contur cat mai bun al obiectelor trebuie sa respecte următoarele caracteristici:

- Obiectele trebuie să fie individuale, separate de o intensitate relativ mare, vizualizate pe un fundal de intensitate mică.
- Umbra obiectelor pe care vrem sa le detectăm în imagine poate influența procesul de binarizare, sursa de iluminare trebuie sa fie perpendiculară pe planul în care se află obiectele.

Scopul binarizării este de a segmenta imaginea în regiunea de interes și de a elimina regiunea de informație nefolosită. În funcție de informația pe care dorim să o segmentăm din imagine, binarizarea va folosi unul sau mai multe praguri de intensitate pentru a izola obiectele de interes. Pentru a determina aceste praguri se folosesc multiple măsurări statistice.

Există mai multe tipuri de binarizare pe care le putem folosi [8]:

- Binarizare cu prag variabil
- Binarizare cu mai multe praguri
- Binarizare globală
- Semi-binarizare

Pentru aplicația de față s-a folosit binarizarea cu prag variabil, acest tip este suficient de precis pentru a identifica obiectele din imagine pentru mai multe stagii de luminozitate.

Am decis ca toate prelucrările și analizele de imagine să fie realizate în limbajul de programare Python deoarece acesta facilitează operațiile cu matrice și are implementat suport pentru librăria OpenCV, librărie pe care o utilizam în analiza imaginilor.

Pentru o binarizare cat mai precisă am decis ca obstacolele, pe care platforma robotică Jaguar trebuie sa le ocolească, să fie de culoare roșie iar fundalul să fie de culoare gri (Figura 3.1).

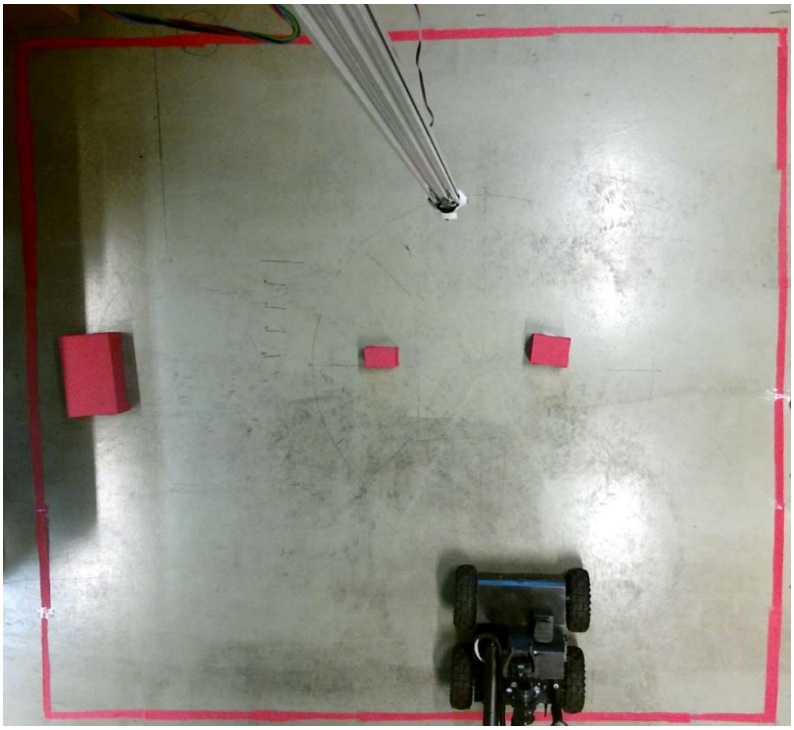


Figura 3.1 Imagine preluată de la camera Kinect

Platforma robotică Jaguar 4X4 dispune de motoare suficient de puternice pentru a produce un impact puternic asupra obstacolelor din fața acestuia iar pentru siguranță am realizat o delimitare a spațiului în care robotul își poate desfășura operațiunile. Acest spațiu se poate observa în Figura 3.1, are culoarea identică cu obiectele pe care acesta trebuie sa le ocolească, principiul fiind același.
După o primă operație de binarizare cu un singur prag se observă atât spațiul de operare dar și obiectele cu aceleași proprietăți de culoare (Figura 3.2).
Aplicația fiind utilizată atât în condiții de luminozitate slabă dar și ridicată, pragul cu care se binarizează imaginea poate fi modificat de către orice utilizator. După operația de binarizare imaginea rezultată o sa fie compusă din trei matrice specifice culorilor roșu, albastru și verde precum imaginea originală. Astfel trebuie realizată o conversie din imagine color în imagine alb-negru pentru ca imaginea finală să fie de o singură matrice simplificând următorii pași ai aplicației.
Rezultatul tuturor proceselor reprezintă o imagine cu două niveluri, alb și negru. Matricea corespunzătoare are dimensiunea imaginii, unui pixel alb îi corespunde numărul întreg 255 iar pentru un pixel negru îi corespunde numărul întreg 0. Această formă finală a prelucrării facilitează construcția algoritmului care calculează cel mai scurt traseu între două puncte dintr-o imagine.

Conturul de culoare negru nu poate fi atins de către robot, spațiul în care acesta își poate desfășura operațiunile este de culoare albă.

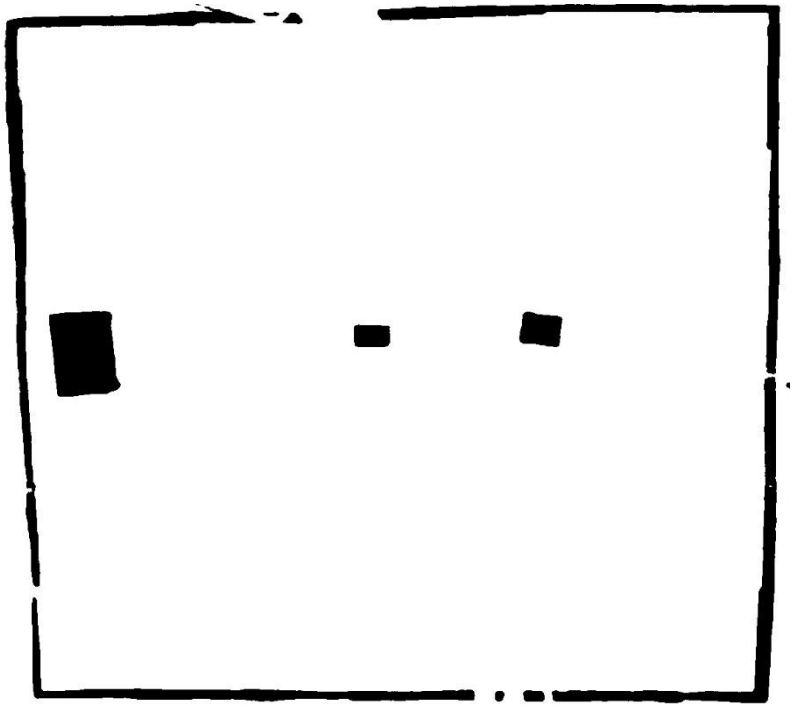


Figura 3.2 Imaginea după procesul de binarizare

Pragul optim pentru procesul de binarizare a fost calculat prin măsurători statistice succesive pe parcursul unei zile în interiorul unei încăperi. Principala sursă de lumină a fost cea naturală. Intervalul de timp dintre 2 măsurători succesive este de doua ore astfel încât nu pot exista rezultate schimbate semnificativ de la o măsurătoare la alta.

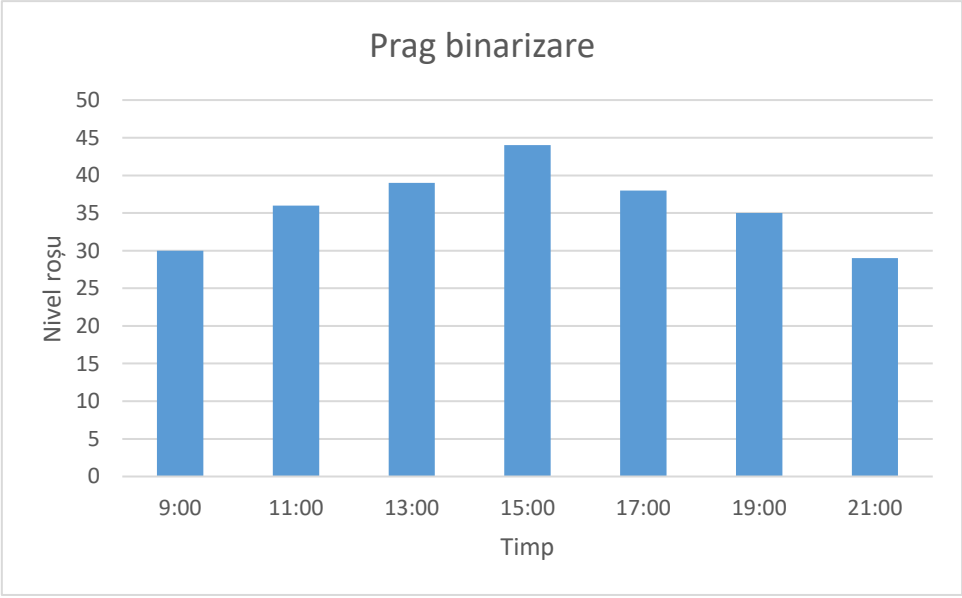


Figura 3.3 Pragul nivelului de culoare

În urma măsurătorilor am decis ca valoarea implicită a pragului sa fie 35 astfel încât să funcționeze indiferent de intervalul oral (nivelul de luminozitate) în care se testează aplicația.

3.2.2 Detecția de contur

O altă metodă de a detecta obiecte într-o imagine este detecția de contur. Conturul unei forme poate fi explicat ca fiind o curbă care unește toate punctele continue pe distanța de frontieră, având aceeași culoare. Pentru rezultate mai bune se pot folosi imagini binare.

Pentru operația de detecție de contur am folosit biblioteca OpenCV. Aceasta pune la dispoziția utilizatorului funcții precum binarizarea cu un singur prag, detecția de contur, transformări între imagini color și monocromatice[9].

Pentru o imagine provenită de la camera Kinect asemănătoare cu Figura 3.1, am obținut detecția de contur din Figura 3.4.

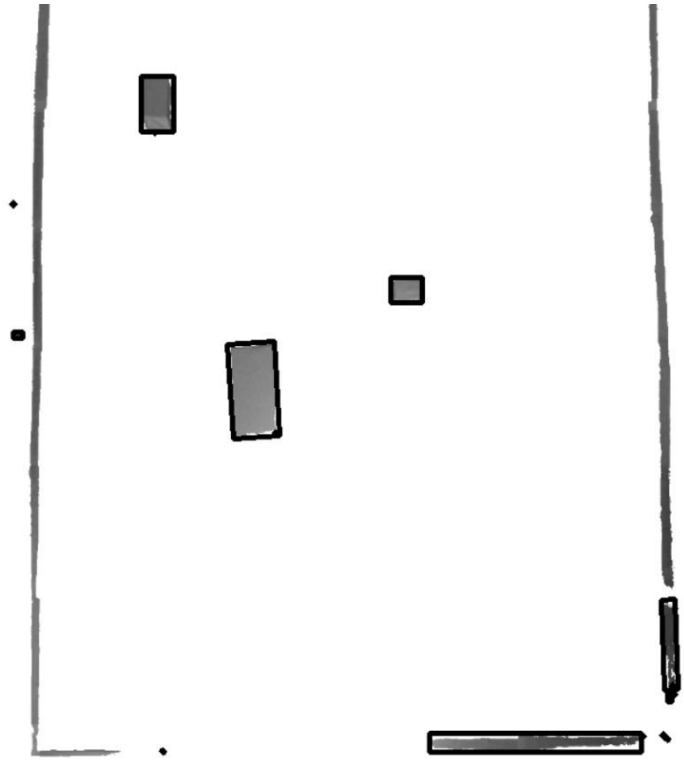


Figura 3.4 Detecția de contur

Rezultatul nu este satisfăcător, cu toate că obiectele din interiorul spațiului de operare au fost detectate cu succes, chenarul a rămas nedetectat.

Scopul final al acestei operațiuni este de a detecta pe o imagine binarizată toate componentele pe care robotul Jaguar trebuie să le ocolească și astfel conturul acestora să fie bine definit și suficient de mare astfel încât să se păstreze o distanță de siguranță între obiecte, chenarul de siguranță și robotul Jaguar.

O alternativă la această prelucrare este transformarea morfologică. Aceste transformări sunt operații simple care se bazează pe forma obiectelor din imagine. Aceste operații sunt de obicei folosite în imagini binarizate.

Transformările morfologice sunt de mai multe tipuri [10]:

- Eroziune
- Dilatare
- Deschidere (eroziune + dilatare)
- Închidere (dilatare + eroziune)
- Gradient morfologic (dilatare – eroziune)

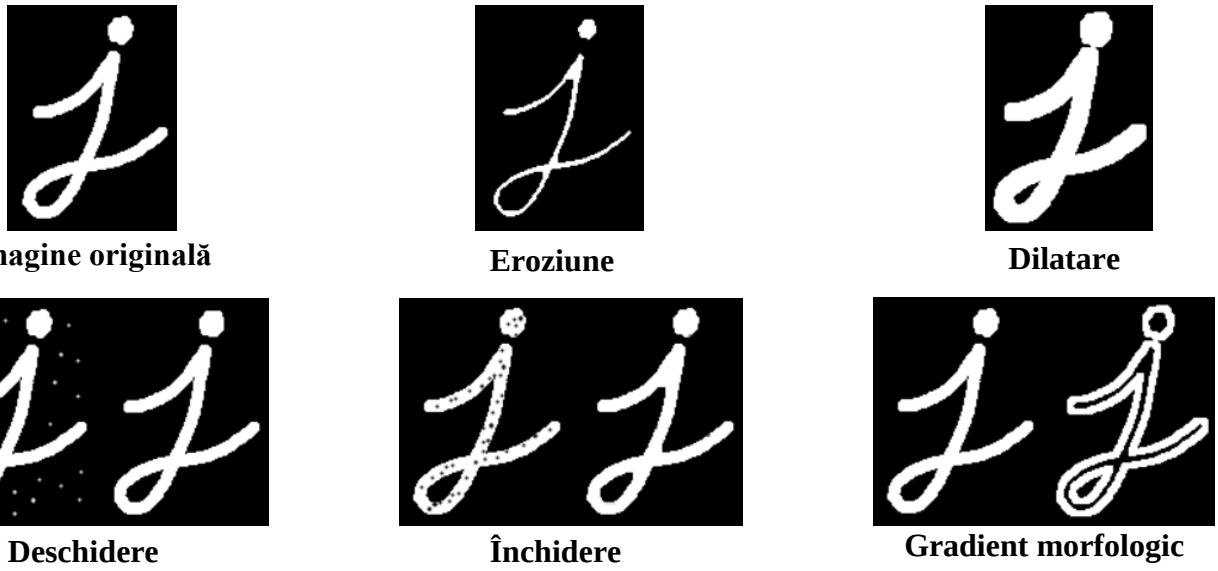


Figura 3.5 Transformări morfologice

Pentru a mări dimensiunea formelor din imagine vom folosi o transformare morfologică de tipul eroziune. Aceasta funcționează precum eroziunea din natură, conturul obiectului din prim-plan va fi erodat după forma acestuia cu ajutorul unui nucleu. Toți pixelii din jurul conturului vor fi aruncați în funcție de dimensiunea nucleului cu care se face transformarea.
În proiectul de față am folosit o transformare de tip eroziune cu un nucleu de 5X5 pixeli compus din valori de 1, valoare ce se identifică vizual ca fiind de culoare albă.

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Figura 3.6 Nucleu folosit în procesul de eroziune

Dimensiunea cu care obiectele își măresc aria poate fi controlată în funcție de numărul de iterații al procesului de eroziune. După transformarea de eroziune imaginea finală procesată arată ca în Figura 3.7.

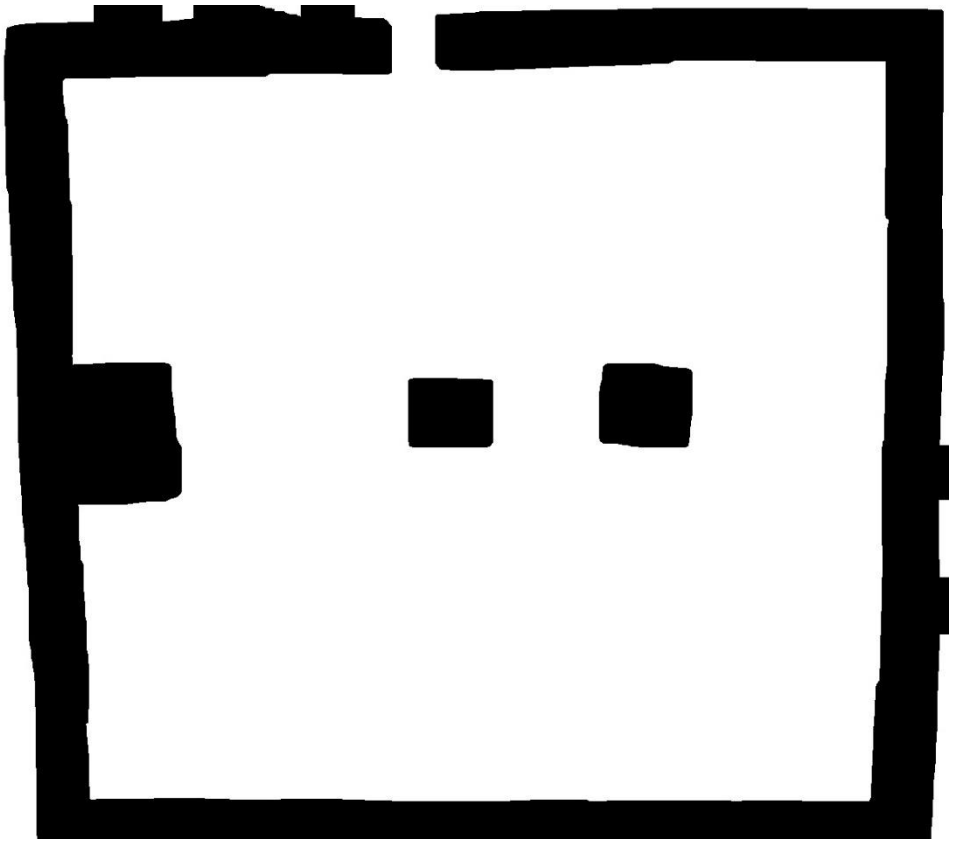


Figura 3.7 Imagine după procesul de eroziune

Imaginea din Figura 3.7 reprezintă forma finală a matricei ce urmează să fie prelucrată de către algoritmul care calculează cel mai scurt traseu dintre două puncte ce pot fi accesate în imagine. În acest moment putem considera platforma mobilă Jaguar ca fiind un pixel din imagine fără a exista posibilitatea unei coliziuni cu obiectele din jur sau de a ieși din zona sigură în care acesta poate opera.

3.3 ALGORITM PENTRU GĂSIREA CELUI MAI SCURT DRUM ÎNTR-O IMAGINE

Algoritmii care calculează cea mai scurtă rută între 2 puncte dintr-o imagine sau graf sunt populari în programarea roboților sau chiar a jocurilor video. Această practică este asemănată în viața reală cu rezolvarea unui labirint. Pentru operatorul uman este foarte ușor de a rezolva un labirint știind mărimea și posibilele căi dintre cele două puncte. Vom vedea că lucrurile nu sunt atât de ușoare în cazul roboților mai ales daca nu există informație a priori despre labirintul care trebuie rezolvat.

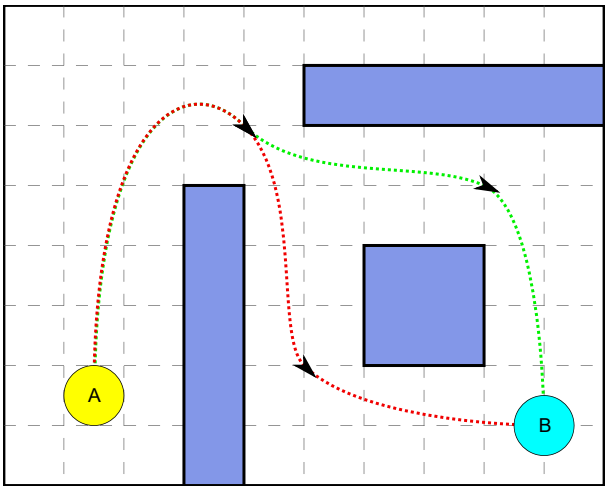


Figura 3.8 Drumul echivalent între punctele A și B [12]

Majoritatea algoritmilor dezvoltați pentru a găsi cea mai simplă rută între două puncte se bazează pe algoritmul lui Dijkstra [12]. Acest algoritm a fost dezvoltat pentru a îndeplini diferite criterii de performanță în funcție de nevoile aplicației:

- Cel mai scurt drum
- Drumul cu costul cel mai mic
- Cel mai rapid drum

Algoritmul Dijakstra [11] determină lungimea cea mai scurtă de la un nod de start (Nodul 1 în Figura 3.9) la toate celelalte noduri ale grafului. Acest algoritm funcționează doar pe grafurile orientate și ponderate. Fiecare muchie din graf are o anumită lungime pe baza căruia se calculează costul.

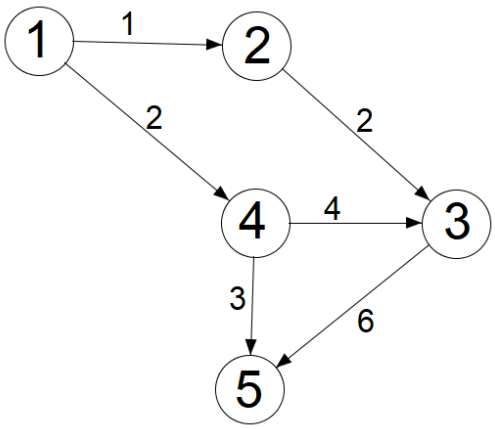


Figura 3.9 Model de graf eligibil pentru algoritmul Dijkstra [11]

3.3.1 Implementarea unui algoritm de tip Dijkstra

Algoritmul folosit în acest proiect are la bază un algoritm Dijkstra pe care l-am adaptat necesităților actuale. Limbajul de programare folosit este Python pentru a parcurge cât mai ușor matricea imagine. În continuare voi explica în detaliu cum am dezvoltat acest algoritm având ca intrare o matrice unidimensională, punctul de start în care se află robotul Jaguar 4X4 și punctul în care trebuie acesta să ajungă.

Fiecare pixel din imagine este privit ca un nod, acest nod are atribute precum coordonatele la care se află, nodul părinte prin care s-a ajuns în punctul actual și trei variabile F, G, H cu ajutorul cărora vom calcula costul trecerii de la un nod la altul.

- Variabila F reprezintă costul total al nodului și se calculează ca fiind suma variabilelor G și H
- Variabila G reprezintă distanța dintre nodul curent și nodul de pornire..
- Variabila H este o variabilă euristică care estimează distanța dintre nodul curent și nodul în care trebuie să ajungă robotul.

Considerăm un caz teoretic în care avem patru obstacole pe care trebuie să le evităm. Nodul start este marcat cu verde iar cel țintă este marcat cu roșu. Obiectele pe care nu trebuie să le atingem sunt de culoare închisă iar drumul cel mai scurt care rezultă după terminarea calculului este evidențiat de culoare albastru (Figura 3.10). Matricea are lungimea egală cu lățimea pentru a nu apărea erori în calculul variabilei G și H. Această restricție va fi respectată și pentru imaginile provenite de la camera Kinect, aceste poze vor fi redimensionate pentru a avea o formă pătratică.

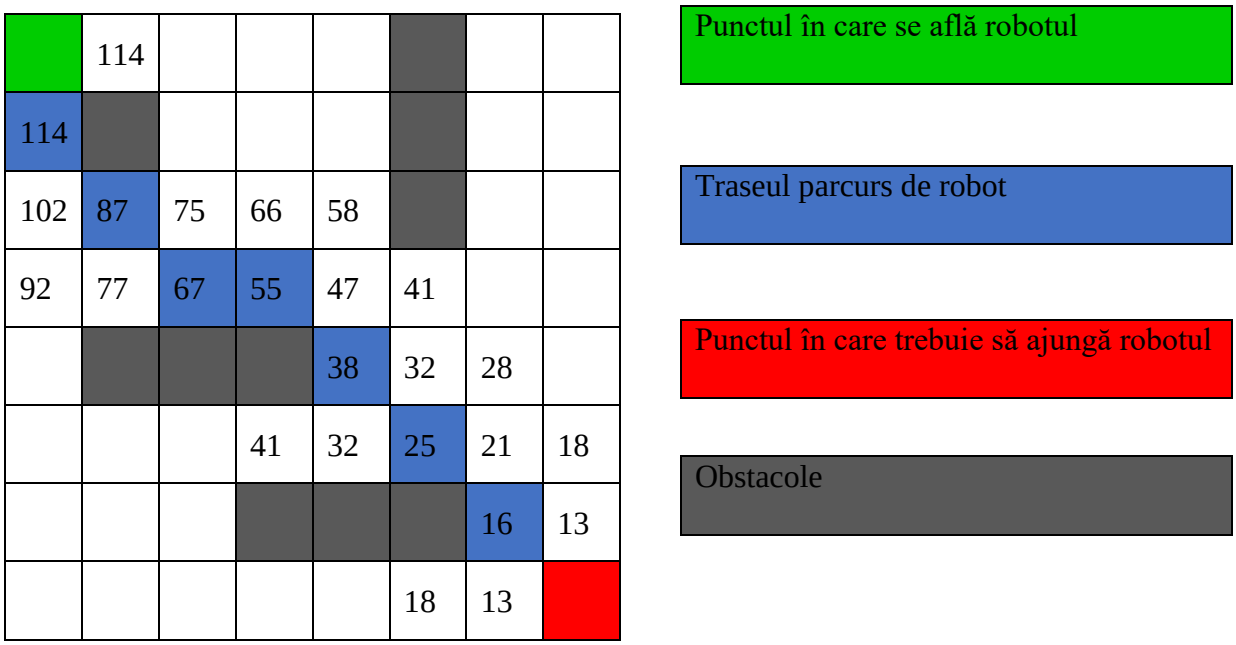


Figura 3.10 Simularea algoritmului de căutare

În Figura 3.10 am simulat funcționarea algoritmului calculând costul pentru fiecare vecin al nodului curent, alegându-l pe cel cu costul cel mai redus. Acest pas a fost repetat pentru fiecare nod vecin al noului nod ales, astfel ajungând până la punctul țintă pe cea mai scurtă distanță.
Primul pas pentru implementarea acestui cod a fost crearea unei clase care să simbolizeze un pixel, clasa am denumit-o sugestiv *Node* și are atributele menționate mai sus.

```
class Node():
    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position

        self.g = 0
        self.f = 0
        self.h = 0
```

Funcția în care am implementat tot algoritmul primește ca parametrii de intrare imaginea, după ce aceasta a fost procesată, și punctele de start și stop, puncte care vor fi inițializate manual de către utilizator printr-un click pe imaginea care îi va fi afișată. Aceste puncte sunt reprezentate de coordonatele pe cele 2 axe. Tipul returnat de funcție este un vector de puncte ce reprezintă fiecare punct ce trebuie parcurs pentru a ajunge în nodul țintă.

```
def pathfind(img, start, end):
    if img[end[0]][end[1]] != 255:
        print("Can't reach the end point!")
        return []
    if img[start[0]][start[1]] != 255:
        print("The start point is wrong!")
        return []

    start_node = Node(None, start)
    start_node.g = start_node.h = start_node.f = 0
    end_node = Node(None, end)
    end_node.g = end_node.h = end_node.f = 0
```

Fiecare element din matricea imagine poate să fie o valoare cuprinsă în intervalul [0,255], unde valoarea 0 este culoarea negru, culoare pe care o au obstacolele și chenarul pentru delimitarea activității, iar 255 este echivalent cu culoare alb, doar pe această culoare îi este permis robotului Jaguar să execute mișcări.
Odată apelată funcția *pathfind* verificăm dacă punctele de start și stop sunt accesibile, platforma robotică nu poate ajunge la un punct pe care nu îl poate accesa. Sunt create două obiecte de tipul *Node* pentru cele 2 puncte primite, punctul de start în care se află robotul și punctul în care trebuie să ajungă. Pentru a ține evidența nodurilor evaluate, prin care am trecut deja și cărora le-am calculat costul, și nodurile care urmează să fie evaluate am declarat doi vectori. În vectorul *closed_list* vom găsi nodurile prin care am trecut și nu mai trebuie să fie evaluate iar în vectorul *open_list* vom adăuga

nodurile pe care urmează să le evaluăm, pe tot parcursul algoritmului vom extrage elemente din această listă.

Daca nu mai avem elemente pe care să le extragem înseamnă că platforma robotică nu poate să ajungă la nodul destinație. Un astfel de caz în care traseul nu poate fi calculat este prezentat în figura următoare.

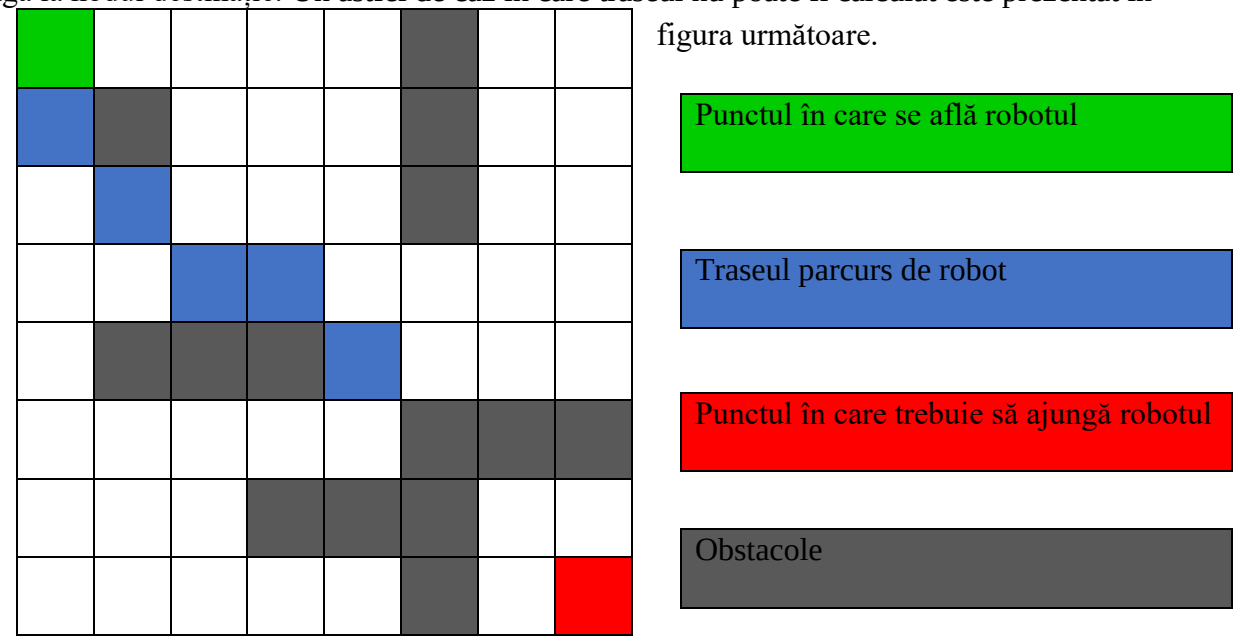


Figura 3.11 Caz imposibil de găsire al traseului

În continuare extragem din lista cu noduri ce așteaptă să fie evaluate, nodul cu cel mai mic cost, acesta reprezentând cea mai bună variantă spre nodul final. Odată găsit acest nod îl scoatem din lista cu noduri ce așteaptă să fie evaluate și îl introducem în lista cu noduri deja evaluate.

```
while len(open_list) > 0:
      current_node = open_list[0]
      current_index = 0

      for index, item in enumerate(open_list):
          if item.f < current_node.f:
              current_node = item
              current_index = index

      open_list.pop(current_index)
      closed_list.append(current_node)
```

Acest algoritm se poate încheia în două moduri. Primul este atunci când lista de noduri date spre evaluare este goală sau punctele de start și stop sunt inaccesibile, în acest caz traseul nu poate fi calculat și este afișată o eroare. Cel de-al doilea caz este atunci când nodul extras este chiar nodul în care trebuie să ajungem, în acest caz traseul a fost construit cu succes și urmează să fie returnat.

Pentru nodul extras, numit și nodul curent, se evaluează toți vecinii acestuia. Un nod are 8 vecini. Fiecare vecin este verificat daca poate fi accesat de către robotul Jaguar și daca reprezintă un nod valabil, să nu fie în afara dimensiunilor imaginii în cazul în care nodul curent se află la marginea acesteia. Dacă unul dintre vecinii nodului curent se află în lista cu noduri deja evaluate înseamnă că acesta a mai fost evaluat și nu are rost să îl reevaluăm.

```
children = []
for new_position in [(0,-1),(0,1),(-1,0),(1,0),(-1,-1),
                     (-1,1),(1,-1),(1,1)]:
    node_position = (current_node.position[0] + new_position[0],
                     current_node.position[1] + new_position[1])

    if node_position[0] > (len(img) - 1) or node_position[0] < 0
    or node_position[1] > (len(img[len(img) - 1]) - 1) or
       node_position[1] < 0:
       continue
    if img[node_position[0]][node_position[1]] != 255:
       continue
    if Node(current_node, node_position) in closed_list:
        continue
```

După ce am verificat punctele vecine și le-am extras doar pe cele valide se creează noduri având ca atribut părinte nodul curent și poziția specifică. Toate aceste noduri vecine sunt stocate într-un vector care urmează să fie parcurs în următorul pas.
Fiecărui vecin din vector îi este calculat costul cu ajutorul variabilelor G și H, dacă acest vecin se află deja în lista cu noduri ce urmează a fi evaluate dar diferă atributul G, care calculează distanța de la nodul start până la nodul curent, nodul curent nu va mai fi introdus în lista cu noduri ce urmează a fi explorate.

```
    new_node=Node(current_node,node_position)
    children.append(new_node)

for child in children:
    child.g = child.parent.g + 1
    child.h = ((child.position[0] - end_node.position[0]) ** 2) +
              ((child.position[1] - end_node.position[1]) ** 2)
    child.f = child.g + child.h

    flag = False
    for open_node in open_list:
```

```
            if child == open_node and child.g >= open_node.g:
                flag = True
        if flag == True:
            continue
        open_list.append(child)
```

3.3.3 Metode de optimizare al algoritmului

Viteza cu care acest algoritm calculează cel mai scurt traseu, între punctul în care se află platforma mobilă și punctul în care vrem să ajungă, depinde de mai mulți factori cum ar fi:

- Dimensiunea imaginii.
- Distanța dintre punctul de start și cel de stop.
- Dificultatea traseului, daca sunt multe obstacole de ocolit.

Pentru a optimiza acest algoritm am ținut cont de factorii care îl îngreunează și am încercat să găsesc o rezolvare pentru fiecare dintre ei.

Prima măsură a fost de a segmenta imaginea primită de la camera Kinect. Imaginea a fost adusă la dimensiunile chenarului în care robotul Jaguar poate funcționa. Astfel am eliminat informația din poză pe care nu o foloseam și timpul de procesare al acesteia s-a micșorat.

O altă măsură a fost de a redimensiona imaginea primită de la camera Kinect, imagine care este recepționată de către program cu rezoluția 1920X1080 pixeli. Am ales ca noua rezoluție a imaginii să fie 500X500 pixeli, astfel am redus cantitatea de pixeli pe care algoritmul trebuie să îi evalueze fără a pierde informație. Motivul pentru care am ales ca lungimea să fie egală cu lățimea pozei este de a diminua cât mai mult eventualele erori de calcul pentru traseele care își au ruta pe direcție diagonală.

Testele făcute cu acest algoritm au avut loc pe un labirint format din maxim 5 obstacole. Din teste am observat că durata de procesare al traseului crește exponențial dacă punctul în care trebuie să ajungă robotul este izolat de obstacole, programul evaluând astfel toate nodurile libere din imagine pentru a găsi o rută către punctul final. Acest caz trebuie evitat în aplicațiile de testare și cele reale.

.

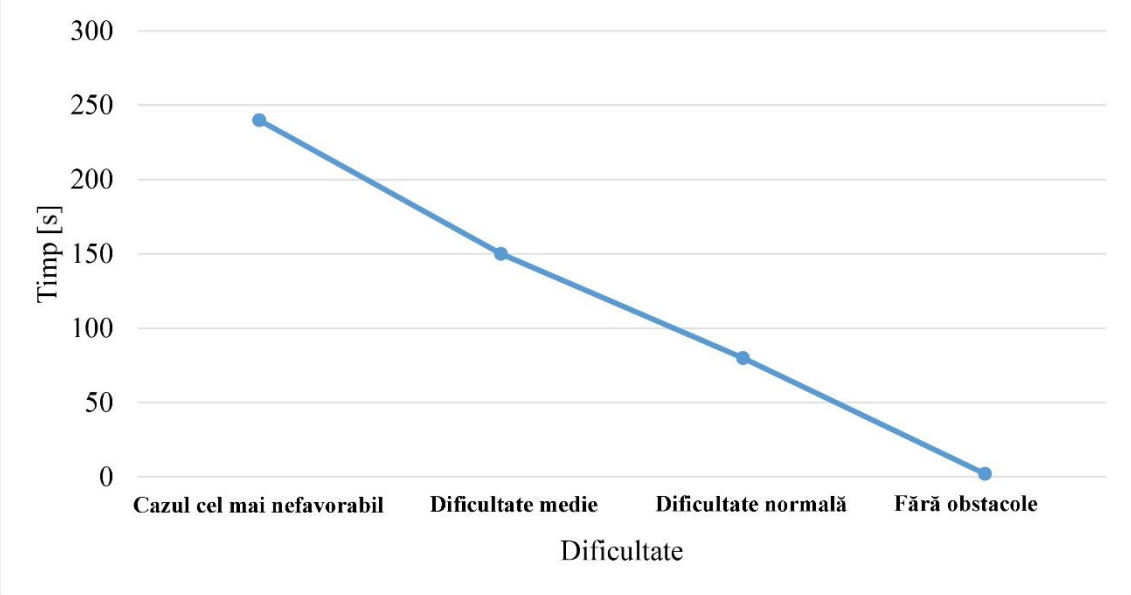


Figura 3.12 Timpul de procesare în funcție de complexitatea algoritmului

CAPITOLUL 4
CONTROLUL PLATFORMEI MOBILE ROBOTICE JAGUAR

4.1 DESCRIERE APLICAȚIE

Aplicația cu ajutorul căreia controlăm robotul Jaguar a fost scrisă în limbajul de programare C# care facilitează programarea orientată pe obiecte. Întregul proiect este format din 4 clase și o interfață. Pentru a ușura cât mai mult munca utilizatorului am implementat doar metodele strict necesare robotului de a își îndeplinii sarcinile. Astfel aplicația este mult mai ușor de înțeles și de operat.

Comenzile se transmit prin intermediul protocolului TCP, computerul gazdă fiind conectat la rețeaua robotului. Odată rulată aplicația, autentificarea către robot se face automat fără intervenția operatorului, acestuia îi rămâne sarcina de a transmite comenzi robotului.

Metodele de siguranță au fost păstrate astfel încât componentele robotului să nu fie suprasolicitate la o utilizare mai îndelungată sau mai activă. Utilizatorul este și el protejat, robotul Jaguar având motoarele oprite cât timp acesta așteaptă o comanda din partea computerului gazdă.

Cele 4 clase care alcătuiesc programul prin care poate fi controlat robotul Jaguar sunt:

- CommandController
- JaguarBaseController
- Program
- Host

4.2 ELEMENTE COMPONENTE

Prima clasă din lista de mai sus, *CommandController*, este folosită pentru a genera comenzile specifice mișcării motoarelor. În această aplicație avem nevoie ca platforma mobilă Jaguar să poată efectua rotiri precise la 45^o, 90^o și 135^o în ambele sensuri de mișcare dar și să parcurgă o distanță precisă măsurată în centimetri. Având acest scop am creat două funcții numite sugestiv *getRotationCommand* și *getMoveCommand*, ambele funcții primesc ca parametrii de intrare unghiul curent la care se află, respectiv distanța curentă parcursă, și unghiul la care vrem să ajungem, respectiv distanța pe care vrem ca robotul să o parcurgă.

La pornirea robotului Jaguar, codificatoarele de la motoare, respectiv modulul IMU sunt resetate și recalibrate astfel încât valoarea acestora tinde spre zero. Din momentul pornirii robotul Jaguar este înregistrată atât distanța parcursă de acesta, cu ajutorul codificatoarelor de la motoare, cât și accelerația și viteza cu ajutorului modulului IMU.

Orientarea robotului este calculată cu ajutorul algoritmului DCM folosind date de la GPS și accelerometru. Unghiurile sub care robotul poate executa mișcări de rotație sunt unghiuri Euler: yaw, pith și roll (Figura 4.1).

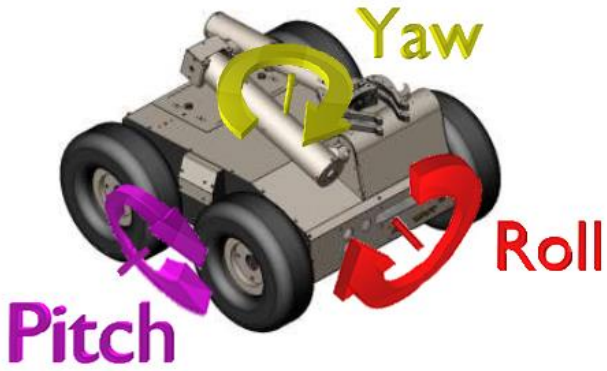


Figura 4.1 Axe de rotație ale platformei mobile

Datele calculate de către robot pentru orientare sunt transmise de către modulul IMU prin intermediul protocolului TCP către computerul gazdă. Un exemplu de pachet primit de către utilizator este următorul:

```
#0,YAW,0.46,GYRO,-42,-18,-11,ACC,4,-29,482,COMP,-858,-1589, 1333,ADC,0,0,0,0
#1,YAW,0.46,GYRO,-40,-20,-11,ACC,4,-28,480,COMP,-859,-1592,-1332,ADC,0,0,0,0
#2,YAW,0.46,GYRO,-41,-17,-11,ACC,3,-30,481,COMP,-858,-1589,-1331,ADC,0,0,0,0
#3,YAW,0.46,GYRO,-41,-16,-11,ACC,6,-29,480,COMP,-858,-1589,-1330,ADC,0,0,0,0
#4,YAW,0.46,GYRO,-41,-18,-11,ACC,4,-31,485,COMP,-859,-1590,-1331,ADC,0,0,0,0
#5,YAW,0.46,GYRO,-42,-15,-11,ACC,1,-28,478,COMP,-858,-1590,-1332,ADC,0,0,0,0
```

Valoarea orientării pe axa normală (YAW) este exprimată în radiani. Pentru a transforma în grade folosim următoarea formulă:

$\alpha_{(degrees)} = \alpha_{(radians)} \, X \, 180° \, / \pi$

Aplicând această formulă pentru pachetul extras mai sus rezultă o orientare a robotului Jaguar de 26.53^o față de locul în care a fost calibrat senzorul. Înainte de a executa o rotație vor fi trimise comenzi de calibrare al senzorului IMU pentru a diminua erorile cu care se rotește platforma robotică. Comanda de rotire a robotului Jaguar este trimisă către acesta în buclă până se atinge unghiul de rotație dorit.

```
public string getRotationCommand(double currentAlpha, double targetAlpha) {
            var err = (int)(targetAlpha * 100 - currentAlpha * 100);
            var command = "";

            err = Math.Min(Math.Max(err, -500), 500);
            command = String.Format("MMW !M {0} {1}", err, err);
            return command;
        }
```

Parametrii cu care este apelată această funcție au unitatea de măsură în grade. Variabila *err* reprezintă diferența dintre unghiul curent și unghiul la care se dorește rotirea robotului, în funcție de mărimea acestei variabile robotul are o mișcare uniformă și se rotește cu viteză mai mică atunci când este aproape de unghiul țintă pentru a fi cât mai precis.

Asemănător cu funcția care construiește comenzile de rotație este și funcția cu ajutorul căreia sunt obținute comenzile pentru mișcarea robotului în linie dreaptă.

```
public string getMoveCommand(double currentDistance, double targetDistance) {
            var doubleErr = targetDistance - currentDistance;
            var command = "";

            var err = (int)Math.Min(Math.Max(doubleErr, -500), 500);
            command = String.Format("MMW !M {0} {1}", err, -err);
            return command;
        }
```

Diferența dintre cele două funcții reprezintă comanda finală pe care o returnează, pentru mișcarea robotului în linie dreaptă, roțile acestuia trebuie sa se miște cu aceeași viteză și în aceeași direcție.

Următoarea clasă din componența programului care are un rol foarte important în funcționarea acestuia este JaguarBaseController. Această clasă reprezintă principalul punct de control al robotului. În această clasă se stabilește conexiunea dintre platforma robotică Jaguar și computerul gazdă, aceasta este o conexiune de tip client-server. Această conexiune va fi detaliata mai târziu în acest capitol, în cadrul sistemului fiind utilizate două astfel de conexiuni. Ip-ul robotului nu se schimbă în cazul unei reporniri al acestuia, astfel acest parametru este memorat în interiorul acestei clase de către un atribut având clasificarea *private*.

Una dintre cele mai importante metode este cea care citește datele transmise de către modulul IMU al robotului Jaguar.

Platforma robotică Jaguar transmite în buclă deschisă date de la senzori către computerul gazdă, aceste date sunt preluate de către computer (care în acest caz reprezintă clientul) și este căutat pachetul în care se transmit date de la modulul IMU. Odată extras un astfel de pachet, identificat după simbolul de început "#" este folosită doar informația din dreptul indicatorului YAW, informația primită este reprezentată în radiani. Datele sunt prelucrate transformând din radiani în grade și retransmise mai departe către program. Valoarea unghiului de rotație primit de la robot este reîmprospătat cu frecvența de 50 Hz, suficient de rapid pentru a ști în permanență unghiul sub care se rotește robotul Jaguar.
Omologul metodei prezentată mai sus pentru codificatoare selectează, din toate pachetele transmise de către robot prin rețea, pachetele care conțin informația provenită de la codificatoarele motoarelor. Identificarea acestor pachete se face prin căutarea liniilor care încep cu "MM0 C" sau "MM1 C", pentru fiecare driver sunt transmise informațiile codificatoarelor celor două motoare atașate. Pentru această aplicație am considerat suficient extragerea datelor de la un singur codificator.
Datele înregistrate de modulul IMU sunt influențate de orice mișcare în spațiu al platformei robotice. După o distanță în linie dreaptă parcursă de Jaguar datele de la modulul IMU sunt eronate și acestea necesită calibrare.
Înainte de efectuarea unei rotiri se așteaptă calibrarea modulului IMU, calibrare declanșată prin transmiterea directă a comenzii: "SYS CAL". Astfel am realizat o metoda special pentru această calibrare care va fi apelată imediat înaintea unei rotiri.
Asemănător cu această calibrare a modulului IMU putem impune codificatoarelor să afișeze o valoare dictată de către utilizator. Pentru a diminua erorile de deplasare înaintea unei deplasări se apelează metoda de resetare a codificatoarelor, această metodă impune acestora valoarea 0.
Este necesar apelarea acestor metode de calibrare deoarece o rotire influențează datele de la codificatoare și o deplasare în linie dreaptă influențează datele de la modulul IMU.
Folosind metodele explicate mai sus am construit metoda finală care trebuie apelată pentru a trimite o comandă de rotire robotului Jaguar.

```
private void rotate(double alpha){
    while (true){
        var currentAlpha = readYawData();
        if (Math.Abs(currentAlpha - alpha) <= calibrationEpsilon){
            sendCommand(stopMotors);
            return;
        }
        var command = commandController.getRotationCommand(currentAlpha, alpha);
        sendCommand(startMotors);
        sendCommand(command);
    }
}
```

În această metodă variabila *calibrationEpsilon* reprezintă eroarea de rotație acceptabilă. Din cauza greutății mari a robotului, la o mișcare de rotație sau de deplasare acesta obține o forță de inerție suficient de mare pentru a depăși valoarea țintă, mișcarea necesitând recentrare.

Analog putem considera și metoda care îi transmite robotului distanța și direcția pe care trebuie să le parcurgă.

```
private void move(double distance){
  while (true){
      var currentDistance = readEncoderData();
      if (Math.Abs(currentDistance - distance) <= encoderEpsilon){
          sendCommand(startMotors);
          sendCommand(stopMotors);
          return;
      }
      var command = commandController.getMoveCommand(currentDistance, distance);
      sendCommand(startMotors);
      sendCommand(command);
  }
}
```

Toate metodele prezentate în această clasă construiesc funcția finală, funcție ce primește ca parametrii unghiul în funcție de care platforma robotică trebuie sa execute o mișcare de rotație și distanța pe care trebuie să o parcurgă în linie dreaptă. După apelarea acestei funcții platforma robotică Jaguar execută mai întâi mișcarea de rotație și după se deplasează cu distanța indicată. În această funcție sunt utilizate și comenzile de calibrare, apelate înainte de operația specifică fiecăreia.

```
public void relativeMove(double alpha, double distance)
{
    waitForCalibration();
    rotate(alpha);
    resetEncoders();
    distance = convertCentiMeterToEncoder(distance);
    move(distance);
}
```

4.3 ARHITECTURA CLIENT-SERVER

În acest proiect sunt folosite 2 astfel de arhitecturi client-server. Una dintre ele se ocupă de comunicația dintre robot și computerul gazdă iar cealaltă face legătura între computerul gazdă și computerul unde are loc procesarea de imagine provenită de la camera Kinect.

Arhitectura client-server s-a realizat cu ajutorul librăriei *System.Net.Sockets*, astfel se va crea un *Socket* de tip TCP pentru aplicația care va juca rolul de client. Computerul gazdă joacă rol de client în comunicația cu robotul Jaguar și rolul de server în comunicația cu stația ce procesează imaginile de la camera Kinect.

Pentru comunicația dintre computerul gazdă și robotul Jaguar stabilim capătul remote al socket-ului:

```
private IPEndPoint remoteEP = new IPEndPoint(IPAddress.Parse(ip), port);
```

Se va crea socket-ul TCP pentru client:

```
private Socket clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

Cu ajutorul acestui socket putem transmite pachete de date prin rețea către platforma robotică.

Obiectul de tip Socket are o metodă specială prin care putem transmite pachete de informație:

```
clientSocket.Send(ASCIIEncoding.UTF8.GetBytes(command + newLine));
```

Pentru a recepționa datele de la senzorii robotului Jaguar ne folosim de:

```
private NetworkStream replyStream = new NetworkStream(clientSocket);
private StreamReader readerReply = new StreamReader(replyStream ;
```

Citind valoarea variabilei *readerReply* vom observa toate pachetele trimise de robotul Jaguar pe această comunicație.

Pentru a nu intra în conflict aplicațiile client-server se vor executa secvențial. Prima comunicație inițiată va fi cu stația care se ocupă de procesarea și detectarea celui mai scurt drum pe care trebuie sa îl parcurgă robotul. În această comunicație computerul gazdă reprezintă serverul iar stația de procesare este clientul. În această aplicație vom folosi aceeași librărie ca mai sus, System.Net.Sockets.

Pentru această comunicație este suficient ca serverul sa primească informația, nefiind nevoie să transmită către client niciun pachet de date.

Datele primite de către server sunt de forma: “sendCommand(alpha,distance)”.

Comunicația cu clientul se va încheia când serverul primește comanda “endCommand”.

Întreaga rutină de comunicație se desfășoară într-o metodă ce returnează o listă cu perechile de comenzi (alpha,distance) pe care programul le transmite mai departe către robot.

Principalele componente ce alcătuiesc sistemul automat pentru evitarea obstacolelor sunt ilustrate în Figura 4.2 împreună cu modul de comunicație dintre acestea.

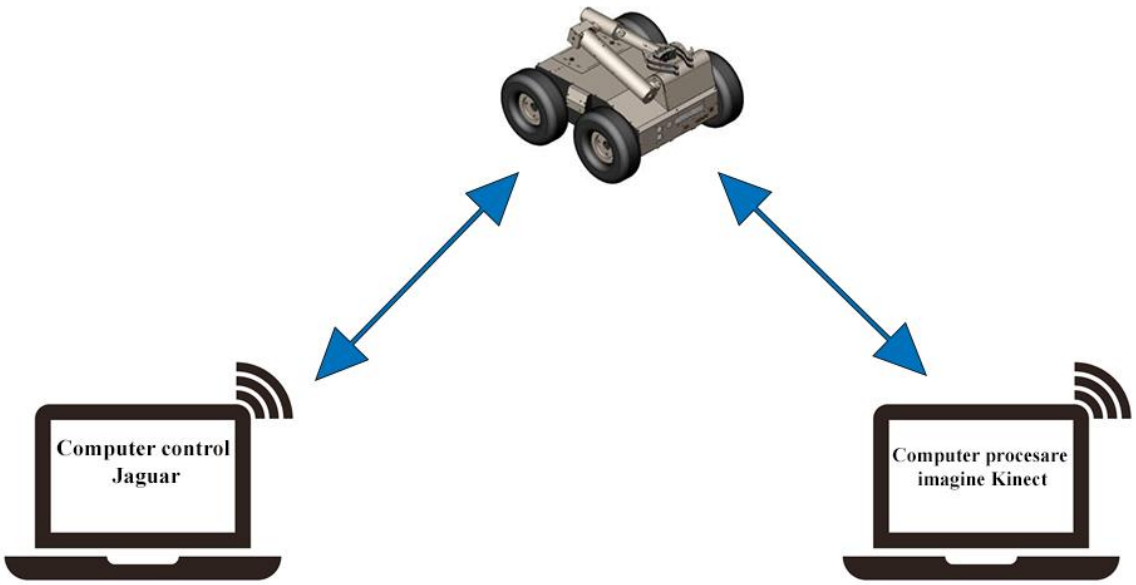


Figura 4.2 Legăturile între componentele sistemului

4.4 TRATAREA ERORILOR DE POZIȚIONARE

Principalele erori ce pot interveni în buna desfășurare a aplicației sunt erorile de poziționare. Acestea au ca sursă poziționarea camerei Kinect față de planul pe care își desfășoară activitatea robotul Jaguar, precizia de rotație și de deplasare a robotului, erori mecanice cum ar fi alinierea greșită a roților.
În Figura 4.3 este reprezentat mediul în care operează platforma robotică Jaguar 4X4 și măsurătorile semnificative cu ajutorul cărora se calculează distanța pe care trebuie să o parcurgă robotul Jaguar.

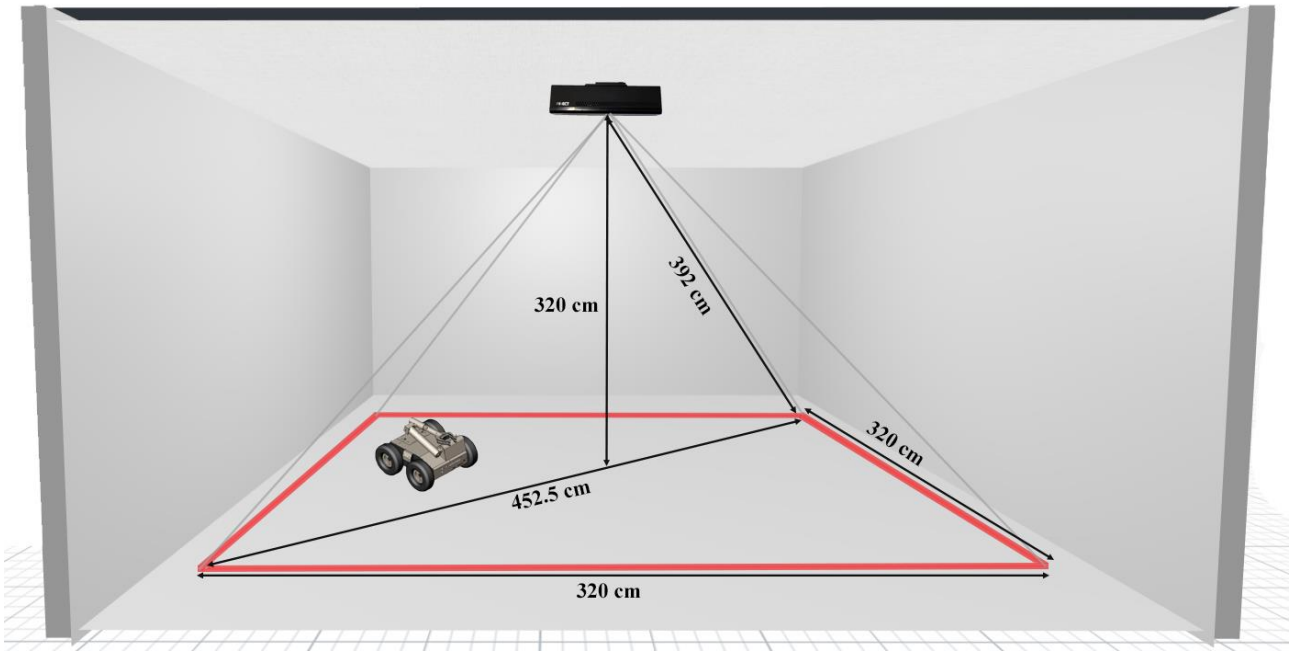


Figura 4.3 Mediul de operare al robotului Jaguar

Calibrările tuturor parametrilor au fost efectuați cu ajutorul acestor măsurători, pentru un alt mediu parametrii vor fi recalibrați în funcție de noile date.
După cum se poate observa platforma robotică Jaguar operează în interiorul unui piramide patrulatere regulate drepte, camera Kinect aflându-se în vârful piramidei. Baza piramidei este un pătrat și fețele laterale sunt triunghiuri isoscele. Înălțimea unește vârful piramidei cu centrul bazei care se află la intersecția diagonalelor bazei.
Pentru o rotație cât mai precisă roțile robotului Jaguar trebuie sa fie aliniate identic una cu cealaltă pentru ca axa de rotație sa fie în mijlocul robotului, astfel diminuăm abaterile de la traseu pentru rotirile imperfecte.
În funcție de înălțimea la care se află camera de unde sunt preluate imaginile, aria pe care robotul Jaguar se poate deplasa se mărește considerabil, cu toate acestea nu putem avea o înălțimea foarte mare deoarece s-ar pierde din acuratețea de detectare al obiectelor de la sol, astfel trebuie sa se găsească o înălțime potrivită pentru a îndeplinii cele două caracteristici.
O altă sursă a erorilor sunt transformările din virgulă mobilă în întregi, pe parcursul prelucrării de imagine sunt mai multe astfel de procese cum ar fi distanța reală calculată de program pe care trebuie să o parcurgă robotul, această distanță este calculată în pixeli după care este corelată în unități reale, centimetri, valoarea finală fiind rotunjită.

Pentru a determina eroarea cu care robotul Jaguar parcurge o distanță în linie dreaptă am efectuat zece măsurători pentru o comandă de avansare cu 150 cm. Rezultatele au fost transpuse în Figura 4.4.

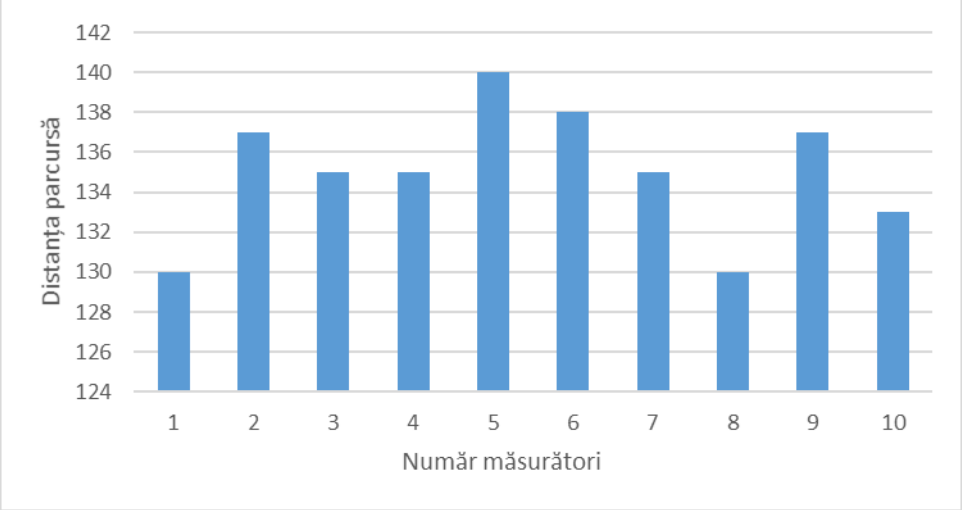


Figura 4.4 Eroarea pentru o mișcare de deplasare

Eroarea obținută pentru această măsurătoare este de 10% care reprezintă aproximativ 15 cm. Pentru a nu exista abateri de la traseul drept pe care trebuie să îl parcurgă robotul Jaguar, roțile acestuia trebuie să fie centrate și puterea transmisă către motoare sa fie egal distribuită.

Mișcarea de rotație a robotului se obține indicând motoarelor opuse să se rotească în antifază. Pentru a determina eroarea cu care robotul efectuează o rotație am efectuat zece măsurători trimițând platformei robotice comanda pentru o rotație la 45^o. Rezultatele obținute pot fi vizualizate în figura următoare.

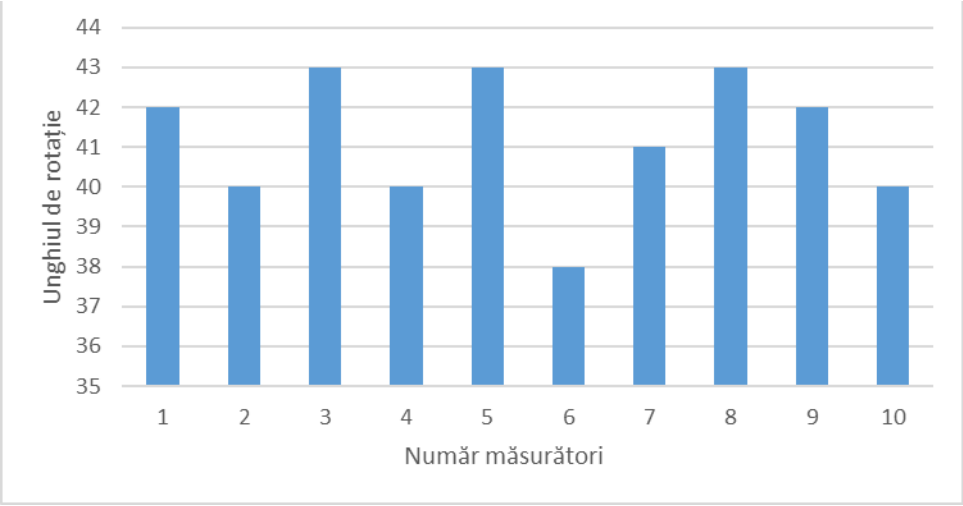


Figura 4.5 Eroarea pentru o mișcare de rotație

Calculând eroarea celor zece măsurători rezultă o eroare finală de aproximativ 11.1 % care reprezintă 5^o. Această eroare poate fi influențată și de factori externi cum ar fi tipul de teren pe care operează robotul Jaguar 4X4, presiunea din roțile acestuia și centrarea roților față de cadrul robotului.

CAPITOLUL 5

SISTEM DE EVITARE AUTOMATĂ A OBSTACOLELOR

5.1 INTRODUCERE

Proiectul de față își propune realizarea unei aplicații de control pentru o platformă robotică mobilă. Aceasta platformă trebuie să evite obstacolele pe care le întâlnește având ca sursă de informare o cameră video Kinect poziționată perpendicular cu planul de mișcare al robotului.

În acest capitol vom discuta despre această aplicație ce rezultă din combinarea tuturor modulelor discutate în capitolele anterioare.

Principalele cerințe pe care trebuie să le îndeplinească aplicația sunt următoarele:

- Găsirea celui mai scurt traseu
- Deplasarea în siguranță a robotului Jaguar 4X4
- Atingerea punctului final dictat de utilizator

5.2 ALGORITM DE FUNCȚIONARE AL SISTEMULUI

Sistemul de evitare automată a obstacolelor cu ajutorul platformei robotice mobile Jaguar 4X4 este format din trei mari componente:

- Robotul Jaguar 4X4
- Computer destinat prelucrării de imagini
- Computerul gazdă care comunică direct cu cele doua componente de mai sus

Pentru a explica acest algoritm mă voi folosi de schema bloc de funcționare (Figura 5.1). Vom considera un caz simplu în care robotul Jaguar trebuie să execute o deplasare spre înainte.

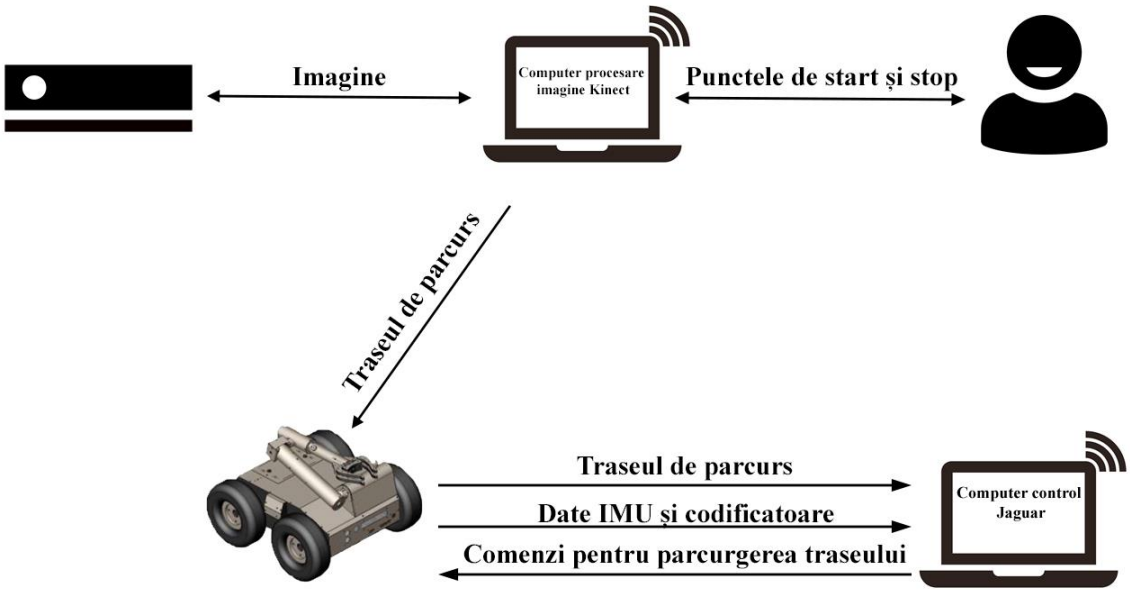


Figura 5.1 Schema de funcționare a sistemului automat de evitare a obstacolelor

Potrivit diagramei (Figura 5.1), computerul care se ocupă de procesarea imaginilor trimite o cerere către camera Kinect, primind ca răspuns ultimul cadru detectat de cameră. Această imagine este afișată utilizatorului, urmând ca acesta să selecteze, cu ajutorul cursorului, punctul în care se află robotul Jaguar și punctul în care trebuie să ajungă acesta. Computerul procesează imaginea și calculează cel mai scurt drum pe care îl poate parcurge robotul până la destinație. Coordonatele pe care trebuie să le parcurgă sunt transmise, prin rețeaua robotului Jaguar, către computerul care controlează robotul. După ce a primit coordonatele traseului computerul de control citește de la robotul Jaguar datele de la senzori, care sunt trimise în buclă deschisa prin rețea cu o frecvență de 50 Hz. Având toate datele necesare computerul construiește și trimite către robotul Jaguar comenzile de control pentru motoare. Robotul Jaguar efectuează mișcările în funcție de comenzile primite.
Această aplicație a fost dezvoltată pentru operațiuni de explorare, terenul ce urmează a fi explorat fiind necunoscut. Singura informație despre teren poate fi obținută de la un aparat de zbor cu sau fără pilot uman (dronă, elicopter). Pentru ca acest sistem să funcționeze corect se presupune că platforma robotică Jaguar 4X4 este amplasată la marginea zonei ce urmează a fi explorată, fața robotului fiind îndreptată către centrul terenului. Această amplasare poate fi efectuată cu ajutorul aparatului de zbor de la care este extrasă informația.

Un scenariu pentru acest sistem îl reprezintă cazul în care se dorește parcurgerea unui deșert de către un pluton militar. Aceste zone deșertice sunt deosebit de periculoase deoarece există posibilitatea de a întâlni o mină explozivă terestră. Cu un aparat de zbor se poate survola zona indicată astfel încât minele să fie expuse și detectate urmând ca informațiile să fie transmise platformei mobile Jaguar pentru construirea traseului în siguranță, urmând ca după parcurgerea zonei de risc de către robotul Jaguar și ajungerea acestuia în siguranță să urmeze trecerea plutonului militar.
Un alt scenariu poate fi închipuit în cazul unei catastrofe naturale. Robotul Jaguar 4X4 poate transporta alimente, unelte, truse de prim ajutor și chiar victime, în cazul copiilor răniți în urma dărâmăturilor. Planul de activitate îl reprezintă aria unui oraș iar observatorul aerian fiind reprezentat de mai multe drone pentru putea acoperii o arie cât mai mare. Datele preluate de a aceste drone pot fi transmise mai multor roboti de tipul Jaguar 4X4 care vor fi trimiși către mai multe puncte de interes.

5.3 DATE EXPERIMENTALE

Pentru a testa această aplicație am realizat două cazuri în care voi pune la încercare sistemul de evitare automată a obstacolelor cu robotul Jaguar 4X4. În aceste două cazuri este exploatată o arie de 9.61 metri pătrați. Obiectele pe care trebuie să le evite platforma robotică Jaguar au forme și dimensiuni diferite, amplasarea acestora fiind aleatoriu în cele două cazuri.
Primul scenariu este reprezentat în Figura 5.2, platforma robotică Jaguar 4X4 este poziționată în colțul din stânga jos. Punctele de start și stop sunt evidențiate cu ajutorul indicatorilor de coloare albastru. Obiectele pe care trebuie să le evite platforma robotică sunt de culoare roșie. Obiectivul pe care trebuie să îl atingă robotul este punctul albastru din colțul drept al imaginii.

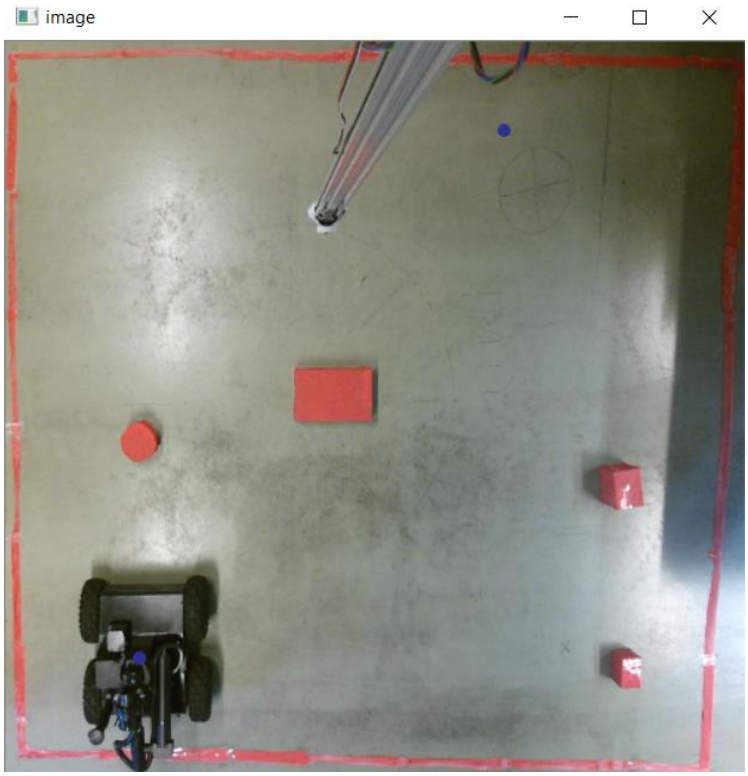


Figura 5.2 Simulare scenariu 1

După prelucrarea imaginii de la camera Kinect, înainte de a transmite cel mai scurt drum găsit către computerul care controlează platforma mobilă Jaguar, este afișat grafic drumul găsit (Figura 5.3).

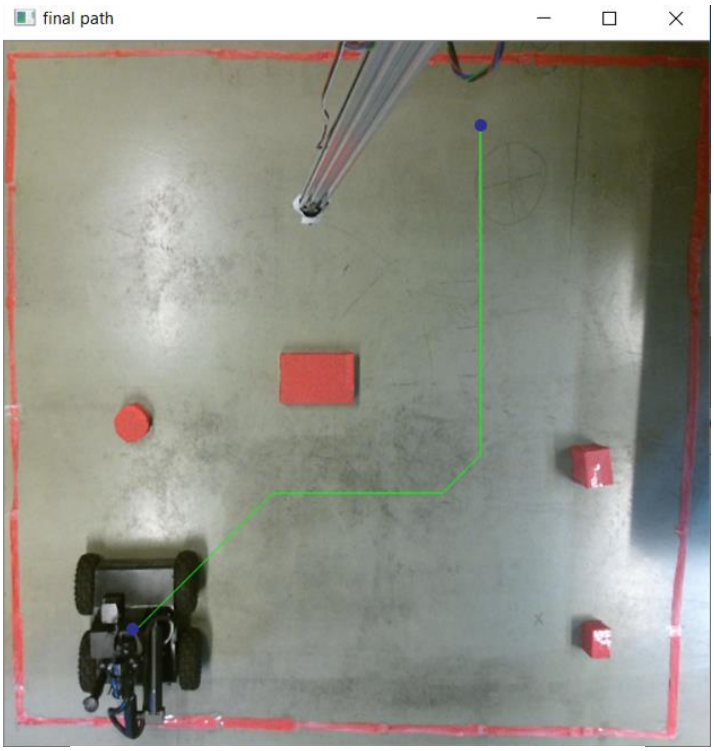


Figura 5.3 Traseu calculat de către sistem

După ce robotul Jaguar a efectuat operațiunile necesare pentru a ajunge în punctul final se poate compara traseul din Figura 5.3, calculat de program, și traseul real pe care l-a parcurs robotul, punctul final este reprezentat în Figura 5.4.

Figura 5.4 Punctul final atins de robotul Jaguar

Pentru acest traseu s-au obținut următoarele date tehnice:

- Timp procesare traseu: 4 secunde
- Timp parcurgere traseu: 18 secunde
- Timp total: 22 secunde
- Distanță parcursă de robot: 396 cm

Pentru cazul al doilea am realizat un al traseu, robotul Jaguar fiind înghesuit între obstacolele pe care trebuie să le evite. În acest test s-a dorit punerea în evidență a capacității robotului de a se strecura în siguranță printre obstacole.

Figura 5.5 Simulare scenariu 2

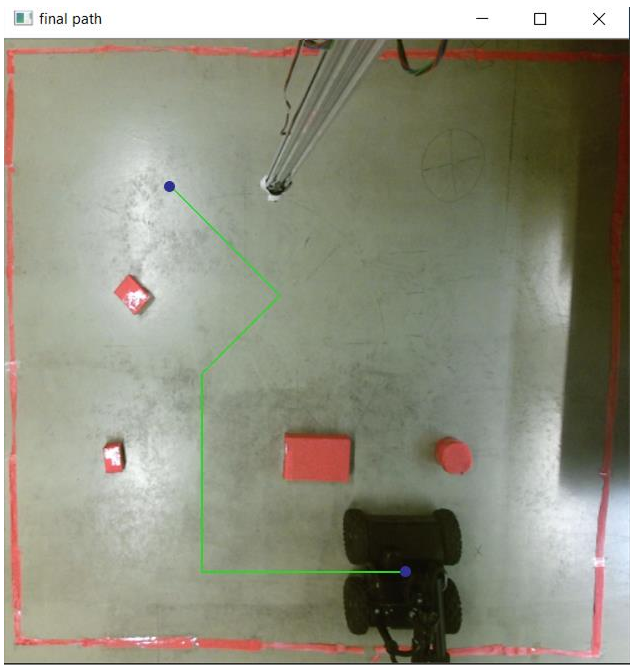


Figura 5.6 Traseu calculat

Rezultate tehnice:

- Timp procesare traseu: 3.4 secunde
- Timp parcurgere traseu: 28 secunde
- Timp total: 31.4 secunde
- Distanță parcursă de robot: 446 cm

CONCLUZII ȘI POSIBILITĂȚI DE DEZVOLTARE

CONCLUZII GENERALE

Obiectivul principal al acestei lucrări este de a dezvolta un sistem automat, capabil să evite obstacole, folosind robotul Jaguar 4X4 pe baza unor secvențe video. Sistemul este compus din trei parți principale: obținerea datelor, algoritmul pentru detectarea traseului cel mai scurt și implementarea mișcărilor pentru robotul Jaguar 4X4.

Pornind de la premisa din introducere, aceea că principalul scop al sistemului este de a oferi siguranță în deplasarea utilizatorilor, am vrut să testez precizia și viteza cu care robotul Jaguar 4X4 poate parcurge un traseu. Testele la care a fost supus sistemul au constat în capacitatea algoritmului de a detecta un traseu cât mai scurt și translatarea de către robot a ariei de observare până la punctul dorit. Cu această configurație, sistemul creat își poate găsi utilizarea în viața reală pentru aplicații militare, oferind siguranță în transport pentru zonele cu risc ridicat. De asemenea poate fi utilizat de către unitățile de salvare în cazul unui dezastru natural pentru a asigura transportul de medicamente și provizii în zonele grav afectate.

Sarcinile efectuate de acest sistem conțin o multitudine de mișcări bazate pe interpretare de date. Imaginile sunt transmise cu ajutorul unei librării dedicate pentru camera Kinect. Traseul este calculat de un algoritm care are la bază algoritmul Dijkstra. După acest pas traseul este interpretat și trimis către programul de control al platformei robotice. Programul pentru controlul robotului Jaguar este scris în totalitate de mine utilizând comenzile specifice pentru microcontrolerul robotului Jaguar.

CONTRIBUȚII PERSONALE

Pentru această lucrare, contribuțiile mele sunt următoarele:

- Am creat un mediu în care robotul Jaguar poate simula un caz real de ocolire a obstacolelor
- Am găsit o metodă de a poziționa camera cât mai sus în spațiul de simulare printr-un sistem glisant
- Am dezvoltat programul pentru controlul robotului Jaguar 4X4 folosind comenzi directe
- Am dezvoltat algoritmul pentru găsirea celui mai scurt traseu
- Am implementat metode pentru detecția obiectelor
- Am implementat interconectarea modulelor: computerul care controlează robotul, computerul pentru prelucrarea de imagine și robotul Jaguar, prin intermediul rețelei fără fir a platformei robotice
- Am programat robotul Jaguar să execute mișcări de rotație și de deplasare bazate pe comenzile primite

POSIBILITĂȚI DE DEZVOLTARE

Pe măsură ce tehnologia evoluează zi de zi, acest proiect trebuie să țină pasul cu noile tehnologii. Acest sistem poate fi îmbunătățit și dezvoltat în continuare cu scopul de a crea o rețea de roboți care comunică între ei și care muncesc în grup pentru a îndeplini un scop comun.

O trăsătură pe care as dori să o adaug în sistem este comunicarea directă a robotului cu un grup de aparate zburătoare, eliminând computerele intermediare, aceste aparate pot să acopere un spațiu considerabil astfel robotul poate sa caute un drum optim într-o arie mult mai mare.

De asemenea procesarea imaginilor poate fi îmbunătățită implementând elemente de inteligență artificială, astfel putem detecta o gamă mult mai mare de obiecte.

O formă finală a acestui proiect este crearea unei rețele de obiecte cum ar fi platforme mobile Jaguar, aparate zburătoare, roboți umanoizi interconectați prin intermediul internetului pentru a lucra împreună să ducă la bun sfârșit o sarcină fără intervenție umană.

REFERINȚE

[1] Manualul de utilizare al platformei robotice Jaguar 4X4:
http://jaguar.drrobot.com/images/Jaguar_4x4_Wheel_Manual.pdf
[2] Documentație Jaguar PMS5006 Protocol
[3] Documentație ROBOTEQ motor controller
[4] Specificații HOKUYO Range Finder:
http://www.robotshop.com/media/files/pdf2/utm-30lx-ew_specification.pdf
[5] Erik Brown, ”Windows Forms Programming with C#”
[6] Ghid de utilizare OpenCV:
http://docs.opencv.org/2.4.13/doc/user_guide/user_guide.html
[7] Documentație cameră video Kinect:
https://smeenk.com/kinect-field-of-view-comparison/
[8] Technical University of Cluj Napoca,”Procesarea imaginilor”:
http://users.utcluj.ro/~rdanescu/pi_c03.pdf
[9] OpenCV,Contour Approximation Method:
https://docs.opencv.org/3.3.1/d4/d73/tutorial_py_contours_begin.html
[10] OpenCV, Morphological Transformation:
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html
[11] Algoritmul Dijkstra:
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
[12] Descriere algoritmi de tip Pathfinding:
https://en.wikipedia.org/wiki/Pathfinding