

PLATFORMĂ ROBOTICĂ MOBILĂ CAPABILĂ DE RECUNOAȘTERE
VIDEO

LUCRARE DE DIPLOMĂ

Prezentată ca cerință parțială pentru obținerea titlului de *Inginer*
în domeniul *Electronică, Telecomunicații și Tehnologia Informației*
Programul de studii *Calculatoare și Tehnologia Informației*

Profesor coordonator:

Ș.L. Dr. Ing. Horia Cucu

Prof. Dr. Ing. Corneliu Burileanu

Student:

Codreanu Radu Adrian

București

2016

Universitatea "Politehnica" din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației
Departamentul Electronică Aplicată și Ingineria Informației

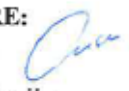
Aprobat Director de Departament :


Prof. Dr. Ing. Sever Pașca

TEMA PROIECTULUI DE DIPLOMĂ
Codreanu Radu Adrian, 442A

1. Titlul temei: Platforma robotica mobila capabila de recunoastere video.
2. Contribuția practică, originală a studentului va consta în (în afara părții de documentare):

Proiectul presupune realizarea unei interfete grafice ce va afișa toate datele primite de la senzori, și va putea trimite date înapoi către microcontroller-ul robotului. Camere video amplasate pe robot vor transmite imagini care vor fi inspectate și procesate prin metode eficiente. Astfel, se va realiza recunoașterea obiectului. Datele rezultate vor fi asociate unor comenzi pe calculatorul gazda, după care se vor transmite microcontroller-ului embedded al robotului care, prin semnale PWM va acționa motoarele, realizându-se astfel urmărirea obiectului.
3. Proiectul se bazează pe cunoștințe dobândite în principal la următoarele 4 discipline: Microcontrolere, Programare Obiect Orientată, Procesare de imagini, Recunoașterea formelor și inteligența artificială.
4. Proprietatea intelectuală asupra proiectului aparține: : Laboratorului de cercetare Speech and Dialogue (Speed) și studentului Codreanu Radu Adrian
5. Locul de desfășurare a activității: UPB, Laboratorul de cercetare Speech and Dialogue (Speed)
6. Realizarea practică rămâne în proprietatea: Laboratorului de cercetare Speech and Dialogue (Speed) și studentului Codreanu Radu Adrian.
7. Data eliberării temei: Noiembrie 2015

CONDUCĂTOR LUCRARE:
Ș.L. Dr. Ing. Horia Cucu 
Prof. Dr. Ing. Corneliu Burileanu 

STUDENT:
Codreanu Radu Adrian 

DECLARAȚIE DE ONESTITATE ACADEMICĂ

Prin prezenta declar că lucrarea cu titlul ”Platformă robotică mobilă capabilă de recunoaștere 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 Inginerie Electronică și Telecomunicații, programul de studii Calculatoare și Tehnologia Informației 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 alta 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 măsurărilor 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, 2016

Codreanu Radu Adrian

CUPRINS

Cuprins	7
Listă figuri	10
Listă tabele	13
Capitolul 1 Introducere	15
1.1 Abstract	15
1.2 Motivația lucrării	16
1.3 Roboți Autonomi	16
1.4 Obiectivul Lucrării	16
Capitolul 2 Platforma Robotică Jaguar	19
2.1 Prezentare	19
2.2 Componente Hardware	21
2.2.1 Controller-ul PMS5006	23
2.2.2 Mesajele cu date de la senzori	23
2.2.3 Modurile de control ale motoarelor	27
2.2.4 Encoder-ul Optic cu incrementare	29
2.2.5 Senzori externi	31
2.3 Componente Software	32
2.3.1. Framework-ul .NET	33

2.3.5	Windows Forms.....	34
2.3.3	Librăria OpenCV	37
2.3.4	Aplicația demo de control –Jaguar Control	42
Capitolul 3	Analiza și Prelucrarea Imaginilor	47
3.1	Introducere	47
3.2	Imagini Digitale	49
3.2.1	Operații pe imagini	49
3.2.2	Filtre liniare	50
3.3	Detecția de Contur	51
3.3.1	Detecția de contur cu filtre liniare	51
3.3.2	Detecția de contur cu filtre neliniare	53
3.3.3	Transformata Hough.....	55
3.4	Segmentarea în spațiul de culoare.....	57
3.4.1	Spațiul de culoare RGB	58
3.4.2	Spațiul de culoare HSV	58
Capitolul 4	Navigarea Autonomă	59
4.1	Introducere	59
4.2	Reprezentări continue	60
4.2.1	Sistem de coordonate local	60
4.2.2	Odometrie	61
4.2.3	Maparea mediului.	65
Capitolul 5	Dezvoltarea și Evoluția Aplicației	68
5.1	Caracteristici Generale.....	68
5.2	Conectarea la platforma robotică Jaguar 4x4.....	69
5.2.1	Conexiunea folosind Sockets.....	71
5.3	Controlul Motoarelor	73
5.4	Scanner-ul Laser	74
5.5	Stream-ul Video	77
5.6	Implementarea funcțiilor OpenCV.....	79
5.6.1	Detecția de cercuri	79
5.6.2	Segmentarea în spațiul de culoare	82
5.7	Deplasarea robotului și harta de ocupare	86
5.7.1	Deplasarea robotului	86
5.7.2	Harta de ocupare	88
5.8	Utilizarea brațului robotic	89
5.9	Modul de funcționare a aplicației finale	95
5.10	Teste și rezultate.....	99
5.10.1	Timpul de poziționare.....	99

5.10.2	Calitatea segmentării.....	100
Concluzii	103	
Referințe	104	
Anexa 1	106	
	Clasa JaguarCtrl.....	106
	Clasa LaserScan.cs.....	171
	Clasa DrRobotMainComm	181

LISTĂ FIGURI

Figură 1 Robotul Jaguar a)	20
Figură 2 Robotul Jaguar b)	20
Figură 3 Diagrama de interconectare.....	22
Figură 4 Motor cu encoder.	28
Figură 5 Motor pentru operații în modul poziție	28
Figură 6 ieșirea unui encoder în cuadratură.	29
Figură 7 Construcția tipică a unui encoder în cuadratură.	30
Figură 8 Conectarea controller-ului la encoder.	30
Figură 9 Aria de detecție a Scanner-ului Laser.	31
Figură 10 Schema bloc de funcționare a sistemului.	33
Figură 11 Arhitectura Emgu CV.....	41
Figură 12 Fereastră de logare	43
Figură 13 Mesaj	43
Figură 14 Interfața aplicației Jaguar Control	44
Figură 15 Controller-ul Gamepad a).....	45
Figură 16 Controller-ul Gamepad b)	45
Figură 17 Informațiile de la senzori.....	45
Figură 18 Structura claselor.....	46
Figură 19 Eșantionarea unei imagini continue.	49

Figură 20 Operații pe imagini.....	50
Figură 21 Filtrul Prewitt.....	53
Figură 22 Filtrul Sobel.....	54
Figură 23 Suprimarea non-maximelor.....	55
Figură 24 Căutare cu rază cunoscută.....	57
Figură 25 Cubul RGB.....	58
Figură 26 Conul HSV.....	58
Figură 27 Propagarea erorii în odometrie.....	65
Figură 28 Hartă de ocupare a unei încăperi.....	66
Figură 29 a) Probabilitatea de ocupare pentru un senzor ideal.....	66
Figură 30 b) Ocuparea hărții pentru un senzor ideal.....	67
Figură 31 ID Wi-Fi al robotului.....	70
Figură 32 Fereastră de conectare la platformă.....	70
Figură 33 Aplicația simplă de control al motoarelor.....	74
Figură 34 Reprezentarea grafică a vectorului de valori pentru Scanner-ul Laser.....	76
Figură 35 Aplicația de control cu integrarea scanner-ului Laser.....	77
Figură 36 Stream video accesat din browser.....	78
Figură 37 Aplicația de control actualizată.....	79
Figură 38 Detecția de cercuri folosind transformata Hough.....	81
Figură 39 Aplicația de detecție a cercurilor și control al robotului.....	82
Figură 40 Segmentarea în spațiul culorilor.....	83
Figură 41 Detectarea unui obiect de o anumită culoare.....	85
Figură 42 Modul de deplasare al robotului.....	87
Figură 43 Implementarea hărții de ocupare a).....	89
Figură 44 Implementarea hărții de ocupare b).....	89
Figură 45 Mișcarea încheieturilor brațului robotic.....	90
Figură 46 Dispunerea în interfață a butoanelor de control al brațului robotic.....	91
Figură 47 Diagrama funcționare a aplicației.....	95
Figură 48 Fereastra de conectare.....	96
Figură 49 Interfața grafică la start-up.....	96
Figură 50 Tab-ul Video Recognition.....	97
Figură 51 Tab-ul Image Processing.....	97
Figură 52 Tab-ul Sensors & Data.....	97
Figură 53 Aplicația în modul Autonom- Detecția obiectului.....	98
Figură 54 Aplicația în modul Autonom- Faza de poziționare și verificare.....	98
Figură 55 Aplicația în modul Autonom- Prinderea obiectului.....	99

Figură 56 Timpul de poziționare.	100
Figură 57 Imaginea de control.	101
Figură 58 Imagine de test a) Figură 59 Imagine de test b)	101
Figură 60 Imagine de test c) Figură 61 Imagine de test d)	101

LISTĂ TABELE

Tabel 1 Componente hardware	21
Tabel 2 Driver-ele de motoare	23
Tabel 3 Mesaje IMU	25
Tabel 4 Comenzi Motor	27
Tabel 5 Platforma Windows	40
Tabel 6 Precizia și reamintirea.....	102

CAPITOLUL 1

INTRODUCERE

1.1 ABSTRACT

Navigarea este știința direcționării unui mobil determinându-i poziția, cursul și distanța parcursă. Navigarea se ocupă cu găsirea drumului până la destinație, evitând coliziunile, conservând energie și respectând un anumit program. Pentru un robot de navigare, scopul este ajungerea la destinație, însă un robot nu va îndeplini doar acest aspect. Astfel, pentru a rezolva diferite probleme, robotul trebuie să poată face și altceva, spre exemplu, un robot de livrare ar trebui să poată livra diferite obiecte, un robot de supraveghere trebuie să analizeze suficient de bine date video, etc.

Navigarea este abilitatea ce permite robotului să ajungă la o anumită destinație.

Obiectivul robotului (scopul) se evidențiază după etapa de navigare.

Această lucrare își propune să studieze aceste două aspecte foarte importante ale roboticii moderne care sunt de asemenea foarte dependente una de cealaltă. Vor fi prezentate metode personale de abordare ale acestora, noțiuni teoretice, soluții de implementare și diferite experimente.

1.2 MOTIVAȚIA LUCRĂRII

Utilitatea roboților capabili de navigare autonomă, atât în medii cunoscute cât și în medii necunoscute, face ca cercetarea în acest domeniu să devină un obiectiv important. După cum se poate observa, în jurul nostru, există deja astfel de roboți autonomi care ne ajută cu treburile zilnice, cum ar fi aspiratoare autonome sau mașini de tuns iarba. Chiar și transportul oamenilor se va face în curând folosind automobile autonome. Deși potențialul este cunoscut, nu există suficient de multă cercetare realizată în acest domeniu. Inginerii și oamenii de știință folosesc diferite tipuri de echipamente pentru a realiza măsurători și a obține informații din diferite medii. Există însă multe situații în care acest lucru nu ar fi posibil sau nu ar fi deloc practic, spre exemplu în zone potențial ostile, sau zone fără posibilitate de acces uman. Pentru aceste situații, utilizarea unor metode de navigare autonomă va permite robotului să exploreze un mediu independent de prezența umană. Robotul va utiliza echipamentele de măsurare, va aduna toate datele și fie le va transmite unui utilizator, fie le va aduce la acesta. Astfel de roboți, capabili de a naviga autonom și de a îndeplini misiuni fără intervenție umană, vor putea salva vieți sau ajuta oameni în diferite situații. Datorită evoluției tehnologiei din ultimul timp, avem acces la o gamă foarte largă de senzori, capabili de performanțe ridicate, avem acces la metode de programare inovative, foarte multe platforme disponibile, dar și acces la foarte multă informație. Astfel, realizarea unei platforme robotice capabile de a îndeplini o sarcină în mod autonom este un pas normal în evoluția curentă.

Robotica nu este interesantă doar din punct de vedere al aplicabilității dar și din punct de vedere al domeniilor pe care le înglobează, domenii cum ar fi, prelucrări de semnale, arhitecturi de microprocesoare, microcontrolere, programare, senzori, rețele de calculatoare, imagistică digitală, etc. Toate aceste cunoștințe legate de aceste domenii, dobândite în cursul facultății, pot fi folosite și aplicate armonios realizând un proiect în domeniul roboticii.

1.3 ROBOȚI AUTONOMI

În principal există trei tipuri de roboți: operați, automați și autonomi. Roboții operați sunt cei care necesită control uman (spre exemplu drone comandate prin radio). Roboții automați realizează activități preprogramate și repetitive în medii controlate (spre exemplu roboții industriali folosiți pentru diferite operații cum ar fi sortarea unor produse). Roboții autonomi îndeplinesc un obiectiv într-un mediu natural/ nestructurat și iau propriile decizii spre îndeplinirea acestui obiectiv (spre exemplu roboți de curierat din spitale sau mașini autonome). Tendința este de a crea roboți cu autonomie cât mai ridicată, pentru ca intervențiile umane să se minimizeze.

Dacă dorim ca în viitor acești roboți, nu numai să asiste la activități precum construcția unor clădiri, sau salvarea unor oameni în cazul unor dezastre naturale, ci să le realizeze singuri, este foarte important ca aceștia să fie capabili să navigheze în medii necunoscute și să perceapă cât mai bine informațiile din exterior.

1.4 OBIECTIVUL LUCRĂRII

Lucrarea dorește să prezinte principalele probleme studiate și cercetate în prezent în domeniul roboticii, și anume problema navigării autonome într-un mediu necunoscut și îndeplinirea unui anumit obiectiv. În cazul de față obiectivul (sau ținta) robotului va fi găsirea unei sfere colorate într-un mediu necunoscut.

Pentru realizarea acestui proiect am folosit platforma robotică mobilă Jaguar 4x4 și aplicația demonstrativă care a însoțit acest robot. O parte foarte importantă a proiectului a constat în studierea acestei platforme, atât a componentelor Hardware cât și Software. Următoarea parte a

constat în deciderea asupra misiunii robotului și studierea posibilităților de implementare a soluțiilor găsite. Deciderea asupra utilizării librăriei de procesare și analiză de imagini, Emgu CV (Wrapper C# pentru Open CV) a făcut posibilă realizarea unor proiecte inițiale ce au ajutat la dezvoltarea acestei lucrări. Navigarea autonomă fiind un domeniu foarte complex a fost abordată simplist inițial urmând ca în final să realizez o cercetare mult mai amănunțită a soluțiilor existente și să decid asupra unor variante inițiale de implementare, ținând cont atât de complexitatea de calcul cât și de eficiență.

CAPITOLUL 2

PLATFORMA ROBOTICĂ JAGUAR

2.1 PREZENTARE

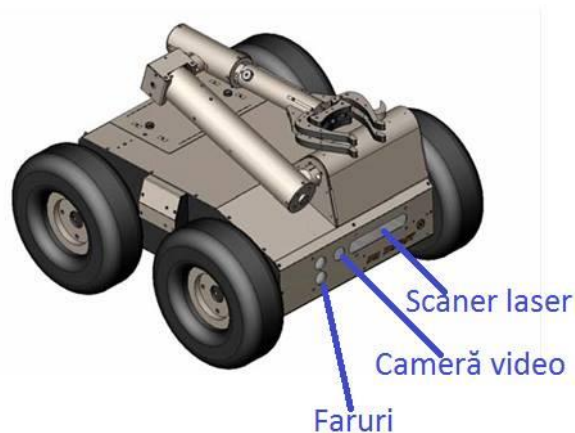
Platforma robotică mobilă Jaguar 4x4 este realizată atât pentru operațiuni de interior, cât și de exterior. Aceasta folosește, pentru mișcare, 4 motoare de 80W, are o greutate de aproximativ 20 de kilograme, atinge o viteză maximă de 11Km/oră, este rezistent la apă și modificări atmosferice. Este proiectat pentru terenuri accidentate fiind capabil să urce o treaptă de până la 110 mm. Robotul este controlat prin conexiune wireless (802.11GP), integrează GPS – pentru exterior, senzor IMU (Inertial Measurement Unit) cu 9 grade de libertate (Giroscop/ Accelerometru/ Busola), Scanner Laser cu raza de acoperire de 30 de metri cu un unghi de aproximativ 240 de grade, pentru navigația autonomă. Camerele integrate (video/audio) și scannerul Laser permit accesarea informațiilor referitoare la mediul înconjurător [1].

Principalele caracteristici ale platformei [1]:

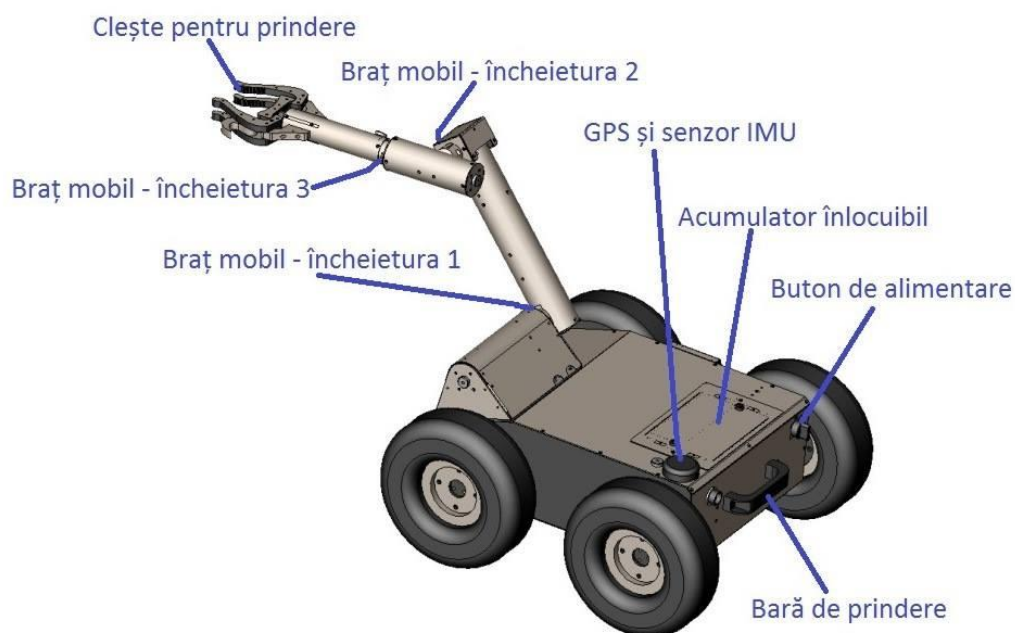
- Platformă mobilă utilă atât pentru aplicații de exterior cât și pentru interior, manevrabilitate ridicată.
- Rezistent la schimbări atmosferice și apă.
- Poate urca rampe de până la 45 de grade sau trepte de maxim 110mm
- Ușor și compact – 20 Kg

- Navigație autonomă cu GPS și IMU
- Rezistă la o cădere de la 1200mm pe beton.
- Scanner Laser integrat
- Camere video cu rezoluție ridicată
- Conexiune Wireless 802.11G
- Aplicație demo –Ready to Use-
- Kit de dezvoltare, protocol de date și coduri sursă, suportând Microsoft® Robotics Studio, Microsoft® Visual Studio, NI LabVIEW®, MATLAB®, Java®

Prezentarea așezării principalelor componente pe platformă:



Figură 1 Robotul Jaguar a)



Figură 2 Robotul Jaguar b)

2.2 COMPONENTE HARDWARE

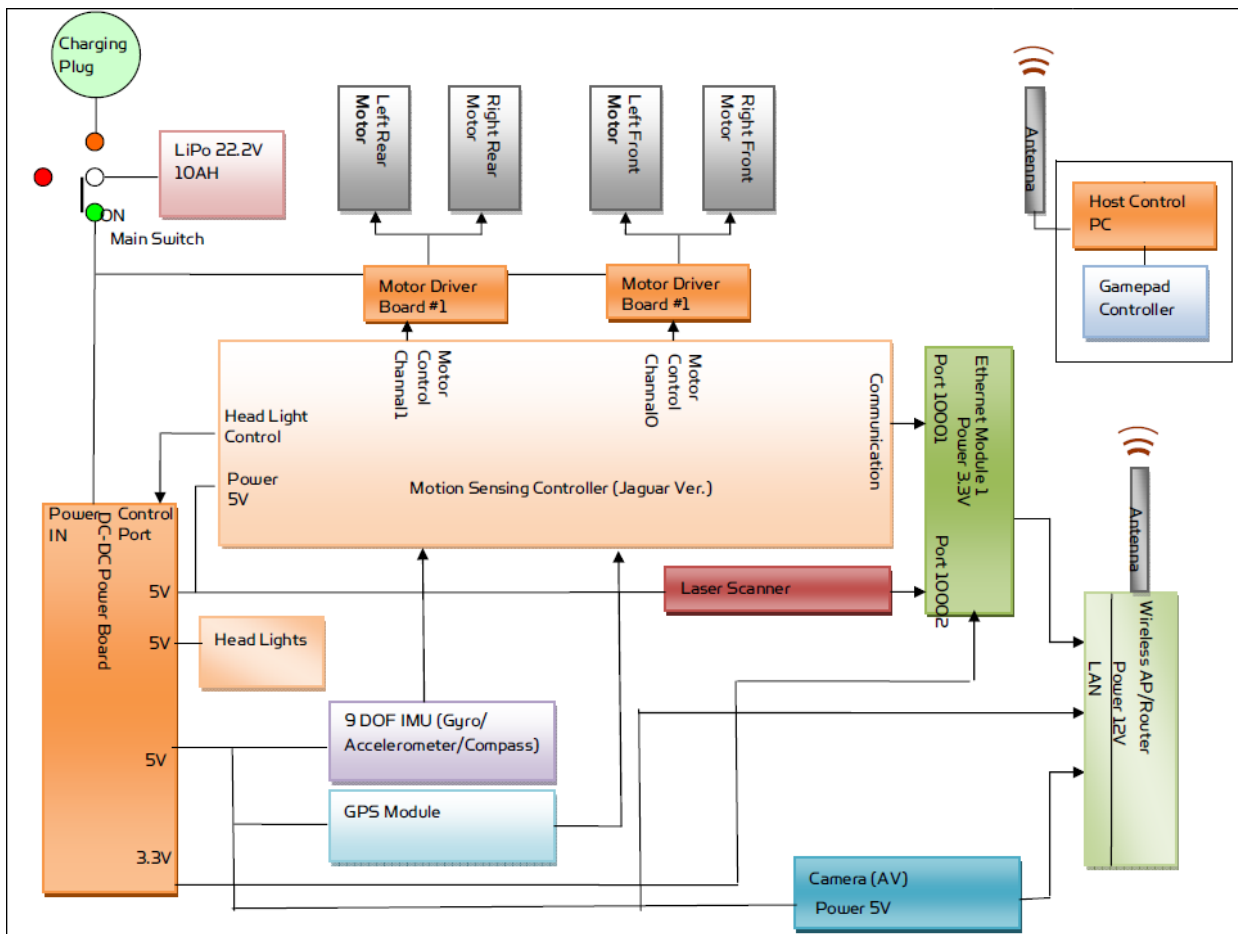
Platforma robotică Jaguar 4x4 pune la dispoziție o gamă largă de componente hardware care ajută la crearea mai multor tipuri de aplicații. Structura hardware a robotului conectează multe tipuri de senzori (Scanner Laser, Senzor IMU, encoder, etc.) , actuatori și module de rețea pentru a facilita capacitățile de comunicare.

În tabelul următor sunt menționate principalele componente hardware ale robotului Jaguar 4x4 [1].

Tabel 1 Componente hardware

JAGUAR 4x4W-ME	Carcasă Jaguar 4x4 Wheel (include motoarele și encoder-ele)	1
JAGUAR-ARM	Braț mobil Jaguar	1
PMS5006	Controller de mișcare și senzori	1
WFS802	Modul de rețea	2
DMD1202	Driver de motoare DC cu canal dual 10A	4
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 22.2 V	1
LPBC5000	Încărcător LiPo de 2A	2
GPC0010	Gamepad Controller	1

Diagrama următoare ilustrează inter-conexiunea între modulele și componentele de bază ale platformei robotice mobile 4x4 [1].



Figură 3 Diagrama de interconectare

Se observă în diagramă modulele de funcționare și componentele principale ale robotului: Modulul Ethernet cu cele două porturi la care se conectează controller-ul de mișcare, respectiv scanner-ul laser. Modulul Ethernet se conectează la router-ul wireless care deține o antenă pentru o arie mai mare de acoperire.

Controller-ul de mișcare și senzori conține două canale (canalul 0 și canalul 1) la care sunt conectate cele două drive-uri de motoare ce controlează motoarele astfel: primul controlează motoarele stânga-spate și dreapta-spate, iar celălalt driver controlează motoarele stânga-față și dreapta-față. Acest controller mai conține și un modul de control al farurilor, care este conectat la portul de control al plăcii de alimentare.

Placa de alimentare primește la intrare 22.2V de la baterie, când comutatorul se află pe poziția "ON" și oferă la ieșire o tensiune de 5V de la care se alimentează controller-ul, scanner-ul laser, farurile, camera, router-ul wireless și, de asemenea, senzorul IMU și GPS-ul care sunt conectate la controller și o tensiune de 3.3V care alimentează modulul Ethernet.

Informațiile se transmit wireless către calculatorul gazdă.

2.2.1 Controller-ul PMS5006

Controller-ul PMS5006 va funcționa ca centru principal de control al robotului Jaguar. Va comunica cu controller-ul master printr-un port serial (port 1 – UART1). Pe sistemul robotic Jaguar, portul serial se va conecta la port 1 al modulului de rețea –module-1. De la computer-ul gazdă, se poate stabili un socket TCP pentru a comunica cu acest controller, folosind IP-ul : 192.168.0.XX și portul 10001. (XX- variază în funcție de platforma robotică, în cazul meu am avut XX = 70) .

Controller-ul PMS5006 poate controla până la 4 Drive de motoare SDC2130 de la RoboteQ prin UART 2, 3, 4, 5. Poate citi curentul de la motoare, puterea de ieșire, tensiunea, temperatura motoarelor, encoder-ele motoarelor și statusul driver-ului. Driver-ul motorului suportă control în buclă deschisă, encodere în modul viteză și buclă închisă cât și encodere în modul poziție și buclă închisă [2].

Inițial, pentru robotul Jaguar 4x4, Driver-ul de motoare-1 va comanda motoarele stânga și dreapta față și Driver-ul de motoare-2 va comanda motoarele stânga și dreapta spate.

Tabel 2 Driver-ele de motoare

Driver-ul de motoare 1	Canalul 0	Motorul stânga față	Encoder 1 și analog input 3 ca senzor de temperatură
	Canalul 1	Motorul dreapta față	Encoder 2 și analog input 4 ca senzor de temperatură
Driver-ul de motoare 2	Canalul 0	Motorul stânga spate	Encoder 1 și analog input 3 ca senzor de temperatură
	Canalul 1	Motorul dreapta spate	Encoder 2 și analog input 4 ca senzor de temperatură

Controller-ul PMS5006 va citi toate datele de la senzorul IMU cu 9 grade de libertate prin portul I2C

Firmware-ul de pe placă va estima orientarea robotului folosind un algoritm DCM și datele de la senzorul IMU și GPS.

În același timp, va citi toate mesajele de la senzorul GPS prin UART-7.

2.2.2 Mesajele cu date de la senzori

Controller-ul PMS va trimite toate datele de la senzori. Există 4 tipuri principale de mesaje cu date de la senzori și toate aceste mesaje de date și vor termina cu "CRLF". (\r\n 0x0d0x0a).

2.2.2.1 Mesaje GPS

Controller-ul PMS5006 va transmite mesajele senzorului GPS către controller-ul master. Pentru a reduce volumul de date setăm GPS-ul să transmită doar propoziții GPRMC.

Orice pachet de date care va începe cu "\$GPRMC", va fi asociat unui mesaj transmis de senzorul GPS.

Formatul datelor GPRMC este prezentat mai jos (exemplu de mesaj transmis de senzorul GPS):

\$GPRMC,220516,A,5133.82,N,00042.24,W,173.8,231.8,130694,004.2,W*70

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

1	220516	Amprenta temporală
2	A	Validitatea: A-Valid, V-Invalid
3	5133.82	Latitudine curentă
4	N	Nord/Sud
5	00042.24	Longitudine curentă
6	W	Est/Vest
7	173.8	Viteză în Nod-uri
8	231.8	Cursul real
9	130694	Data
10	004.2	Variația
11	W	Est/Vest
12	*70	checksum

Rata de actualizare a acestui mesaj este de 5Hz (Modul GPS Garmin GPS 18X-5Hz)

2.2.2.2 Mesajele de date IMU

Controller-ul PMS5006 [2] va transmite toate datele de la modulul senzorial IMU și valoarea estimată a orientării. Orice pachet de date care va începe cu "#" va fi asociat unui mesaj de date IMU. Formatul unui mesaj de tip IMU este după cum urmează:

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

xx,xxx,xxx

După "#", fiecare șir este delimitat de ",".

Tabel 3 Mesaje IMU

1 Numărul de secvență al pachetului de date	2 Yaw -fixat
3 Valoarea orientării estimată in Rad	4 Gyro - fixat
5 Gyro X valoare citită	6 Gyro Y valoare citită
7 Gyro Z valoare citită	8 Accel -fixat
9 Accelerometru X valoare citită	10 Accelerometru Y valoare citită
11 Accelerometru Z valoare citită	12 Comp - fixat
13 Busolă digitală X valoare citită	14 Busolă digitală Y valoare citită
15 Busolă digitală Z valoare citită	16 ADC (nefolosit)

Mesajul este updatat cu o rată de aproximativ 50Hz.

2.2.2.3 Mesaje de la senzorii si Driver-ele motoarelor

Controller-ul PMS5006 colectează datele de la driver-ul de motoare și le transmite controller-ului Master [2]s. Din moment ce PMS5006 poate controla până la 4 Driver-e de motoare SDC2130, mesajele vor începe cu:

- "MM0" – înseamnă că datele sunt primite de la Driver-ul de motoare 1
- "MM1" – înseamnă că datele sunt primite de la Driver-ul de motoare 2
- "MM2" – înseamnă că datele sunt primite de la Driver-ul de motoare 2
- "MM3" – înseamnă că datele sunt primite de la Driver-ul de motoare 2

Controller-ul PMS5006 va trimite comenzi către Driver-ul de motoare pentru a accesa datele de la senzorii motoarelor.

Spre exemplu, daca citim datele:

"MM0 V=125,246,4750" (a se observa spațiul dintre "MM0" și mesajul real.)

Decodând, tensiunea pentru Driver-ul de motoare 1 este de 12.5V , tensiunea totală este 24.6 V și ieșirea regulatorului de 5V de pe placă este de 4.750V.

PMS5006 va trimite 10 comenzi in buclă pentru fiecare Driver de motoare la o frecvență de 50Hz, deci rata de reîmprospătare pentru fiecare dată de la senzor este de aproximativ 5Hz.

Formatul a 10 interogări pentru datele de la senzori arată in felul următor:

"?A", "?AI", "?C", "?D", "?FF", "?p", "?S", "?T", "?V", "?CR".

Canalul de intrare analog 3,4 va fi folosit pentru senzorul de temperatură al motoarelor.

2.2.2.4 Comenzi de control

Toate comenzile de sistem sau de control al motoarelor se termină cu “CRLF” (\r\n 0x0d0a) [2].

- Comanda System “PING”

Formatul: PING

Funcționează ca un pachet de tip heart beat, controller-ul Master trebuie sa trimită acest mesaj către PMS5006 la o rată de aproximativ 4Hz, altfel PMS5006 va opri toate motoarele –consideră că a pierdut comunicarea cu controller-ul master.

- Comanda de control GPIO

Formatul: SYS GPO n

Unde ‘n’ este un număr pe 8 biți. Această comandă va controla un port IO. Este rezervat pe robotul Jaguar.

- Comanda de control de putere

Formatul: SYS MMC n

Unde ‘n’ este un număr pe 8 biți. Această comandă va controla un canal de putere de pe controller-ul PMS5006. Doar bit-ul 7 este disponibil pentru robotul Jaguar, bit7 = 1 va aprinde luminile, bit7 = 0 va stinge luminile. Restul biților sunt rezervați.

- Comanda de control pentru senzorul IMU

Formatul: SYS CAL

Această comandă va face ca PMS5006 să recalculeze offset-ul senzorului Gyro. Dacă se găsește o valoare a Gyro mai mare ca 10 când robotul stă nemișcat, trebuie să trimitem această comandă către PMS5006.

- Comanda de control GPS

Formatul: EGPS n

Această comandă îi va spune controller-ului PMS5006 dacă să folosească GPS sau nu. Pentru n=1 Enable GPS, pentru n=0 Disable GPS în algoritmul DCM.

- Setare valoare inițială DCM

Format: DCM n

Această comandă va seta valoarea inițială pentru DCM. “n” este valoarea direcției setată între -180~+180.

- Comenzi de control pentru motoare

Din moment ce PMS5006 poate controla până la 4 Drive de motoare SDC2130, există 5 tipuri de comenzi.

Tabel 4 Comenzi Motor

MM0 (comandă motor)	PMS5006 va trimite o comandă către driver-ul de motoare 1
MM1 (comandă motor)	PMS5006 va trimite o comandă către driver-ul de motoare 2
MM2 (comandă motor)	PMS5006 va trimite o comandă către driver-ul de motoare 3 (nu este folosit)
MM3 (comandă motor)	PMS5006 va trimite o comandă către driver-ul de motoare 4 (nu este folosit)
MMW (comandă motor)	PMS5006 va trimite o comandă către driver-ele 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 toate comenzile de control și configurare listate în manualul SDC2130, spre exemplu:

"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 buclă deschisă. Dacă este nevoie de control de viteză sau poziție trebuie trimise comenzi de schimbare a modurilor control către driver-ul de motoare [2]s.

2.2.3 Modurile de control ale motoarelor

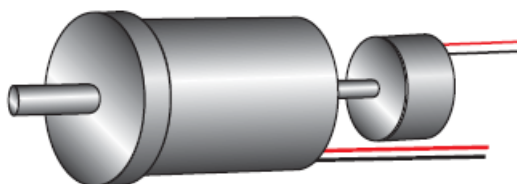
Pentru fiecare motor, controller-ul suportă mai multe moduri de control. În configurația inițială a controller-ului este setat controlul în buclă deschisă pentru fiecare motor (mai puțin motoarele brațului care sunt setate în modul de control al poziției). Modul se poate schimba folosind "Roborun PC utility" [3].

- Modul de control în buclă deschisă

În acest mod, controller-ul 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 [3].

- Modul de control în buclă închisă

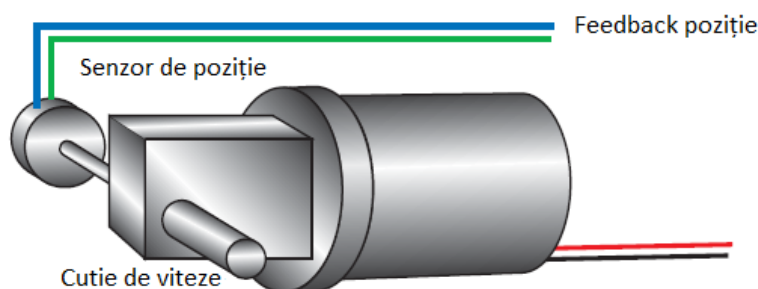
În acest mod, ilustrat în figura de mai jos, un encoder optic este folosit pentru a măsura viteza actuală a motorului. Dacă viteza se schimbă datorită unei schimbări în sarcină, controller-ul va compensa automat puterea de ieșire. Acest mod este de preferat în modul de control de precizie sau în aplicații pentru roboți autonomi [3].



Figură 4 Motor cu encoder.

- 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 decelerează pentru a ajunge ușor la destinația dorită. Acest mod face posibilă crearea unui "Jumbo Servo" care poate fi folosit pentru a mișca un braț robotic [3].



Figură 5 Motor pentru operații în modul poziție

- Modul de control în buclă închisă cu numărătoarea poziției

În acest mod, un encoder este atașat la motor ca în figura de mai sus. Apoi, controller-ul poate fi comandat să miște motorul la o valoare specifică de numărători, folosind accelerația, viteza și decelerația definite de utilizator [3].

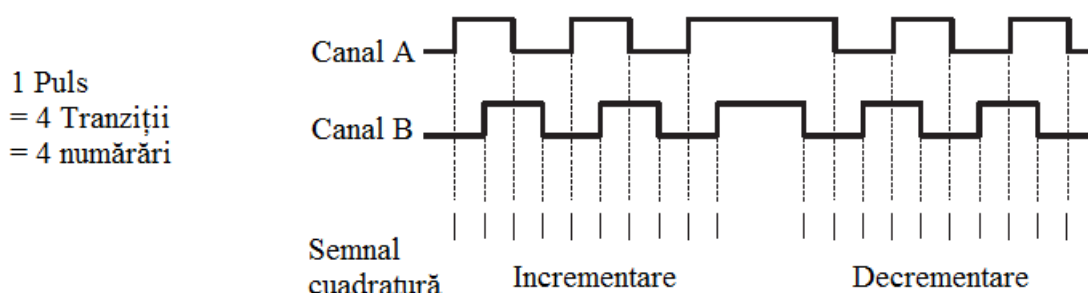
- Modul de tracțiune

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].

2.2.4 Encoder-ul Optic cu incrementare

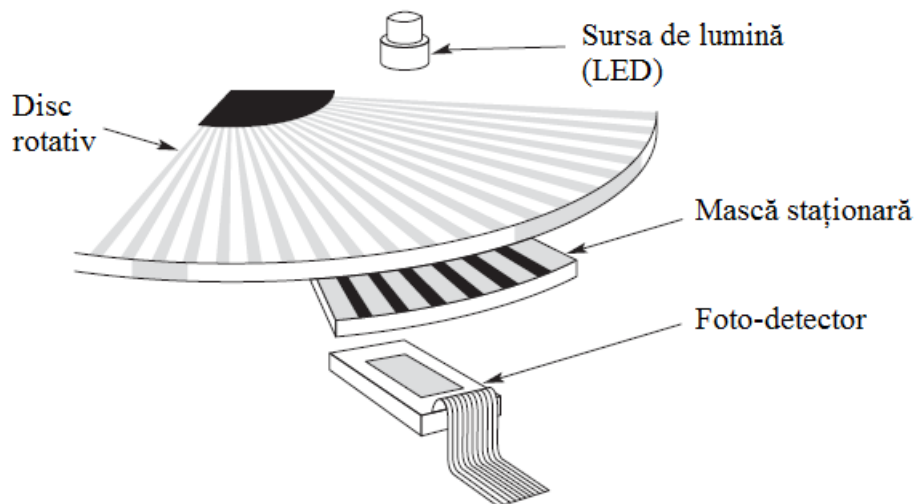
Encoder-ele optice cu incrementare sunt module care ajută la măsurarea vitezei și a distanței parcurse pentru un motor. Spre deosebire de encoder-ele absolute care returnează un număr pe mai mulți biți (depinzând de rezoluție), encoder-ele cu incrementare trimit pulsuri când se rotesc. Numărând aceste pulsuri se poate transmite câte revoluții au avut loc la mișcarea motoarelor. Viteza rotației poate fi determinată din intervalul de timp scurs între două pulsuri sau după numărul de pulsuri dintr-o perioadă de timp. Datorită faptului că sunt dispozitive digitale, encoder-ele cu incrementare vor măsura distanța și viteza cu acuratețe foarte mare [3].

Din moment ce motoarele se pot mișca atât în față cât și în spate, este necesar să diferențiem modul în care pulsurile sunt numărate astfel ca ele vor incrementa sau decremента un numărator din aplicație. Encoder-ele în cuadratură au două canale, A și B, care sunt defazate electric cu 90 de grade. Astfel direcția de rotație poate fi determinată prin monitorizarea relației de fază dintre cele două canale. În plus, cu un encoder cu două canale, o multiplicare de patru ori a rezoluției este realizată numărând fronturile crescătoare și descrescătoare a fiecărui canal (A și B). Spre exemplu, un encoder care produce 250 de pulsuri per revoluție (PPR) poate genera 1000 de numărări pe revoluție (Count Per Revolution- CPR) după cuadratură.



Figură 6 Ieșirea unui encoder în cuadratură.

Figura de mai jos arată modul tipic de construcție a unui encoder în cuadratură. Cât timp discul se rotește în fața unei măști staționare, blochează lumina de la un LED. Lumina care trece prin mască este receptată de un foto-detector. Două foto-detectoare sunt plasate unul lângă celălalt astfel că lumina ce trece prin mască să lovească un detector, apoi pe celălalt pentru a produce pulsurile defazate cu 90 de grade.

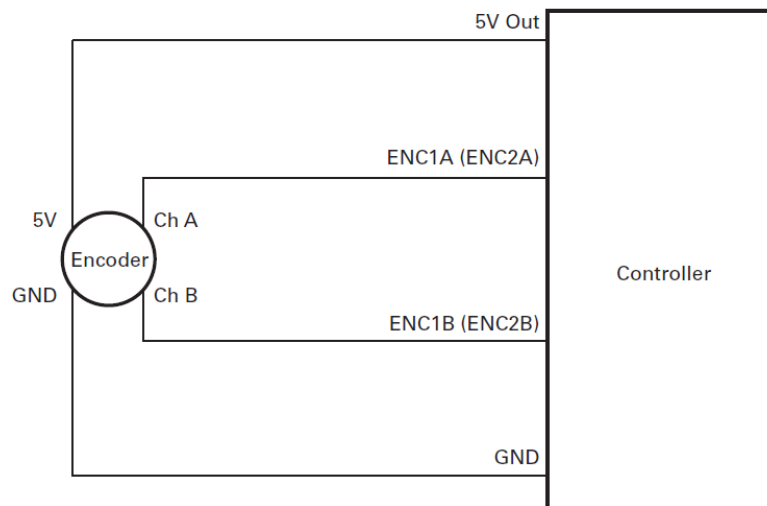


Figură 7 Construcția tipică a unui encoder în cuadratură.

Când sunt folosite în aplicații de poziționare, controller-ul trebuie să miște motorul până când o limită este atinsă. Această poziție este apoi folosită ca poziția 0 de referință pentru toate mișcările următoare [3].

Encoder-ele folosite îndeplinesc următoarele condiții:

- Două ieșiri în cuadratură (Canalul A, Canalul B).
- 3.0V minim între Nivelul de 0 și Nivelul de 1 la ieșirea în cuadratură.
- 5V DC. Consum de 50mA sau mai puțin pentru fiecare encoder.



Figură 8 Conectarea controller-ului la encoder.

2.2.5 Senzori externi

Senzorii externi sunt senzori care ajută la obținerea informațiilor din mediul care înconjoară robotul (spre exemplu: distanța până la un obstacol).

Robotul Jaguar 4x4 este de asemenea dotat cu astfel de senzori:

- Două camere video IP – Axis IP Camera
- Scanner Laser de distanță – Hokuyo Laser Range Finder UTM-30LX

2.2.5.1 Camerele video

Platforma robotică este dotată cu două camere video, una din ele este amplasată la baza robotului, iar cealaltă este amplasată pe brațul robotic [2].

Camerele sunt de tip Axis IP, ceea ce înseamnă că stream-ul video poate fi accesat via IP.

Acestea sunt de asemenea și protejate cu username și parolă.

IP-ul camerelor este:

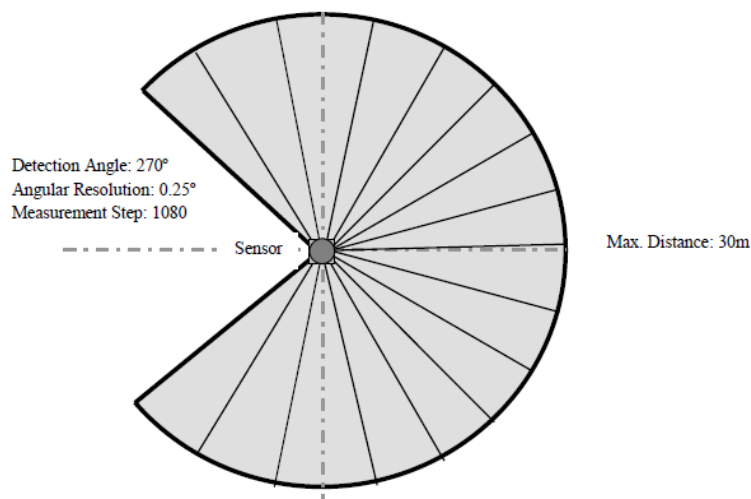
Pentru camera de pe bază: 192.168.0.75 cu portul 8081 ,ID: root, pass: drrobot

Pentru camera de pe braț: 192.168.0.74 cu portul 8082, ID: root, pass: drrobot

Camerele au o rezoluție de 640x480, un maxim de 30 de frame-uri pe secundă și un canal audio bidirecțional.

2.2.5.2 Scanner-ul Laser de distanță

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



Figură 9 Aria de detecție a Scanner-ului Laser.

În proiectul de față acest senzor a fost folosit pentru determinarea distanțelor până la un obstacol și realizarea unei hărți de ocupare.

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

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

Un mesaj de cerere de la computer-ul gazdă include include un cod de comandă. Răspunsul și SRM-ul 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 Statusul, Timpul și Distanța.
- GE – realizează funcția de achiziție a distanței și a intensității, va returna Statusul, Timpul, Distanța și Intensitatea.

Exemplu de comenzi fără măsurare:

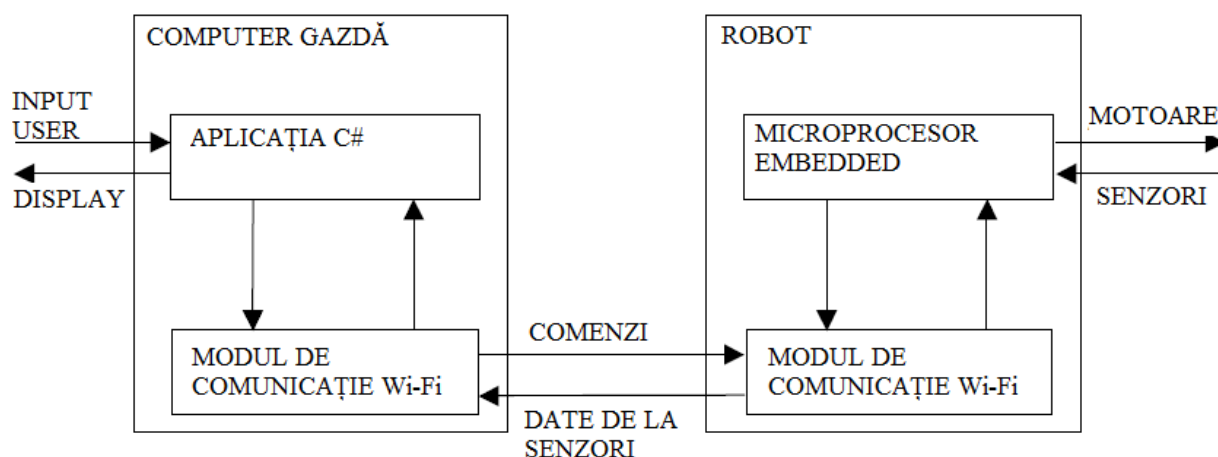
- BM – comandă de tipul State Transition (tranziție de stare) , realizează tranziția în modul de măsurare (nu se pot realiza măsurători fără tranziția în această stare). Va trimite către computerul gazdă un mesaj cu status-ul dispozitivului.
- RS – comandă de resetare, realizează resetarea senzorului.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

Codurile de comandă prezentate sunt printre cele mai utilizate, cu ajutorul acestora putem realiza o rutină de execuție pentru obținerea măsurătorilor de distanță [4].

2.3 COMPONENTE SOFTWARE

Aplicația de control a robotului rulează pe un computer gazdă, trimite comenzi via Wi-Fi către controller-ul 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 aibe la bază același limbaj si mediu de programare ca și aplicația demonstrativă.

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



Figură 10 Schema bloc de funcționare a sistemului.

Pe computerul gazdă rulează o aplicație C# ce va transmite semnale de comandă către microprocesorul embedded al robotului folosind o conexiune wireless (între computer și router-ul wireless al robotului). Folosind aceste semnale, microprocesorul va acționa motoarele. De asemenea microprocesorul primește date de la senzori și le transmite mai departe către aplicația C# (prin aceeași conexiune wireless). Aplicația va decoda mesajele primite și va folosi datele de la senzori fie pentru afișarea lor, fie pentru luarea deciziilor ce influențează acționarea actuatorilor.

În subcapitolele următoare voi prezenta diferitele tehnologii utilizate în dezvoltarea acestei aplicații, diferitele metode de implementare cât și rezultatele obținute.

2.3.1. Framework-ul .NET

Deși Framework-ul .NET se bazează pe limbaje de programare și medii de dezvoltare care au fost create cu mai mulți ani în urmă, acesta este mai recent, iar succesul său pe piață va crește cu trecerea timpului. Internetul a redefinit modul în care programatorii gândesc interfețele aplicațiilor și dezvoltarea lor. Cu succesul Internet-ului, un produs sau o interfață va apărea direct într-un browser web decât într-o aplicație tradițională Windows. Limbajul de programare Java a avut mult succes în medii de tip UNIX sau server web, și este limbajul ales pentru multe aplicații web în prezent [5].

Astfel Microsoft a încercat să revoluționeze modul în care gândim despre dezvoltarea unei aplicații. Rezultatul fiind Framework-ul .NET și limbajul de programare C#.

Limbajul C# a fost reconstruit complet cu ideea de a păstra flexibilitatea limbajelor C și C++. Multe erori comune de tip runtime în C++ sunt erori de compilare în C#.

Alte caracteristici includ tipul de date string ca fiind built-in, lipsa variabilelor globale, și integrarea erorilor critice de sistem și de aplicație într-un model comun de excepții.

Framework-ul .NET este o interfață de programare și un mediu de execuție pentru sisteme de operare Windows, și o mare parte din framework-ul propriu-zis a fost scris în C#.

Framework-ul .NET include aproape toate tehnologiile și mediile de dezvoltare care au evoluat în timp, de la COM la XML și ASP la Visual Studio. Aceste tehnologii sunt recreate și reinventate sub un singur framework. Deși compatibilitatea inversă nu s-a pierdut complet, framework-ul .NET redefineste clasele și metodele pentru aceste tehnologii și produsele care le utilizează. În particular, framework-ul include un nou suport pentru dezvoltarea aplicațiilor

Windows, acces la site-uri web, comunicații remote pentru programe, interacțiuni cu baze de date, securitate și alte tehnologii.

2.3.5 Windows Forms

După cum am menționat mai devreme, toate obiectele din .NET Framework sunt scrise în C# și sunt organizate în *namespace-uri*. În dezvoltarea aplicațiilor de tip windows se va folosi namespace-ul `System.Windows.Forms`.

Înainte de a discuta despre clase specifice, este foarte important să ne familiarizăm cu trei termeni importanți în framework-ul .NET dar și în namespace-ul Windows Forms de asemenea: *componente, containere și control* [5].

O *componentă* este un obiect care permite distribuirea între aplicații. Clasa `Component` încapsulează această noțiune, și este baza pentru majoritatea membrilor din namespace-ul Windows Forms.

Un *container* este un obiect care poate ține una sau mai multe componente. Un container este pur și simplu un mecanism de grupare, și asigură că seturi de componente sunt încapsulate și manipulate în moduri similare. Containerele sunt folosite în namespace-ul Windows Forms de fiecare dată când este nevoie de un grup de obiecte. Clasa `Container` încapsulează conceptul de container.

Un *control* este o componentă cu aspect vizual. În namespace-ul Windows Forms, un control este o componentă care prezintă o interfață grafică pe Windows desktop. Clasa `Control` stă la baza aplicațiilor Windows Forms. Este de menționat și faptul că namespace-ul `System.Web.UI` definește o clasă `Control` pentru reprezentarea grafică a obiectelor ce apar pe o pagină Web.

În mod general, orice interfață grafică vedem pe un Windows desktop este un control, și orice obiect din spate este o componentă.

Majoritatea elementelor vizuale ale unei interfețe grafice, cum ar fi butoanele, casete de text, casete de dialog, sunt reprezentate de clase de control.

Voi prezenta în continuare următoarele aspecte ale programării C#:

- Clasa `Form`
- Execuția programului: cum execută un program framework-ul .NET
- Control: cum fiecare control este o clasă distinctă.
- Evenimente: folosirea evenimentelor C# pentru procesarea acțiunilor utilizatorului.

2.3.2.1 Namespace-uri și clase

Am discutat mai devreme despre utilizarea namespace-urilor în .NET pentru a defini un scop pentru un set de clase și alte tipuri. În program folosim cuvântul cheie `namespace` pentru a declara un nou namespace, spre exemplu `namespace JaguarControl`.

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 definește un nume de clasă și o colecție de

membri pentru a reprezenta și operarea în clasă. O clasă este unul din tipurile posibile într-un namespace.

Clasele în C# suportă moștenirea singulară (single inheritance), astfel clasa moștenește de la o altă clasă. În program putem defini o clasă care să moștenească din clasa `Form`, care se găsește în namespace-ul `System.Windows.Forms`.

Clasa `Form` stă la baza aplicațiilor Windows în framework-ul .NET. Reprezintă orice tip de fereastră dintr-o aplicație, de la casete de dialog la Interfețe de documente multiple (MDI – Multiple Document Interface). Clasa `Form` aduce posibilitatea de a afișa, așeza diferite tipuri de controale și care interacționează cu fereastra de aplicație.

Clasele în .NET conțin unul sau mai mulți membri care definesc comportamentul și caracteristicile clasei. Membri unei clase pot fi constante, câmpuri, metode, proprietăți, evenimente, operatori, constructori și declarații de tip nested [5].

2.3.2.2 Constructori și metode

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`, `protected` și `private` [5].

Primul membru se numește *constructor*, și funcționează ca un constructor în C++. Dacă constructorul inițializează o nouă instanță a unei clase se numește *instance constructor*. Un constructor de instanță fără parametri se numește *default constructor*. C# suportă de asemenea constructori statici pentru a inițializa clasa.

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.

2.3.2.3 Tipuri C#

Cuvântul cheie *new* este utilizat pentru a inițializa orice tip în C#. Acesta include clase și structuri cât și tipuri simple ca `int`, sau enumerări.

Există două clasificări pentru tipuri în C#, fiecare cu un anumit comportament de inițializare. *Value types* conține respectivele date pentru tip. Acesta include tipuri *built-in* cum ar fi `int`, `char`, `bool` dar și structuri create folosind cuvântul cheie `struct`. Tipurile de valori sunt de obicei mici ceea ce face ușor stocarea lor în stivă sau în obiectul care le conține [5].

Tipurile de referință conțin o referință la datele tipului. Funcționează asemănător cu pointerii din C++, cu diferența ca este implicit în C#. Toate clasele din C# sunt tipuri de referință, ca și tipurile *built-in* `object` și `string`. Compilatorul convertește 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].

2.3.2.4 Intrarea în program

Fiecare program C# își începe execuția în funcția `Main`, la fel ca în C, C++ și Java. Această funcție este punctul de start al aplicației. După ce sistemul de operare Windows crează un nou proces. Inițializează diferite structuri de date interne, și încarcă programul executabil în memorie, programul este invocat apelând, acest punct de start.

Punctul de start în C# este similar cu funcția `main` din C și C++ cu excepția că în C# trebuie să fie un membru static al clasei.

Compilatorul C# folosește prima instanță a funcției `main` pe care o localizează ca punct de start pentru program.

2.3.2.5 Clasa Application

Clasa `Application` este folosită pentru a gestiona aplicațiile, thread-urile, și mesajele Windows. Această clasă este utilizată în mod comun pentru a afișa formularul inițial într-o aplicație și să aștepte pentru acțiuni ale utilizatorului [5].

Clasa `Application` este un obiect care încapsulează membrii statici necesari pentru gestionarea și procesarea formularelor. Această clasă este închisă, însemnând că această clasă nu poate fi moștenită. Clasa `Application` este parte a namespace-ului `System.Windows.Forms`. Nu se poate crea o instanță a acestei clase.

```
public sealed class Application
```

Clasa `Application` are metode pentru a începe și a opri aplicații și thread-uri, și pentru a procesa mesaje Windows, după cum urmează [5]:

- `Run` pornește o buclă de mesaj a aplicației pe thread-ul curent, face formularul vizibil.
- `Exit` sau `ExitThread` oprește o buclă de mesaj.
- `DoEvents` procesează mesajele în timp ce programul se află într-o buclă.
- `AddMessageFilter` adaugă un filtru de mesaje pentru monitorizarea mesajelor Windows.

2.3.2.6 Execuția unui program

La execuția unui program, sistemul de operare crează și inițializează un proces care [5]:

- Folosește metoda `Main` ca punct de start pentru execuție, care:
 - Instanțiază o instanță a clasei folosind cuvântul cheie `new`.
 - Invocă constructorul de instanță pentru formular.

- Înapoi în metoda `Main`, metoda `Application.Run` este apelată cu noul obiect creat ca parametru, și:
 - Afișează formularul ca fereastră de aplicație.
 - Așteaptă și procesează mesaje sau interacțiuni ale utilizatorului.
- Când fereastra de aplicație se închide:
 - Metoda `Application.Run` returnează
 - Metoda `Main` returnează
 - Se iese din program.

2.3.2.7 Shortcut-uri

Prima schimbare care poate fi observată într-un cod C# este folosirea cuvântului cheie `using` la începutul unui program [5].

```
using System;

using System.Windows.Forms;
```

Programatorii caută mereu shortcut-uri, astfel au făcut și cei de la Microsoft, rezultatul a fost cuvântul cheie `using`.

Cuvântul `using` joacă defapt două roluri în C#. Primul este ca o directivă pentru a specifica un shortcut, al doilea este pentru a ne asigura ca 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.

Spre exemplu: mai sus am discutat despre `Application`, o funcție poate apela `System.Windows.Forms.Application.Run`. Putem folosi directiva `using` pentru a scurta acest apel la `Application.Run`. Forma lungă se numește nume complet calificat (fully qualified name) pentru că întregul namespace este specificat.

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

2.3.3 *Librăria OpenCV*

OpenCV (Open Source Computer Vision) este o librărie licențiată BSD, open-source, care include sute de algoritmi de computer vision.

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 multi dimensionale `Mat` și funcții 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, histogramme, etc.
- **Video** – un modul de analiză video care include estimări de mișcare, extrageri de fundal, și algoritmi de tracking 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, codec-uri pentru imagini și video, simple capabilități UI.
- **Gpu** – algoritmi de accelerare GPU.
- Alte module ca wrapper-uri pentru Python sau C#.

2.3.3.1 Concepte API

Namespace-ul cv

Toate clasele și funcțiile OpenCV sunt plasate în namespace-ul `cv`. Astfel, pentru a accesa această funcționalitate din codul nostru, trebuie să folosim directiva `using namespace cv`.

Management-ul automat al memoriei

OpenCV folosește memoria automat. Multe structuri de date ca `vector`, `Mat`, folosite de funcții și metode au destructori care dezalocă buffer-urile de memorie când este nevoie. Acest lucru înseamnă că destructorii nu dezalocă mereu memoria, ca în cazul `Mat`, se ia în considerare posibilele distribuiri de date. Un destructor decrementează numărătorul de referință asociat cu buffer-ul de date pentru matrice. Buffer-ul este dezallocat dacă și doar dacă numărătorul de referință este 0, atunci când alte structuri fac referință la același buffer. Similar, când o instanță `Mat` este copiată, nu se copiază defapt date propriu-zise. În schimb, numărătorul de referință este incrementat pentru a reține că există un alt deținător al aceleiași date. Există de asemenea și o metodă `Mat.clone` care crează o copie completă a matricei [6].

Alocarea automată a datelor de ieșire

OpenCV dezalocă memorie automat, dar tot în mod automat, alocă memorie pentru parametri de ieșire ai funcțiilor. Deci, dacă o funcție are unul sau mai multe șiruri de intrare, și unul sau mai multe șiruri de ieșire, șirurile de ieșire sunt automat alocate sau realocate. Marimea și tipul șirurilor sunt determinate din marimea și tipul șirurilor de intrare [6].

Tipuri fixe de pixeli

Template-urile (Șabloanele) sunt o caracteristică importantă în C++ care permit implementarea unor structuri de date și algoritmi foarte puternici și eficienți. Totuși, folosirea intensivă a acestor template-uri pot mări dramatic timpul de compilare și dimensiunea codului. Poate fi în regulă pentru algoritmi de bază, dar nu prea eficient pentru librării de computer vision, unde un singur algoritm poate ajunge la câteva mii de linii de cod. Din acest motiv, dar și pentru simplificarea dezvoltării unor legături sau wrapper-e pentru alte limbaje, cum ar fi Python, Java, Matlab sau C#, care nu folosesc template-uri deloc, sau folosesc foarte puțin,

implementarea actuală a OpenCV este bazată pe polimorfism și runtime dispatching. În acele locuri unde runtime dispatching nu este prea eficient sau ar fi mult prea încet, implementarea actuală introduce mici clase template, metode și funcții [6].

În consecință, există set fix, limitat de tipuri de date primitive pe care librăria poate opera. Astfel, elementele din șiruri pot avea unul din următoarele tipuri [6]:

- Întregi fără semn pe 8 biți (uchar)
- Întregi cu semn pe 8 biți (schar)
- Întregi fără semn pe 16 biți (ushort)
- Întregi cu semn pe 16 biți (short)
- Întregi cu semn pe 32 biți (int)
- Număr în virgulă mobilă pe 32 biți (float)
- Număr în virgulă mobilă pe 64 biți (double)
- Un tuplu de elemente unde toate elementele au același tip (unul din cele de mai sus). Un șir în care elementele sunt astfel de tupluri, se numesc șiruri multi-canal. Numărul maxim de canale este definit de constanta CV_CN_MAX care este 512.

Erorile

OpenCV folosește excepții pentru a semnaliza erori critice. Când datele de intrare au formatul corect și face parte din câmpul de valori specificat, dar algoritmul nu are succes (spre exemplu, un algoritm de optimizare nu converge), este returnat un cod de eroare special (de obicei, o variabilă booleană) [6].

Excepțiile pot fi instanțe ale clasei `Exception` sau derivatele ei.

Excepțiile sunt de obicei aruncate fie folosind macro-ul `CV_Error(errcode, descriere)`, sau varianta sa asemănătoare cu `printf` `CV_Error_(errcode, printf-spec, (printf-args))`, sau folosind macro-ul `CV_Assert(condition)` care verifică condiția și aruncă o excepție când nu este satisfăcută. Datorită management-ului automat de memorie, toate buffer-ele intermediare sunt automat dealocate în cazul unei erori. Trebuie adăugat un `try` pentru a prinde (catch) excepțiile, dacă este nevoie [6].

Exemplu try – catch:

```
try
{
    VideoStream.NewFrame+=new
    NewFrameEventHandler(ProcessFrame_NewFrame);
    _blobDetector = new CvBlobDetector();

    string sourceURL = "http://192.168.0.74:8082/axis-
    cgi/mjpg/video.cgi";

}
catch (NullReferenceException excpt)
{
    MessageBox.Show(excpt.Message); }
```

2.3.3.2 Emgu CV

Emgu CV este o platformă .Net și încapsulează (wrapper) librăria OpenCV despre care am menționat în subcapitolul anterior. Astfel permite funcțiilor OpenCV să fie apelate din limbaje compatibile .NET, cum ar fi C# , Visual Basic, VC++, IronPython etc. Wrapper-ul poate fi compilat de Visual Studio, Xamarin Studio și Unity, poate funcționa pe Windows, Linux, Mac OS X, iOS, Android și Windows Phone

Caracteristicile pentru platforma de Windows pot fi Observate în tabelul de mai jos [8].

Tabel 5 Platforma Windows

Nume	Emgu CV (open Source)	Emgu CV pentru Windows (Comercial)		Emgu CV pentru Unity
Tool de dezvoltare	Visual Studio 2010 și mai noi	Visual Studio 2010 și mai noi	Visual Studio 2013 și mai noi	Unity 3D Pro v5.1 și mai noi
Platformă	Windows	Windows	Windows 8.1	Windows Desktop Standalone
Arhitectură CPU suportată	i386, x64	i386, x64	i386, x64	i386(Editor), x64(Standalone)
Procesare CUDA GPU	✓	✓	X	✓
OpenCL(GPU&CPU)	✓	✓	X	✓
Tesseract OCR	✓	✓	✓	✓
Compilat cu Intel C++ Compiler	X	✓	X	X
Exception Handling	✓	✓	✓	✓
Debugger Visualizer	✓	✓	X	X
Emgu.CV.UI	✓	✓	X	X
Licență	GPL	Licență comercială		Licență comercială

Emgu CV este scris complet în C#. Poate fi compilat în Mono și astfel poate rula pe orice platforma suportată de Mono, incluzând iOS, Android , Windows Phone, Mac OS X și Linux.

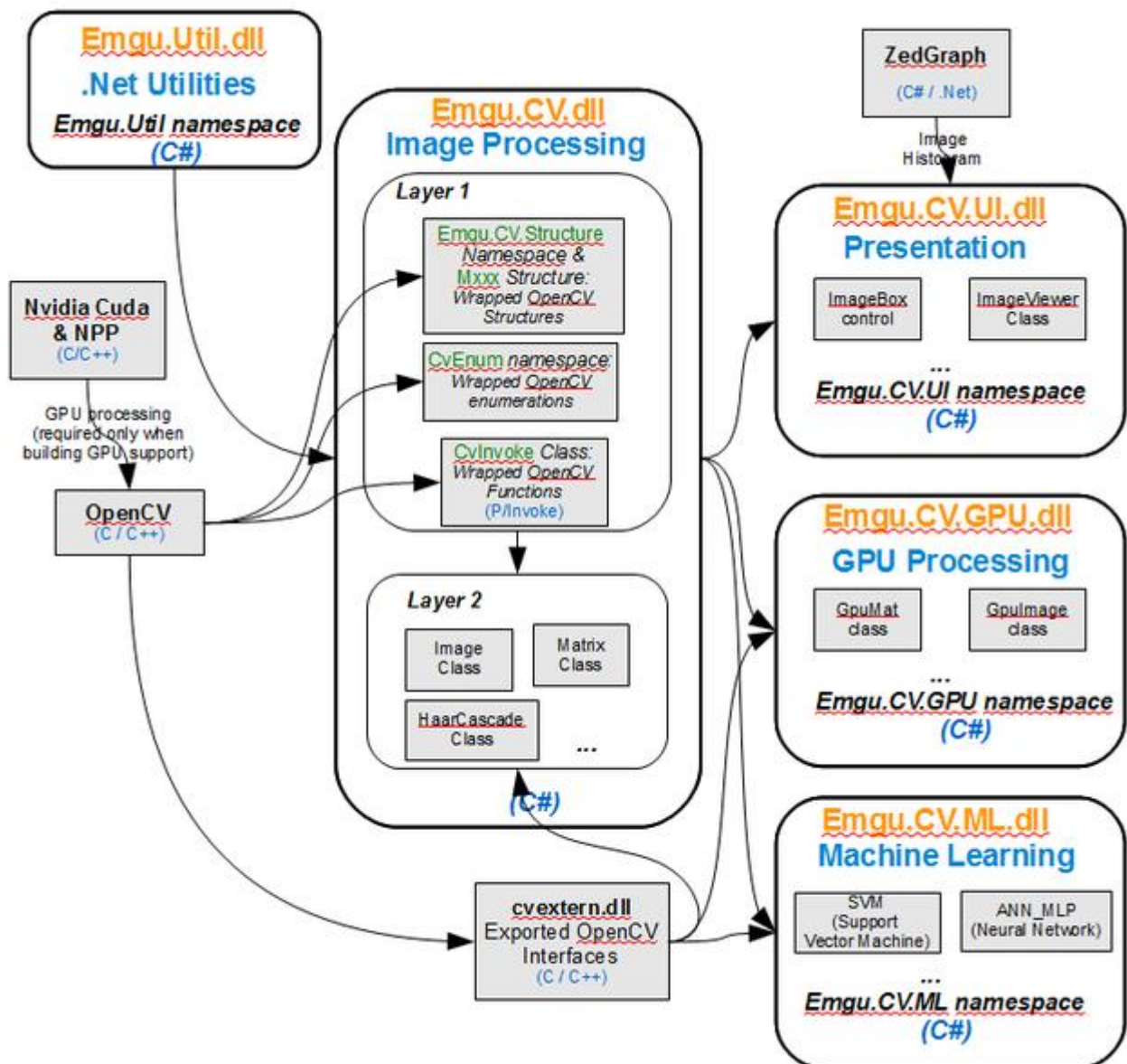
Avantaje:

- Clasa Image cu Generic Color și Depth
- Garbage collection automat
- Imagine XML serializabilă
- Posibilitatea de a alege fie clasa Image sau funcții invocate direct din OpenCV
- Operații generice pe pixeli

Arhitectura EmguCV

Emgu CV are două straturi de încapsulare, după cum se poate observa în figura de mai jos [8]:

- Stratul de bază (layer 1) conține mapări pentru funcții, structuri și enumerări și le reflectă direct pe cele din OpenCV.
- Al doilea strat (layer 2) conține clase din .NET.



Figură 11 Arhitectura Emgu CV

2.3.4 Aplicația demo de control –Jaguar Control

2.3.4.1 Descriere

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 via Wi-Fi la platforma robotică.

Astfel, pe computer-ul gazdă trebuie instalate următoarele programe [1]:

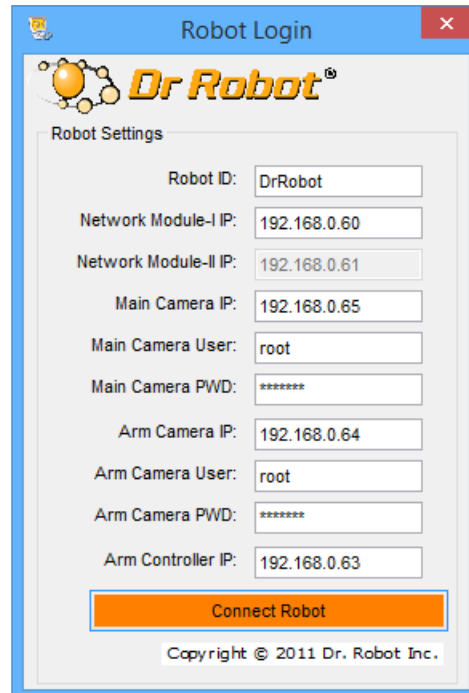
- Programul ”Jaguar Control” – instalat folosind executabilul Setup.exe
- Programul ”Google Earth”
- SDK pentru camerele Axis

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

- Reîmprospătează datele citite de la encodelele 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 stream-ul video de la camerele Axis.

2.3.4.2 Mod de utilizare

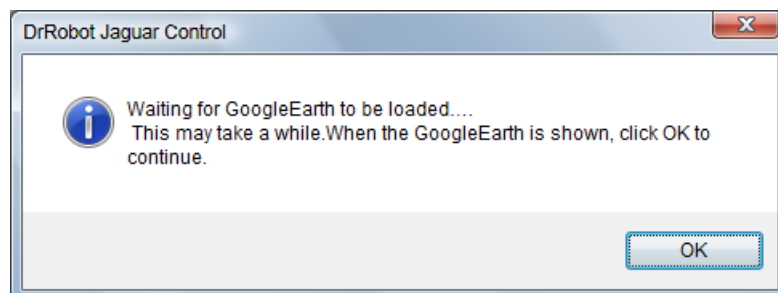
Odată pornit programul, va fi afișată fereastra de logare [1]:



Figură 12 Fereastră de logare

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 unor routere, astfel fiecare are asociată o adresă IP. După apăsarea butonului **Connect Robot** aplicația realizează conexiunea Wi-Fi cu robotul prin intermediul socketelor TCP, odată conectați se pot transmite și primi date de la fiecare din componentele prezentate în capitolul 2.2.

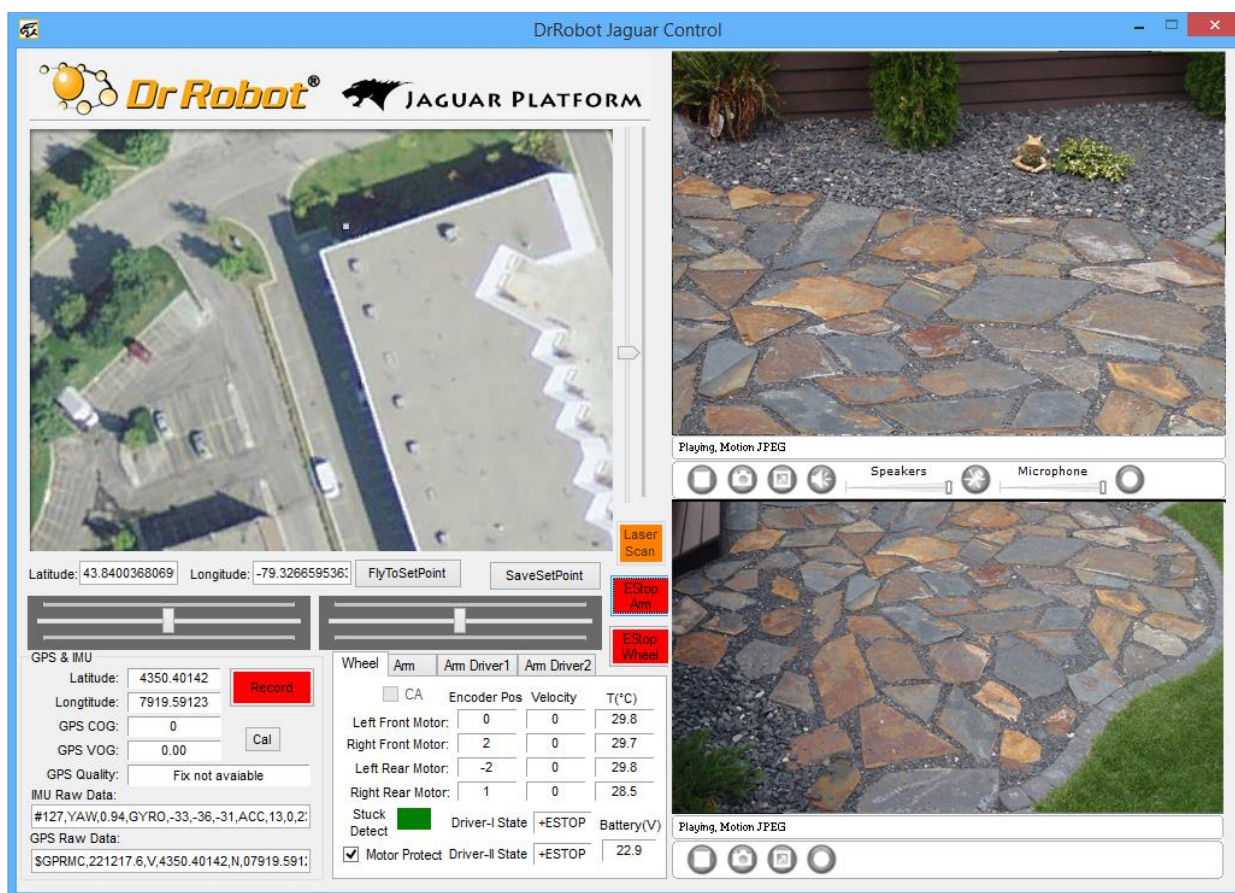
După această etapă va fi afișat următorul mesaj [1]:



Figură 13 Mesaj

Se așteaptă verificarea conexiunii la internet pentru încărcarea hașurilor Google Earth. Google Earth suportă de asemenea și utilizarea offline, dar harta trebuie obținută online în prealabil.

După încărcare (care e posibil să dureze destul de mult în modul offline) va apărea interfața completă a aplicației:

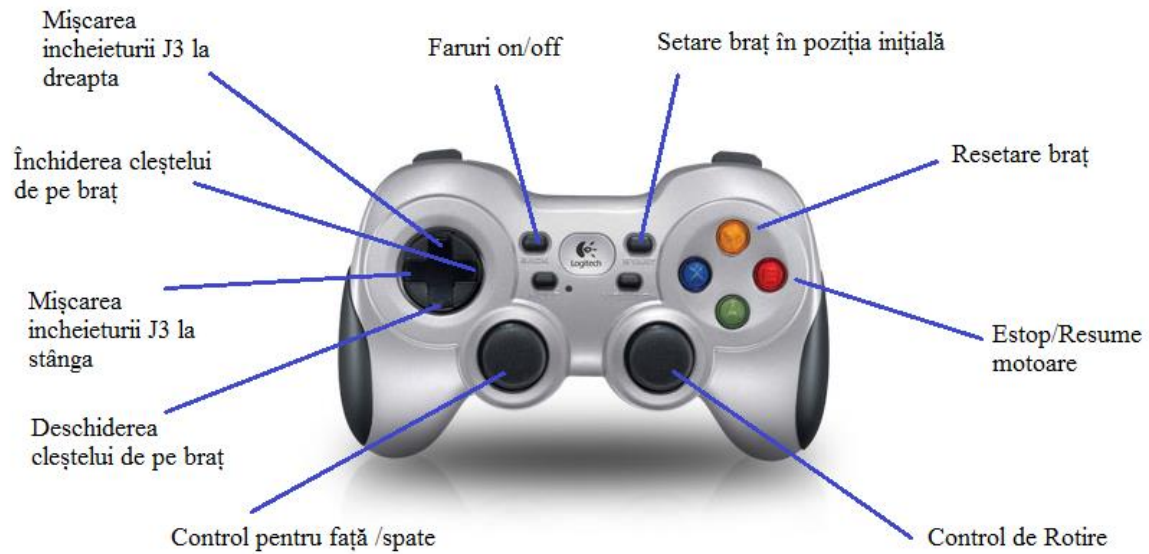


Figură 14 Interfața aplicației Jaguar Control

În această interfață se poate observa stream-ul camerelor Axis în partea din dreapta, și datele de la senzori în partea din stânga, de asemenea, se pot observa și trackbar-urile care sunt folosite pentru controlul robotului (față / spate, stânga / dreapta)

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 mapate pe un controller Gamepad (cuprins în pachetul robotului).

Controller-ul Gamepad și butoanele de control sunt prezentate în figurile a) și b) de mai jos [1] :



Figură 15 Controller-ul Gamepad a)



Figură 16 Controller-ul Gamepad b)

Controller-ul Gamepad poate fi folosit doar când programul rulează și este în focus.

În această interfață, poate cele mai importante informații sunt datele primite de la senzori, sau de la Driver-ele de motoare. Aceste date sunt afișate în casete de text cu etichetele aferente. Deși aceste date nu sunt utilizate în controlul robotului, modul în care acestea sunt accesate este foarte important. Date ca poziția encoderelor sau temperatura motoarelor pot fi foarte folositoare în dezvoltarea și realizarea de aplicații cu roboți autonomi.

Wheel	Arm	Arm Driver1	Arm Driver2
☐ CA	Encoder Pos	Velocity	T(°C)
Left Front Motor:	0	0	29.8
Right Front Motor:	2	0	29.7
Left Rear Motor:	-2	0	29.8
Right Rear Motor:	1	0	28.5
Stuck Detect	☒ Driver-I State	+ESTOP	Battery(V)
☒ Motor Protect	Driver-II State	+ESTOP	22.9

Figură 17 Informațiile de la senzori

În această parte a aplicației sunt afișate informațiile de la motoare. Dacă robotul folosește bateriile LiPo incluse, Robotul trebuie oprit la o tensiune de sub 22.2V (afișarea tensiunii bateriei în aplicație fiind foarte importantă din acest motiv)

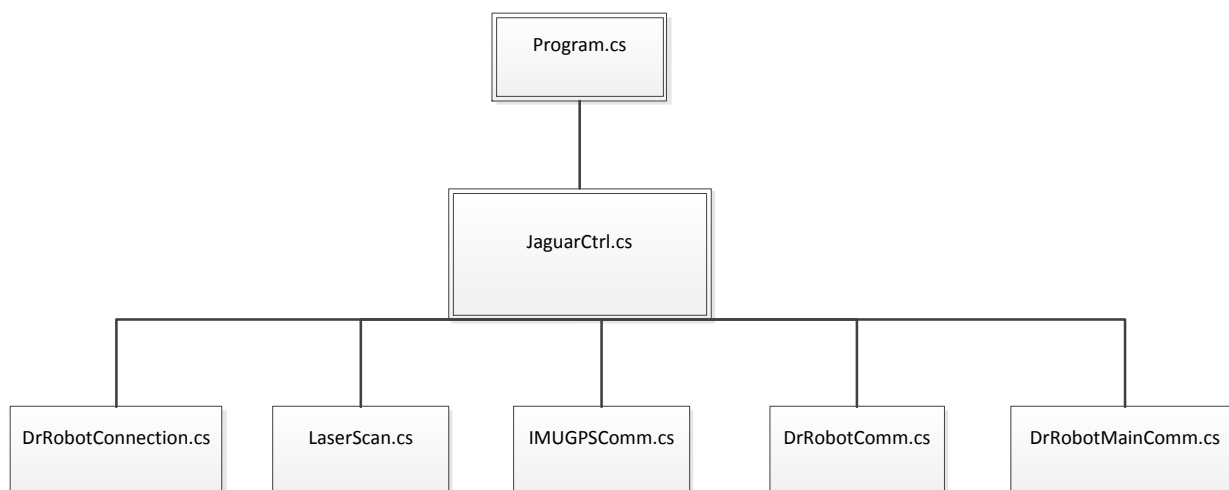
2.3.4.3 Dezvoltarea Software-ului

Aplicația mai sus prezentată a fost scrisă în totalitate în C#, tot codul sursă fiind pus la dispoziție pentru a putea fi studiat și înțeles modul de utilizare și programare pentru această platformă robotică.

În programarea robotului, nu vom atinge codul embedded din microprocesorul robotului, ci vom programa totul din computerul gazdă. Aplicația C# va rula pe acest computer și folosind o conexiune Wi-Fi vom realiza o controlare Open loop a robotului.

În folderul aplicației se pot găsi toate clasele utilizate în dezvoltarea acestei aplicații:

- DrRobotComm.cs
- DrRobotMainComm.cs
- DrRobotRobotConnection.cs
- IMUGPSComm.cs
- JaguarCtrl.cs
- LaserScan.cs
- Program.cs



Figură 18 Structura claselor

Arhitectura generală a codului este prezentată în figura de mai sus. De observat este că fișierul Application este `Program.cs` care instanțiază o fereastră de dialog: `JaguarCtrl`. În `JaguarCtrl`, se crează instanțieri ale diferitelor clase, aici are loc cea mai importantă parte a programării.

Vom folosi toate aceste informații pentru a ne crea propria aplicație. Vom pleca de la aceeași structură și vom face modificări pe parcurs. Dezvoltarea aplicației este prezentată în capitolul 5.

CAPITOLUL 3

ANALIZA ȘI PRELUCRAREA IMAGINILOR

3.1 INTRODUCERE

Tehnologia digitală modernă a făcut posibilă manipularea de semnale multidimensionale cu sisteme care pot fi de la simple circuite digitale la computere paralele avansate. Scopul acestui tip de manipulare poate fi împărțit în trei categorii [15]:

- Procesare de Imagini imagine la intrare -> imagine la ieșire
- Analiză de Imagini imagine la intrare -> măsurători la ieșire
- Înțelegere de Imagini imagine la intrare -> descriere de nivel înalt la ieșire

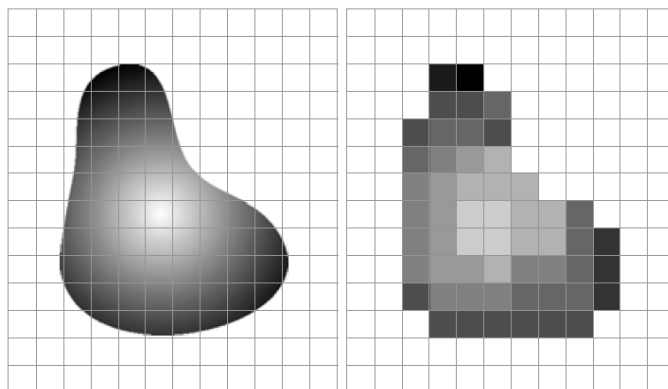
Imaginea poate fi definită ca o funcție de două variabile, spre exemplu $f(x,y)$ cu f fiind amplitudinea imaginii la pozițiile reale de coordonate (x,y) . O imagine poate fi considerată compusă din sub-imagini referite uneori ca regiuni de interes, sau pur și simplu regiuni. Acest lucru reflectă faptul ca imaginile conțin în mod frecvent colecții de obiecte fiecare putând fi baza unei regiuni. Într-un sistem sofisticat de procesare de imagini poate fi posibilă aplicarea unor operații specifice de procesare pentru acele regiuni. Astfel o parte a imaginii (regiune) poate fi procesată pentru a reduce un anumit tip de zgomot, cât timp altă parte poate fi procesată pentru a scoate în evidență unele caracteristici [15].

Amplitudinile unei imagini vor fi în majoritatea cazurilor , fie numere reale, fie numere întregi. Numere întregi sunt de obicei rezultatul unui proces de cuantizare care convertește o gamă continuă într-un număr discret de nivele.

3.2 IMAGINI DIGITALE

O imagine digitală $a[m,n]$ descrisă într-un spațiu discret 2D este derivată dintr-o imagine analogică $a(x,y)$ într-un spațiu 2D continuu, printr-un proces de eșantionare care este frecvent denumit digitizare.

Efectele digitizării se pot observa în imaginea următoare [9]:



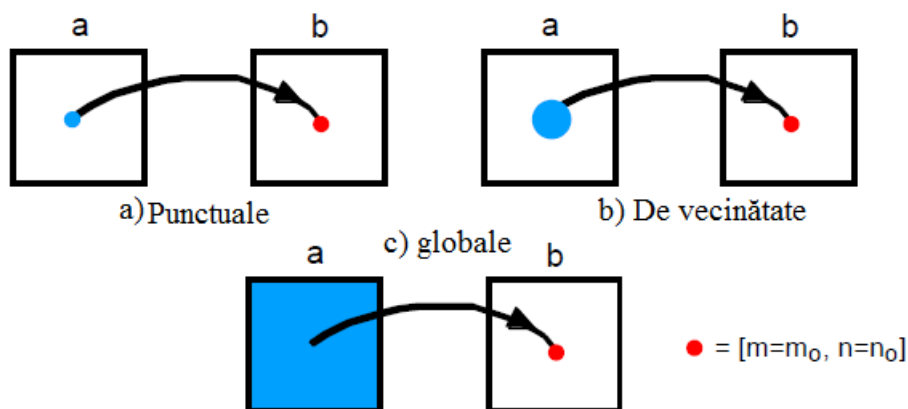
Figură 19 Eșantionarea unei imagini continue.

Imaginea prezentată în figura de mai sus a fost împărțită într-un număr discret de rânduri și coloane. Valoarea asociată fiecărui pixel este luminozitatea medie a pixelului rotunjit la cel mai apropiat întreg. Procesul de reprezentare a amplitudinii unui semnal 2D la anumite coordonate ca un întreg cu L nivele de gri diferite se numește cuantizare în amplitudine, sau mai simplu, cuantizare.

3.2.1 Operații pe imagini

Există multe posibilități de a clasifica sau caracteriza operațiile pe imagini. Tipurile de operații pot fi aplicate imaginilor digitale pentru a transforma o imagine de intrare $f[m,n]$ într-o imagine de ieșire $g[m,n]$ pot fi clasificate în trei categorii [9]:

- Operații punctuale – valoarea de ieșire la coordonate specifice este dependentă doar de valoarea de intrare de la aceleași coordonate.
- Operații pe vecinătate – valoarea de ieșire de la coordonate specifice este dependentă de valorile de intrare din vecinătatea acelorași coordonate.
- Operații globale/ integrale - valoarea de ieșire de la coordonate specifice este dependentă de toate valorile din imaginea de intrare.



Figură 20 Operații pe imagini.

Aceste tipuri de operații pot facilita procesarea imaginii în sensul extragerii caracteristicilor utile, sau evidențierea regiunilor de interes. Astfel de transformări stau la baza diferitelor tipuri de filtre [9].

3.2.2 Filtre liniare

Majoritatea imaginilor sunt afectate într-o anumită măsură de zgomot, care este o variație neexplicată a datelor. Analiza de imagini este de obicei simplificată dacă acest zgomot ar putea fi filtrat. Termenul de filtru se aplică de asemenea și operațiilor care accentuează anumite caracteristici dintr-o imagine. Astfel acceptând această definiție mai largă, filtrele de imagini pot fi folosite pentru a evidenția contururi – adică delimitări între obiecte sau părți de obiecte, dintr-o imagine. Filtrele ajută la interpretarea vizuală a imaginilor, și pot fi de asemenea folosite ca precursor pentru mai multe tipuri de procesare digital, cum ar fi segmentarea [15].

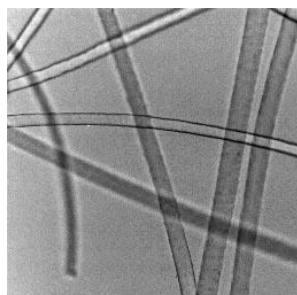
Multe metode, operează pe fiecare pixel separat (operații punctuale). Filtrele schimbă valoarea unui pixel luând în considerare valorile pixelilor vecini acestuia.

- Spre exemplu o fereastră glisantă de 3x3 care realizează operația de mediere – calculează media pixelilor din fereastra de 3x3 și înlocuiește valoarea pixelului central cu această medie.

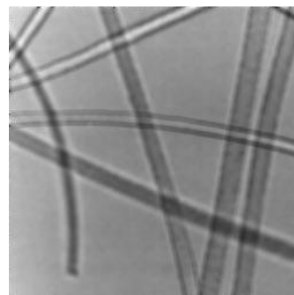


fereastră de mediere.

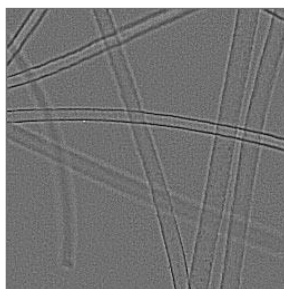
Rezultatul aplicării acestui filtru, este reducerea zgomotului din imagine, dar de asemenea aplică și efectul de blur asupra conturilor obiectelor din imagine.



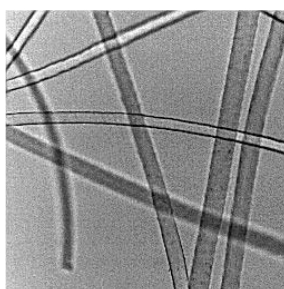
-----fereastră de mediere----->



- Dacă imagine filtrată cu filtrul precedent (fereastră glisantă de mediere) este scăzută din imaginea inițială (pixel cu pixel) se obține filtrul Laplacian, și are ca efect scoatere în evidență a conturilor din imagine.



- Dacă ieșirea unui filtru Laplacian este adunată la imaginea inițială (pixel cu pixel). Pentru ochiul uman, imaginea pare mai clară, deoarece tranzițiile de la contururi s-au accentuat.



Filtrele prezentate mai sus, sunt toate, filtre liniare, deoarece valorile de ieșire sunt combinații liniare ale pixelilor din imaginea inițială. Aplicarea a două filtre liniare asupra unei imagini are același efect cu aplicarea lor în ordine inversă [15].

3.3 DETECȚIA DE CONTUR

3.3.1 Detecția de contur cu filtre liniare

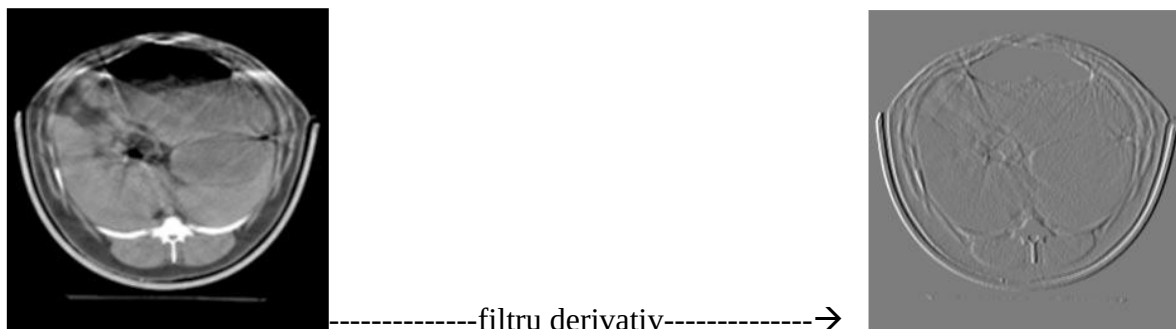
Când ne ocupăm de o regiune sau un obiect, există mai multe posibilități în ceea ce privește o reprezentare compactă care poate facilita manipularea sau aplicarea unor măsurători pe obiect.

În fiecare caz ne asumăm că începem cu o reprezentare a imaginii ca cea prezentată mai sus. Există câteva tehnici pentru a reprezenta o regiune sau un obiect realizând o descriere de contur [10].

Pentru a scoate în evidență conturile din imagine este necesar ca unele ponderi din filtru să fie negative. În particular, un set de ponderi care au suma egală cu zero, care produc, ca ieșire diferențe între valorile pixelilor din imaginea de intrare, numim acest filtru, un filtru derivativ [10]. Un filtru derivativ va determina, la o poziție din imaginea de ieșire diferența dintre valorile pixelilor pe coloane pe fiecare parte de la acea locație în imaginea originală. Astfel ieșirea filtrului va avea magnitudine mare (fie negativă fie pozitivă) dacă există o diferență notabilă între valorile pixelilor de la stânga și de la dreapta locației pixelului curent. Un astfel de set de ponderi este prezentat mai jos [16]:

$$w = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Putem realiza și o ponderare a valorilor, înmulțind cu o anumită valoare subunitară pentru a obține și o ușoară atenuare a zgomotului.



Deși filtrele derivative de ordin 1, prezentate anterior, pot răspunde la contururi doar pe o anumită direcție, există și filtrele derivative de ordin 2 care reușesc să delimiteze contururi în orice direcție și totuși rămânând liniare. Considerăm următoarea matrice:

$$w = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Acesta este același tip de filtru despre care am menționat mai devreme (filtrul Laplacian) observăm rezultatul în imaginile de mai jos.



Dacă realizăm și o binarizare după un anumit prag, putem obține o imagine binară în care contururile sunt mult mai bine evidențiate.

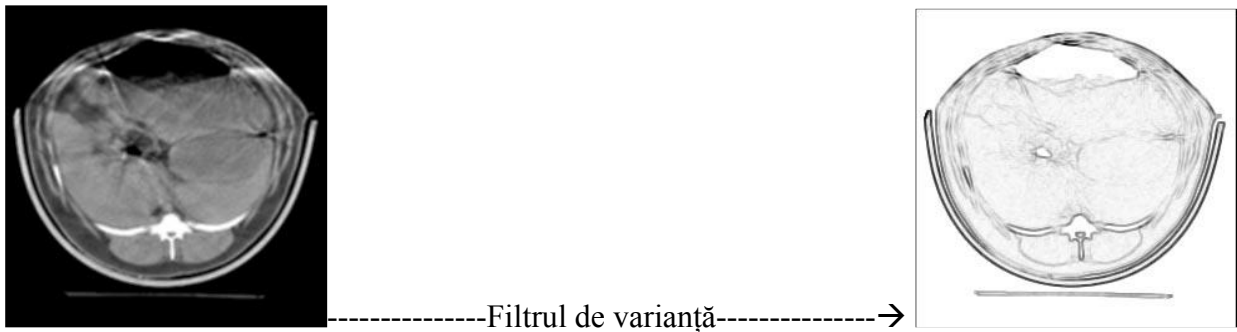


3.3.2 Detecția de contur cu filtre neliniare

Un filtru de contur, ar trebui să producă un răspuns semnificativ la o anumită locație din imagine dacă pixelii din vecinătate indică un șablon sistematic de schimbare a valorilor. Filtrele linare considerate anterior sunt folositoare pentru a găsi contururi doar pe o anumită direcție, doar filtrul Laplacian a fost util pentru toate direcțiile. În continuare vom considera doar câteva filtre de contur, relevante pentru proiectul de față [16].

Filtrul de varianță

Filtrul de varianță, reprezintă o măsură a prezenței unui contur care poate fi calculată foarte rapid. Cum era de așteptat, vecinătățile care conțin contururi au varianță mai mare, dar zgomotul din imaginea inițială poate fi observat în imaginea de ieșire. Rezultatele acestui filtru sunt reprezentate mai jos [16].

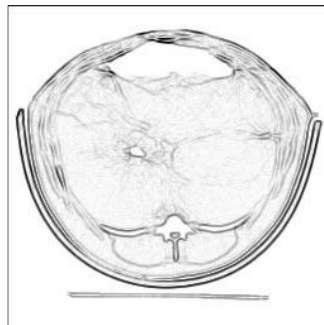


Filtre de gradient

Filtrul de mai sus, cât și multe alte filtre de acest tip scot în evidență nu doar marginile obiectului din imagine, dar și zgomotul. Pentru a reduce acest zgomot putem aplica filtre de netezire pe diferite variante ale imaginii, dar o altă variantă, mult mai robustă și eficientă este estimarea gradientului [10].

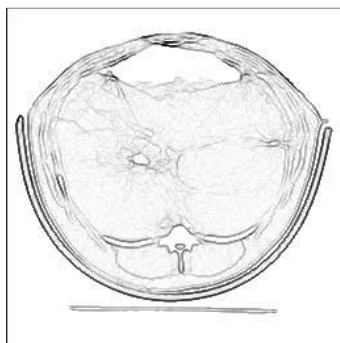
$$\sqrt{\left(\frac{\partial f_{ij}}{\partial x}\right)^2 + \left(\frac{\partial f_{ij}}{\partial y}\right)^2}$$

Putem construi un filtru care să estimeze gradientul, înlocuind derivatele parțiale din formula gradientului cu estimatele lor. Această estimare a gradientului maxim este cunoscut ca *filtrul Prewitt*. În figura de mai jos putem observa rezultatul aplicării acestui filtru. Observăm că acest filtru răspunde la contururi în toate direcțiile.



Figură 21 Filtrul Prewitt

Filtrul Sobel este foarte asemănător , diferența fiind că în estimarea gradientului maxim se adaugă o pondere mai mare pixelilor din apropierea pixelului curent.



Figură 22 Filtrul Sobel

Filtrul Canny

Algoritmul Canny de detecție a conturilor este împărțit în mai multe etape [7]:

- Reducerea zgomotului
- Găsirea gradientului de intensitate
- Suprimarea non-maximelor
- Trasarea conturului și thresholding hysteresis

Algoritmul își propune:

- Marcarea cât mai multor contururi reale din imagine.
- Marcarea unui contur o singură dată.

1) Reducerea zgomotului:

Pentru a asigura că imaginea nu este afectată de pixeli de zgomot. Filtrul este bazat pe prima derivată a unei Gaussiene:

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Unde x este distanța de la origine la axa orizontală, y este distanța de la origine la axa verticală iar sigma este deviația standard a distribuției gaussiene [10].

2) Găsirea gradientului de intensitate

Contururile pot fi orientate în diferite direcții → Canny folosește patru filtre:

- Orizontal – 90 de grade
- Vertical – 0 grade
- 2 Diagonale – 45 de grade și 135 de grade.

Operatorul de detecție a conturului returnează o valoare pentru prima derivată pe direcția orizontală (Gy) și pe direcția verticală (Gx). Astfel, gradientul de contur și direcția pot fi determinate:

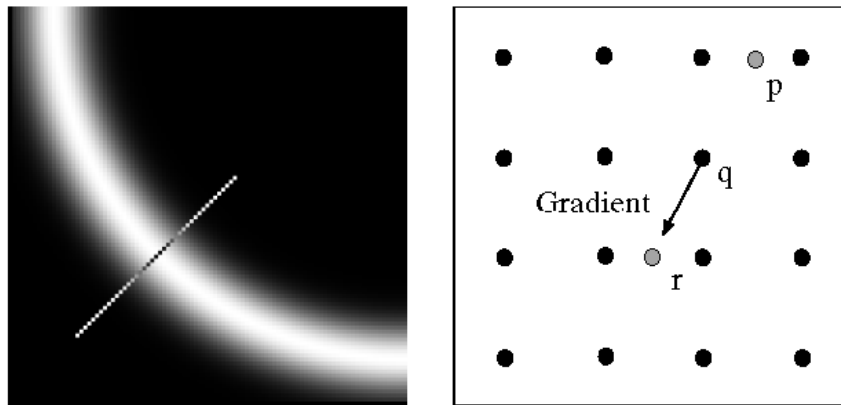
$$G = \sqrt{G_x^2 + G_y^2} \quad \text{și} \quad \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Operatorul de detecție a conturului folosit în algoritm este un operator Sobel (prezentat mai sus). Este definit matematic astfel:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A, \quad \text{și} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

3) Suprimarea non-maximelor

Se realizează o căutare pentru a determina dacă magnitudinea gradientului este un maxim local pe direcția gradientului, acest lucru este reprezentat în figura de mai jos [10].



Figură 23 Suprimarea non-maximelor

Se creează astfel un set de puncte de contur sub forma unei imagini binare.

4) Trasarea conturului și thresholding hysteresis

Ne asumăm faptul că gradientii de intensitate au o probabilitate mult mai mare de a corespunde unor contururi.

Thresholding-ul care folosește hysteresis are nevoie de două valori de intrare: marginea superioară și marginea inferioară [10].

După ce se încheie această etapă avem o imagine filtrată Canny.

3.3.3 Transformata Hough

Această transformată își propune găsirea caracteristicilor liniare dintr-o imagine, dar poate fi de asemenea folosită pentru a găsi cercuri în imagine [7].

În spațial unei imagini, o linie dreaptă este descrisă de următoarea ecuație

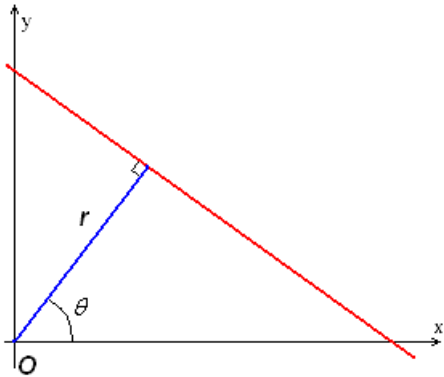
$$y = mx + b$$

Și poate fi afișată graphic pentru fiecare pereche de puncte (x,y)

Transformând punctele din imagine în spațiul parametrilor, și considerând doar parametrii (m,b) o linie poate fi descrisă ca un punct.

Totuși, liniile verticale pot fi nemărginite în acești parametric, astfel ar fi mai bine să alegem alt set de parametri pentru a ne descrie linia.

În spațiul Hough folosim parametrii r și θ , unde r este distanța de la origine la linie și θ este unghiul vectorului până la cel mai apropiat punct.

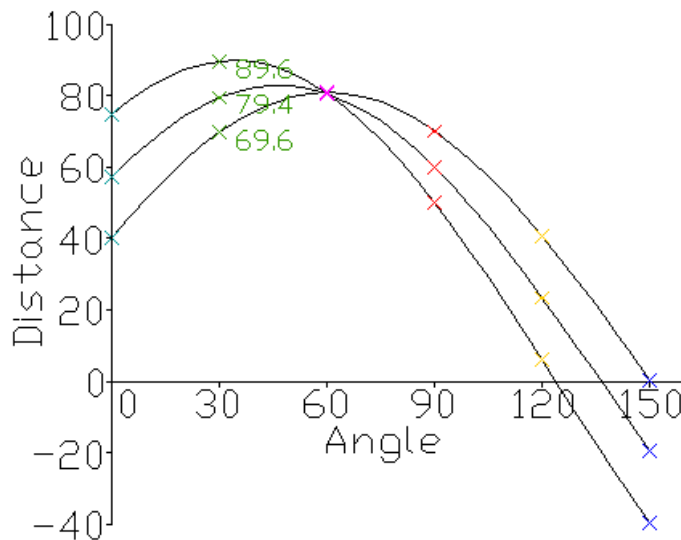


Folosind această parametrizare putem să scriem:

$$y = (-\cos\theta/\sin\theta)x + (r/\sin\theta), \text{ sau } r = x\cos\theta + y\sin\theta$$

Această nouă ecuație va corespunde cu o curbă sinusoidală în planul (r,θ) , care este unică pentru punctul (r,θ) .

Dacă două curbe (în spațiul Hough) se intersectează, atunci punctul de intersecție corespunde unei linii în spațiul original al imaginii.



Deteția punctelor coliniare devine o problemă de detecție a curbelor concurente.

Acest nou, unic punct (r,θ) poate fi convertit înapoi în spațiul imaginii (x,y) și apoi este desenată o linie în imagine.

În cazul detecției de cercuri se folosește același algoritm, trebuie însă reconsiderată ecuația inițială astfel, vom avea:

$$(x - a)^2 + (y - b)^2 = R^2$$

Unde (a,b) este centrul cercului, și r este raza.

sau putem scrie:

$$x = a + R\cos(\theta)$$

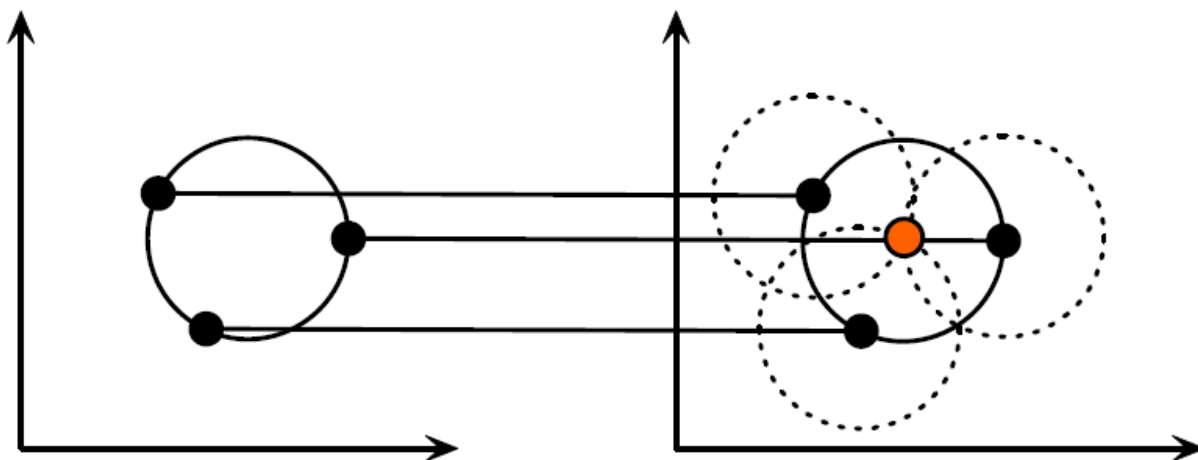
$$y = b + R\sin(\theta)$$

Când unghiul θ parcurge 360 de grade, punctele (x,y) vor descrie perimetrul unui cerc.

Dacă o imagine conține multe puncte, și multe dintre ele se suprapun peste perimetrul unui cerc, atunci programul de căutare trebuie să returneze tripletul (a,b,R) pentru a descrie fiecare cerc.

Astfel apare o problemă legată de resursele consummate (memorie, timp) datorată căutării în trei dimensiuni.

Dacă cercurile căutate au o rază cunoscută am putea reduce problema la o căutare 2D.



Figură 24 Căutare cu rază cunoscută

Fiecare punct din spațiul geometric (partea stângă) generează un cerc în spațiul parametrilor (partea dreaptă). Cercurile din spațiul parametrilor intersecționează dubletul (a,b) care este centrul în spațiul geometric.

3.4 SEGMENTAREA ÎN SPAȚIUL DE CULOARE

În momentul de față se există multe metode de segmentare pentru a extrage informații legate de culoare din imagine. În acest câmp de lucru informația de culoare a fost intens folosită ca o proprietate de percepție a obiectelor pentru a separa un obiect de un fundal. Există diferite reprezentări ale culorii în imaginile digitale, fiecare cu caracteristici speciale.

Culoarea ca proprietate vizuală de percepție a obiectelor, este importantă în codarea imaginilor, computer vision, dar și procesarea video. Datorită nevoilor acestor tipuri de aplicații, există mai multe metode folosite pentru a reprezenta culoarea, fiecare bazată pe altă idee matematică, cu avantajele sau dezavantajele ei.

Calitatea segmentării poate fi judecată doar ținând cont de aplicația finală. Spre exemplu, dacă un sistem de recunoaștere de obiecte beneficiază sau nu de segmentare. Totuși o segmentare bună ar trebui să respecte câteva proprietăți.

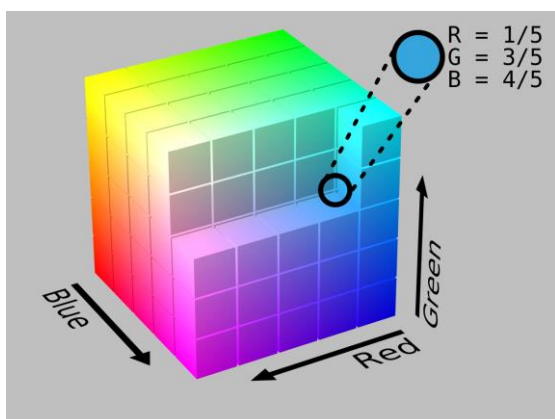
Există foarte multe spații de culoare propuse în literatură. Fiecare cu proprietățile lor, avantaje, limitări și arii de aplicații. Ne vom axa doar pe cele mai uzuale, care sunt destul de folosite în procesarea de imagini: RGB și HSV.

Segmentarea realizându-se setând anumite intervale de valori pentru fiecare dimensiune a spațiului. Deși foarte răspândit RGB nu are rezultate așa bune ca HSV, aceste două spații urmând să fie descrise sumar în subcapitolele următoare.

3.4.1 Spațiul de culoare RGB

Datorită simplității spațiului de culoare RGB, acesta este cel mai des folosit. Este reprezentat de roșu (R), verde (G), și albastru (B). Culoarele finale sunt rezultatul combinării acestor trei culori.

Spațiul de culoare RGB poate fi reprezentat de cubul RGB prezentat în imaginea de mai jos



Figură 25 Cubul RGB

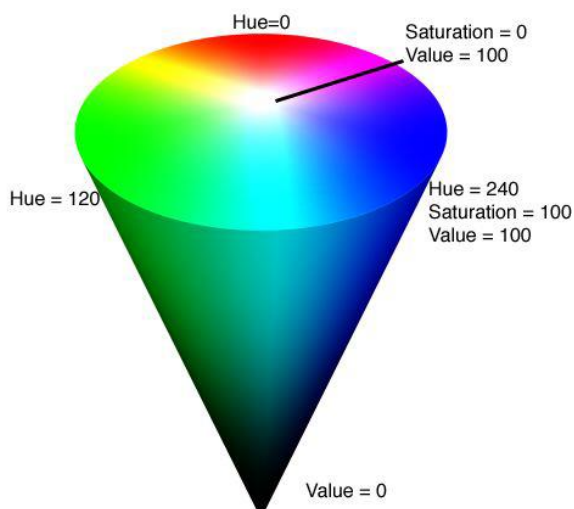
După cum poate fi observat în această imagine există 3 dimensiuni ale acestui spațiu, astfel o anumită culoare dintr-o imagine poate fi reprezentată folosind proporții ale fiecărui plan din acest spațiu, spre exemplu, culoarea aproape albastră reprezentată în imaginea alăturată este formată din 4/5 Albastru, 3/5 Verde și 1/5 Roșu.

Putem astfel obține toate culorile spectrului.

3.4.2 Spațiul de culoare HSV

Spațiul de culoare HSV (adică Hue, Saturație și Valoare) separă intensitățile și le reprezintă independent. Hue descrie poziția culorii într-un spectru de 360 de grade. Saturația descrie puritatea culorii: măsoară diferența dintre culoare și o valoare grayscale de aceeași intensitate. Valoarea, reprezentând al treilea canal, măsoară luminozitatea.

Acest spațiu poate fi reprezentat prin ceea ce se numește conul HSV, reprezentat în figura de mai jos:



Figură 26 Conul HSV

Conceptual, spațiul de culoare HSV este un con. Văzut din partea ciculară a conului, valorile Hue sunt reprezentate de unghiul relativ la valoarea de 0 grade care este reprezentată de obicei de culoarea roșu. Saturația este reprezentată ca distanța de la centrul cercului. Culorile foarte saturate se află la extremitățile conului, cât timp lipsa saturației este concentrată în centru. Luminozitatea, sau valoarea, este reprezentată de poziția pe verticală, astfel că la vârful conului este valoarea 0, iar la baza conului este valoarea

maximă (de obicei 100).

CAPITOLUL 4

NAVIGAREA AUTONOMĂ

4.1 INTRODUCERE

Navigarea autonomă primește din ce în ce mai multă atenție și importanță în diferite arii de aplicații moderne. Astfel în această capitol ne propunem să descriem modurile de navigare autonomă existente, diferite metode de implementare și posibile soluții pentru proiectul de față. Nu vom lua în considerare metode de autocalizare folosind tehnologii de tip GPS datorită faptului că aplicația își propune aplicabilitate de interior, sau zone în care dispozitivele GPS nu sunt utile.

Principala problemă în navigarea într-o zonă fără GPS este că sistemul trebuie să fie capabil să își estimeze poziția și viteza, detectând structuri de mediu necunoscute cu acuratețe ridicată și întârzieri scăzute pentru a controla robotul într-o manieră cât mai stabilă.

Proiectul își propune realizarea unei aplicații ce se va conecta la un UGV (Unmanned Ground Vehicle) , aplicația va rula pe un computer și va avea rol de interfață. Robotul va folosi senzorii disponibili pentru a scana mediul înconjurător, va trimite datele la computer, unde acestea vor fi procesate pentru a realiza o descriere grafică a mediului traversat.

Sistemele robotice avansate, interpretează informațiile senzoriale pentru a identifica trasee de navigat corespunzătoare, dar și obstacole mobile sau imobile. Aceste sisteme realizează de obicei o mapare a mediului bazată pe input senzorial, permițând robotului să țină evidența propriei poziții chiar și când se află în medii necunoscute. Pentru un robot autonom, capacitatea de a naviga în mediul de acțiune este poate cea mai importantă caracteristică. În general navigarea poate fi definită ca o combinație a trei competențe de bază: localizarea, planificarea traseului și controlul mișcărilor.

Localizarea denotă abilitatea robotului de a-și determina propria poziție și orientare într-un plan de referință. Planificarea traseului de parcurs definește capacitatea de a calcula o secvență de comenzi de mișcare adecvate pentru a ajunge la destinația dorită. Potențialele arii de aplicații ale navigării autonome includ: automobile autonome, metode de ghidare pentru orbi, explorarea regiunilor periculoase, etc.

4.2 REPREZENTĂRI CONTINUE

Pentru a controla un robot trebuie să putem reprezenta stările robotului cu variabile cuantificabile. Având în vedere descrierea stării, modelăm mișcarea robotului cu ecuații diferențiale: Kinematics [11].

După ce obținem ecuațiile Kinematice putem dezvolta o lege de control care va trimite robotul la o locație dorită.

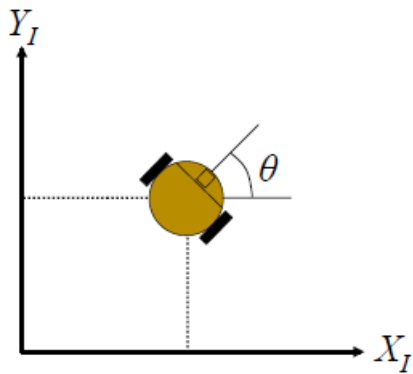
Pentru a controla robotul vom folosi sisteme de coordonate:

- Sistem de coordonate global
- Sistem de coordonate local

4.2.1 Sistem de coordonate local

Realizăm o ancorare a unei coordonate în sistemul nostru (poziția robotului). Cu această coordonată trebuie să descriem starea robotului [11]:

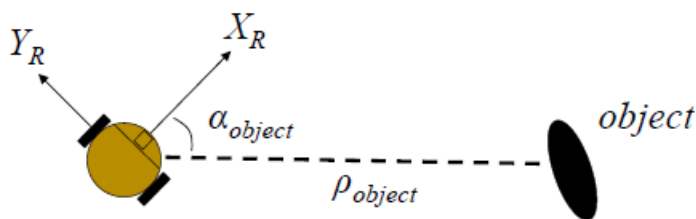
$S = [x \ y \ \theta]$,astfel starea inițială a robotului va fi $S = [0 \ 0 \ 0]$



Sistemul de coordonate local este folositor când dorim să măsurăm distanța până la un obiect din mediu.

Spre exemplu putem considera detecția unui perete folosind Scanner-ul Laser.

Măsuratoarea va fi făcută relativ la coordonatele locale ale robotului (ρ_{object} , α_{object})



Putem calcula poziția obiectului a cărei distanță am măsurat-o în coordonate locale astfel:

$$x_{object}, R = \rho_{object} \cos(\alpha_{object})$$

$$y_{object}, R = \rho_{object} \sin(\alpha_{object})$$

Există mai multe metode de determinare a poziției robotului, una din aceste metode este Odometria.

4.2.2 Odometrie

Odometria presupune folosirea senzorilor motoarelor pentru determinarea poziției robotului și a distanței parcurse [11].

După cum am prezentat mai devreme, în capitolul 2.2.4, motoarele robotului Jaguar 4x4 sunt dotate cu encodere optice, acest lucru ne poate ajuta în aplicarea tehnicilor de odometrie.

Odometria este utilizată datorită simplității în implementare, dar vine cu un dezavantaj destul de mare, în sensul că erorile se pot acumula în timp.

Sursele de erori în cazul odometriei pot proveni de la:

- Rezoluție limitată
- Diametrul roților inegal
- Variații în punctul de contact al roții
- Contact cu inegal cu podeaua, sau surse de frecare variabile.

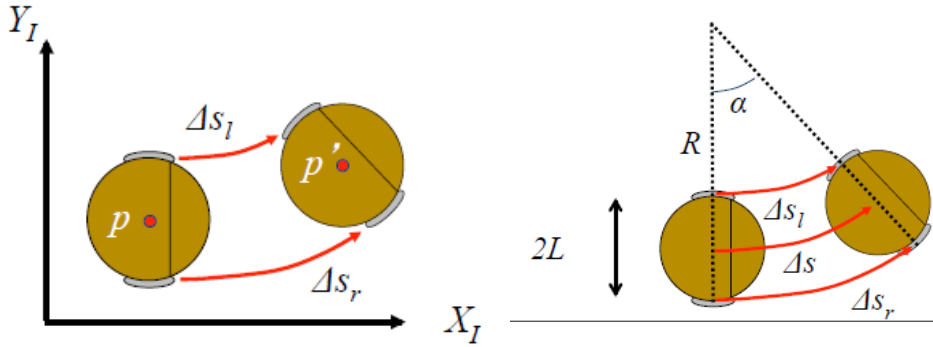
Erorile pot fi:

- Deterministe – erorile de acest tip pot fi eliminate prin calibrare potrivită

- Non-deterministe – erorile trebuie să fie descrise de modele de eroare și vor determina în permanență la estimări imprecise ale poziției robotului.

4.2.2.1 Modelarea mișcării

Dacă robotul pornește din poziția p , și roțile din stânga și dreapta se mișcă pe distanța Δs_r , Δs_l , care este poziția rezultată p' ?



Pentru a descrie, vom modela schimbarea în unghiul de mișcare și si distanța parcursă de robot Δs . Ca în figura de mai sus (partea dreaptă).

Ne asumăm că robotul se mișcă pe un arc circular de rază constantă (pentru simplificare).

Astfel, obținem:

$$\Delta s_l = R\alpha \quad \Delta s_r = (R+2L)\alpha \quad \Delta s = (R+L)\alpha$$

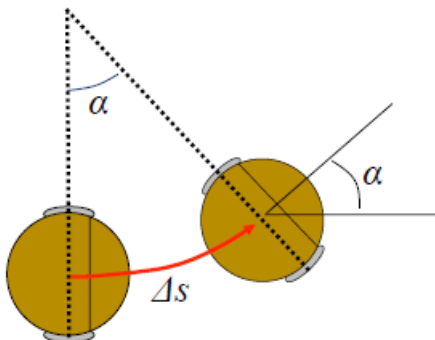
Din primele două ecuații obținem:

$$\begin{aligned} R\alpha &= \Delta s_l \\ L\alpha &= (\Delta s_r - R\alpha)/2 = \Delta s_r / 2 - \Delta s_l / 2 \end{aligned}$$

Înlocuim în cea de a treia ecuație, pentru a obține:

$$\begin{aligned} \Delta s &= (R+L)\alpha \\ &= R\alpha + L\alpha \\ &= \Delta s_l + \Delta s_r / 2 - \Delta s_l / 2 \\ &= \Delta s_l / 2 + \Delta s_r / 2 \\ &= (\Delta s_l + \Delta s_r) / 2 \end{aligned}$$

Pentru a obține variația în unghiul $\Delta\theta$, observăm că este egal cu rotația în jurul centrului cercului. Astfel putem spune că $\Delta\theta = \alpha$.



Deci rezolvăm ecuația pentru α din primele două ecuații:

$$\Delta sl = R\alpha \quad \Delta sr = (R+2L)\alpha$$

Rezultă:

$$\begin{aligned} sl / R &= \Delta sr / (R+2L) \\ (R+2L) \Delta sl &= R \Delta sr \\ 2L \Delta sl &= R (\Delta sr - \Delta sl) \end{aligned}$$

$$R = 2L \Delta sl / (\Delta sr - \Delta sl)$$

Înlocuim R astfel:

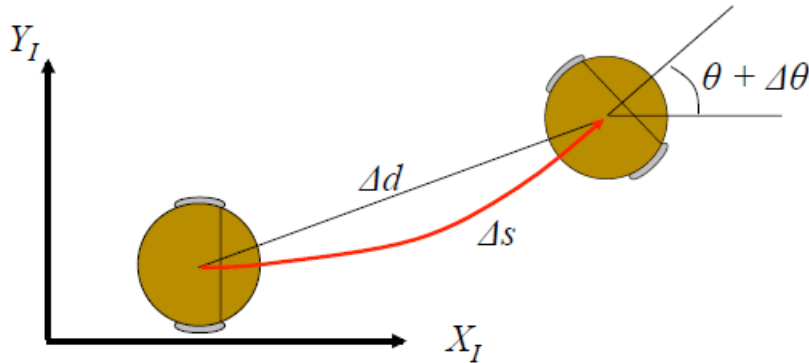
$$\begin{aligned} \alpha &= \Delta sl / R \\ &= \Delta sl (\Delta sr - \Delta sl) / (2L \Delta sl) \\ &= (\Delta sr - \Delta sl) / 2L \end{aligned}$$

Deci obținem:

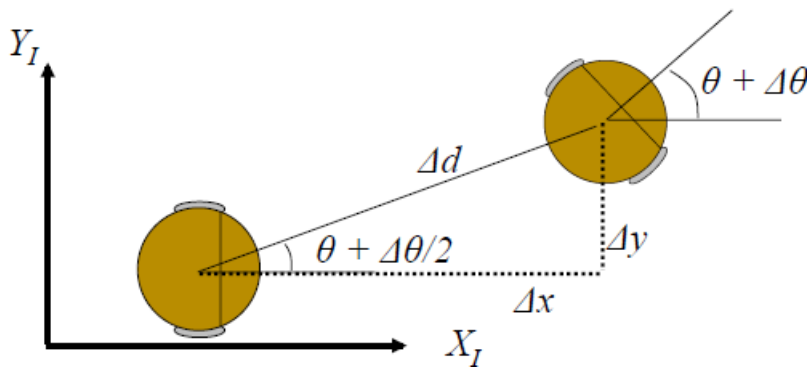
$$\Delta\theta = (\Delta sr - \Delta sl) / 2L$$

Acum, pentru că am obținut $\Delta\theta$ și Δs putem calcula poziția.

Vom folosi un nou segment de lungime Δd . Reprezentat în imaginea de mai jos.



Și acum calculăm schimbarea poziției ca funcție de Δd .



Astfel, obținem:

$$\begin{aligned} \Delta x &= \Delta d \cos(\theta + \Delta\theta/2) \\ \Delta y &= \Delta d \sin(\theta + \Delta\theta/2) \end{aligned}$$

Dacă presupunem că mișcarea este destul de mică atunci putem aproxima Δd cu Δs și obținem:

$$\Delta x = \Delta s \cos(\theta + \Delta\theta/2)$$

$$\Delta y = \Delta s \sin(\theta + \Delta\theta/2)$$

Am obținut astfel coordonatele poziției robotului folosind odometria.

4.2.2.2 Modelarea incertitudinii în mișcare

Din rezultatele obținute mai devreme extragem esențialul:

$$\Delta x = \Delta s \cos(\theta + \Delta\theta/2)$$

$$\Delta y = \Delta s \sin(\theta + \Delta\theta/2)$$

$$\Delta\theta = (\Delta sr - \Delta sl)/2L$$

$$\Delta s = (\Delta sl + \Delta sr)/2$$

Vom considera termenii care conțin delta ca erori în mișcarea roților și vom vedea cum se propagă în erorile de poziționare.

Vom exemplifica presupunând că robotul se va mișca un metru pe axa X.

Dacă:

$$\Delta s = 1 + e_s$$

$$\Delta\theta = 0 + e_\theta$$

Unde e_s și e_θ sunt erorile.

Observând ecuațiile de poziție atât pentru x cât și pentru y putem observa că ambele vor fi afectate de această eroare.

Dar se observă că termenul $\Delta\theta$ afectează în mod diferit cele două ecuații, una din ele având sinus, cealaltă cosinus, astfel, presupunem $e_\theta = 2$ grade și eroarea pe s nulă.

Obținem:

$$\Delta x = \cos(\theta + \Delta\theta/2) = 0.9998$$

$$\Delta y = \sin(\theta + \Delta\theta/2) = 0.0175$$

Robotul trebuia să se miște 1 metru pe axa X, și 0 metri pe axa Y.

Calculând erorile obținem

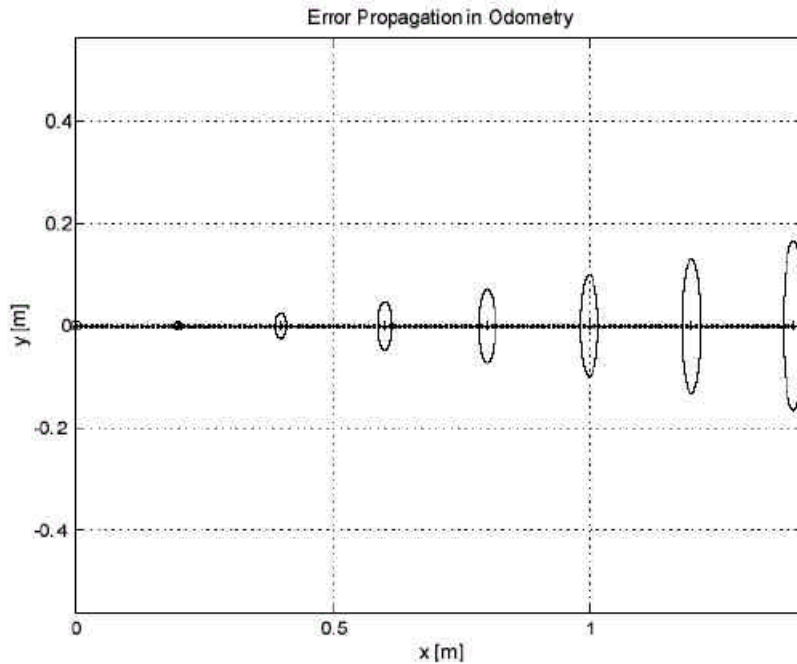
$$e_x = 0.0002$$

$$e_y = -0.0175$$

Observație:

În modul, eroarea pe axa Y este mai mare!

Erorile perpendiculare pe direcția de mers cresc mult mai mult [11].



Figură 27 Propagarea erorii în odometrie.

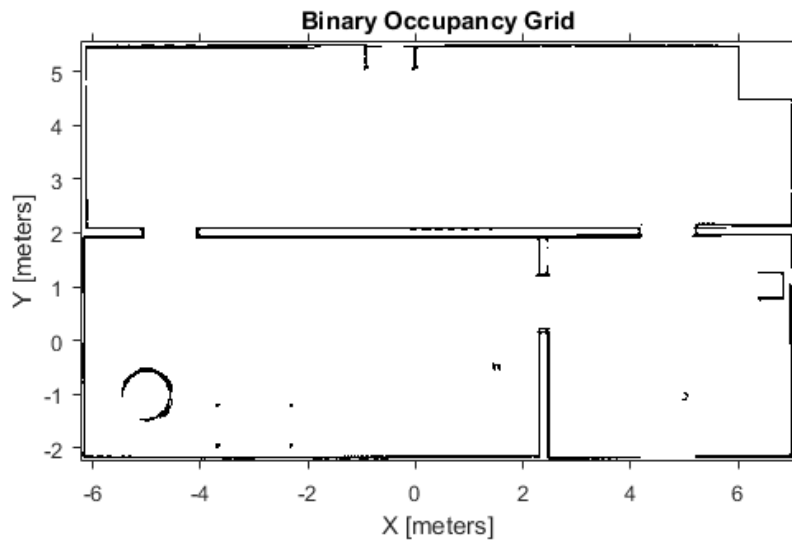
4.2.3 Maparea mediului.

Pentru ca robotul să se poate localiza și deplasa într-un mediu necunoscut, acesta trebuie să reușească percepția detaliilor importante din mediul respectiv. Printre cele mai importante astfel de detalii sunt obstacolele. După cum am arătat mai devreme, folosirea odometriei pentru determinarea poziției robotului presupune utilizarea coordonatelor, și mai exact coordonate polare, ne vom folosi de acest lucru în realizarea unei hărți a mediului în care se deplasează robotul. Există mai multe metode de realizare a unei hărți și foarte mulți algoritmi disponibili, însă, pentru a realiza o implementare cât mai sigură și simplă, vom discuta doar despre un singur tip de mapare.

4.2.3.1 Harta de ocupare

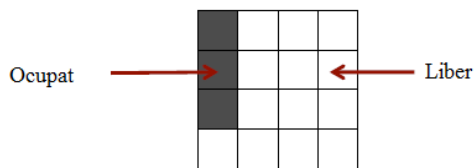
Algoritmii existenți de realizare a hărților de ocupare realizează discretizarea elementelor din mediu în celule, se obține astfel o reprezentare compactă și rigidă. Această mulțime de celule poate fi asociată unei matrici cu o dimensiune mare, fiecare element al acestei matrici poate fi: 1 – dacă există un obstacol în acel punct sau 0 – dacă locul respectiv este liber, se poate realiza și o hartă cu 3 valori, a treia valoare corespunzând spațiului necercetat. Se realizează astfel un model non-parametric care folosește resurse de memorie substanțiale, în schimb nu se bazează pe detecția de caracteristici [17].

O astfel de hartă de ocupare poate fi observată în figura de mai jos:



Figură 28 Hartă de ocupare a unei încăperi.

După cum se poate observa , bordurile reprezentate cu negru sunt obstacolele, ce este reprezentat cu alb reprezintă loc liber.



Fiecare celulă este o variabilă binară aleatoare care modelează ocuparea.

Distribuția de probabilitate a hărții este dată de produsul celulelor.

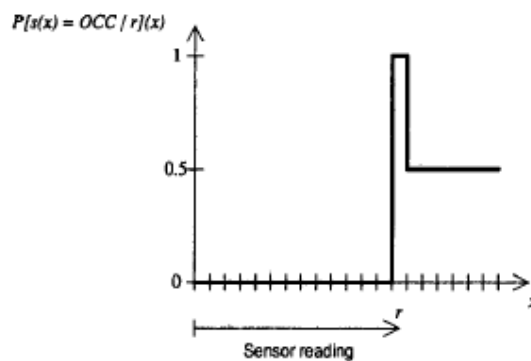
$$\text{Astfel: } p(m) = \prod_i p(m_i)$$

Unde $p(m)$ reprezintă harta iar $p(m_i)$ reprezintă celula.

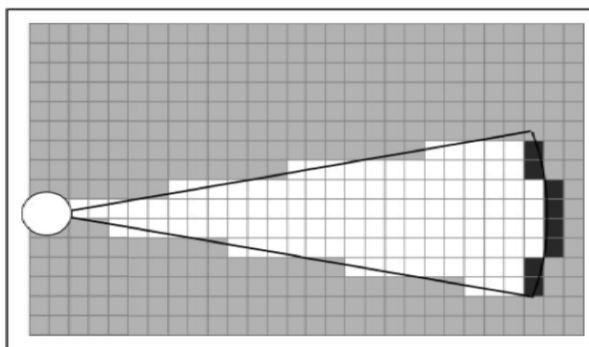
Folosind datele de la senzor $z_{1:t}$ și poziția $x_{1:t}$ la un anumit moment de timp, estimarea hărții se va face astfel:

$$p(m | z_{1:t}, x_{1:t}) = \prod_i p(m_i | z_{1:t}, x_{1:t})$$

Astfel folosind datele citite de la senzorul de distanță, vom obține ceva asemănător figurii b). Senzorul folosit în acest proiect este cel prezentat în capitolul 2.2.5.2.



Figură 29 a) Probabilitatea de ocupare pentru un senzor ideal.



Figură 30 b) Ocuparea hărții pentru un senzor ideal.

CAPITOLUL 5

DEZVOLTAREA ȘI EVOLUȚIA APLICAȚIEI

5.1 CARACTERISTICI GENERALE

Proiectul de față își propune realizarea unei aplicații de control pentru o platformă robotică mobilă. Această aplicație însă, va trebui să îndeplinească câteva obiective, și să respecte unele condiții de bază. Robotul va trebui să navigheze autonom într-un mediu necunoscut (fără hartă inițială) să realizeze o hartă a acestui mediu și să reușească detecția unui obiect ce respectă un set de caracteristici.

Aplicația va fi realizată în totalitate în limbajul de programare C# folosind mediul Microsoft Visual Studio 2010 .NET Framework.

Această aplicație va dispune de o interfață grafică ce va permite utilizatorului vizualizarea în timp real a datelor trimise de senzori, a hărții realizate și a stream-ului video de la camerele robotului. De asemenea aplicația va trebui să ofere posibilitatea controlului manual al robotului cât și posibilitatea de setare a modului autonom.

Aplicația va trebui să comunice prin conexiune Wi-Fi cu robotul, să trimită comenzile necesare de acționare a motoarelor și să primească toate datele necesare de la senzori. Se va folosi schema bloc reprezentată în Figura 10 din capitolul 2.3.

Astfel principalele caracteristici ale sistemului format vor fi:

- Pentru platforma robotică :
 - Achiziție de date de la encoderele motoarelor
 - Achiziție de date de la Scanner-ul Laser
 - Captură Video

- Pentru aplicația C#:
 - Achiziția de input de la utilizator
 - Afișarea datelor de la senzorii robotului
 - Interpretarea și procesarea datelor de la senzori pentru realizarea mișcării autonome în mediu
 - Procesarea și analiza imaginilor primite de la camerele robotului pentru detecția obiectului căutat
 - Trimiterea comenzilor necesare pentru acționarea motoarelor. Comenzi care sunt, fie bazate pe input de la utilizator , fie bazate pe informațiile procesate de la senzori.

Dezvoltarea acestui proiect s-a realizat secvențial. Datorită faptului că nu au existat alte proiecte realizate în prealabil cu această platformă etapa de documentare și familiarizare cu platforma a fost foarte importantă. Documentația robotului nu aduce detalii referitoare la dezvoltarea de aplicații, dar este însoțită de codul sursă al aplicației demonstrative prezentată în capitolul 2.3.4.

Astfel primul pas a fost familiarizarea cu platforma robotică în sine – determinarea tuturor componentelor hardware și utilizarea aplicației demonstrative.

Următorul pas a constat în familiarizarea cu limbajul de programare C#, și mediul de programare Microsoft Visual Studio 2010, din nou fără cunoștințe anterioare. Având în vedere lipsa de comentarii pentru majoritatea liniilor de cod din codul sursă al aplicației demo, dificultatea în înțelegerea logicii și a fluxului de execuție al codului a crescut considerabil.

După aceste prime etape de bază a urmat realizarea unui prototip inițial de aplicație care să realizeze partea de conectare cu robotul.

5.2 CONECTAREA LA PLATFORMA ROBOTICĂ JAGUAR 4x4

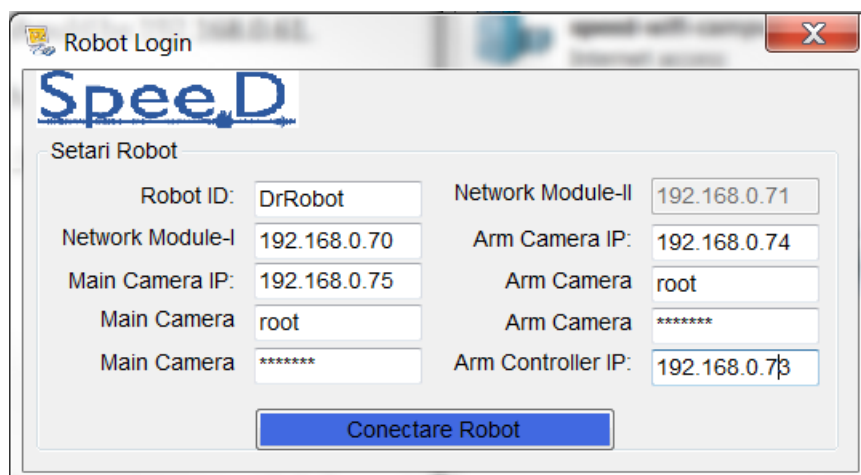
După cum a fost prezentat în capitolul 2.3.4, aplicația de control se va conecta la platforma robotică folosind o conexiune Wi-Fi. Robotul este echipat cu un router Wi-Fi care asignează fiecărei componentă principală conectată la acesta un anumit IP. IP-urile fiecărui modul vor fi folosite pentru a trimite sau primi date. Primul pas, pentru conectarea la router-ul robotului va fi schimbarea adresei IP a modului de rețea Wireless al computer-ului gazdă. IP-ul acestuia va trebui să fie static și să se afle în gama de IP-uri corectă. IP-ul modului de rețea prezent pe robot este 192.168.0.70 și restul componentelor primesc IP-uri până la 192.168.0.75. Astfel vom seta IP-ul modului de rețea Wireless la 192.168.0.76.

Pornirea robotului va determina pornirea router-ului intern și astfel va fi vizibil în lista de conexiuni posibile Wi-Fi (fiecare din roboți are asignat un ID unic, robotul folosit în acest proiect are asignat ID-ul: DriJaguarB).



Figură 31 ID Wi-Fi al robotului.

IP-urile componentelor principale pot fi observate în fereastra de conectare la robot.



Figură 32 Fereastră de conectare la platformă.

Clasa folosită pentru realizarea inițializării câmpurilor de IP-uri ale robotului este clasa `DrRobotRobotConnection.cs`. În această clasă sunt folosite metode de citire și scriere în fișiere de tip XML.

Astfel, se citește un fișier de configurare numit `OutDoorRobotConfig_4x4.xml` pentru setarea acestor IP-uri.

Se citesc datele din acest document și se scriu în text box-urile care pot fi observate în figura de mai sus. Dacă se modifică datele din text box, după apăsarea butonului *Conectare Robot*, se vor scrie datele modificate în fișierul XML.

Citirea sau scrierea datelor din XML se face folosind `System.Xml`, care pune la dispoziție funcțiile `ReadXml()` și `WriteXml()`.

Citirea datelor din textbox-uri se face astfel:

```
txtRobotID.Text = row.RobotID;
txtRobotIP.Text = row.RobotIP;
```

```

txtGPSIP.Text = row.GPSIP;
txtCameraIP.Text = row.CameraIP;
txtCameraPWD.Text = row.CameraPWD;
txtCameraID.Text = row.CameraUser;
txtDomeCameraID.Text = row.AimCameraUser;
txtDomeCameraIP.Text = row.AimCameraIP;
txtDomeCameraPWD.Text = row.AimCameraPWD;
txtArmIP.Text = row.ManipulatorIP;

```

Iar pentru scriere se vor inversa în jurul operatorului de atribuire ("=").

5.2.1 Conexiunea folosind Sockets

Pentru a realiza transmisia și recepția datelor sau a diferitelor mesaje, se va folosi System.Net.Sockets. Astfel se va crea un Socket de tip TCP pentru aplicația noastră care va juca rol de aplicație de tip client. De asemenea se vor crea și thread-uri diferite pentru transmisie și recepție. Clasa responsabilă de această conexiune va fi clasa DrRobotMainComm.

Această clasă va defini o serie de funcții pentru stabilirea conexiunii, transmiterea de mesaje și recepția de mesaje.

Pentru stabilirea conexiunii se va stabili capătul remote al socket-ului prin determinarea adresei IP din câmpurile prezentate mai devreme:

```
IPEndPoint remoteEP = new IPEndPoint(ipAddress, remotePort);
```

Se va crea socket-ul TCP:

```

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

```

Vom crea un nou thread pentru datele primite ,vom conecta socket-ul și vom porni thread-ul nou creat:

```

receiveThread = new Thread((dataReceive));
receiveThread.CurrentCulture = new CultureInfo("en-US");

clientSocket.Connect(remoteEP);
receiveThread.Start();

```

Funcția definită în această clasă pentru accesarea datelor primite va fi funcția dataReceive(). Acestea va rula `while` (StayConnected) , unde StayConnected este o variabilă de tip boolean care se setează ca fiind true după stabilirea cu succes a conexiunii.

```

if (commMethod == DrRobotConnectionMethod.WiFiComm)
{
    // wifi
    if (!replyStream.DataAvailable)
    {
        Thread.Sleep(20);
    }
}

```

```
        else
        {
            recStr = readerReply.ReadLine();
            processData();
        }
    }
```

Se stabilește tipul conexiunii în funcție de `DrRobotConnectionMethod` astfel:

```
public enum DrRobotConnectionMethod
{
    SerialComm = 0x01,
    WiFiComm = 0x02, // vom folosi conexiune WiFi.
}
```

Funcția `processData()` va realiza reîmprospătarea mesajelor:

```
void processData()
{
    if (recStr.Length == 1)
    {
        // mesaj ack, "+", "-"
        if (recStr == "+")
            _form.UpdateAckMsg(comID + "VALID");
        else if (recStr == "-")
            _form.UpdateAckMsg(comID + "INVALID");
    }
    else if (recStr.Length > 0)
    {
        _form.UpdateRcvDataMsg(comID + recStr);
    }
}
```

Pentru a transmite mesaje folosind conexiunea realizată, această clasă definește funcția `SendCommand(string)`. Această funcție va trimite datele de tip string după convertirea lor în date de tip byte:

```
Cmd += "\r\n"; // Cmd este argumentul de tip string al funcției
byte[] cmdData = ASCIIEncoding.UTF8.GetBytes(Cmd);

if (commMethod == DrRobotConnectionMethod.WiFiComm)
{
    if ((cmdData != null) && (clientSocket != null))
    {
        try
        {
            if (clientSocket.Send(cmdData) == cmdData.Length)
            {
                // ...
            }
        }
        catch { }
    }
}
```

```

        else
        {
            return false;
        }
    }
}

```

Astfel, folosind aceste funcții putem realiza conexiunea WiFi cu platforma robotică, iar comunicarea se va face prin intermediul mesajelor. Fiecare modul conectat la modulul principal de rețea are un anumit protocol de comunicare, acestea au fost discutate în capitolele anterioare.

5.3 CONTROLUL MOTOARELOR

Următorul pas în realizarea aplicației, după stabilirea conexiunii cu platforma robotică, a fost trimiterea de comenzi pentru acționarea motoarelor. Folosind informațiile prezentate în capitolul 2.2.1 putem crea în interfața grafică a aplicației un set de butoane pentru acționarea motoarelor.

În aplicația demo de control, motoarele sunt controlate folosind track-bar-uri, astfel o valoare returnată de poziția track-bar-ului va determina modificarea vitezei unui motor , prin modificarea semnalelor de tip PWM.

Astfel:

```

forwardPWM = trackBarForwardPower.Value;
turnPWM = trackBarTurnPower.Value;

```

Comenzile trimise motoarelor se vor face folosind funcția SendCommand(string) din clasa drrobotMainComm prezentată în subcapitolul anterior.

Din datele de catalog al driver-ului de motoare (cum a fost prezentat în capitolul 2.2.1) putem observa formatul mesajelor de control, astfel:

Vom atribui:

```

cmd1 = forwardPWM + turnPWM;
cmd2 = forwardPWM - turnPWM;

```

Și vom trimite mesajul:

```

string strCmd = "MMW " + "!M " + cmd1.ToString() + " "
+ (cmd2).ToString();
drrobotMainComm.SendCommand(strCmd);

```

MMW – va determina acționarea tuturor motoarelor iar valorile vor depinde de cmd1 și cmd2 (driver-ul 1 și driver-ul 2 /cele două motoare din față respectiv cele două motoare din spate).

Pentru a opri motoarele vom trimite mesajul:

```

string strCmd = "MMW !M 0 0";
drrobotMainComm.SendCommand(strCmd);

```

Folosind aceste informații am realizat o interfață simplă care controlează motoarele prin intermediul unui set butoane.

A fost creat un buton pentru deplasarea în față cu viteză constantă , deplasarea în spate cu viteză constantă, rotire la stânga pentru o perioadă fixată de timp și o viteză constantă, rotire la dreapta

pentru o perioadă fixată de timp și o viteză constantă și un buton pentru oprirea tuturor motoarelor.

Fiind un design instabil și nesigur (pentru că nu folosim metode de siguranță sau citiri ale senzorilor) testele au fost făcute cu robotul suspendat.



Figură 33 Aplicația simplă de control al motoarelor.

5.4 SCANNER-UL LASER

După etapa de control al motoarelor folosind un set simplu de butoane și comenzi am abordat partea de conexiune la scanner-ul laser și integrarea sa în aplicația de control. În primă fază am realizat conectarea la scanner-ul laser și folosind protocolul de comunicație descris în capitolul 2.2.5.2 am reușit realizarea unor măsurători inițiale.

Scanner-ul laser va folosi tot un tip de conexiune Wi-Fi:

```
private DrRobotConnectionMethod laserCommMethod =  
DrRobotConnectionMethod.WiFiComm;
```

Setăm o constantă de tip string pentru a fi folosită drept comandă de scanare (descrie în 2.2.5.2) :

```
private const string SCANCOMMAND = "GD0000108001";
```

Ca și în clasa de conectare, vom deschide un socket TCP separat pentru a schimba mesaje:

```
clientSocketLaser = new TcpClient();
```

În funcția HandleClientLaser() vom realiza toată procesarea de date necesară. Vom accesa stream-ul de informații primit prin socket-ul nou creat:

```
cmdStreamLaser = clientSocketLaser.GetStream();  
readerLaser = new BinaryReader(cmdStreamLaser);  
writerLaser = new BinaryWriter(cmdStreamLaser);
```

Citim stream-ul de date în bufferul bytes de tip byte:

```
byte[] bytes = new byte[MAX_BUFFER_LENGTH];  
// Read poate returna orice între 0 și numBytesToRead (variabilă  
// declarată).  
// Această metodă va bloca până când se va citi un byte.  
processLen = readerLaser.Read(bytes, 0, bytes.Length);
```

```
Array.Copy(bytes, 0, decodeBuff, lastPos, processLen);
```

```

processLen += lastPos;
keepSearch = true;
endPos = 0;
lastPos = processLen;

```

Apoi verificând statusul și lungimea mesajului primit, putem citi datele, le vom converti în codare de tip ASCII și le vom scrie în vectorul dataArray:

```

if ((status == "00") && (msg.Length == DATAPACKAGE_LEN))
{
    //presupunem că toate datele sunt corecte
    while (lineCnt < DATALINE - 1)
    {
        temp = dataRead.ReadLine();
temp = temp.Remove(64, 1);
        tempData =
System.Text.Encoding.ASCII.GetBytes(temp);
        Array.Copy(tempData, 0, dataArray, lineCnt *
64, tempData.Length);
        lineCnt++;
    }
    if (lineCnt == DATALINE - 1)
    {
        temp = dataRead.ReadLine();
        temp = temp.Remove(LASTLINE_DATA_LEN, 1);
//Ultima linie este formată din 40 bytes de date + suma.
        tempData =
System.Text.Encoding.ASCII.GetBytes(temp);
        Array.Copy(tempData, 0, dataArray, lineCnt *
64, tempData.Length);
    }
}

```

Pentru a converti acest șir nou creat într-o distanță măsurată în milimetri vom parcurge vectorul dataArray[], și vom rescrie elementele în vectorul disData[].

```

for (int i = 0; i < DISDATALEN; i++)
{
    disData[i] = ((long)(dataArray[3 * i] -
0x30) << 12) + (((long)(dataArray[3 * i + 1] - 0x30)) << 6) +
(long)(dataArray[3 * i + 2] - 0x30);
    if (disData[i] < Math.Abs(LASER_OFFSET_X *
1000)) disData[i] = (long)(MAX_LASER_DIS * 1000);
    laserSensorData.DisArrayData[i] =
((double)disData[i]) / 1000;
}

```

Astfel am realizat cu success nu doar conexiunea cu scanner-ul laser dar și citirea datelor măsurate.

Pentru a putea începe măsurătorile însă va trebui să schimbăm modul scanner-ului folosind o comandă de tranziție de stare, și anume, BM (descrisă în capitolul 2.2.5.2). Vom folosi funcția sendCommandLaser(string).

```

private void sendCommandLaser(string s)
{

```

```

        if (laserCommMethod ==
DrRobotConnectionMethod.WiFiComm)
        {
            try
            {
                s = s + new string((char)13, 1);

                ASCIIEncoding ascCode = new ASCIIEncoding();
                writerLaser.Flush();
                byte[] bytes = ascCode.GetBytes(s);
                writerLaser.Write(bytes, 0, bytes.Length);

            }
            catch
            {
            }
        }
    }

```

Astfel comanda de schimbare de stare în modul de măsurare poate fi apelată astfel:

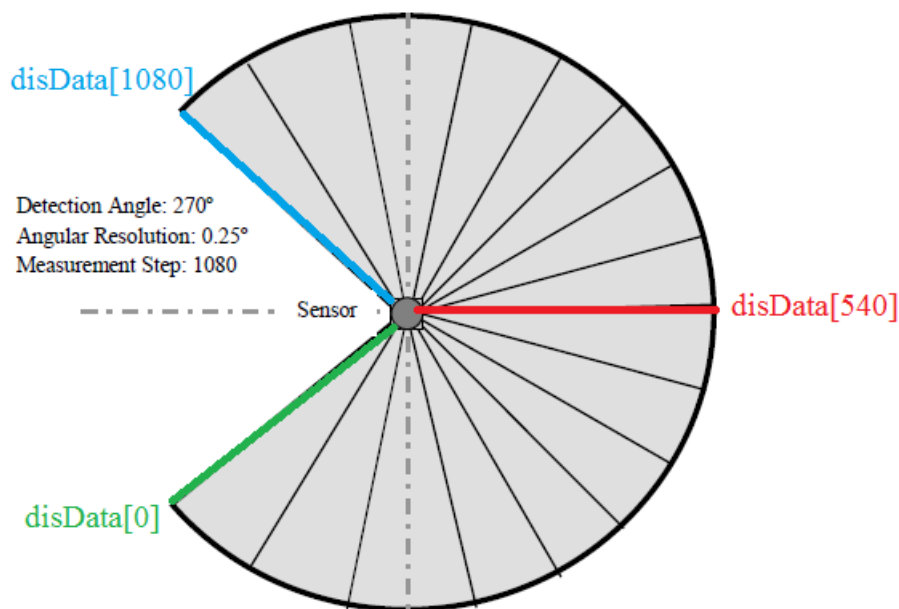
```

sendCommandLaser("BM");
sendCommandLaser(SCANCOMMAND); //Pentru a începe scanarea.

```

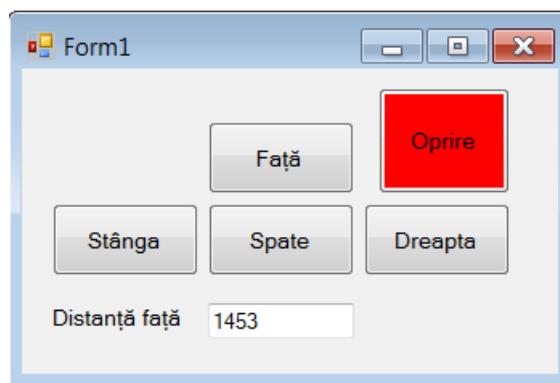
Vectorul `disData[ ]` conține 1080 de elemente, fiecare din aceste elemente reprezintă o direcție a unui fascicul laser, astfel valoarea unui astfel de element va determina distanța până la primul obstacol pe direcția respectivă.

Spre exemplu: `disData[540]` va fi un număr ce va reprezenta distanța în milimetri până la primul obstacol pentru fasciculul laser din mijloc. (reprezentare în figura de mai jos)



Figură 34 Reprezentarea grafică a vectorului de valori pentru Scanner-ul Laser.

Am folosit aceste date pentru a realiza interfața următoare a aplicației de control, unde am integrat procesarea datelor de la scanner-ul laser. Datele au fost folosite pentru a măsura distanța din fața robotului. Și programul a fost modificat pentru a face robotul să se oprească dacă în fața sa se află un obstacol la o distanță mai mică de un metru.



Figură 35 Aplicația de control cu integrarea scanner-ului Laser.

În imaginea de mai sus se poate observa, sub setul de butoane initial, un textbox. În acest textbox este afișată valoarea variabilei `disData[540]`, în cazul de față indică o distanță de 1 metru și 45 de centimetri până la cel mai apropiat obstacol.

Integrarea acestui senzor în sistem este foarte importantă, iar caracteristicile senzorului deschid orizontul către foarte multe posibilități de programare și utilizare.

5.5 STREAM-UL VIDEO

Proiectul își propune realizarea operațiilor de procesare și analiză de imagini pe imaginile primite de la camerele robotului.

Camerele platformei robotice sunt camere Axis și folosesc conexiune ethernet, stream-ul video se trimite via IP.

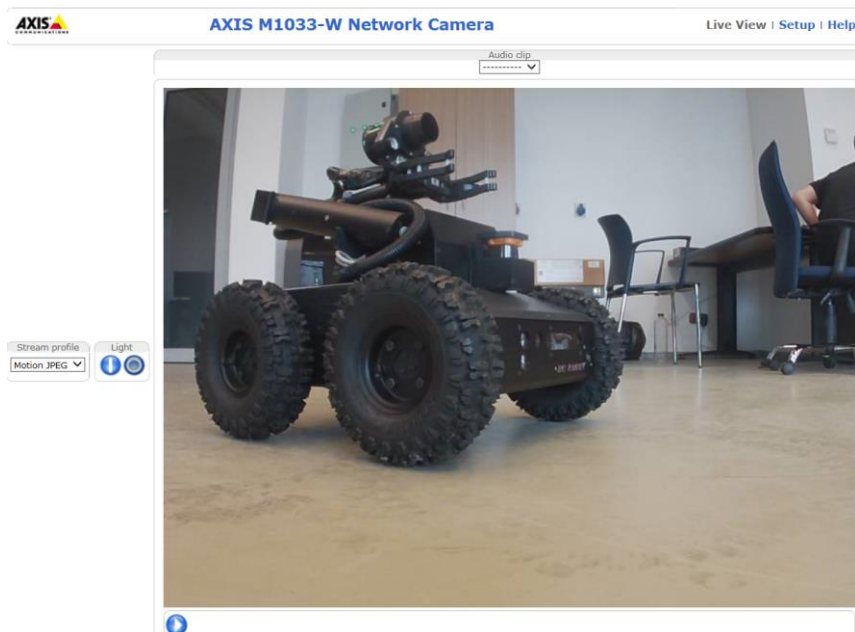
După cum a fost prezentat în capitolul 2.2.5.1 camerele sunt protejate cu username și parolă.

O dată conectați la robot (via WiFi), putem accesa stream-ul video într-un browser web, la adresa IP corespunzătoare camerei, după ce introducem datele de autentificare.

Astfel, pentru a prelua acest stream video din aplicația noastră trebuie găsită o metodă de a ne conecta folosind datele de autentificare.

În aplicația demonstrativă stream-ul video este preluat folosind o librărie specială pentru acest tip de camere `AxAXISMEDIACONTROLlib`. Problema este că folosind funcțiile acestei librării se crează o fereastră în care este afișat conținutul media, dar imaginile nu pot fi preluate și convertite în imagini care pot fi folosite de librăria OpenCV pentru realizarea procesării și analizei.

Astfel a fost nevoie de o altă soluție pentru preluarea stream-ului video.



Figură 36 Stream video accesat din browser.

După cum se poate observa (în partea stângă a ecranului) stream-ul video este codat Motion JPEG (MJPEG). Acest lucru a constituit o problemă deoarece nu am reușit să găesc o funcție (în librăriile folosite deja) pentru a accesa și afișa imaginile într-un ImageBox (un control caracteristic al Microsoft Visual Studio). Din ImageBox pot fi extrase imagini pentru a fi prelucrate folosind funcții din librăria OpenCV.

După încercarea mai multor soluții posibile am decis că cea mai simplă soluție din punct de vedere al implementării software este folosirea librăriei AForge. Această librărie este tot o librărie de prelucrare și analiză de imagini însă nu este chiar Open Source.

```
using AForge.Video; //Folosim AForge.Video

private MJPEGStream VideoStream = new MJPEGStream(); //Declaram
VideoStream de tip MJPEGStream

this.Load += new EventHandler(frmMain_Load); //Încărcăm datele
folosind funcția frmMain_Load

try
{
    VideoStream.NewFrame += new
    NewFrameEventHandler(ProcessFrame_NewFrame);
} // Accesăm un nou frame.
```

În funcția frmMain_Load() vom inițializa atributele importante ca adresa IP pentru stream-ul video (VideoStream.Source), datele de autentificare și apoi vom porni stream-ul.

```
private void frmMain_Load(object sender, System.EventArgs e)
{
    VideoStream.Source = "http://192.168.0.75:8081/axis-
    cgi/mjpg/video.cgi?resolution=320x240&req_fps=30&.mjpg";
    VideoStream.Login = "root";
    VideoStream.Password = "drrobot";

    VideoStream.Start();
```

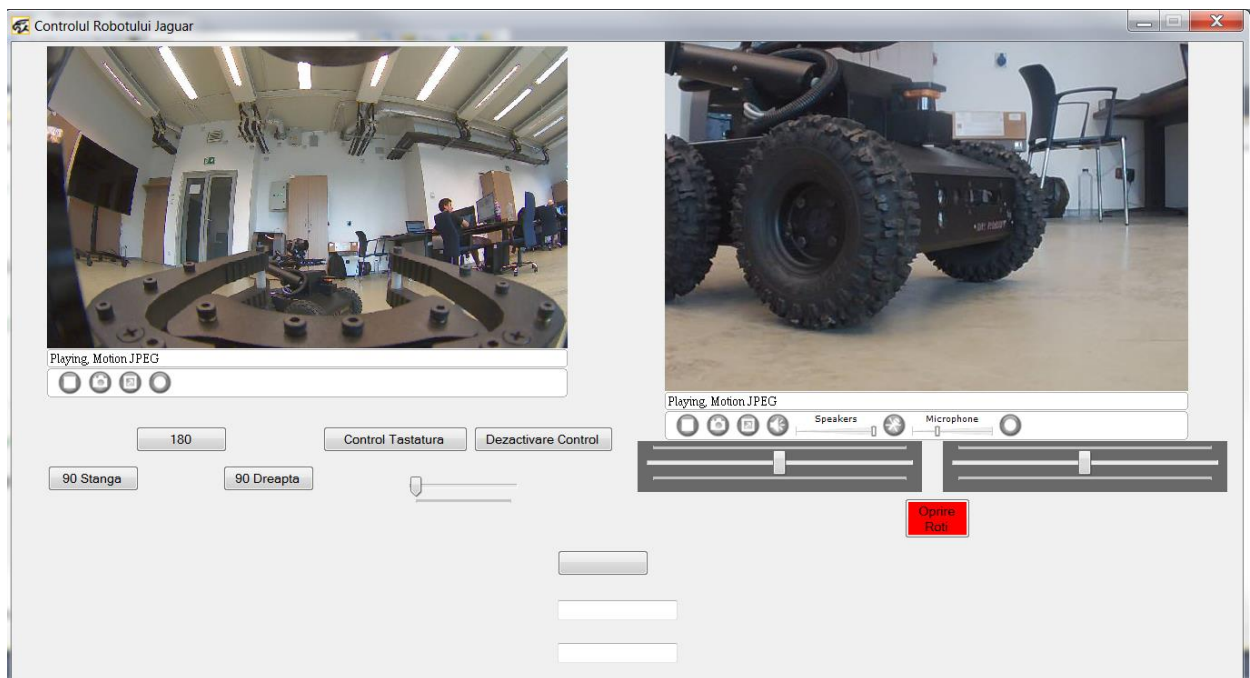
```

VideoStream1.Source =
"http://192.168.0.74:8082/axis-
cgi/mjpg/video.cgi?resolution=320x240&req_fps=30&.mjpg";
VideoStream1.Login = "root";
VideoStream1.Password = "drrobot";

VideoStream1.Start();
}

```

După finalizarea acestui pas, și implementarea în aplicația proprie a codului pentru captura video am realizat un upgrade al interfeței inițiale, adăugând mai multe tipuri de controale, metode de vizualizare a stream-ului video, textbox-uri pentru a citi date de la senzori, toate acestea au fost necesare pentru a realiza cât mai multe teste și experimente.



Figură 37 Aplicația de control actualizată.

5.6 IMPLEMENTAREA FUNCȚIILOR OPENCV

Librăria OpenCV după cum a fost prezentată în capitolul 2.3.3 este o librărie care înglobează o serie de funcții și algoritmi de analiză și prelucrare a imaginilor. Pentru acest proiect s-a propus detecția unui obiect cu caracteristici simple. Astfel robotul va trebui să navigheze autonom până la descoperirea acestui obiect.

S-a hotărât ca obiectul căutat să fie un obiect care poate fi ușor de indentificat din mai multe unghiuri sau ipostaze. O formă geometrică tridimensională care are aceeași formă în spațiul bidimensional din mai multe unghiuri, este sfera. Astfel s-a decis găsirea unei funcții potrivite pentru detecția de cercuri, implementarea în codul aplicației de control a robotului și eventuala ajustare a parametrilor pentru o detecție cât mai bună.

5.6.1 Detecția de cercuri

În cazul detecției de cercuri s-a deci folosirea transformatei Hough discutată la capitolul 3.3.3. Implementarea unei astfel de funcții în aplicația de control a presupus întâi adăugarea librăriei EmguCV (wrapper C# pentru OpenCV –discutat în 2.3.3.2).

```
using Emgu.CV;  
using Emgu.CV.Cvbk;  
using Emgu.CV.UI;  
using Emgu.CV.CvEnum;  
using Emgu.CV.Structure;
```

Astfel avem la dispoziție funcțiile de prelucrare de imagini. Pentru a putea prelucra imaginile de la stream-ul video, acestea sunt capturate folosind librăria Aforge (descrie în 5.5)

Astfel apelul:

```
VideoStream.NewFrame+=new  
NewFrameEventHandler(ProcessFrame_NewFrame1);
```

Va determina execuția funcției:

```
void ProcessFrame_NewFrame1(object sender,  
AForge.Video.NewFrameEventArgs eventArgs1)
```

Având în vedere că frame-urile au fost preluate folosind librăria Aforge, imagini pot fi convertite în imagini de tip *Bitmap* și ulterior în imagini de tip *Image<Bgr, Byte>* folosite de Emgu CV pentru prelucrare:

```
Bitmap FrameData1 = new Bitmap(eventArgs1.Frame);  
Image<Bgr, Byte> myImage1 = new Image<Bgr, Byte>(FrameData1);
```

Vom converti imaginea color în imagine cu nivele de gri, apoi vom aplica un filtru de netezire pentru a reduce zgomotul.

```
Image<Gray, Byte> graySoft = myImage.Convert<Gray,  
Byte>().PyrDown().PyrUp();  
Image<Gray, Byte> gray = graySoft.SmoothGaussian(3);  
CvInvoke.GaussianBlur(gray, smoothedFrame, new Size(3, 3), 1);
```

Vom afișa imaginea rezultată folosind funcția

```
imageBox4.Image = gray; //afișăm imaginea "gray" în imageBox4
```

Pentru a observa contururile din imagine vom folosi funcția *Canny()* care realizează aplicarea algoritmului Canny descrie la 3.3.2.

```
Image<Gray, Byte> cannyEdges1 = median.Canny(cannyThreshold.Intensity,  
cannyThresholdLinking.Intensity);
```

Parametrii *cannyThreshold* și *cannyThresholdLinking* sunt inițializați anterior și pot fi modificați pentru a găsi o valoare cât mai bună pentru condițiile de exterior de la un anumit moment de timp.

Cea mai importantă funcție în această parte a programului va fi funcția de detecție de cercuri, *HoughCircles*:

```
CircleF[] Circles = median.Clone().HoughCircles(  
cannyThreshold,  
  
circleAccumulatorThreshold,
```

```
Resolution,
MinDistance,
MinRadius,
MaxRadius
) [0];
```

Parametrii acestei funcții sunt inițializați anterior și pot fi modificați pentru găsirea unor valori optime. Parametrul `circleAccumulatorThreshold` reprezintă un acumulator care va reține câte puncte sunt necesare pentru a detecta un cerc, adică câte puncte ale unui contur se află la aceeași distanță de centru. Setarea unei valori mari pentru acest acumulator va rezulta în detecția cât mai precisă a cercurilor însă vor fi detectate mult mai rar. Setarea acumulatorului la o valoare mică va determina detecția de cercuri chiar și acolo unde probabil acestea nu sunt. Se caută o variantă de mijloc, optimă pentru diferite situații.

`Resolution` reprezintă rezoluția acumulatorului folosit în detectarea centurilor cercurilor.

`MinDistance` reprezintă distanța minimă între două centruri de cercuri detectate.

`MinRadius` reprezintă raza minimă pentru ca un cerc să fie detectat.

`MaxRadius` reprezintă raza maximă pentru ca un cerc să fie detectat.

Pentru a obține și o confirmare grafică a detecției unui cerc într-o imagine vom desena un cerc folosind funcția:

```
foreach (CircleF circle in Circles)
{
    CvInvoke.Circle(img1, Point.Round(circle.Center),
        (int)circle.Radius, new Bgr(Color.Red).MCvScalar, 2);}
```

Rezultatul afișării imaginii rezultate poate fi observat în figura de mai jos:



Figură 38 Detecția de cercuri folosind transformata Hough.

O parte foarte interesantă a acestei funcții este că ea returnează o serie de parametri foarte importanți. Funcția returnează raza cercului găsit, coordonatele centrului cercului cât și numărul de cercuri găsite în imagine.

Datorită faptului că în această aplicație dorim detecția unui singur obiect vom seta parametrul funcției `MinDistance` cu o valoare foarte mare pentru a detecta un singur cerc în toată imaginea.

Vom folosi valoarea razei și coordonatele centrului returnate de funcție pentru a obține informații despre poziția obiectului.

Informațiile astfel obținute vor fi folosite pentru a trimite comenzile necesare către motoarele robotului pentru ca acesta să se poziționeze la o anumită distanță de obiect.

Primele teste realizate folosind o astfel de implementare au fost făcute montând un laptop pe robot, acesta rula aplicația și controla robotul.



Figură 39 Aplicația de detecție a cercurilor și control al robotului.

5.6.2 Segmentarea în spațiul de culoare

După cum s-a dovedit în urma testelor realizate folosind aplicația cu detecție de cercuri, detecția se făcea greoi și programul răspundea foarte greu sau deloc la schimbări bruște ale poziției cercului în imagine, schimbări care pot fi cauzate și de deplasarea robotului în timpul căutării, făcând astfel găsirea obiectului mult mai puțin probabilă. În plus detecția unui cerc nu se poate face de la distanță foarte mare întrucât conturul acestuia nu se poate evidenția suficient de mult. De asemenea imaginea este foarte afectată de zgomot iar conturile altor obiecte pot afecta detecția conturului obiectului căutat.

În consecință am decis utilizarea unei noi constrângeri pentru obiectul de căutat, adăugarea unei noi caracteristici care să îl poată separa (nu doar prin formă) de celelalte obiecte din jur.

Deși există foarte multe soluții la dispoziție, cum ar fi folosirea unui obiect cu mai multe caracteristici și crearea unui model pentru recunoaștere, dar cea mai simplă soluție găsită, atât din punct de vedere al implementării cât și din punct de vedere al volumului de calcul a fost segmentarea în spațiul de culoare.

Culoarea obiectului căutat îl deosebește de foarte multe din obiectele înconjurătoare, astfel binarizarea imaginii în funcție de culoarea obiectului poate reduce foarte mult din informațiile inutile din imagine, în unele cazuri, după o segmentare optimă, cantitatea de informație din imagine se poate reduce doar la informația dată de obiectul căutat, adică doar informația utilă.

Implementarea s-a făcut folosind tot librăria Emgu CV. Imaginea rezultată din stream-ul video a fost convertită din imagine de tip BGR (sau RGB) în imagine de tip HSV (Hue, Saturație, Valoare), descrierile acestor spații de culoare s-au făcut în capitolele 3.4.1 respectiv 3.4.2.

```
Image<Hsv, Byte> hsvimg = img1.Convert<Hsv, Byte>();
```

Folosind funcția InRange am putut realiza segmentarea cu un set de praguri presetat de utilizator.

```
Image<Gray, byte> imageHSV = hsvimg.InRange(lowerLimit,
upperLimit);
```

Unde:

```
Hsv lowerLimit = new Hsv(a1, b1, c1);
Hsv upperLimit = new Hsv(a2, b2, c2);
```

Variabilele a1, b1, c1, a2, b2, c2 se setează folosind funcțiile delegate :

```
SetT1(a1);
SetT2(b1);
SetT3(c1);
SetT4(a2);
SetT5(b2);
SetT6(c2);
```

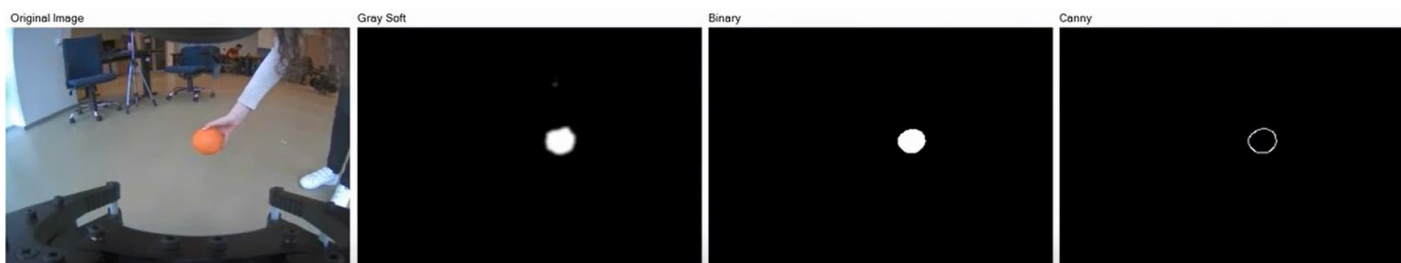
Aceste funcții modifică valorile pragurilor în funcție de valorile returnate de poziția unor track-bar-uri din interfața grafică.

Spre exemplu:

```
private void SetT5(float i)
{
    if (this.textBox1.InvokeRequired)
    {
        SetTe4 d = new SetTe4(SetT5);
        this.Invoke(d, new object[] { i });
    }
    else
    {
        this.b2 = trackBar5.Value;
        this.label18.Text = trackBar5.Value.ToString();
    }
}
```

Această funcție va seta valoarea variabilei b2.

Rezultatele obținute în urma segmentării au fost mulțumitoare, se pot observa imaginile obținute, cărora li se aplică o serie de filtre, în figura de mai jos:



Figură 40 Segmentarea în spațiul culorilor.

O pată de o anumită culoare poate fi detectată de la o distanță mai mare decât cea la care poate fi detectat un cerc, astfel, s-a dorit găsirea unei soluții pentru detecția rapidă a unei pete de culoare.

S-a decis utilizarea funcției de BlobTracking din EmguCV disponibilă în:

```
using Emgu.CV.VideoSurveillance;
```

Implementarea acestei funcții s-a realizat după cum urmează.

Au fost declarate variabilele necesare:

```
private static BackgroundSubtractor _fgDetector;  
private static Emgu.CV.Cvbe.CvBlobDetector _blobDetector;  
float maxblob;  
CvBlob largestContour;
```

Pentru inițializare s-a pus în try-ul principal:

```
_fgDetector = new  
Emgu.CV.VideoSurveillance.BackgroundSubtractorMOG2();  
_blobDetector = new CvBlobDetector();
```

S-au convertit imaginile și apoi au fost aplicate filtrele corespunzătoare, după care a fost realizată binarizarea imaginii.

```
Image<Gray, byte> imageHSV = hsvimg.InRange(lowerLimit,  
upperLimit);  
  
Image<Gray, Byte> graySoft =  
imageHSV.Convert<Gray, Byte>().PyrDown().PyrUp();  
Image<Gray, Byte> gray =  
graySoft.SmoothGaussian(3);  
gray = gray.AddWeighted(graySoft, 1.5, -0.5, 0);  
  
Image<Gray, Byte> bin = gray.ThresholdBinary(new  
Gray(70), new Gray(255));
```

După care urmează detecția propriu-zisă și afișarea pe imagine a unui contur pentru a încadra obiectul găsit, programul a fost realizat astfel încât să fie detectat doar cel mai mare blob din imagine:

```
CvBlobs blobs = new CvBlobs();  
_blobDetector.Detect(median, blobs);  
blobs.FilterByArea(1, int.MaxValue);  
maxblob = 0;  
#endregion  
  
if (blobs.Count != 0 && autonom == true)  
{  
    Checkautonom(false);  
  
    foreach (var pair in blobs)  
    {  
        if (pair.Value.Area > maxblob)  
        {  
            maxblob = pair.Value.Area;  
            largestContour = pair.Value;  
        }  
    }  
}
```

```

    }
}
CvBlob blo = largestContour;

CvInvoke.Rectangle(img, blo.BoundingBox, new
MCvScalar(0.0, 50.0, 255.0), 2);

```

Funcția va returna dimensiunile dreptunghiului în care va fi încadrat obiectul. Vom primi informații despre poziția centrului și aria dreptunghiului.

Aria va fi folosită pentru a decide asupra distanței până la obiect (poziția pe axa Z), iar coordonatele centrului vor fi folosite pentru a determina poziția pe axa X.

Se va putea implementa asemănător implementării detecției de cercuri, un program pentru controlul robotului în funcție de poziția și dimensiunea obiectului din imagine.



Figură 41 Detectarea unui obiect de o anumită culoare.

Implementarea acestei funcții de deplasare se va realiza astfel:

```

if (blo.Area > 4500 && blo.Area < 6500)
{
    if (!protectMotorTemp)
    {
        SetSpeed(0);
        UnCheckautonom(true);
        Checkautonom(false);
    }
}
else
{
    UnCheckautonom(false);
}
if (blo.Area <= 5000)
{
    if (!protectMotorTemp)
    {
        SetSpeed(65);
    }
}
if (blo.Area >= 6000)
{

```

```
        if (!protectMotorTemp)
        {
            SetSpeed(-65);
        }
    }

    if (blo.Centroid.X >= 90 && blo.Centroid.X <= 140)
    {
        if (!protectMotorTemp)
        {
            SetTurn(0);
        }
    }

    if (blo.Centroid.X > 140)
    {
        if (!protectMotorTemp)
        {
            SetTurn(200);
        }
    }

    if (blo.Centroid.X < 90)
    {
        if (!protectMotorTemp)
        {
            SetTurn(-200);
        }
    }
}

else
{
    Checkautonom(true);
}

}
```

Unde Funcțiile SetTurn(float) și SetSpeed(float) realizează acționarea motoarelor, pentru întoarcere stânga/ dreapta, respectiv deplasare față/ spate.

Se observă de asemenea și utilizarea fanionului de verificare a temperaturii motoarelor pentru a ne asigura funcționarea în condiții sigure.

5.7 DEPLASAREA ROBOTULUI ȘI HARTA DE OCUPARE

În modul său de deplasare autonom, robotul trebuie să depisteze obstacolele și să evite coliziunea cu acestea. Deși inițial s-a dorit o implementare mai complexă a modului de parcurgere a mediului, s-a decis asupra unui mod de deplasare simplist din motive legate de timpul necesar de implementare.

5.7.1 Deplasarea robotului

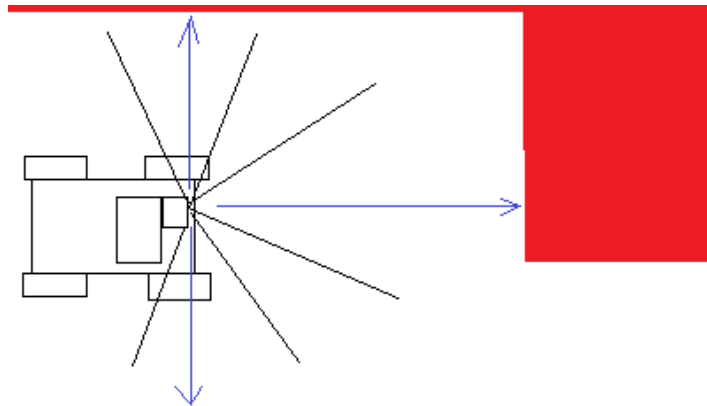
Robotul va folosi informațiile primite de la scanner-ul Laser pentru a realiza detecția obiectelor din jurul său și pentru a lua deciziile necesare evitării lor. Algoritmul de deplasare este unul foarte simplu și se bazează strict pe informațiile din vectorul disData[] prezentat în capitolul 5.4.

Deși foarte simplu această funcție realizează o acoperire destul de bună a unei încăperi. Există posibilitatea blocării în anumite puncte însă unele variabile au fost ajustate astfel încât să se reducă aceste probabilități.

Se împarte în 3 părți aria de acoperire a scanner-ului laser, astfel vor rezulta 3 vectori. Un vector ce va corespunde părții din față, un vector ce va corespunde lateralei din partea dreaptă, iar celălalt lateralei din partea stângă.

Se vor parcurge acești vectori și se va cauta valoarea minimă dintre valorile elementelor curente.

În timpul deplasării, dacă se descoperă, pentru vectorul ce reprezintă partea din față, un minim mai mic decât o valoare presetată (de obicei 1000 sau 1500 –reprezentând 1 metru respectiv 1,5 metri) se va opri robotul și se va seta fanionul de verificare_laterală. O dată setat acest fanion se vor verifica ariile laterale scanate de laser și va comanda rotirea robotului către partea cu valori minime mai mari. Spre exemplu, conform imaginii de mai jos, robotul detectează un obstacol în față și decide rotirea către dreapta datorită faptului că există un obstacol în partea stângă.



Figură 42 Modul de deplasare al robotului.

Implementarea acestei funcții se va realiza astfel:

```
bool verificare_laterală = false;
bool verificat = false;

if (minim_laser < 1600) // fata < 1200 || disData[570] < 1200 ||
{
    trackBarForwardPower.Value = 0;
    trackBarTurnPower.Value = 0;
    verificare_laterală = true;
    textBox4.Text = "S-a oprit robotul";
}
else
{
    // textBox5.Text = "";
    trackBarForwardPower.Value = 150;
    trackBarTurnPower.Value = 0;
    textBox4.Text = "robotul CAUTA";
    verificare_laterală = false;
}

if (verificare_laterală)
{
    if (stanga < dreapta)
    {
        trackBarTurnPower.Value = -500;
    }
}
```

```

        verificat = true;
        verificare_laterala = false;
        //textBox4.Text = "robotul stanga";
    }

    else
    {
        trackBarTurnPower.Value = 500;

        verificat = false;
        verificare_laterala = false;
        //textBox4.Text = "robotul dreapta"
    }

```

5.7.2 Harta de ocupare

În ideea realizării unui mod de deplasare mult mai intuitiv și eficient s-a decis realizarea unei hărți de ocupare (prezentată în capitolul 4.2.3.1). Deși încă nu joacă un rol în deplasarea propriu-zisă a robotului înțelegerea modului de implementare și a teoriei poate ajuta în dezvoltarea unei aplicații care să se bazeze pe astfel de informații.

Principiul se bazează pe crearea unei matrici în care elementele de la anumite poziții își vor modifica valorile corespunzător cu valorile distanțelor returnate de senzorul laser.

Se vor folosi coordonate polare, poziția robotului însă nu va fi estimată, acesta va rămâne reprezentat în centrul hărții.

Vom extrage informațiile legate de unghi, și coordonatele pe axele X și Y doar din datele de la scanner-ul laser. Astfel cunoscând fiecare pas al unghiului pentru fasciculele laser consecutive putem determina unghiurile pentru fiecare din cele 1080 de elemente.

```

for (int i = laserConfigData.StartStep; i < laserConfigData.EndStep; i++)
{
    if ((laserSensorData.DisArrayData[i] >=
laserConfigData.MinDis)
        && (laserSensorData.DisArrayData[i] <
laserConfigData.MaxDis))
    {
        // Vom actualiza celulele neocupate, MAPRATIO este
        folosit pentru a trece din metri în centimetri.
        angle = i * laserConfigData.AngleStep +
laserConfigData.StartAngle; // calculul unghiului în funcție de AngleStep.
        //x
        dblX1 = laserSensorData.DisArrayData[i] * MAPRATIO *
Math.Sin(angle) + laserConfigData.OffsetX;
        //y
        dblY1 = -laserSensorData.DisArrayData[i] * MAPRATIO *
Math.Cos(angle) + laserConfigData.OffsetY;

```

Vom face trecerea în coordonate globale, nu vom actualiza poziția robotului, aceasta va rămâne în permanență (0,0,0).

```

        dblX2 = dblX1 * Math.Cos(currentPos.robotDir) - dblY1 *
Math.Sin(currentPos.robotDir);
        dblY2 = dblX1 * Math.Sin(currentPos.robotDir) + dblY1
* Math.Cos(currentPos.robotDir);

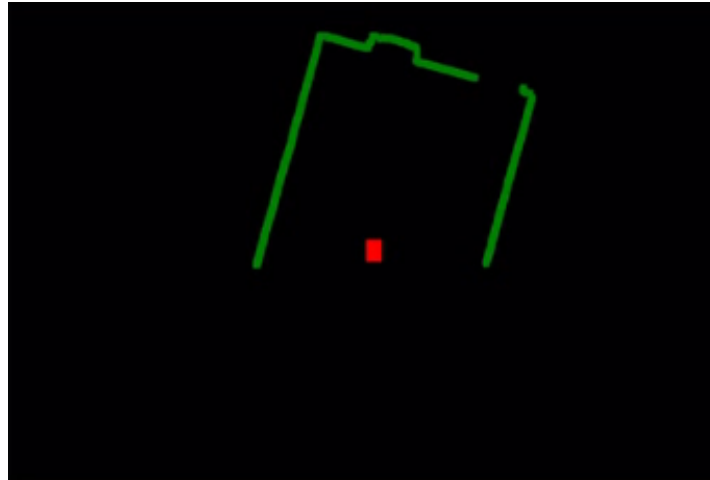
        end_x = (int) (dblX2 + Math.Sign(dblX2) * 0.501) +
MAPCENT;
        end_y = (int) (dblY2 + Math.Sign(dblY2) * 0.501) +
MAPCENT;

```

Pentru a actualiza harta folosind datele rezultate vom apela funcția:

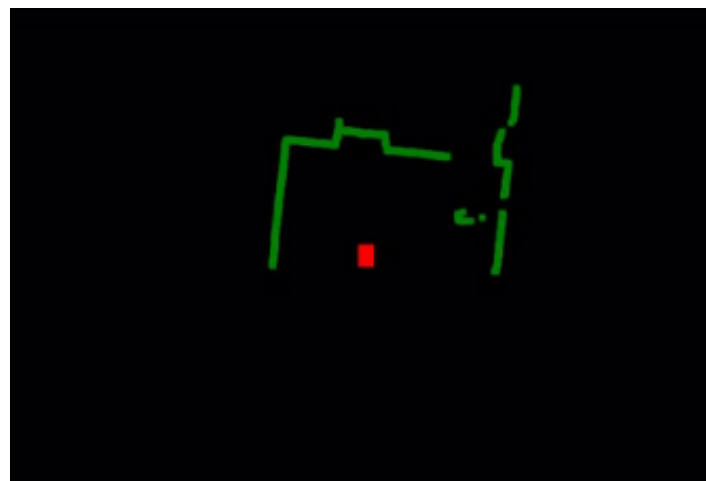
```
OCC_MAP_EMPTY_Update(start_x, start_y, end_x, end_y);
```

În această funcție vom realiza actualizarea hărții pentru toate punctele detectate. Partea matematică și interpretarea datelor în această parte a programului a fost deja implementată, vom exemplifica vizual rezultatul acestei implementări.



Figură 43 Implementarea hărții de ocupare a).

Se poate observa în figura de mai sus, reprezentat cu roșu este robotul iar cu verde sunt pereții unei camere.



Figură 44 Implementarea hărții de ocupare b).

În imaginea b) se poate observa că poziția robotului este aceeași (în centrul imaginii) dar datorită deplasării în mediu acesta a actualizat datele citite de la senzorul laser.

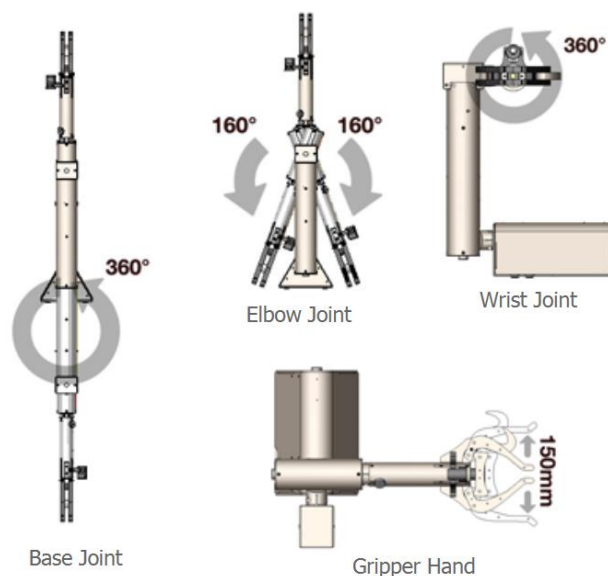
În momentul de față deplasarea robotului nu este influențată de informațiile din hartă dar implementările viitoare vor fi realizate în sensul obținerii unei astfel de deplasări. Folosind informațiile de odometrie prezentate în capitolul 4.2.2 putem actualiza poziția robotului în hartă și obținerea unei hărți a unei încăperi.

5.8 UTILIZAREA BRAȚULUI ROBOTIC

Platforma robotică jaguar 4x4 este de asemenea dotată cu un braț robotic, se pot observa detalii despre așezarea și distribuția încheieturilor în imaginile de la capitolul 2.1. Pentru a fi posibilă prinderea obiectului căutat după etapa de detecție a fost necesară studierea modului de funcționare a brațului robotic. Dificultatea a provenit din faptul că în interfața grafică a aplicației demonstrative nu existau butoane de control pentru braț. Singurul mod prin care brațul putea fi

acționat era prin utilizarea Gamepad-ului (2.1). Cum am menționat mai devreme, pentru a putea face ca aplicația să funcționeze în totalitate este nevoie de controlul brațului, astfel am decis realizarea unor butoane de control pentru braț în interfața grafică a robotului, cu scopul de a mă familiariza cu modul de acționare și pentru a putea face posibilă acționarea autonomă.

Motoarele brațului sunt setate (by default) să funcționeze în modul Position Control –descriș la capitolul 2.2.3, acest mod folosește encoder-ele optice pentru determinarea poziției. Controlul brațului se va face, în consecință, fie incrementând valorile encoderelor, fie setând o anumită valoare pentru aceste encodere pentru a trimite brațul la o poziție dorită. Brațul nu dispune de toate gradele de libertate, acesta având la dispoziție doar mișcările în planul XY, mișcarea pe axa Z fiind posibilă rotind baza robotului (folosind roțile). Modul de mișcare al brațului se poate observa în figura de mai jos.



Figură 45 Mișcarea încheieturilor brațului robotic.

Din figură putem observa că există două încheieturi importante pentru a mișca brațul, Base Joint și Elbow Joint. Acestea sunt folosite pentru a deplasa brațul în sus sau în jos.

Datorită faptului că brațul este dotat cu o camera video (deasupra cleștelui robotului) a fost posibilă folosirea informațiilor din imagine pentru a realiza detecția de formă prezentată în 5.6.1. Astfel informațiile legate de poziția pe axa Y a centrului cercului detectat au putut fi folosite pentru a mișca încheietura Joint 2 (Elbow Joint), iar aria cercului detectat a fost folosită pentru a controla încheietura Joint 1 (Base Joint). Efectul dorit a fost acela ca pentru o arie și o poziție pe axa Y dorită brațul să se apropie de obiectul căutat până când este suficient de aproape pentru a putea fi prins.

Pentru a trimite o comandă de mișcare unui motor dotat cu encoder al brațului se va seta o valoare de mișcare astfel:

```
jointMotorData[0].jointCmd = -200;
drRobotCommMot1.SendCommand("!PR 1 " +
jointMotorData[0].jointCmd.ToString() + "\r");
```

Trimiterea acestei comenzi către driver-ul de control al motoarelor brațului va determina mișcarea încheieturii 1 (Joint 1- Base Joint) cu aproximativ 13 grade în sus.

Am decis utilizarea unei metode incrementale sau decrementale de utilizare a acestui mod de control, astfel valoarea encoder-ului va crește sau va scădea cu o valoare constantă (mică – 1 sau 2 grade) într-o anumită perioadă de timp, pentru a atinge poziția dorită. Poziția dorită nu

este una fixă, ea depinde de poziția obiectului în imagine, și variază ușor în funcție de etapa de poziționare care permite un interval de eroare.

Implementarea controlului brațului a fost realizată în interfața grafică. Se poate observa în imaginea de mai jos, modul de dispunere al butoanelor de control.



Figură 46 Dispunerea în interfață a butoanelor de control al brațului robotic.

Implementarea pentru butonul "sus Joint1" se poate observa mai jos:

```
private void button12_Click(object sender, EventArgs e)
{
    ...
    jointMotorData[0].jointCtrl = true;
    jointMotorData[0].jointCmd = -JOINT1_CMD-20;
    if (!jointMotorData[0].protect)
        drRobotCommMot1.SendCommand("!PR 1 " +
        jointMotorData[0].jointCmd.ToString() + "\r");
}
```

Se poate observa folosirea fanionului `jointMotorData[0].protect` acesta va verifica dacă encoderul motorului indică aceeași poziție ca poziția la care a fost trimis brațul, dacă acestea nu coincid înseamnă că brațul este blocat sau obstrucționat de un obiect, dacă acesta este cazul atunci nu se vor mai trimite comenzi de mișcare. Aceasta este o metodă de a ne asigura de buna funcționare a motoarelor prin intermediul encoderelor. Alte mijloace de asigurare a bunei funcționări, ca verificarea senzorilor de temperatură, sunt de asemenea folosite.

În următoarea etapă, după ce am reușit controlul brațului folosind niște butoane simple am decis implementarea codului necesar mișcării autonome a brațului pentru a reuși poziționarea față de un obiect sferic.

Setând astfel un interval pentru raza dorită și un interval pentru pozițiile pe axa Y putem mișca brațul din încheieturile 1 și 2 pentru a ajunge lângă obiectul căutat, folosind metoda de detecție de cercuri și camera brațului.

Părți ale codului folosit pentru controlul brațului în mod autonom este prezentat mai jos, au fost adăugate comentarii pentru a face înțelegerea codului mult mai ușoară.

```
CircleF[] Circles = median.Clone().HoughCircles(
    cannyThreshold,
    circleAccumulatorThreshold,
    Resolution,
    MinDistance,
    MinRadius,
    MaxRadius)[0]; //Funcția de recunoaștere de formă
```

```

if (Circles.Length == 1 && poz_J1 == true && checkBox2.Checked == true)
{
    cerc_gasit = true; //sunt verificate și setate anumite fanioane care
    controlează fluxul de execuție al programului.

    foreach (CircleF circle in Circles)
    {
        a = circle.Center; //se salvează centrul și raz
        b = circle.Radius;
        CvInvoke.Circle(img1, Point.Round(circle.Center), (int)circle.Radius, new
        Bgr(Color.Red).MCvScalar, 2);
        imageBox2.Image = img1; //desenăm și afișăm în imagine cercul.
        SetText1(b.ToString())
        if (a.X > 130 && a.X < 190) //verificăm dacă robotul s-a poziționat
        corect pe axa X, dacă da putem începe rutina de prindere.
        {
            if (b < 53)
            {
                if (b < 30)
                {
                    JOINT1_CMD = 4;
                    JOINT2_CMD = 4; //setăm cu ce valoare vom incrementa
                    /decrementa encoderele , astfel pentru o valoare mare a razei vom mișca mai rapid, iar
                    pentru o valoare mai mică a razei vom mișca mai ușor brațul

                }
                else
                {
                    JOINT1_CMD = 1;
                    JOINT2_CMD = 1;
                }
                ...

                jointMotorData[0].jointCtrl = true;
                jointMotorData[0].jointCmd = -JOINT1_CMD;
                jointMotorData[0].jointJoy = true;
            }
        }
        if (!jointMotorData[0].protect)
            drRobotCommMot1.SendCommand("!PR 1 " +
jointMotorData[0].jointCmd.ToString() + "\r"); //se trimite comenzile
        if (b > 70) //comenzi în funcție de valorile razei -Joint 1
        {
            ....
            jointMotorData[0].jointCtrl = true;
            jointMotorData[0].jointCmd = JOINT1_CMD;
            jointMotorData[0].jointJoy = true;
        }
    }
    if (!jointMotorData[0].protect)
        drRobotCommMot1.SendCommand("!PR 1 " +
jointMotorData[0].jointCmd.ToString() + "\r");
}

if (a.Y < 165) //comenzi în funcție de valoarea lui Y -Joint2
{ ...
    jointMotorData[1].jointCmd = -JOINT2_CMD;
    jointMotorData[1].jointCtrl = true;
    joyButtonCnt[1] = 0;
    jointMotorData[1].jointJoy = true;
}

```

```

    }

    if (!jointMotorData[1].protect)
    {
        //textBox5.Text = "miscare sus";

        drRobotCommMot1.SendCommand("!PR 2 " +
jointMotorData[1].jointCmd.ToString() + "\r");

    }
}
if (a.Y > 177)
{
    ...
    jointMotorData[1].jointCmd = JOINT2_CMD;
    jointMotorData[1].jointCtrl = true;
    joyButtonCnt[1] = 0;
    jointMotorData[1].jointJoy = true;
}

if (!jointMotorData[1].protect)
{
    drRobotCommMot1.SendCommand("!PR 2 " +
jointMotorData[1].jointCmd.ToString() + "\r");
}
}
if (a.Y <= 177 && a.Y >= 165) //intervalul de acceptare,
oprim acționarea brațului.
{
    ...
    jointMotorData[1].jointCmd = 0;
    jointMotorData[1].jointCtrl = true;
    joyButtonCnt[1] = 0;
    jointMotorData[1].jointJoy = true;
}

if (!jointMotorData[1].protect)
{
    drRobotCommMot1.SendCommand("!PR 2 " +
jointMotorData[1].jointCmd.ToString() + "\r");
}
}
if (b >= 52 && b <= 70)
{
    counter_cerc++;
    if (counter_cerc > 30) //metodă de verificare a acurateții
    {
        if (test == true)
        {
            test = false;
        }
        if (prindere == true)
        {
            prindere = false;
            strange(); //începe rutina de strângere.

```

}

Motorul care acționează Gripper-ul (Cleștele) brațului nu este setat în modul Position Control ci este în modul PWM, astfel se va trimite o valoare care va fi proporțională cu viteza lui de mișcare. Acest lucru a făcut dificilă implementarea rutinei de prindere deoarece nu există nici o posibilitate de a verifica când să opri motorul (nu avem un senzor de presiune pe clește sau alți senzori asemănători) astfel motorul trebuie să strângă obiectul găsit pentru o perioadă de timp suficient de lungă pentru a strânge suficient de tare. Am decis astfel utilizarea unui timer. Implementarea actuală nu conține un mod de adaptare a timpului de strângere în funcție de dimensiunile obiectului găsit, dar o astfel de implementare va fi efectuată ulterior.

Timpul de prindere va fi setat în prealabil. Implementarea a fost dificilă deoarece una din clasele principale care pun la dispoziție timere, și anume clasa Timer nu permite oprirea timer-ului după o perioadă fixată, s-a decis astfel utilizarea System.Timers.

Se va declara un timer:

```
public System.Windows.Forms.Timer tempo= new
System.Windows.Forms.Timer();
```

În funcția strange() vom porni timer-ul, care după o perioadă de timp va declanșa executarea unui eveniment:

```
tempo.Interval = 7000;
tempo.Enabled = true;
tempo.Elapsed +=new ElapsedEventHandler(tempo_Elapsed);
tempo.Start();
```

În evenimentul astfel declanșat vom opri timer-ul și vom trimite o comandă de oprire a gripper-ului:

```
void tempo_Elapsed(object sender, ElapsedEventArgs e) {

    tempo.Enabled = false;
    tempo.Stop();
    //comandă de oprire a motorului gripper-ului.
    drRobotCommMot2.SendCommand("!G 2 0\r");
    terminat.Enabled = true;
    terminat.Elapsed +=new ElapsedEventHandler(terminat_Elapsed);
    terminat.Start();}
```

Terminat_Elapsed este un eveniment, declanșat de alt timer care realizează rutina de finalizare a misiunii (ridicarea brațului).

Această implementare a permis astfel prinderea obiectului căutat folosind brațul robotului și informațiile din imagine, după realizarea fazei de căutare și poziționare.

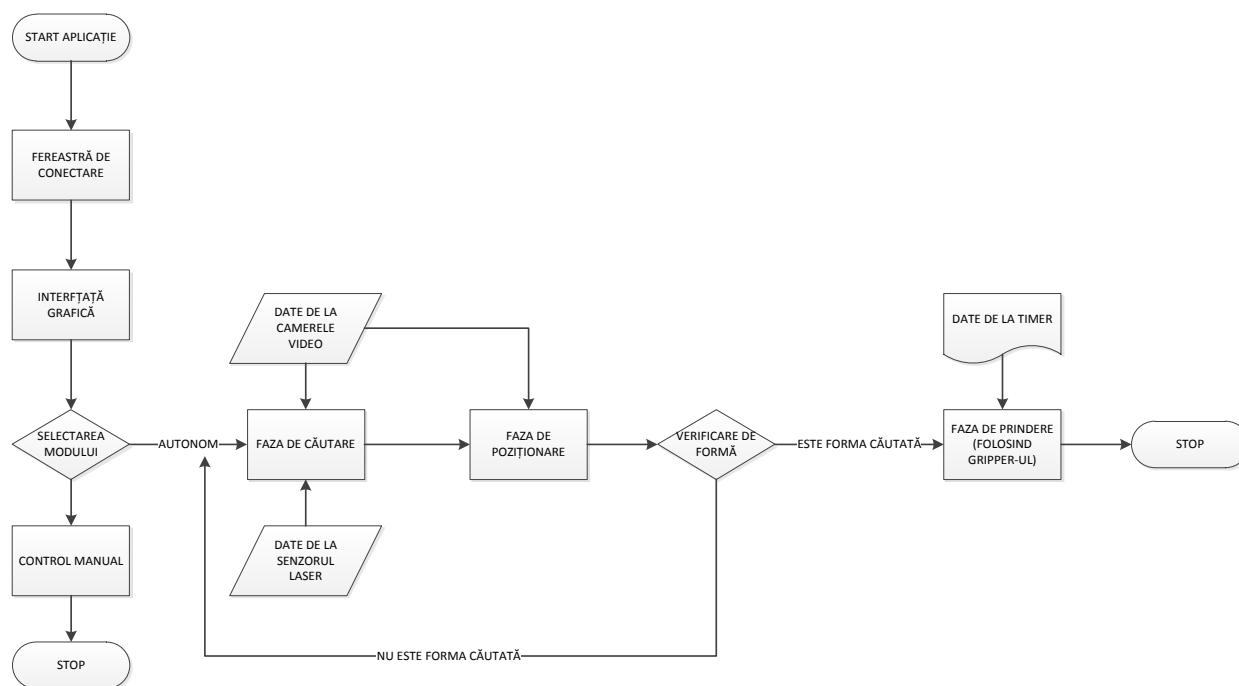
Astfel toate aceste implementări prezentate anterior au putut fi folosite pentru a crea o aplicație ce va permite funcționarea autonomă a robotului și îndeplinirea misiunii sale.

5.9 MODUL DE FUNCȚIONARE A APLICAȚIEI FINALE

Implementarea actuală a proiectului unește toate aspectele discutate anterior în acest capitol într-o aplicație care poate realiza navigarea autonomă și detecția de obiecte, au fost realizate o serie de teste din care s-a tras concluzia că, în interacția curentă aplicația rulează suficient de bine, reușind atingerea obiectivului.

Fazele de funcționare ale aplicației sunt după cum urmează:

- Faza 1: Faza de căutare, în această fază robotul caută obiectul în mediu folosind algoritmul de căutare descris în capitolul 5.7.1.
- Faza 2: Faza de poziționare, în această fază robotul a descoperit un obiect de culoarea celui căutat (capitolul 5.6.2) și se poziționează la o anumită distanță față de acesta.
- Faza 3: După poziționare, datorită apropierii de obiect se poate realiza verificarea formei acestuia folosind transformata Hough (capitolul 5.6.1). Dacă se decide că obiectul este cel căutat se realizează rutina de prindere. Altfel se reia căutarea.



Figură 47 Diagrama funcționare a aplicației.

Diagrama de mai sus exemplifică într-un mod general fluxul de execuție în cazul utilizării aplicației în interacția curentă.

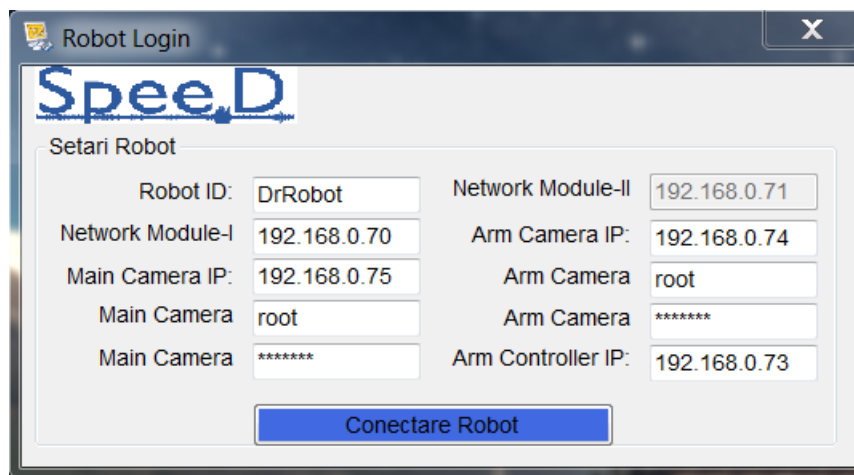
Interfața grafică a fost remodelată pentru a cuprinde cât mai mult informații și pentru a putea permite utilizatorului acces la cât mai multe informații.

Având în vedere diagrama de funcționare de mai sus, vom exemplifica, vizual, modul de utilizare și fluxul de funcționare:



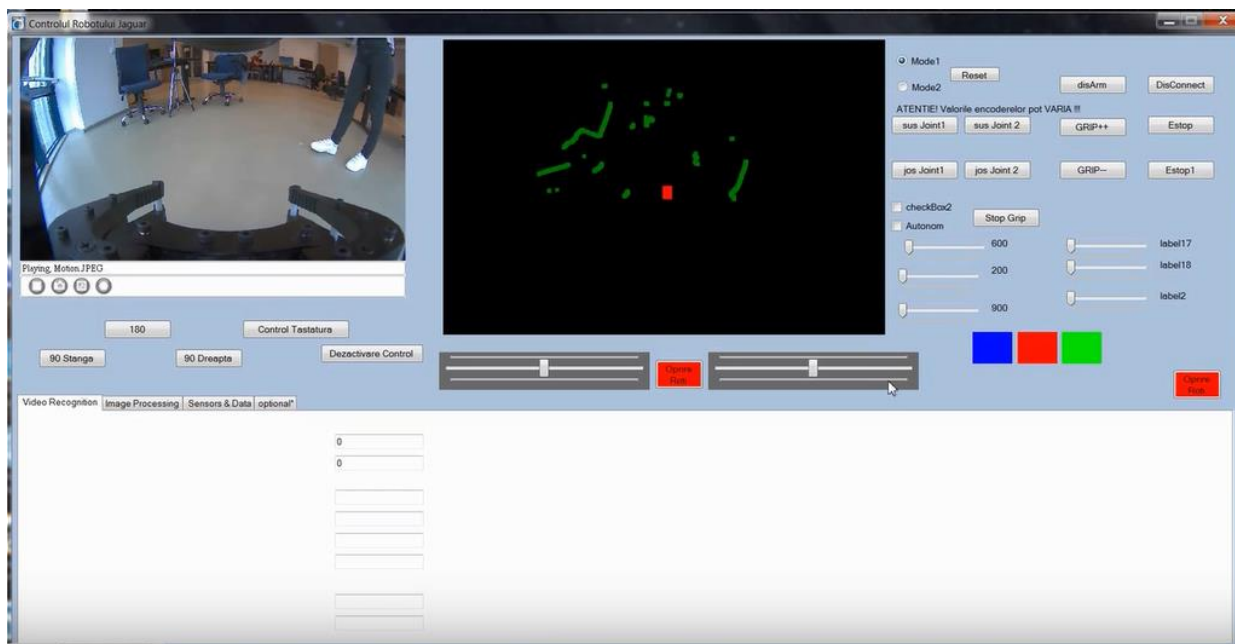
Se va porni aplicația folosind executabilul:

Va apărea o fereastră de conectare la robot, pentru a funcționa robotul trebuie pornit, și computer-ul gazdă trebuie conectat la robot via Wi-Fi (computer-ul trebuie să aibe setat un IP static pentru placa de rețea Wireless):



Figură 48 Fereastra de conectare.

După apăsarea butonului conectare, va apărea , după o scurtă durată de timp, interfața principală:



Figură 49 Interfața grafică la start-up.

Se poate observa că în partea de jos , se află o listă de tab-uri acestea vor permite afișarea de informații în diferite moduri de utilizare. În partea din dreapta sus se pot observa două butoane Radio care permit interschimbarea modurilor. Mode1 –Modul de control manual și Mode2 – Modul de control Autonom.

*Dacă nu apare harta generată cu ajutorul senzorului laser este necesară restartarea aplicației întrucât robotul nu s-a conectat la scanner-ul Laser !

Trackbar-urile din partea dreaptă pun la dispoziția utilizatorului posibilitatea de a segmenta imaginea în spațiul de culori HSV. Există deja un set de valori presetate pentru trei culori și setările vor fi actualizate la apăsarea acelor butoane (Butoanele –Albastru, Roșu și Verde).

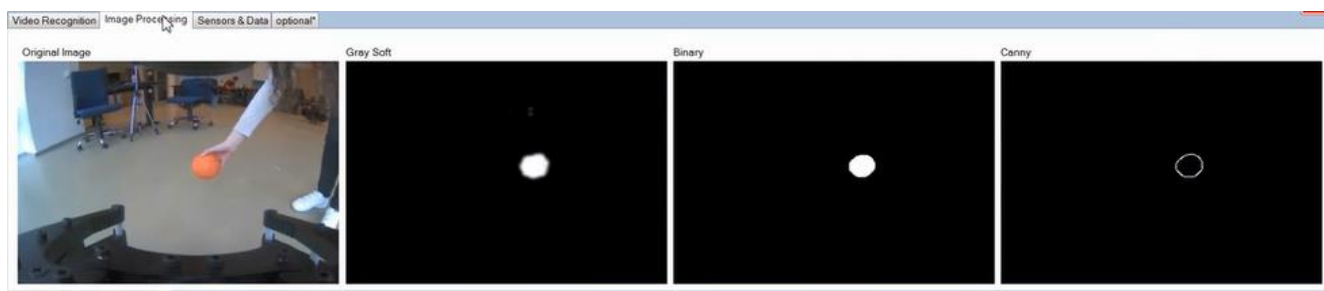
Sunt prezente de asemenea și butoanele de control pentru braț dar și trackbar-urile pentru controlul motoarelor roților. Există și butoanele care permit controlul de la tastatură (folosind săgețile pentru deplasare, și tasta Space pentru oprire) dar și primele butoane create pentru rotiri la stânga și la dreapta.

Tab-ul Video Recognition devine disponibil la trecerea în modul autonom (selectarea Mode2).



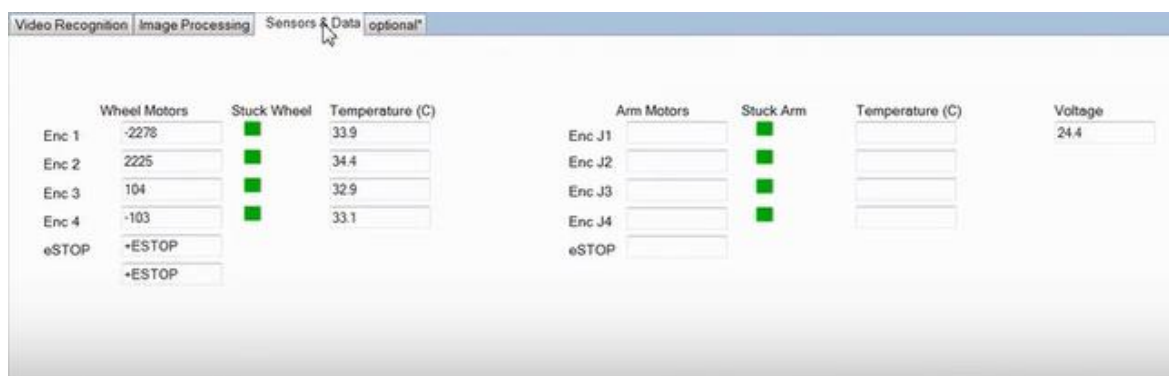
Figură 50 Tab-ul Video Recognition

Tab-ul Image processing la fel ca și Video Recognition devine disponibil în modul 2.



Figură 51 Tab-ul Image Processing

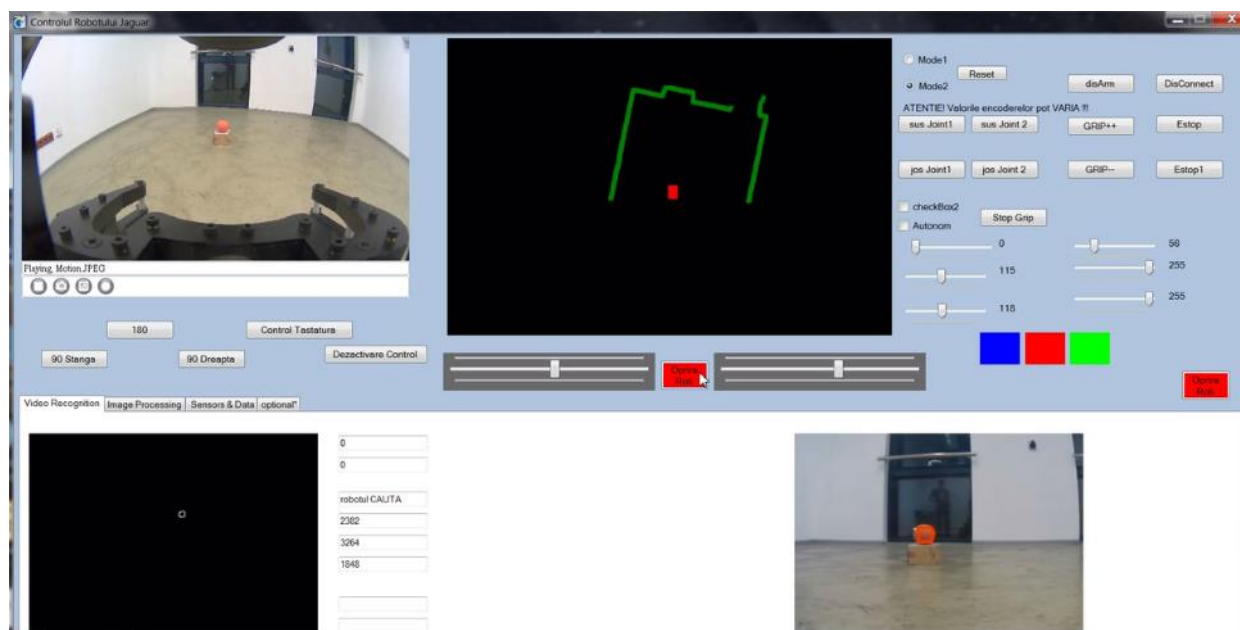
Tab-ul Sensors & Data va afișa starea motoarelor și a encoder-elor, va fi afișată și tensiunea de la baterie pentru a ne asigura că robotul funcționează în parametri normali.



Figură 52 Tab-ul Sensors & Data

După setarea robotului în modul autonom (Mode2) acesta va începe căutarea obiectului, se va folosi de informații de la senzorul de distanță Laser pentru a evita obstacolele și folosind camerele va încerca detecția obiectului căutat.

Dacă obiectul a fost detectat va apărea, în Tab-ul Video Recognition, obiectul încadrat într-un dreptunghi portocaliu.

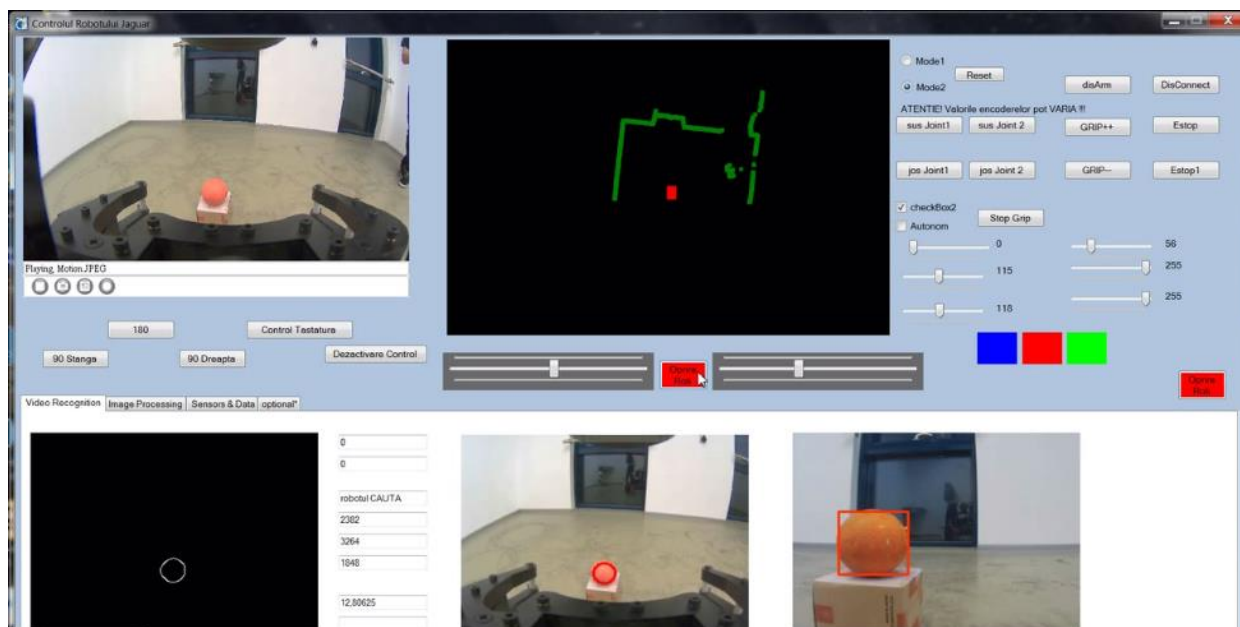


Figură 53 Aplicația în modul Autonom- Detecția obiectului.

Odată detectat obiectul, robotul va executa rutina de poziționare, astfel se va ajunge la distanța dorită față de obiect.

În cazul testului de față, poziționarea s-a terminat după 6 secunde de la încheierea fazei de căutare.

După faza de poziționare se va verifica forma obiectului.



Figură 54 Aplicația în modul Autonom- Faza de poziționare și verificare.

Se poate observa în imaginea din mijloc a Tab-ului Video Recognition cum obiectul a fost recunoscut ca fiind un cerc, astfel, brațul va încerca să se poziționeze pentru a prinde obiectul.

În seria de imagini de mai jos putem observa cum brațul se apropie de obiect, și de asemenea se poate observa cum algoritmul de detecție a cercului reușește să estimeze cercul chiar dacă acesta

este doar jumătate (datorită poziției Gripper-ului în imagine)- acest lucru a fost posibil setând un număr mai mic de puncte necesare în acumulatorul descris la 5.6.1.



Figură 55 Aplicația în modul Autonom- Prinderea obiectului.

Astfel, misiunea robotului este completă, acesta a reușit în mod autonom, să parcurgă un spațiu dintr-o încăpere evitând obstacole, să detecteze obiectul căutat de la o distanță suficient de mare, și să se poziționeze suficient de bine pentru a reuși apoi prinderea obiectului.

Succesul misiunii în acest moment este foarte dependent de factori externi ca luminozitatea sau frecarea cu suprafața de contact, aceste aspecte nu au fost acoperite de implementarea curentă.

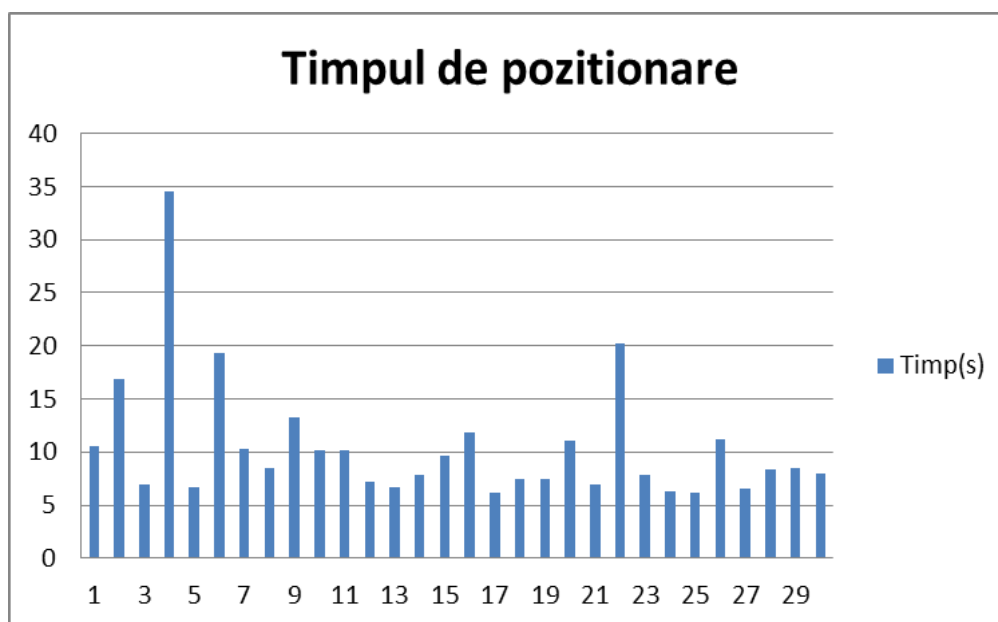
Evident că în interacția curentă a aplicației, diferitele faze de control nu sunt optimizate din punct de vedere al implementării. În continuare ne propunem dezvoltarea acestor tip de aplicații și optimizarea lor pentru misiuni în condiții reale.

5.10 TESTE ȘI REZULTATE

După cum am menționat mai devreme, reușita misiunii este foarte dependentă de variațiile condițiilor de exterior, în următoarele teste vom verifica, cât de mult sunt afectate rezultatele de aceste variații.

5.10.1 Timpul de poziționare

Frecarea cu suprafața de contact și umflarea inegală a roților pot determina apariția de erori în faza de poziționare, astfel modificând, uneori drastic, timpul în care robotul reușește să se poziționeze. Au fost realizate 30 de măsurători ale timpului trecut între momentul în care robotul detectează obiectul (intrarea în faza de poziționare) și finalizarea fazei de poziționare. Măsurătorile au fost făcute folosind aceleași setări pentru segmentarea imaginii pe parcursul tuturor testelor, de asemenea robotul a plecat din aceeași poziție, de fiecare dată, astfel putem spune că variația timpului de poziționare este datorată contactului imperfect cu podeaua. În graficul următor se poate observa timpul de poziționare (în secunde) pe axa Y, iar pe axa X avem testele în ordinea efectuării lor (de la 1 la 30).



Figură 56 Timpul de poziționare.

Observăm din acest graphic că timpul variază în jurul valorii de 8 secunde, există într-adevăr și câteva teste în care timpul de poziționare a trecut de 20 de secunde, dar aceste fiind rare. Se poate observa de asemenea că în toate cele 30 de teste consecutive, poziționarea s-a încheiat cu succes, astfel putem concluziona că intervalele au fost setate corespunzător.

5.10.2 Calitatea segmentării

Vom analiza calitatea segmentării realizate, folosind obiectul de test (sfera portocalie). Obiectul nu are culoare uniformă deci astfel nu vom putea avea niciodată o segmentare perfectă. De asemenea un factor care influențează foarte tare calitatea segmentării este luminozitatea din mediu. Variația luminozității va induce erori după cum vom vedea în testele realizate.

Pentru teste am folosit imagini prelucrate de aplicația de control a robotului. Imaginile au fost preluate de la camerele robotului, apoi procesate. În această procesare a avut loc și segmentarea în spațiul de culoare dar și aplicarea unor filtre folosite pentru a reduce zgomotul din imagine.

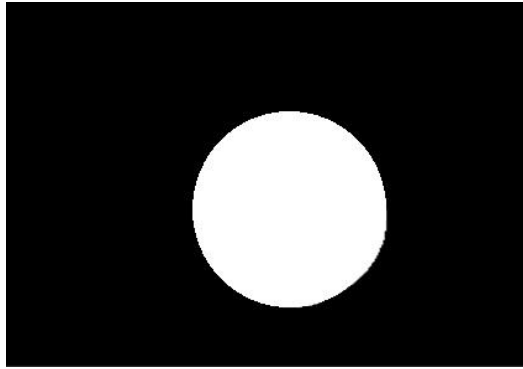
Pentru a măsura calitatea segmentării vom folosi măsurile de precizie și reamintire astfel [10]:

TP –True Positive va reprezenta numărul de pixeli ce aparțin obiectului

FP –False Positive va reprezenta numărul de pixeli de fundal segmentați ca făcând parte din obiect

FN –False Negative va reprezenta numărul de pixeli din obiect segmentați ca fiind fundal.

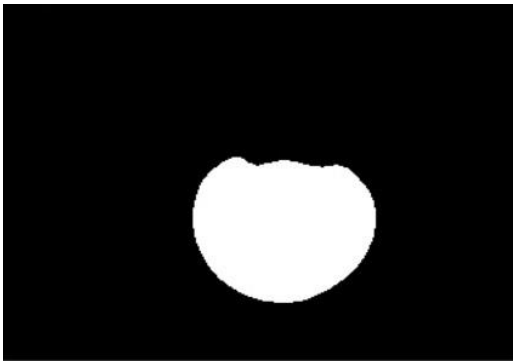
Imaginea cu Segmentarea 100% corectă a fost estimată și arată astfel:



Figură 57 Imaginea de control.

Vom folosi această imagine pentru a compara cu rezultatele segmentării noastre. Segmentarea a fost făcută folosind praguri apreciate vizual de către utilizator astfel încât să se obțină cât mai puține puncte de zgomot. (Imaginea a fost filtrată cu un filtru median și astfel au fost eliminate multe puncte de zgomot.)

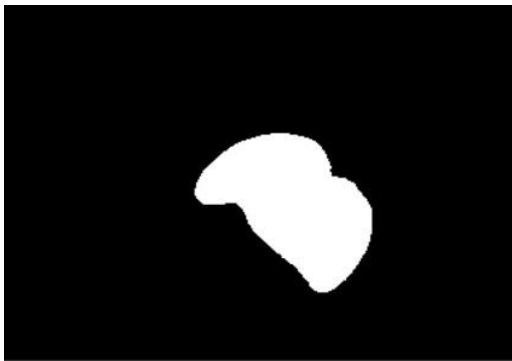
Imaginile de test sunt următoarele:



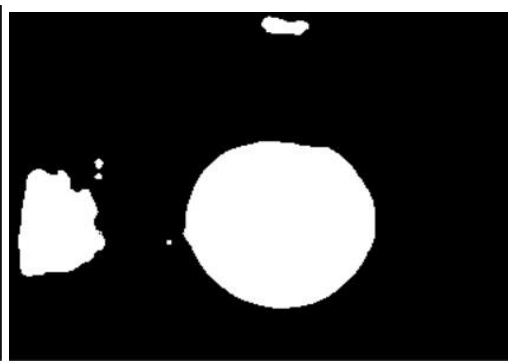
Figură 58 Imagine de test a)



Figură 59 Imagine de test b)



Figură 60 Imagine de test c)



Figură 61 Imagine de test d)

Imaginea a) a fost rezultatul segmentării manuale (la o anumită luminositate), după creșterea ușoară a luminosității s-a obținut imaginea b), la luminositatea maximă din timpul zilei s-a obținut imaginea d). Spre lăsarea serii a fost capturată imaginea c) unde luminositatea a fost mult mai scăzută.

Pentru a calcula precizia și reamintirea din aceste imagini am folosit formulele [10]:

$$\text{Precizie} = TP / (TP + FP)$$

$$\text{Reamintire} = TP / (TP + FN)$$

Pentru a obține numărul de pixeli albi sau negri din imagine, am folosit limbajul de programare și mediul Matlab. Am importat imaginile folosind funcții puse la dispoziție de Laboratorul de Analiză și Prelucrare de Imagini [10].

Am parcurs liniile și coloanelor matricelor rezultate și folosind un contor am numărat punctele de valoare 255 și punctele de valoare 0 (alb și negru):

Am obținut următoarele rezultate:

Tabel 6 Precizia și reamintirea

Imaginea	Puncte considerate obiect-255	Puncte considerate fundal-0	Precizia	Reamintirea
Imaginea de control	19524	105426	-	-
Imaginea de test a)	14181	110768	1	0.85
Imaginea de test b)	17418	107531	0.9026	0.9026
Imaginea de test c)	11032	113918	1	0.69
Imaginea de test d)	22898	102044	0.88	~ 0.89

Din rezultatele obținute putem observa că deși cu precizie mare nu obținem foarte multe puncte din obiect, astfel putem trage concluzia că o segmentare bună pentru cazul nostru va fi cea cu o precizie bună dar și o reamintire bună, deci segmentarea din imaginea b). Facem astfel un compromis pentru a obține rezultate bune din ambele puncte de vedere.

Urmărim astfel să obținem o segmentare ca cea din imaginea de test b), aceste informații ne vor fi folositoare pentru realizarea unei segmentări automate, nefiind nevoie de segmentare manuală.

În imaginea b) deși avem puncte considerate ca fiind puncte de obiect, ele fiind de fundal, avem o precizie ridicată dar și la fel și reamintirea astfel putem obține un contur mult mai precis folosit de algoritmul de recunoaștere a formei. Algoritmul de încadrare a unui blob a fost modificat pentru a încadra doar cel mai mare blob din imagine astfel, punctele de lângă obiect vor fi ignorate.

CONCLUZII

Această lucrare și-a propus realizarea unei aplicații de control capabilă să trimită și să primească datele necesare de la o platformă robotică pentru a îndeplini un obiectiv în mod autonom. Având în vedere faptul că nu au mai fost puse la dispoziție alte exemple, nu au mai fost realizate proiecte (în cadrul acestei facultăți) folosind această platformă și lipsa de experiență în utilizarea limbajului de programare C#, modul actual de funcționare al aplicației poate fi considerat un success. Robotul reușește să se deplaseze autonom într-o încăpere, să evite obstacolele întâlnite, să detecteze obiectul căutat și să îl prindă folosind brațul robotic. Deși implementarea nu este una optimă, iar algoritmi folosiți sunt destul de simpli, această variantă poate constitui o bază pentru cercetări și proiecte viitoare.

Testele realizate au demonstrat buna funcționare a fluxului de execuție, iar rezultatele obținute au ajutat la îmbunătățirea calității aplicației prin modificarea parametrilor din program.

Aplicația pune la dispoziție două moduri de utilizare, sunt afișate grafic datele de la senzori și harta de ocupare, sunt prezentate de asemenea atât imaginile primite de la camere cât și imaginile procesate, făcând astfel interfața grafică creată, un mod util și interactiv de controlare a robotului.

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:
<https://www.roboteq.com/index.php/docman/motor-controllers-documents-and-files/documentation/user-manual/7-nextgen-controllers-user-manual/file>
- [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] Aplicații OpenCV:
http://docs.opencv.org/2.4/opencv_tutorials.pdf
- [8] Documentație Emgu CV:
<http://www.emgu.com/wiki/index.php/Documentation>
- [9] M. Zamfir, Cursul și laboratorul –Prelucrearea Imaginilor.
- [10] C. Vertan, Cursul și laboratorul –Analiza Imaginilor.
- [11] C. Clark, Cursul –Autonomous Robot Navigation, Harvey Mudd College
- [12] Laboratorul E190Q –Harvey Mudd College
<https://www.hmc.edu/lair/E190Q/>
- [13] C. Stoica, Cursul –Programare Distribuită
- [14] [https://msdn.microsoft.com/en-us/library/w89fhyex\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/w89fhyex(v=vs.110).aspx)
- [15] T. Young, J. Gerbrands, J. van Vliet, “Fundamentals of Image Processing” - Delft University of Technology

[16] Filtre liniare și neliniare:

<http://www.bioss.ac.uk/people/chris/ch3.pdf>

[17] Harta de ocupare, proiect al universității Stanford:

<http://robots.stanford.edu/papers/thrun.occ-journal.pdf>

ANEXA 1

CLASA JAGUARCTRL

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Timers;
using Timer = System.Timers.Timer;
using System.Runtime.InteropServices;
using EARTHLib;
using System.Drawing.Drawing2D;
using System.IO;
using Microsoft.DirectX.DirectInput;
using System.Threading;
using Emgu.CV;
using Emgu.CV.CvB;
using Emgu.CV.UI;
using Emgu.CV.CvEnum;
using Emgu.CV.Structure;
using Emgu.CV.VideoSurveillance;

//using AForge.Controls;
using AForge.Video;
//using AForge;

using DirectShowLib;

using System.Net;
using System.Web;

namespace DrRobot.JaguarControl
{
    public partial class JaguarCtrl : Form
```

```

{
    DrRobotRobotConnection drRobotConnect = null;
    RobotConfig robotCfg = null;
    RobotConfig.RobotConfigTableRow jaguarSetting = null;
    private const string configFile =
"c:\\DrRobotAppFile\\OutDoorRobotConfig_4x4.xml";

    private const double TEMPERATURE_TH = 60.0;

    public System.Windows.Forms.Timer timp = new
System.Windows.Forms.Timer();
    public System.Windows.Forms.Timer timp_laser = new
System.Windows.Forms.Timer();
    public System.Windows.Forms.Timer timp_laser2 = new
System.Windows.Forms.Timer();
    public System.Windows.Forms.Timer timp_gripper= new
System.Windows.Forms.Timer();
    public System.Windows.Forms.Timer timp_harta = new
System.Windows.Forms.Timer();
    public System.Windows.Forms.Timer timp_brat = new
System.Windows.Forms.Timer();

    Timer tempo = new Timer(7000);
    Timer terminat = new Timer(3000);

    float a2, b2, c2;
    float a1, b1, c1;
    private static BackgroundSubtractor _fgDetector;
    private static Emgu.CV.CvB.CvBlobDetector _blobDetector;
    float maxblob;
    CvBlob largestContour;

    bool prindere = true;
    bool fanion_gripper=true;
    bool verificare = true;
    bool poz_J1 = true;
    bool poz_J2 = true;
    bool test = true;
    bool test1 = true;
    bool autonom = true;
    bool btnEstop_ev = true;

    public int mode = 1;    // 1-User Control
                        // 2-Autonomous
                        // 3-...

    public Capture _capture = null;
    private bool _captureInProgress;
    float c;
    int powerIO = 0x01;

    private MJPEGStream VideoStream = new MJPEGStream();
    private MJPEGStream VideoStream1 = new MJPEGStream();

    private EventArgs Autonomie = new EventArgs();
    int contor1=0;
    int contor2=0;
    bool cerc_gasit = false;

    #region variable for Manipulator

```

```
private int[] JOINT_CIRCLECNT = new int[4] { 5700, 5700, 3724, 756 };
//1:285,1:285,1:49,1:27

private DrRobotComm drRobotCommMot1 = null;
private DrRobotComm drRobotCommMot2 = null;
private MotorDriverData motorDriverData1 = new MotorDriverData();
private MotorDriverData motorDriverData2 = new MotorDriverData();

private const string MOT1ID = "MOT1";
private const string MOT2ID = "MOT2";

private delegate void UpdateUIDelegate(string Msg);

private bool armFlag = false;
private int pingCh = 1;
private int pingCnt = 0;
public class JointMotorData
{
    public int pwmOutput = 0;
    public int encodeSpeed = 0;
    public int encoderPos = 0;
    public int encoderDir = 0;
    public int circleCnt = 100;
    public double resolution = 10;
    public bool protect = false;
    public double angle = 0;
    public int iniPos = 0;
    public int limitPos1 = 0;
    public int limitPos2 = 0;
    public bool stuckFlag = false;
    public int cmdDir = 1;
    public int preCmdDir = 1;
    public int manipulatorStuckCnt = 0;
    public int jointCmd = 0;
    public bool jointJoy = false;
    public bool jointCtrl = false;
    public double motorTemperature;
}
private JointMotorData[] jointMotorData = new JointMotorData[4];
private const double ARMLEN1 = 31.0;
private const double ARMLEN2 = 31.0;
private double HRes = 0;
private double VRes = 0;
private const double manipulatorRatio = 15.0;
private double[] manipulatorPos = new double[2] { 0, 0 };
private enum Joystick_Ctrl_Mode { WheelCtrl, CameraCtrl,
ManipulatorCtrl, FlipperControl };
private Joystick_Ctrl_Mode joystick_Ctrl_Mode =
Joystick_Ctrl_Mode.WheelCtrl;
private Joystick_Ctrl_Mode button_Ctrl_Mode =
Joystick_Ctrl_Mode.ManipulatorCtrl;

private const int MANIPULATOR_STUCK_TH = 10;
private const double MANIPULATOR_STUCKVELTH = 50;
private const double MANIPULATOR_STUCK_PWMTH = 100;
int counter_cerc = 0;

bool running;

private int resetDelayCnt = 0;
```

```

private bool resetStatue = false;
private const int RESET_TIME = 5000;
private int JOINT1_CMD = 1;
private int JOINT2_CMD = 1;
private const int JOINT3_CMD = 20;
private const int JOINT4_CMD = 330;

private const double POS_STEP = 1;
private const int POS_CTRL_TIME = 800;
private int manPosCtrlCnt = 0;
private bool blnManPosOver = true;
private const int JOYPOS_DELAY = 1;
private bool setIniPos = false;
private const int HALF_RANGE = 16384;
private const double J1_OFFSET = 13.0;
private const double J2_OFFSET = 13.0;
private double[] JOINT_INI_ANGLE = new double[4] { Math.PI - Math.PI
* J1_OFFSET / 180, Math.PI - Math.PI * J2_OFFSET / 180, 0, 0 };
private const int JOINT2_PRE_RESET = 20;
private static Timer loopTimer;
#endregion

private SetColor setColor = new SetColor();

public class SetColor
{
    public string SetRed = "0x7f0000ff";
    public string SetGreen = "0x7f00ff00";
    public string SetBlue = "0x7fff0000";
}
#region motor variable define
private bool blnEStop = false;
private DrRobotMainComm drrobotMainComm = null;
private const string COMM1_ID = "COMM1";
public MotorDriverData[] motorDriverData = new MotorDriverData[4];
private const string SENSOR_PACKAGE_ID = "#";
private const string GPS_PACKAGE_ID = "$GPRMC";
private const string MOTOR_PACKAGE_ID = "MM";
private string[] MM_PACKAGE_ID = new String[4] { "MM0", "MM1", "MM2",
"MM3" };
private delegate void UpdateMainUIDelegate(string Msg);

public class MotorData
{
    public int pwmOutput = 0;
    public int encodeSpeed = 0;
    public int encoderPos = 0;
    public int encoderDir = 0;
}
private MotorData[] armMotor = new MotorData[2];
private int[] armEncoder = new int[2] { 0, 0 };
private int[] preArmEncoder = new int[2] { 0, 0 };
private bool firstEncoderData = true;
private const int ARM_CIRCLE_CNT = 1140; // 5*4*285 /5 = 1140
private double[] armPosAngle = new double[2] { 0, 0 };
private double[] preArmPos = new double[2] { 0, 0 };
private int[] armResetPos = new int[2] { 0, 0 };
private double[] armPosStart = new double[2] { 0, 0 };

public MotorData leftFrontWheelMotor = new MotorData();
public MotorData rightFrontWheelMotor = new MotorData();

```

```
public MotorData leftRearWheelMotor = new MotorData();
public MotorData rightRearWheelMotor = new MotorData();

private int forwardPower = 0;
private int a = 0;
private int b = 0;
private int turnPower = 0;
int armCmd1 = 0;
int armCmd2 = 0;
int exeTime = 1000;
private bool armJoy2 = false;
private bool armJoy1 = false;

private const short armChannel1 = 0;
private const short armChannel2 = 1;
private const int ARM_DELAY_CNT = 40;
private const int ARM_VEL1 = 200;
private const int ARM_VEL2 = 1000;

//for temperature sensor
private double[] resTable = new
double[25]{114660,84510,62927,47077,35563,27119,20860,16204,12683,10000,
7942,6327,5074,4103,3336,2724,2237,1846,1530,1275,1068,899.3,760.7,645.2,549.
4};
private double[] tempTable = new double[25] { -20, -15, -10, -5, 0,
5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95,
100 };
private const double FULLAD = 4095;

private double stuckVelTH = 50;
private double stuckAcc = 0;
private double stuckPWMTH = 12000;
private double stuckAccTH = 50;
#endregion

#region Axis Camera Control
string CompleteURL(string theMediaURL, string theMediaType)
{
    string anURL = theMediaURL;
    if (!anURL.EndsWith("/")) anURL += "/";

    if (theMediaType == "mjpeg")
    {
        anURL += "axis-cgi/mjpg/video.cgi";
    }
    else if (theMediaType == "mpeg4")
    {
        anURL += "mpeg4/media.amp";
    }
    else if (theMediaType == "mpeg2-unicast")
    {
        anURL += "axis-cgi/mpeg2/video.cgi";
    }
    else if (theMediaType == "mpeg2-multicast")
    {
        anURL += "axis-cgi/mpeg2/video.cgi";
    }

    anURL = CompleteProtocol(anURL, theMediaType);
    return anURL;
}
```

```

string CompleteProtocol(string theMediaURL, string theMediaType)
{
    if (theMediaURL.IndexOf("://") >= 0) return theMediaURL;

    string anURL = theMediaURL;

    if (theMediaType == "mjpeg")
    {
        anURL = "http://" + anURL;
    }
    else if (theMediaType == "mpeg4")
    {
        anURL = "axrtsphhttp://" + anURL;
    }
    else if (theMediaType == "mpeg2-unicast")
    {
        anURL = "http://" + anURL;
    }
    else if (theMediaType == "mpeg2-multicast")
    {
        anURL = "axsdp" + anURL;
    }

    return anURL;
}

#endregion
#region form funtion
public JaguarCtrl(DrRobotRobotConnection form, RobotConfig
robotConfig)
{

    drRobotConnect = form;
    robotCfg = robotConfig;

    jaguarSetting =
(RobotConfig.RobotConfigTableRow) robotCfg.RobotConfigTable.Rows[0];
    InitializeComponent();
    radioButton1.Checked = true;
    timp_laser.Tick += new EventHandler(Start_Laser);
    timp_laser2.Tick += new EventHandler(Start_Scan);
    timp_harta.Tick += new EventHandler(Tick_harta);
    //timp_brat.Tick += new EventHandler(timp_bratl);

    timp_brat.Interval = 3000;

    timp_harta.Interval = 300;
    timp_laser.Interval = 1500;
    timp_laser2.Interval = 1000;
    timp_harta.Start();
    timp_laser.Start();
    timp_laser2.Start();

    timp.Tick += new EventHandler(Autonom);
    timp.Interval = 80;
    timp.Start();

    this.Load += new EventHandler(frmMain_Load);

    // CvInvoke.UseOpenCL = false;

```

```
        try
        {

            VideoStream.NewFrame += new
NewFrameEventHandler(ProcessFrame_NewFrame);
            VideoStream1.NewFrame += new
NewFrameEventHandler(ProcessFrame_NewFrame1);

            _fgDetector = new
Emgu.CV.VideoSurveillance.BackgroundSubtractorMOG2();
            _blobDetector = new CvBlobDetector();

            // _capture = myImage;

            string sourceURL = "http://192.168.0.74:8082/axis-
cgi/mjpg/video.cgi";
            // _capture = new Capture("http://192.168.0.74:8082/axis-
cgi/mjpg/video.cgi");
            // Capture _Capture = new
CvInvoke.cvCreateFileCapture("http://192.168.0.74:8082/axis-
cgi/mjpg/video.cgi");
            // Image<Bgr, Byte> imgOriginal = new Image<Bgr,
byte>(_capture.ImageGrabbed);*/
            // _capture.ImageGrabbed += ProcessFrame;

        }
        catch (NullReferenceException excpt)
        {
            MessageBox.Show(excpt.Message);
        }

    }

    private void JaguarCtrl_Shown(object sender, EventArgs e)
    {
        drRobotConnect.Hide();
        this.Focus();
    }

    private void JaguarCtrl_FormClosed(object sender, FormClosedEventArgs
e)
    {
        //yAMC.Stop();

        try
        {
            if (drRobotConnect != null)
                drRobotConnect.Close();
        }
        catch
        {
        }
    }
}
```

```

private void setArmControl(bool set)
{
    if (set)
    {
        button9.Text = "Disconnect";
    }
    else
    {
        button9.Text = "Connect";
    }
    button10.Enabled = set;
    button11.Enabled = set;
    btnQueryCmd.Enabled = set;
    btnQueryCmd1.Enabled = set;
}

private void JaguarCtrl_Load(object sender, EventArgs e)
{
    //for flipper arm
    for (int i = 0; i < 2; i++)
    {
        armMotor[i] = new MotorData();
    }

    //for manipulator
    for (int i = 0; i < 4; i++)
    {
        jointMotorData[i] = new JointMotorData();
        jointMotorData[i].circleCnt = JOINT_CIRCLECNT[i];
        jointMotorData[i].resolution = jointMotorData[i].circleCnt /
(2 * Math.PI);
        jointMotorData[i].iniPos = 0;
        jointMotorData[i].angle = Math.PI;
        jointMotorData[i].motorTemperature = 0;
    }
    armFlag = false;
    armFlag = connectArmController();
    setArmControl(armFlag);
    if (armFlag)
    {
        //btnQueryCmd_Click(null, null);
        // btnQueryCmd1_Click(null, null);
        pingCh = 1;
        tmrPingMotorDriver.Enabled = true;
    }

    drrobotMainComm = new DrRobotMainComm(this, COMM1_ID);
    for (int i = 0; i < 4; i++)
    {
        motorDriverData[i] = new MotorDriverData();
    }

    bool res = drrobotMainComm.StartClient(jaguarSetting.RobotIP,
jaguarSetting.Port1); //now use tcp first
    // bool res = drRobotComm1.SerialOpen(1, 115200); //now use
tcp first

    if (res)
    {
        tmrPing.Enabled = true;
    }
}

```

```
//for Axis camera
try
{
    // Set properties, deciding what url completion to use by
    MediaType.
        myAMC.MediaUsername = jaguarSetting.CameraUser;
        myAMC.MediaPassword = jaguarSetting.CameraPWD;
        myAMC.MediaType = "mjpeg";
        myAMC.MediaURL = CompleteURL(jaguarSetting.CameraIP + ":" +
jaguarSetting.CameraPort, "mjpeg");
        myAMC.AudioConfigURL = "http://" + jaguarSetting.CameraIP +
":" + jaguarSetting.CameraPort + "/axis-
cgi/view/param.cgi?camera=1&action=list&group=Audio,AudioSource.A0";

        /*
        myAMC.PTZControlURL = "http://" + jaguarSetting.CameraIP +
":" + jaguarSetting.CameraPort + "/axis-cgi/com/ptz.cgi";
        myAMC.UIMode = "ptz-absolute";
        myAMC.ShowToolbar = true;
        myAMC.StretchToFit = true;
        myAMC.EnableContextMenu = true;
        myAMC.ToolbarConfiguration = "default,+ptz";
        */
        myAMC.ToolbarConfiguration = "default, + audiocontrol,+ rec";

        myAMC.AudioReceiveStop();
        // Start the streaming
        myAMC.Play();
        myAMC.AudioReceiveURL = "http://" + jaguarSetting.CameraIP +
":" + jaguarSetting.CameraPort + "/axis-cgi/audio/receive.cgi";
        myAMC.Volume = 100;
        //myAMC.AudioReceiveStart();
        myAMC.AudioTransmitURL = "http://" + jaguarSetting.CameraIP +
":" + jaguarSetting.CameraPort + "/axis-cgi/audio/transmit.cgi";
    }
    catch (Exception ex)
    {
        MessageBox.Show(this, "Nu exista Stream: " + ex.Message);
    }

//for camera2
try
{
    // Set properties, deciding what url completion to use by
    MediaType.
        myCamera.MediaUsername = jaguarSetting.AimCameraUser;
        myCamera.MediaPassword = jaguarSetting.AimCameraPWD;
        myCamera.MediaType = "mjpeg";
        myCamera.MediaURL = CompleteURL(jaguarSetting.AimCameraIP +
":" + jaguarSetting.AimCameraPort, "mjpeg");
        myCamera.AudioConfigURL = "http://" +
jaguarSetting.AimCameraIP + ":" + jaguarSetting.AimCameraPort + "/axis-
cgi/view/param.cgi?camera=1&action=list&group=Audio,AudioSource.A0";

        myCamera.PTZControlURL = "http://" +
jaguarSetting.AimCameraIP + ":" + jaguarSetting.AimCameraPort + "/axis-
cgi/com/ptz.cgi";
        myCamera.UIMode = "ptz-absolute";
        myCamera.ShowToolbar = true;
        myCamera.StretchToFit = true;
        myCamera.EnableContextMenu = true;
```

```

        myCamera.ToolbarConfiguration = "default,+ptz";
        myCamera.EnableJoystick = false;
        myCamera.ToolbarConfiguration = "default, + audiocontrol,+
rec";

        myCamera.AudioReceiveStop();
        // Start the streaming
        myCamera.Play();
        //textBox6.Text = jaguarSetting.CameraPort.ToString();
        //textBox7.Text = jaguarSetting.CameraIP.ToString();

    }
    catch (Exception ex)
    {
        MessageBox.Show(this, "Nu exista Stream: " + ex.Message);
    }

    if ((jaguarSetting.LaserRangeIP != "0.0.0.0") &&
(jaguarSetting.LaserRangePort != 0))
    {

        startCommLaser();
        /* btnTurnOn.Enabled = true;
        btnLaserScan.Enabled = true;
        trackBarDetectDis.Enabled = true;
        checkBoxCA.Enabled = true;*/
    }
    else
    {
        // checkBoxCA.Enabled = false;
        /*   trackBarDetectDis.Enabled = false;
        btnTurnOn.Enabled = false;
        btnLaserScan.Enabled = false;*/
    }
    // drawLaserBackground();
    MOTDIR = jaguarSetting .MotorDir;

}

private void myAMC_OnError(object sender,
AxAXISMEDIACONTROLLib._IAxisMediaControlEvents_OnErrorEvent e)
{

}

private void JaguarCtrl_FormClosing(object sender,
FormClosingEventArgs e)
{
    tmrDisplay.Enabled = false;
    tmrDrawing.Enabled = false;
    tmrJoyStickPoll.Enabled = false;
    try
    {
        if (drrobotMainComm != null)
        {
            drrobotMainComm.Close();
        }
    }
}

```

```
        catch
        {
        }
        //stopCommunicationLaser();
        //disconnectArmController();
    }

    #endregion

    #region Joystick variables
    #region variable for Joystick Control
    private bool forceStopManipulator = false;
    private bool protectMotorTemp = false;
    private bool protectMotorStuck = false;
    private const int NOCONTROL = -32768;

    private bool lightOn = false;
    private int expandIO = 0x77;
    private const short LEFTWHEELCHANNEL = 3;           //3
    private const short RIGHTWHEELCHANNEL = 4;          //4
    public static JoystickState[] joyState = new JoystickState[2];
    private static int[] preSetButton = new int[4] { -1, -1, -1, -1 };
    private static int[] joyButtonCnt = new int[4] { 0, 0, 0, 0 };
    private const int JOYDELAY = 10;

    private Device[] applicationDevice = new Device[2];

    private bool blnJoyStick = true;
    private const int MAXPWM = 16384;
    private const int MINPWM = 4000;
    private const int INIPWM = 16384;
    private int MOTDIR = 1;
    private const int CMD_INT_TIME = 10;

    private bool lightCtrl = true;
    private double drvThrottle = 1.0;
    #endregion

    #endregion

    #region Comenzi
    private void trackBarForwardPower_ValueChanged(object sender,
EventArgs e)
    {
        drvRobot_trackbar();
    }

    private void trackBarTurnPower_ValueChanged(object sender, EventArgs
e)
    {
        drvRobot_trackbar();
    }

    private void drvRobot_trackbar()
    {
        int forwardPWM = 0;
        int turnPWM = 0;
        int cmd1 = 0, cmd2 = 0;
        if (!protectMotorTemp)
        {
            forwardPWM = trackBarForwardPower.Value;
            turnPWM = trackBarTurnPower.Value;
        }
    }
    }
```

```

    }
    else
    {
        forwardPWM = 0;
        turnPWM = 0;
    }
    cmd1 = forwardPWM + turnPWM;
    cmd2 = forwardPWM - turnPWM;
    sendMotorCmd(cmd1, cmd2, cmd1, cmd2);
}

private void sendMotorCmd(int cmd1, int cmd2, int cmd3, int cmd4)
{
    cmd1 = (cmd1 < -1000 ? -1000 : cmd1);
    cmd1 = (cmd1 > 1000 ? 1000 : cmd1);

    cmd2 = (cmd2 < -1000 ? -1000 : cmd2);
    cmd2 = (cmd2 > 1000 ? 1000 : cmd2);

    cmd3 = (cmd3 < -1000 ? -1000 : cmd3);
    cmd3 = (cmd3 > 1000 ? 1000 : cmd3);

    cmd4 = (cmd4 < -1000 ? -1000 : cmd4);
    cmd4 = (cmd4 > 1000 ? 1000 : cmd4);

    string strCmd = "MMW " + "!M " + cmd1.ToString() + " " + (-
cmd2).ToString();
    drrobotMainComm.SendCommand(strCmd);
}

private void sendEStopCmd()
{
    string strCmd = "MMW !M 0 0";
    drrobotMainComm.SendCommand(strCmd);
    strCmd = "MMW !EX";
    drrobotMainComm.SendCommand(strCmd);
}

private void sendEStopRelease()
{
    string strCmd = "MMW !MG";
    drrobotMainComm.SendCommand(strCmd);
}

#endregion

#region buttons

private void DrawDataPic(PictureBox picCtrl, string dataName, Color
penColor, double[] data, int startPos, int endPos, int length, int maxData)
{
    Bitmap bmp = new Bitmap(picCtrl.Width, picCtrl.Height);
    Graphics g = Graphics.FromImage(bmp);

    int width = bmp.Width - 4;
    int height = bmp.Height - 4;
    Pen drawAxisPen = new Pen(Color.Black);
    Pen drawDataPen = new Pen(penColor);

```

```
g.Clear(System.Drawing.Color.White);
g.SmoothingMode =
System.Drawing.Drawing2D.SmoothingMode.HighQuality;
GraphicsPath path = new GraphicsPath();

g.DrawLine(drawAxisPen, 0, height / 2, width, height / 2);
int x = 0;
int y = 0;
int lastX = 0;
int lastY = height / 2;
double scaleY = ((double)height / 2) / maxData;
double scaleX = (double)width / data.Length;
int dataPos = 0;

for (int i = 0; i < length; i++)
{
    x = (int)(i * scaleX);

    dataPos = i + startPos;
    if (dataPos >= length)
    {
        dataPos = dataPos - length;
    }
    y = (int)(data[dataPos] * scaleY) + height / 2;
    g.DrawLine(drawDataPen, lastX, lastY, x, y);
    lastX = x;
    lastY = y;
}

String drawString = dataName;
// Create font and brush.
Font drawFont = new Font("Arial", 8, FontStyle.Bold);
SolidBrush drawBrush = new SolidBrush(Color.Red);
// Create point for upper-left corner of drawing.
float fontPosX = 3.0F;
float fontPosY = 3.0F;
// Set format of string.
StringFormat drawFormat = new StringFormat();
//drawFormat.FormatFlags = StringFormatFlags.DirectionVertical;

// Draw string to screen.
g.DrawString(drawString, drawFont, drawBrush, fontPosX, fontPosY,
drawFormat);

picCtrl.Image = bmp;
}

private void button8_Click(object sender, EventArgs e)
{
    if (button8.Text == "EStop Arm")
    {
        button8.BackColor = Color.Green;
        try
        {
            forceStopManipulator = true;
            if ((drRobotCommMot1 != null) && armFlag)
            {
                drRobotCommMot1.SendCommand("!EX\r");
            }
        }
    }
}
```

```

        if ((drRobotCommMot2 != null) && armFlag)
        {
            drRobotCommMot2.SendCommand("!EX\r");
        }

    }
    catch
    {
    }
    button8.Text = "Res Arm";
    forceStopManipulator = true;
}
else
{
    try
    {
        forceStopManipulator = false;
        if ((drRobotCommMot1 != null) && armFlag)
        {
            drRobotCommMot1.SendCommand("!MG\r");
        }
        if ((drRobotCommMot2 != null) && armFlag)
        {
            drRobotCommMot2.SendCommand("!MG\r");
        }
    }
    catch
    {
    }

    button8.BackColor = Color.Red;
    button8.Text = "EStop Arm";
    forceStopManipulator = false;
}
}

private void checkBoxCA_CheckedChanged(object sender, EventArgs e)
{
    if (checkBoxCA.Checked)
    {
        //lower down the robot driving power
        if (jaguarSetting.RobotType.ToLower() != "jaguar")
            drvThrottle = 0.65;
    }
    else
    {
        drvThrottle = 1.0;
    }
}
#endregion

#region comunicatii

private bool connectArmController()
{
    bool res = false;

```

```
        drRobotCommMot1 = new DrRobotComm(this, MOT1ID);
        //drRobotCommMot1.SerialOpen(1, 115200);
        res = drRobotCommMot1.StartClient(jaguarSetting.ManipulatorIP,
jaguarSetting.ManipulatorPort1);
        drRobotCommMot2 = new DrRobotComm(this, MOT2ID);

        res = drRobotCommMot2.StartClient(jaguarSetting.ManipulatorIP,
jaguarSetting.ManipulatorPort2);

        return res;
    }
    public void UpdateAckMsg(string Msg)
    {
        Invoke(new UpdateUIDelegate(UpdateACKUI), Msg);
    }

    private void UpdateACKUI(string Msg)
    {
        if (Msg.StartsWith(MOT1ID))
        {
            lblACKMessage.Text = Msg.Remove(0, MOT1ID.Length);
        }
        else if (Msg.StartsWith(MOT2ID))
        {
            lblACKMessage1.Text = Msg.Remove(0, MOT2ID.Length);
        }
    }

    public void UpdateRcvDataMsg(string Msg)
    {
        Invoke(new UpdateUIDelegate(UpdateRCVUI), Msg);
    }

    private void UpdateRCVUI(string Msg)
    {
        if (Msg.StartsWith(MOT1ID))
        {
            Msg = Msg.Remove(0, MOT1ID.Length);
            lblRcvData.Text = Msg;

            if (Msg.StartsWith("A="))
            {
                //here is the current message
                try
                {
                    Msg = Msg.Remove(0, 2);
                    string[] strData = Msg.Split(':');
                    motorDriverData1.motAmp1 = double.Parse(strData[0]) /
10;
                    lblMotAmp1.Text =
motorDriverData1.motAmp1.ToString("0.00");
                    motorDriverData1.motAmp2 = double.Parse(strData[1]) /
10;
                    lblMotAmp2.Text =
motorDriverData1.motAmp2.ToString("0.00");
                }
                catch
                {
                }
            }
            else if (Msg.StartsWith("AI="))
            {

```

```

        //here is the current message
        try
        {
            Msg = Msg.Remove(0, 2);
            string[] strData = Msg.Split(':');
            motorDriverData1.ai3 = double.Parse(strData[2]);
            jointMotorData[0].motorTemperature =
Trans2Temperature(motorDriverData1.ai3);
            textBox25.Text =
jointMotorData[0].motorTemperature.ToString("0.0");
            motorDriverData1.ai4 = double.Parse(strData[3]);
            jointMotorData[1].motorTemperature =
Trans2Temperature(motorDriverData1.ai4);
            textBox26.Text =
jointMotorData[1].motorTemperature.ToString("0.0");

        }
        catch
        {
        }
    }
    else if (Msg.StartsWith("C="))
    {
        //here is the current message
        try
        {
            Msg = Msg.Remove(0, 2);
            string[] strData = Msg.Split(':');
            motorDriverData1.motEncP1 = int.Parse(strData[0]);
            jointMotorData[0].encoderPos =
motorDriverData1.motEncP1;
            textBox15.Text =
motorDriverData1.motEncP1.ToString();
            // jointMotorData[0].angle = trans2Angle(0);

            motorDriverData1.motEncP2 = int.Parse(strData[1]);
            jointMotorData[1].encoderPos =
motorDriverData1.motEncP2;
            textBox16.Text =
motorDriverData1.motEncP2.ToString();
            //jointMotorData[1].angle = trans2Angle(1);
            //encoderPositionProtect12();
            lblJ1Angle.Text = (jointMotorData[0].angle / Math.PI
* 180).ToString("0.00");
            lblJ2Angle.Text = (jointMotorData[1].angle / Math.PI
* 180).ToString("0.00");
            //drawManipulator(jointMotorData[0].angle,
jointMotorData[1].angle);
            //manipulatorPos =
getPositionXY(jointMotorData[0].angle, jointMotorData[1].angle);

            lblPosX.Text = manipulatorPos[0].ToString("0.00");
            lblPosY.Text = manipulatorPos[1].ToString("0.00");
        }
        catch
        {
        }
    }
    else if (Msg.StartsWith("D="))
    {
        //here is the current message
        try
        {

```

```
Msg = Msg.Remove(0, 2);
byte data = byte.Parse(Msg);
int test = data & 0x4;
if ((data & 0x4) != 0)
{
    motorDriverData1.di3 = 1;
}
else
{
    motorDriverData1.di3 = 0;
}
if ((data & 0x8) != 0)
{
    motorDriverData1.di4 = 1;
}
else
{
    motorDriverData1.di4 = 0;
}
//      lblDI3.Text =
motorDriverData1.di3.ToString();
//      lblDI4.Text =
motorDriverData1.di4.ToString();
}
catch
{
}
}
else if (Msg.StartsWith("DO="))
{
    //here is the current message
    try
    {
        Msg = Msg.Remove(0, 3);
        byte data = byte.Parse(Msg);
        if ((data & 0x1) != 0)
        {
            motorDriverData1.do1 = 1;
        }
        else
        {
            motorDriverData1.do1 = 0;
        }
        if ((data & 0x2) != 0)
        {
            motorDriverData1.do2 = 1;
        }
        else
        {
            motorDriverData1.do2 = 0;
        }
        //      lblDO1.Text = motorDriverData1.do1.ToString();
        //      lblDO2.Text = motorDriverData1.do2.ToString();
    }
    catch
    {
    }
}
else if (Msg.StartsWith("P="))
{
    //here is the current message
    try
    {
```

```

        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');
        motorDriverData1.motPower1 = int.Parse(strData[0]);
        jointMotorData[0].pwmOutput =
motorDriverData1.motPower1;
        lblArmJ1PWM.Text =
motorDriverData1.motPower1.ToString();
        motorDriverData1.motPower2 = int.Parse(strData[1]);
        jointMotorData[1].pwmOutput =
motorDriverData1.motPower2;
        lblArmJ2PWM.Text =
motorDriverData1.motPower2.ToString();
        //manipulatorStuckDetect12();
    }
    catch
    {
    }
}
else if (Msg.StartsWith("S="))
{
    //here is the current message
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');
        motorDriverData1.motEncS1 = int.Parse(strData[0]);
        jointMotorData[0].encodeSpeed =
motorDriverData1.motEncS1;
        lblArmJ1Vel.Text =
motorDriverData1.motEncS1.ToString();
        motorDriverData1.motEncS2 = int.Parse(strData[1]);
        jointMotorData[1].encodeSpeed =
motorDriverData1.motEncS2;
        lblArmJ2Vel.Text =
motorDriverData1.motEncS2.ToString();
    }
    catch
    {
    }
}
else if (Msg.StartsWith("T="))
{
    //here is the current message
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');

        motorDriverData1.ch1Temp = double.Parse(strData[0]);
        //    lblCh1Temp.Text =
motorDriverData1.ch1Temp.ToString();
        motorDriverData1.ch2Temp = double.Parse(strData[1]);
        //    lblCh2Temp.Text =
motorDriverData1.ch2Temp.ToString();
    }
    catch
    {
    }
}
else if (Msg.StartsWith("V="))
{
    //here is the current message
    try

```

```
        {
            Msg = Msg.Remove(0, 2);
            string[] strData = Msg.Split(':');
            motorDriverData1.drvVoltage =
double.Parse(strData[0]) / 10;
            lblDrvVol.Text =
motorDriverData1.drvVoltage.ToString();
            motorDriverData1.batVoltage =
double.Parse(strData[1]) / 10;
            lblBatVol0.Text =
motorDriverData1.batVoltage.ToString();
            motorDriverData1.reg5VVoltage =
double.Parse(strData[2]) / 1000;
            lbl5VVol.Text =
motorDriverData1.reg5VVoltage.ToString();
        }
        catch
        {
        }
    }
    else if (Msg.StartsWith("MMOD="))
    {
        //here is the current message
        try
        {
            Msg = Msg.Remove(0, 5);
            string[] strData = Msg.Split(':');

            motorDriverData1.model = int.Parse(strData[0]);
            if (motorDriverData1.model != 3)
            {
                //disable all joint 1 command
                jointMotorData[0].protect = true;
            }
            if (motorDriverData1.model == 0)
            {
                lblDriverModel1.Text = "Open Loop";
            }
            else if (motorDriverData1.model == 1)
            {
                lblDriverModel1.Text = "SpeedCtrl";
            }
            else if (motorDriverData1.model == 2)
            {
                lblDriverModel1.Text = "R_PosCtrl";
            }
            else if (motorDriverData1.model == 3)
            {
                lblDriverModel1.Text = "Pos_Ctrl";
            }
            else if (motorDriverData1.model == 1)
            {
                lblDriverModel1.Text = "Tor_Ctrl";
            }
        }

        motorDriverData1.mode2 = int.Parse(strData[1]);
        if (motorDriverData1.mode2 != 3)
        {
            //disable all joint 1 command
            jointMotorData[1].protect = true;
        }
        if (motorDriverData1.mode2 == 0)
```

```

        {
            lblDriverModel2.Text = "Open Loop";
        }
        else if (motorDriverData1.mode2 == 1)
        {
            lblDriverModel2.Text = "SpeedCtrl";
        }
        else if (motorDriverData1.mode2 == 2)
        {
            lblDriverModel2.Text = "R_PosCtrl";
        }
        else if (motorDriverData1.mode2 == 3)
        {
            lblDriverModel2.Text = "Pos_Ctrl";
        }
        else if (motorDriverData1.mode2 == 1)
        {
            lblDriverModel2.Text = "Tor_Ctrl";
        }
    }
    catch
    {
    }
}
else if (Msg.StartsWith("FF="))
{
    //here is the current message
    try
    {
        Msg = Msg.Remove(0, 3);
        byte data = byte.Parse(Msg);
        motorDriverData1.statusFlag = data;
        if ((data & 0x1) != 0)
        {
            checkBoxOverHeat.Checked = true;
        }
        else
        {
            checkBoxOverHeat.Checked = false;
        }
        if ((data & 0x2) != 0)
        {
            checkBoxOverVol.Checked = true;
        }
        else
        {
            checkBoxOverVol.Checked = false;
        }
        if ((data & 0x4) != 0)
        {
            checkBoxUnderVol.Checked = true;
        }
        else
        {
            checkBoxUnderVol.Checked = false;
        }
        if ((data & 0x8) != 0)
        {
            checkBoxShort.Checked = true;
        }
        else
        {

```

```
        checkBoxShort.Checked = false;
    }
    if ((data & 0x10) != 0)
    {
        checkBoxEStop.Checked = true;
    }
    else
    {
        checkBoxEStop.Checked = false;
    }
    if ((data & 0x20) != 0)
    {
        //    checkBoxSepexFault.Checked = true;
    }
    else
    {
        //    checkBoxSepexFault.Checked = false;
    }
    if ((data & 0x40) != 0)
    {
        //    checkBoxPromFault.Checked = true;
    }
    else
    {
        //    checkBoxPromFault.Checked = false;
    }
    if ((data & 0x80) != 0)
    {
        //    checkBoxConfigFault.Checked = true;
    }
    else
    {
        //    checkBoxConfigFault.Checked = false;
    }
}
catch
{
}
}
else if (Msg.StartsWith(MOT2ID))
{
    Msg = Msg.Remove(0, MOT2ID.Length);
    lblRcvData1.Text = Msg;

    if (Msg.StartsWith("A="))
    {
        //here is the current message
        try
        {
            Msg = Msg.Remove(0, 2);
            string[] strData = Msg.Split(':');
            motorDriverData2.motAmp1 = double.Parse(strData[0]) /
10;
            lblMot1Amp1.Text =
motorDriverData2.motAmp1.ToString("0.00");
            motorDriverData2.motAmp2 = double.Parse(strData[1]) /
10;
            lblMot1Amp2.Text =
motorDriverData2.motAmp2.ToString("0.00");
        }
        catch
        {

```

```

    }
}
else if (Msg.StartsWith("AI="))
{
    //here is the analog input message
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');
        motorDriverData2.ai3 = double.Parse(strData[2]);
        jointMotorData[2].motorTemperature =
Trans2Temperature(motorDriverData2.ai3);
        textBox27.Text =
jointMotorData[2].motorTemperature.ToString("0.0");
        motorDriverData2.ai4 = double.Parse(strData[3]);
        jointMotorData[3].motorTemperature =
Trans2Temperature(motorDriverData2.ai4);
        textBox28.Text =
jointMotorData[3].motorTemperature.ToString("0.0");
    }
    catch
    {
    }
}
else if (Msg.StartsWith("C="))
{
    //here is the encoder position count message
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');
        motorDriverData2.motEncP1 = int.Parse(strData[0]);
        jointMotorData[2].encoderPos =
motorDriverData2.motEncP1;
        textBox17.Text =
motorDriverData2.motEncP1.ToString();

        motorDriverData2.motEncP2 = int.Parse(strData[1]);
        jointMotorData[3].encoderPos =
motorDriverData2.motEncP2;
        textBox18.Text =
motorDriverData2.motEncP2.ToString();
        //encoderPositionProtect34();
    }
    catch
    {
    }
}
else if (Msg.StartsWith("D="))
{
    //here is the digital input message
    try
    {
        Msg = Msg.Remove(0, 2);
        byte data = byte.Parse(Msg);
        int test = data & 0x4;
        if ((data & 0x4) != 0)
        {
            motorDriverData2.di3 = 1;
        }
        else
        {
            motorDriverData2.di3 = 0;
        }
    }
    catch
    {
    }
}
}
}

```

```
    }
    if ((data & 0x8) != 0)
    {
        motorDriverData2.di4 = 1;
    }
    else
    {
        motorDriverData2.di4 = 0;
    }
    // lblDI31.Text = motorDriverData2.di3.ToString();
    // lblDI41.Text = motorDriverData2.di4.ToString();
}
catch
{
}
}
else if (Msg.StartsWith("DO="))
{
    //here is the digital output message
    try
    {
        Msg = Msg.Remove(0, 3);
        byte data = byte.Parse(Msg);
        if ((data & 0x1) != 0)
        {
            motorDriverData2.do1 = 1;
        }
        else
        {
            motorDriverData2.do1 = 0;
        }
        if ((data & 0x2) != 0)
        {
            motorDriverData2.do2 = 1;
        }
        else
        {
            motorDriverData2.do2 = 0;
        }
        // lblDO11.Text = motorDriverData2.do1.ToString();
        // lblDO21.Text = motorDriverData2.do2.ToString();
    }
    catch
    {
    }
}
else if (Msg.StartsWith("P="))
{
    //here is the motor power (0 ~ 1000) message
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');
        motorDriverData2.motPower1 = int.Parse(strData[0]);
        jointMotorData[2].pwmOutput =
motorDriverData2.motPower1;
        lblArmJ3PWM.Text =
motorDriverData2.motPower1.ToString();
        motorDriverData2.motPower2 = int.Parse(strData[1]);
        jointMotorData[3].pwmOutput =
motorDriverData2.motPower2;
        lblArmJ4PWM.Text =
motorDriverData2.motPower2.ToString();
    }
    catch
    {
    }
}
```

```

        //manipulatorStuckDetect34();
    }
    catch
    {
    }
}
else if (Msg.StartsWith("S="))
{
    //here is the encoder speed message
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');
        motorDriverData2.motEncS1 = int.Parse(strData[0]);
        jointMotorData[2].encodeSpeed =
motorDriverData2.motEncS1;
        lblArmJ3Vel.Text =
motorDriverData2.motEncS1.ToString();

        motorDriverData2.motEncS2 = int.Parse(strData[1]);
        jointMotorData[3].encodeSpeed =
motorDriverData2.motEncS2;
        lblArmJ4Vel.Text =
motorDriverData2.motEncS2.ToString();
    }
    catch
    {
    }
}
else if (Msg.StartsWith("T="))
{
    //here is the motor driver board temperature message
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');

        motorDriverData2.ch1Temp = double.Parse(strData[0]);
        //    lblCh1Temp1.Text =
motorDriverData2.ch1Temp.ToString();
        motorDriverData2.ch2Temp = double.Parse(strData[1]);
        //    lblCh2Temp1.Text =
motorDriverData2.ch2Temp.ToString();
    }
    catch
    {
    }
}
else if (Msg.StartsWith("V="))
{
    //here is the power voltage message
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');
        motorDriverData2.drvVoltage =
double.Parse(strData[0]) / 10;
        lblDrvVol1.Text =
motorDriverData2.drvVoltage.ToString();
        motorDriverData2.batVoltage =
double.Parse(strData[1]) / 10;
        textBox20.Text =
motorDriverData2.batVoltage.ToString();
    }
    catch
    {
    }
}

```

```
motorDriverData2.reg5VVoltage =  
double.Parse(strData[2]) / 1000;  
lbl5VVol1.Text =  
motorDriverData2.reg5VVoltage.ToString();  
}  
catch  
{  
}  
}  
else if (Msg.StartsWith("MMOD="))  
{  
    //here is the current message  
    try  
    {  
        Msg = Msg.Remove(0, 5);  
        string[] strData = Msg.Split(':');  
        motorDriverData2.model = int.Parse(strData[0]);  
        if (motorDriverData2.model != 3)  
        {  
            //disable all joint 1 command  
            jointMotorData[2].protect = true;    //keep it in  
position control  
        }  
        if (motorDriverData2.model == 0)  
        {  
            lblDriverMode21.Text = "Open Loop";  
        }  
        else if (motorDriverData2.model == 1)  
        {  
            lblDriverMode21.Text = "SpeedCtrl";  
        }  
        else if (motorDriverData2.model == 2)  
        {  
            lblDriverMode21.Text = "R_PosCtrl";  
        }  
        else if (motorDriverData2.model == 3)  
        {  
            lblDriverMode21.Text = "Pos_Ctrl";  
        }  
        else if (motorDriverData2.model == 1)  
        {  
            lblDriverMode21.Text = "Tor_Ctrl";  
        }  
        motorDriverData2.mode2 = int.Parse(strData[1]);  
        if (motorDriverData2.mode2 != 0)  
        {  
            //disable all joint 1 command  
            jointMotorData[3].protect = true;    //keep it  
in open loop control  
        }  
        if (motorDriverData2.mode2 == 0)  
        {  
            lblDriverMode22.Text = "Open Loop";  
        }  
        else if (motorDriverData2.mode2 == 1)  
        {  
            lblDriverMode22.Text = "SpeedCtrl";  
        }  
        else if (motorDriverData2.mode2 == 2)  
        {  
            lblDriverMode22.Text = "R_PosCtrl";  
        }  
    }  
}
```

```

        else if (motorDriverData2.mode2 == 3)
        {
            lblDriverMode22.Text = "Pos_Ctrl";
        }
        else if (motorDriverData2.mode2 == 1)
        {
            lblDriverMode22.Text = "Tor_Ctrl";
        }
    }
    catch
    {
    }
}

else if (Msg.StartsWith("FF="))
{
    //here is the current message
    try
    {
        Msg = Msg.Remove(0, 3);
        byte data = byte.Parse(Msg);
        motorDriverData2.statusFlag = data;
        if ((data & 0x1) != 0)
        {
            checkBoxOverHeat1.Checked = true;
        }
        else
        {
            checkBoxOverHeat1.Checked = false;
        }
        if ((data & 0x2) != 0)
        {
            checkBoxOverVol1.Checked = true;
        }
        else
        {
            checkBoxOverVol1.Checked = false;
        }
        if ((data & 0x4) != 0)
        {
            checkBoxUnderVol1.Checked = true;
        }
        else
        {
            checkBoxUnderVol1.Checked = false;
        }
        if ((data & 0x8) != 0)
        {
            checkBoxShort1.Checked = true;
        }
        else
        {
            checkBoxShort1.Checked = false;
        }
        if ((data & 0x10) != 0)
        {
            checkBoxEStop1.Checked = true;
        }
        else
        {
            checkBoxEStop1.Checked = false;
        }
        if ((data & 0x20) != 0)

```

```
        {
            //checkBoxSepexFault1.Checked = true;
        }
        else
        {
            //checkBoxSepexFault1.Checked = false;
        }
        if ((data & 0x40) != 0)
        {
            //checkBoxPromFault1.Checked = true;
        }
        else
        {
            //checkBoxPromFault1.Checked = false;
        }
        if ((data & 0x80) != 0)
        {
            //checkBoxConfigFault1.Checked = true;
        }
        else
        {
            //checkBoxConfigFault1.Checked = false;
        }
    }
    catch
    {
    }
}

}

#endregion

#region main communication decode message here

public void UpdateMainRcvDataMsg(string Msg)
{
    Invoke(new UpdateMainUIDelegate(UpdateMainRCVUI), Msg);
}

private void UpdateMainRCVUI(string Msg)
{
    //process message here

    Msg = Msg.Remove(0, COMM1_ID.Length);

    if (Msg.StartsWith("\n"))
        Msg = Msg.Remove(0, 1);

    /*if (Msg.StartsWith(SENSOR_PACKAGE_ID))
    {
        // sensor package here,
#seqNum,Yaw,_,GYRO,_,_,_,ACC,_,_,_,COMP,_,_,_,PRES,_,
        textBoxRCV.Text = Msg;
        //processIMUMsg(Msg);
    }*/
    /* else if (Msg.StartsWith(GPS_PACKAGE_ID))
    {
        //GPS sensor here
```

```

        textBoxGPSRCV.Text = Msg;
        // processGPSMsg(Msg);
    }*/
else if (Msg.StartsWith(MOTOR_PACKAGE_ID))
{
    //motor driver board message here
    processMotorMsg(Msg);
}
else
{
    //other system message here
}

}

private void processMotorMsg(string Msg)
{
    //motor driver board sensor package
    //maybe there are 4 :MM0,MM1,MM2,MM3
    for (int i = 0; i < 4; i++)
    {
        if (Msg.StartsWith(MM_PACKAGE_ID[i]))
        {
            Msg = Msg.Remove(0, MM_PACKAGE_ID[i].Length + 1);
            if (Msg.StartsWith("A="))
            {
                //here is the current message
                try
                {
                    Msg = Msg.Remove(0, 2);
                    string[] strData = Msg.Split(':');
                    motorDriverData[i].motAmp1 =
double.Parse(strData[0]) / 10;
                    motorDriverData[i].motAmp2 =
double.Parse(strData[1]) / 10;
                    if (i == 0)
                    {
                        // lblMot1Amp.Text =
motorDriverData[i].motAmp1.ToString();
                        // lblMot2Amp.Text =
motorDriverData[i].motAmp2.ToString();
                    }
                    else if (i == 1)
                    {
                        // lblMot3Amp.Text =
motorDriverData[i].motAmp1.ToString();
                        // lblMot4Amp.Text =
motorDriverData[i].motAmp2.ToString();
                    }
                }
                catch
                {
                }
            }
            else if (Msg.StartsWith("AI="))
            {
                //here is the analog input message,
                try
                {
                    Msg = Msg.Remove(0, 3);
                    string[] strData = Msg.Split(':');
                    motorDriverData[i].ai3 =
double.Parse(strData[2]);
                }
            }
        }
    }
}

```

```
motorDriverData[i].ai4 =
double.Parse(strData[3]);
//use 10K resistor to GND, translate to
temperature
double res =
Trans2Temperature(motorDriverData[i].ai3);
motorDriverData[i].motTemp1 = res;
res = Trans2Temperature(motorDriverData[i].ai4);
motorDriverData[i].motTemp2 = res;
if (i == 0)
{
    textBox21.Text =
motorDriverData[i].motTemp1.ToString("0.0");
    textBox22.Text =
motorDriverData[i].motTemp2.ToString("0.0");
}
else if (i == 1)
{
    textBox23.Text =
motorDriverData[i].motTemp1.ToString("0.0");
    textBox24.Text =
motorDriverData[i].motTemp2.ToString("0.0");
}
}
catch
{
}
}
else if (Msg.StartsWith("C="))
{
    //here is the encoder position message
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');
        motorDriverData[i].motEncP1 =
int.Parse(strData[0]);
        motorDriverData[i].motEncP2 =
int.Parse(strData[1]);
        if (i == 0)
        {
            textBox10.Text =
motorDriverData[i].motEncP1.ToString();
            textBox11.Text =
motorDriverData[i].motEncP2.ToString();
        }
        else if (i == 1)
        {
            textBox12.Text =
motorDriverData[i].motEncP1.ToString();
            textBox13.Text =
motorDriverData[i].motEncP2.ToString();
        }
    }
    catch
    {
}
}
else if (Msg.StartsWith("D="))
{
    //here is the digital input message
    try
    {
```

```

        Msg = Msg.Remove(0, 2);
        byte data = byte.Parse(Msg);
        int test = data & 0x4;
        if ((data & 0x4) != 0)
        {
            motorDriverData[i].di3 = 1;
        }
        else
        {
            motorDriverData[i].di3 = 0;
        }
        if ((data & 0x8) != 0)
        {
            motorDriverData[i].di4 = 1;
        }
        else
        {
            motorDriverData[i].di4 = 0;
        }
    }
    catch
    {
    }
}
else if (Msg.StartsWith("DO="))
{
    //here is the digital output message
    try
    {
        Msg = Msg.Remove(0, 3);
        byte data = byte.Parse(Msg);
        if ((data & 0x1) != 0)
        {
            motorDriverData[i].do1 = 1;
        }
        else
        {
            motorDriverData[i].do1 = 0;
        }
        if ((data & 0x2) != 0)
        {
            motorDriverData[i].do2 = 1;
        }
        else
        {
            motorDriverData[i].do2 = 0;
        }
    }
    catch
    {
    }
}
else if (Msg.StartsWith("P="))
{
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');
        motorDriverData[i].motPower1 =
int.Parse(strData[0]);
        motorDriverData[i].motPower2 =
int.Parse(strData[1]);
    }
    catch
    {
    }
}

```

```
        if (i == 0)
        {
            ///      lblMot1Power.Text =
motorDriverData[i].motPower1.ToString();
            //      lblMot2Power.Text =
motorDriverData[i].motPower2.ToString();
        }
        else if (i == 1)
        {
            //      lblMot3Power.Text =
motorDriverData[i].motPower1.ToString();
            //      lblMot4Power.Text =
motorDriverData[i].motPower2.ToString();
        }
    }
    catch
    {
    }
}
else if (Msg.StartsWith("S="))
{
    //here is the velocity message
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');
        motorDriverData[i].motEncS1 =
int.Parse(strData[0]);
        motorDriverData[i].motEncS2 =
int.Parse(strData[1]);
        if (i == 0)
        {
            lblVel1.Text =
motorDriverData[i].motEncS1.ToString();
            lblVel2.Text =
motorDriverData[i].motEncS2.ToString();
        }
        else if (i == 1)
        {
            lblVel3.Text =
motorDriverData[i].motEncS1.ToString();
            lblVel4.Text =
motorDriverData[i].motEncS2.ToString();
        }
    }
    catch
    {
    }
}
else if (Msg.StartsWith("T="))
{
    //here is the motor driver board temperature message
    try
    {
        Msg = Msg.Remove(0, 2);
        string[] strData = Msg.Split(':');
        motorDriverData[i].ch1Temp =
double.Parse(strData[0]);
        motorDriverData[i].ch2Temp =
double.Parse(strData[1]);
    }
}
```

```

        catch
        {
        }
    }
    else if (Msg.StartsWith("V="))
    {
        //here is the power voltage message
        try
        {
            Msg = Msg.Remove(0, 2);
            string[] strData = Msg.Split(':');
            motorDriverData[i].drvVoltage =
double.Parse(strData[0]) / 10;
            motorDriverData[i].batVoltage =
double.Parse(strData[1]) / 10;
            motorDriverData[i].reg5VVoltage =
double.Parse(strData[2]) / 1000;
            if (i == 0)
            {
                textBox20.Text =
motorDriverData[i].batVoltage.ToString();
            }
        }
        catch
        {
        }
    }
    else if (Msg.StartsWith("CR="))
    {
        //here is the encoder relative reading message
        try
        {
            Msg = Msg.Remove(0, 3);
            string[] strData = Msg.Split(':');
            motorDriverData[i].motEncCR1 =
int.Parse(strData[0]);
            motorDriverData[i].motEncCR2 =
int.Parse(strData[1]);
        }
        catch
        {
        }
    }
    else if (Msg.StartsWith("FF="))
    {
        //here is the motor driver board message
        try
        {
            Msg = Msg.Remove(0, 3);
            byte data = byte.Parse(Msg);
            motorDriverData[i].statusFlag = data;
            string errorStr = "";
            if ((data & 0x1) != 0)
            {
                //checkBoxOverHeat.Checked = true;
                errorStr = "OH";
            }
            else
            {
                //checkBoxOverHeat.Checked = false;
            }
            if ((data & 0x2) != 0)

```

```
{
    //checkBoxOverVol.Checked = true;
    errorStr += "+OV";
}
else
{
    //checkBoxOverVol.Checked = false;
}
if ((data & 0x4) != 0)
{
    //checkBoxUnderVol.Checked = true;
    errorStr += "+UV";
}
else
{
    //checkBoxUnderVol.Checked = false;
}
if ((data & 0x8) != 0)
{
    //checkBoxShort.Checked = true;
    errorStr += "SHT";
}
else
{
    //checkBoxShort.Checked = false;
}
if ((data & 0x10) != 0)
{
    //checkBoxEStop.Checked = true;
    errorStr += "+ESTOP";
    blnEStop = true;
}
else
{
    //checkBoxEStop.Checked = false;
    blnEStop = false;
}
if ((data & 0x20) != 0)
{
    //checkBoxSepexFault.Checked = true;
    errorStr += "SEPF";
}
else
{
    //checkBoxSepexFault.Checked = false;
}
if ((data & 0x40) != 0)
{
    //checkBoxPromFault.Checked = true;
    errorStr += "+PromF";
}
else
{
    //checkBoxPromFault.Checked = false;
}
if ((data & 0x80) != 0)
{
    //checkBoxConfigFault.Checked = true;
    errorStr += "+ConfF";
}
else
{

```

```

        //checkBoxConfigFault.Checked = false;
    }

    if (errorStr.Length > 1)
    {
        if (i == 0)
            textBox14.Text = errorStr;
        else if (i == 1)
            textBox29.Text = errorStr;
    }
    else
    {
        if (i == 0)
            textBox14.Text = "OK";
        else if (i == 1)
            textBox29.Text = "OK";
    }
    }
    catch
    {
    }
}
else if (Msg.StartsWith("+"))
{
    //right command ack
    if (i == 0)
    {
        //    lblMD1Ack.Text = "+";
    }
    else if (i == 1)
    {
        //    lblMD2Ack.Text = "+";
    }
}
else if (Msg.StartsWith("-"))
{
    //right command ack
    if (i == 0)
    {
        //    lblMD1Ack.Text = "-";
    }
    else if (i == 1)
    {
        //    lblMD2Ack.Text = "-";
    }
}
}
}
}

private double Trans2Temperature(double adValue)
{
    //for new temperature sensor
    double tempM = 0;
    double k = (adValue / FULLAD);
    double resValue = 0;
    if (k != 0)
    {
        resValue = 10000 * (1 - k) / k;        //AD value to resistor
    }
    else
    {
        resValue = resTable[0];
    }
}

```

```
    }

    int index = -1;
    if (resValue >= resTable[0])           //too lower
    {
        tempM = -20;
    }
    else if (resValue <= resTable[24])
    {
        tempM = 100;
    }
    else
    {
        for (int i = 0; i < 24; i++)
        {
            if ((resValue <= resTable[i]) && (resValue >= resTable[i
+ 1]))
            {
                index = i;
                break;
            }
        }
        if (index >= 0)
        {
            tempM = tempTable[index] + (resValue - resTable[index]) /
(resTable[index + 1] - resTable[index]) * (tempTable[index + 1] -
tempTable[index]);
        }
        else
        {
            tempM = 0;
        }
    }

    return tempM;
}

private void tmrPing_Tick(object sender, EventArgs e)
{
    try
    {
        if (drrobotMainComm != null)
        {
            drrobotMainComm.SendCommand("PING");
        }
    }
    catch
    {
    }
}

#endregion
#region buttons control
private void btnEStopWheel_Click(object sender, EventArgs e)
{
    if (btnEStopWheel.Text == "Oprire Roti")
    {
        btnEStopWheel.Text = "Activare Roti";
        sendEStopCmd();
    }
    else
    {

```

```

        btnEStopWheel.Text = "Oprire Roti";
        sendEStopRelease();
    }
}

private void btnCalIMU_Click(object sender, EventArgs e)
{
    drrobotMainComm.SendCommand("SYS CAL");
}

public class MotorDriverData
{
    public double motAmp1 = 0;
    public double motAmp2 = 0;
    public int motPower1 = 0;
    public int motPower2 = 0;
    public int motEncP1 = 0;
    public int motEncP2 = 0;
    public int motEncS1 = 0;
    public int motEncS2 = 0;
    public double motTemp1 = 0;
    public double motTemp2 = 0;
    public double drvVoltage = 0;
    public double batVoltage = 0;
    public double reg5VVoltage = 0;
    public double ai3 = 0;
    public double ai4 = 0;
    public int di3 = 0;
    public int di4 = 0;
    public int do1 = 0;
    public int do2 = 0;
    public double intTemp = 0;
    public double ch1Temp = 0;
    public double ch2Temp = 0;
    public int statusFlag = 0;
    public int model = 0;
    public int mode2 = 0;
    public int motEncCR1 = 0;
    public int motEncCR2 = 0;
    public MotorDriverData()
    {
        motAmp1 = 0;
        motAmp2 = 0;
        motPower1 = 0;
        motPower2 = 0;
        motEncP1 = 0;
        motEncP2 = 0;
        motEncS1 = 0;
        motEncS2 = 0;
        motTemp1 = 0;
        motTemp2 = 0;
        drvVoltage = 0;
        batVoltage = 0;
        reg5VVoltage = 0;
        ai3 = 0;
        ai4 = 0;
        di3 = 0;
        di4 = 0;
        do1 = 0;
        do2 = 0;
        intTemp = 0;
    }
}

```

```
        ch1Temp = 0;
        ch2Temp = 0;
        statusFlag = 0;
        model = 0;
        mode2 = 0;
        motEncCR1 = 0;
        motEncCR2 = 0;
    }
}

private void apasare(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Space)
    {
        trackBarForwardPower.Value = 0;
        trackBarTurnPower.Value = 0;

    }
    if (e.KeyCode == Keys.Up)
    {
        trackBarForwardPower.Value = trackBarForwardPower.Value +
100;

    }

    if (e.KeyCode == Keys.Down)
    {
        trackBarForwardPower.Value -= 100;

    }

    if (e.KeyCode == Keys.Left)
    {
        trackBarTurnPower.Value -= 30;

    }

    if (e.KeyCode == Keys.Right)
    {
        trackBarTurnPower.Value += 30;
    }
}

private void trackBarForwardPower_ValueChanged_1(object sender,
EventArgs e)
{
    drvRobot_trackbar();
}

private void textBox1_TextChanged(object sender, EventArgs e)
{

```

```
}

private void button1_Click(object sender, EventArgs e)
{
    textBox1.Select();
}

private void button2_Click(object sender, EventArgs e)
{
    trackBarForwardPower.Value = 0;
    trackBarTurnPower.Value = 0;
}

private void button3_Click(object sender, EventArgs e)
{
    trackBarTurnPower.Value = 300;

    for (int i = 0; i < 1340; i++)
    {
        Thread.Sleep(1);
    }
    trackBarTurnPower.Value = 0;
    sendEStopRelease();
}

private void button4_Click(object sender, EventArgs e)
{
    trackBarTurnPower.Value = -300;

    for (int i = 0; i < 1340; i++)
    {
        Thread.Sleep(1);
    }
    trackBarTurnPower.Value = 0;
    sendEStopRelease();
}

private void button5_Click(object sender, EventArgs e)
{
    trackBarTurnPower.Value = 480;

    for (int i = 0; i < 1340; i++)
    {
        Thread.Sleep(1);
    }
    trackBarTurnPower.Value = 0;
    sendEStopRelease();
}

private void button6_Click(object sender, EventArgs e)
{
}

delegate void SetTextCallback(string text);

private void SetText(string text)
{

```

```
        if (this.textBox3.InvokeRequired)
        {
            SetTextCallback d = new SetTextCallback(SetText);
            this.Invoke(d, new object[] { text });
        }
        else
        {
            this.textBox9.Text = text;
        }
    }

    delegate void SetTextCallback13(string text);

    private void SetText1(string text)
    {
        if (this.textBox8.InvokeRequired)
        {
            SetTextCallback13 d = new SetTextCallback13(SetText1);
            this.Invoke(d, new object[] { text });
        }
        else
        {
            this.textBox8.Text = text;
        }
    }

    delegate void SetSpeedCallback(int t);

    private void SetSpeed(int t)
    {
        if (this.trackBarForwardPower.InvokeRequired)
        {
            SetSpeedCallback s = new SetSpeedCallback(SetSpeed);
            this.Invoke(s, new object[] { t });
        }
        else
        {
            this.trackBarForwardPower.Value = t;
        }
    }

    }

    delegate void SetTurnCallback(int z);

    private void SetTurn(int z)
    {
        if (this.trackBarTurnPower.InvokeRequired)
        {
            SetTurnCallback s = new SetTurnCallback(SetTurn);
            this.Invoke(s, new object[] { z });
        }
        else
        {
            this.trackBarTurnPower.Value = z;
        }
    }
}
```

```

delegate void SetvarCallback(int v);

private void Setvar(int v)
{
    if (this.trackBarTurnPower.InvokeRequired)
    {
        SetvarCallback s = new SetvarCallback(Setvar);
        this.Invoke(s, new object[] { v });
    }
    else
    {
        this.trackBarTurnPower.Value += v;
    }
}

delegate void ReduceCallback(int r);
private void Reduce(int r)
{
    if (this.trackBarTurnPower.InvokeRequired &&
this.trackBarForwardPower.InvokeRequired)
    {
        ReduceCallback s = new ReduceCallback(Setvar);
        this.Invoke(s, new object[] { r });
    }
    else
    {
        if (this.trackBarForwardPower.Value != 0)
        {
            if (this.trackBarForwardPower.Value > 0)
            {
                this.trackBarForwardPower.Value -= r;
            }
            if (this.trackBarForwardPower.Value < 0)
            {
                this.trackBarForwardPower.Value += r;
            }
        }
    }
}

}
delegate void Checkauto(bool z);

private void Checkautonom(bool z)
{
    if (this.checkBox1.InvokeRequired)
    {
        Checkauto s = new Checkauto(Checkautonom);
        this.Invoke(s, new object[] { z });
    }
    else
    {
        this.checkBox1.Checked = z;
    }
}
}

```

```
delegate void UnCheckauto(bool z);

private void UnCheckautonom(bool z)
{
    if (this.checkBox1.InvokeRequired)
    {
        UnCheckauto s = new UnCheckauto(UnCheckautonom);
        this.Invoke(s, new object[] { z });
    }
    else
    {
        this.checkBox2.Checked = z;
    }
}

delegate void ArmCallback(int y);
private void Arm(int y)
{
    if (this.trackBarTurnPower.InvokeRequired &&
this.trackBarForwardPower.InvokeRequired)
    {
        ArmCallback s = new ArmCallback(Setvar);
        this.Invoke(s, new object[] { y });
    }
    else
    {
        drRobotCommMot1.SendCommand("!PR 2 " +
jointMotorData[1].jointCmd.ToString() + "\r");
    }
}

}

#endregion

#region trackbars EMGU
delegate void SetTextCallback123(string text);

delegate void SetT(float i);

private void SetT1(float i)
{
    if (this.textBox1.InvokeRequired)
    {
        SetT d = new SetT(SetT1);
        this.Invoke(d, new object[] { i });
    }
    else
    {
        this.a1 = trackBar1.Value;
        this.label11.Text = trackBar1.Value.ToString();
    }
}
```

```

    }
}
delegate void SetTel1(string i);

private void SetT22(string i)
{
    if (this.textBox1.InvokeRequired)
    {
        SetTel1 d = new SetTel1(SetT22);
        this.Invoke(d, new object[] { i });
    }
    else
    {
        this.textBox9.Text = i;
    }
}
}
delegate void SetTel(float i);

private void SetT2(float i)
{
    if (this.textBox1.InvokeRequired)
    {
        SetTel d = new SetTel(SetT2);
        this.Invoke(d, new object[] { i });
    }
    else
    {
        this.b1 = trackBar2.Value;
        this.label12.Text = trackBar2.Value.ToString();
    }
}
}
delegate void SetTe2(float i);

private void SetT3(float i)
{
    if (this.textBox1.InvokeRequired)
    {
        SetTe2 d = new SetTe2(SetT3);
        this.Invoke(d, new object[] { i });
    }
    else
    {
        this.c1 = trackBar3.Value;
        this.label15.Text = trackBar3.Value.ToString();
    }
}
}
delegate void SetTe3(float i);

private void SetT4(float i)
{
    if (this.textBox1.InvokeRequired)
    {
        SetTe3 d = new SetTe3(SetT4);
        this.Invoke(d, new object[] { i });
    }
}

```

```
        else
        {
            this.a2 = trackBar6.Value;
            this.label17.Text = trackBar6.Value.ToString();
        }
    }
    delegate void SetTe4(float i);

    private void SetT5(float i)
    {
        if (this.textBox1.InvokeRequired)
        {
            SetTe4 d = new SetTe4(SetT5);
            this.Invoke(d, new object[] { i });
        }
        else
        {
            this.b2 = trackBar5.Value;
            this.label18.Text = trackBar5.Value.ToString();
        }
    }
    delegate void SetTe6(float i);

    private void SetT6(float i)
    {
        if (this.textBox1.InvokeRequired)
        {
            SetTe6 d = new SetTe6(SetT6);
            this.Invoke(d, new object[] { i });
        }
        else
        {
            this.c2 = trackBar4.Value;
            this.label2.Text = trackBar4.Value.ToString();
        }
    }
}
#endregion
#region EMGU CV

void ProcessFrame_NewFrame1(object sender,
AForge.Video.NewFrameEventArgs eventArgs1)
{
    if (mode == 2)
    {
        #region variabile procesare CERC

        Bitmap FrameData1 = new Bitmap(eventArgs1.Frame);
        Image<Bgr, Byte> myImage1 = new Image<Bgr, Byte>(FrameData1);
        // pictureBox1.Image = myImage;

        Image<Bgr, Byte> img1 = myImage1;
        Image<Hsv, Byte> hsvimg = img1.Convert<Hsv, Byte>();
        SetT1(a1);
        SetT2(b1);
        SetT3(c1);
        SetT4(a2);
        SetT5(b2);
        SetT6(c2);
```

```

        Hsv lowerLimit = new Hsv(a1, b1, c1);
        Hsv upperLimit = new Hsv(a2, b2, c2);
        //SetText1(lowerLimit.ToString());

        Image<Gray, byte> imageHSV = hsvimg.InRange(lowerLimit,
upperLimit);

        Image<Gray, Byte> graySoft = imageHSV.Convert<Gray,
Byte>().PyrDown().PyrUp();
        Image<Gray, Byte> gray = graySoft.SmoothGaussian(3);
        gray = gray.AddWeighted(graySoft, 1.5, -0.5, 0);
        imageBox5.Image = img1;
        imageBox4.Image = graySoft;

        Image<Gray, Byte> bin = gray.ThresholdBinary(new Gray(70),
new Gray(255));

        Gray cannyThreshold = new Gray(100);
        Gray cannyThresholdLinking = new Gray(100);
        Gray circleAccumulatorThreshold = new Gray(47);

        Image<Gray, Byte> cannyEdges =
gray.Canny(cannyThreshold.Intensity, cannyThresholdLinking.Intensity);

        Mat smoothedFrame = new Mat();
        Mat MedianFrame = new Mat();
        CvInvoke.GaussianBlur(gray, smoothedFrame, new Size(3, 3),
1); //filter out noises
        CvInvoke.MedianBlur(imageHSV, MedianFrame, 15);
        Image<Gray, Byte> median = MedianFrame.ToImage<Gray, Byte>();
        imageBox3.Image = median;
        //Load the image from file

        Mat grayFrame1 = new Mat();
        CvInvoke.CvtColor(img1, grayFrame1,
ColorConversion.Bgr2Gray);

        Mat smallGrayFrame1 = new Mat();
        CvInvoke.PyrDown(grayFrame1, smallGrayFrame1);
        Mat smoothedGrayFrame1 = new Mat();
        CvInvoke.PyrUp(smallGrayFrame1, smoothedGrayFrame1);

        //Image<Gray, Byte> smallGrayFrame = grayFrame.PyrDown();
        //Image<Gray, Byte> smoothedGrayFrame =
smallGrayFrame.PyrUp();
        Mat cannyFrame1 = new Mat();
        CvInvoke.Canny(smoothedGrayFrame1, cannyFrame1, 100, 60);
        PointF a;
        float b;
        double Resolution = 2;
        double MinDistance = 1000.0;
        int MinRadius = 0;
        int MaxRadius = 1000;

        Image<Gray, Byte> graySoft1 = img1.Convert<Gray,
Byte>().PyrDown().PyrUp();
        Image<Gray, Byte> gray1 = graySoft1.SmoothGaussian(3);
        gray1 = gray1.AddWeighted(graySoft1, 1.5, -0.5, 0);

```

```
Image<Gray, Byte> bin1 = gray1.ThresholdBinary(new Gray(60),
new Gray(255));

// Gray cannyThreshold = new Gray(100);
// Gray cannyThresholdLinking = new Gray(100);
// Gray circleAccumulatorThreshold = new Gray(160);

Image<Gray, Byte> cannyEdges1 =
median.Canny(cannyThreshold.Intensity, cannyThresholdLinking.Intensity);

imageBox6.Image = cannyEdges1;
//timp_brat.Tick += new EventHandler(timp_brat_Tick);

CircleF[] Circles = median.Clone().HoughCircles(
    cannyThreshold,
    circleAccumulatorThreshold,
    Resolution, //Resolution of the
accumulator used to detect centers of the circles
    MinDistance, //min distance
    MinRadius, //min radius
    MaxRadius //max radius
)[0]; //Get the circles from the
first channel

//textBox4.Text=Circles.ToString();
#endregion
imageBox1.Image = cannyEdges1;

if (test == false && test1 == true)
{
    timp_brat_Tick();
    // timp_brat.Start();
}
// SetText(Circles.Length.ToString());
if (checkBox2.Checked == true && verificare == true)
{
    if(btnEstop_ev==true){
        btnEstop_ev = false;
        //btnEStopWheel_ev();
        sendEStopCmd();
    }

    #region miscare initiala BRAT !
    verificare = false;
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;
    bool armCtrl2 = false;
    bool armCtrl1 = false;

    //byte[] buttons = joyState[0].GetButtons();
    preSetButton[0] = 1;
    preSetButton[1] = 1;
    //move manipulator J1
    jointMotorData[0].cmdDir = 1;
    jointMotorData[0].jointCmd = -200;
    if (preSetButton[0] != 4)
    {
```

```

        jointMotorData[0].jointCmd = -200;
        jointMotorData[0].jointCtrl = true;
        joyButtonCnt[0] = 0;
        jointMotorData[0].jointJoy = true;
    }
    else
    {
        joyButtonCnt[0]++;
        if (joyButtonCnt[0] == JOYDELAY)
        {
            jointMotorData[0].jointCtrl = true;
            jointMotorData[0].jointCmd = -200;
            jointMotorData[0].jointJoy = true;
        }
        if (!jointMotorData[0].protect)
            drRobotCommMot1.SendCommand("!PR 1 " +
jointMotorData[0].jointCmd.ToString() + "\r");

        #endregion
    }

    if (Circles.Length == 1 && poz_J1 == true &&
checkBox2.Checked == true)
    {
        cerc_gasit = true;
        //Checkautonom(false);

        foreach (CircleF circle in Circles)
        {

            a = circle.Center;
            b = circle.Radius;

            CvInvoke.Circle(img1, Point.Round(circle.Center),
(int)circle.Radius, new Bgr(Color.Red).MCvScalar, 2);
            imageBox2.Image = img1;
            SetText1(b.ToString());
            // SetText(b.ToString());

            if (a.X > 130 && a.X < 190)
            {
                if (b < 53)
                {
                    // circleAccumulatorThreshold = new Gray(40);

                    if (b < 30)
                    {
                        JOINT1_CMD = 4;
                        JOINT2_CMD = 4;
                    }
                    else
                    {

```

```
        JOINT1_CMD = 1;
        JOINT2_CMD = 1;
    }
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;
    bool armCtrl2 = false;
    bool armCtrl1 = false;

    //byte[] buttons = joyState[0].GetButtons();
    preSetButton[0] = 1;
    preSetButton[1] = 1;
    //move manipulator J1
    jointMotorData[0].cmdDir = 1;
    jointMotorData[0].jointCmd = -JOINT1_CMD;
    if (preSetButton[0] != 4)
    {

        jointMotorData[0].jointCmd = -JOINT1_CMD;
        jointMotorData[0].jointCtrl = true;
        joyButtonCnt[0] = 0;
        jointMotorData[0].jointJoy = true;
    }
    else
    {
        joyButtonCnt[0]++;
        if (joyButtonCnt[0] == JOYDELAY)
        {
            jointMotorData[0].jointCtrl = true;
            jointMotorData[0].jointCmd = -

            jointMotorData[0].jointJoy = true;
        }
    }
    if (!jointMotorData[0].protect)
        drRobotCommMot1.SendCommand("!PR 1 " +
jointMotorData[0].jointCmd.ToString() + "\r");
    }
    if (b > 70)
    {
        // circleAccumulatorThreshold = new Gray(80);
        //for arm control
        jointMotorData[0].jointCmd = 0;
        jointMotorData[1].jointCmd = 0;
        bool armCtrl2 = false;
        bool armCtrl1 = false;

        //byte[] buttons = joyState[0].GetButtons();
        preSetButton[0] = 1;
        preSetButton[1] = 1;
        jointMotorData[0].cmdDir = -1;
        jointMotorData[0].jointCmd = JOINT1_CMD;
        if (preSetButton[0] != 6)
        {
            jointMotorData[0].jointCmd = JOINT1_CMD;
            jointMotorData[0].jointCtrl = true;
            joyButtonCnt[0] = 0;
            jointMotorData[0].jointJoy = true;
        }
        else
        {
            joyButtonCnt[0]++;
            if (joyButtonCnt[0] == JOYDELAY)
            {
```

```

        jointMotorData[0].jointCtrl = true;
        jointMotorData[0].jointCmd =

JOINT1_CMD;

        jointMotorData[0].jointJoy = true;
    }
}
if (!jointMotorData[0].protect)
    drRobotCommMot1.SendCommand("!PR 1 " +
jointMotorData[0].jointCmd.ToString() + "\r");
}

if (a.Y < 165)
{
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;
    bool armCtrl2 = false;
    bool armCtrl1 = false;

    //byte[] buttons = joyState[0].GetButtons();
    preSetButton[0] = 1;
    preSetButton[1] = 1;

    jointMotorData[1].cmdDir = 1;
    jointMotorData[1].jointCmd = -JOINT2_CMD;
    if (preSetButton[1] != 7)
    {
        //textBox4.Text = "introduc datele de
miscare";

        jointMotorData[1].jointCmd = -JOINT2_CMD;
        jointMotorData[1].jointCtrl = true;
        joyButtonCnt[1] = 0;
        jointMotorData[1].jointJoy = true;
    }

    if (!jointMotorData[1].protect)
    {
        //textBox5.Text = "miscare sus";

        drRobotCommMot1.SendCommand("!PR 2 " +
jointMotorData[1].jointCmd.ToString() + "\r");
        //Arm(jointMotorData[1].jointCmd);
    }
}
if (a.Y > 177)
{
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;

    preSetButton[0] = 1;
    preSetButton[1] = 1;

    jointMotorData[1].cmdDir = -1;
    jointMotorData[1].jointCmd = JOINT2_CMD;
    if (preSetButton[1] != 7)
    {
        // textBox4.Text = "introduc datele de
miscare";

```

```
        jointMotorData[1].jointCmd = JOINT2_CMD;
        jointMotorData[1].jointCtrl = true;
        joyButtonCnt[1] = 0;
        jointMotorData[1].jointJoy = true;
    }

    if (!jointMotorData[1].protect)
    {
        //textBox5.Text = "miscare jos";

        drRobotCommMot1.SendCommand("!PR 2 " +
jointMotorData[1].jointCmd.ToString() + "\r");
        //Arm(jointMotorData[1].jointCmd);

    }
}
if (a.Y <= 177 && a.Y >= 165)
{

    // poz_J1 = false;
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;

    preSetButton[0] = 1;
    preSetButton[1] = 1;

    jointMotorData[1].cmdDir = -1;
    jointMotorData[1].jointCmd = 0;
    if (preSetButton[1] != 7)
    {
        // textBox4.Text = "introduc datele de
miscare";

        jointMotorData[1].jointCmd = 0;
        jointMotorData[1].jointCtrl = true;
        joyButtonCnt[1] = 0;
        jointMotorData[1].jointJoy = true;
    }

    if (!jointMotorData[1].protect)
    {
        //textBox5.Text = "miscare jos";

        drRobotCommMot1.SendCommand("!PR 2 " +
jointMotorData[1].jointCmd.ToString() + "\r");
        //Arm(jointMotorData[1].jointCmd);

    }

}
if (b >= 52 && b <= 70)
{
    counter_cerc++;
    if (counter_cerc > 30)
    {
        if (test == true)
        {
            test = false;
        }
        if (prindere == true)
        {
```

```

        prindere = false;
        strange();

    }
    // poz_J2 = false;
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;
    bool armCtrl2 = false;
    bool armCtrl1 = false;

    //byte[] buttons =

joyState[0].GetButtons();

    preSetButton[0] = 1;
    preSetButton[1] = 1;
    jointMotorData[0].cmdDir = -1;
    jointMotorData[0].jointCmd = 0;
    if (preSetButton[0] != 6)
    {
        jointMotorData[0].jointCmd = 0;
        jointMotorData[0].jointCtrl = true;
        joyButtonCnt[0] = 0;
        jointMotorData[0].jointJoy = true;
    }
    else
    {
        joyButtonCnt[0]++;
        if (joyButtonCnt[0] == JOYDELAY)
        {
            jointMotorData[0].jointCtrl =

true;

            jointMotorData[0].jointCmd = 0;
            jointMotorData[0].jointJoy =

true;

        }
    }
    if (!jointMotorData[0].protect)
        drRobotCommMot1.SendCommand("!PR 1 "
+ jointMotorData[0].jointCmd.ToString() + "\r");

    }

}

/*if (b >= 45 && b <= 60)
{
    if (fanion_gripper == true)
    {
        timp_gripper.Interval = 3000;
        timp_gripper.Start();
        strange();
        timp_gripper.Tick += new

EventHandler(Stop_grip);

    }

}*/

```

```
        Thread.Sleep(25);
    }
}

else
{
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;

    preSetButton[0] = 1;
    preSetButton[1] = 1;

    jointMotorData[1].cmdDir = -1;
    jointMotorData[1].jointCmd = 0;
    if (preSetButton[1] != 7)
    {
        // textBox4.Text = "introduc datele de miscare";
        jointMotorData[1].jointCmd = 0;
        jointMotorData[1].jointCtrl = true;
        joyButtonCnt[1] = 0;
        jointMotorData[1].jointJoy = true;
    }

    if (!jointMotorData[1].protect)
    {
        //textBox5.Text = "miscare jos";

        drRobotCommMot1.SendCommand("!PR 2 " +
jointMotorData[1].jointCmd.ToString() + "\r");
        //Arm(jointMotorData[1].jointCmd);
    }

    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;
    bool armCtrl2 = false;
    bool armCtrl1 = false;

    //byte[] buttons = joyState[0].GetButtons();
    preSetButton[0] = 1;
    preSetButton[1] = 1;
    jointMotorData[0].cmdDir = -1;
    jointMotorData[0].jointCmd = 0;
    if (preSetButton[0] != 6)
    {
        jointMotorData[0].jointCmd = 0;
        jointMotorData[0].jointCtrl = true;
        joyButtonCnt[0] = 0;
        jointMotorData[0].jointJoy = true;
    }
    else
    {
```

```

        joyButtonCnt[0]++;
        if (joyButtonCnt[0] == JOYDELAY)
        {
            jointMotorData[0].jointCtrl = true;
            jointMotorData[0].jointCmd = 0;
            jointMotorData[0].jointJoy = true;
        }
    }
    if (!jointMotorData[0].protect)
        drRobotCommMot1.SendCommand("!PR 1 " +
jointMotorData[0].jointCmd.ToString() + "\r");

    }
}

void ProcessFrame_NewFrame(object sender,
AForge.Video.NewFrameEventArgs eventArgs)
{
    if (mode == 2)
    {
        #region variabile procesare CULOARE
        //drawOccMap();
        Mat frame = new Mat();
        // _capture.Retrieve(frame, 0);
        Bitmap FrameData = new Bitmap(eventArgs.Frame);
        Image<Bgr, Byte> myImage = new Image<Bgr, Byte>(FrameData);
        // pictureBox1.Image = myImage;

        Image<Bgr, Byte> img = myImage;
        //Image<Bgr, Byte> img = frame.ToImage<Bgr, Byte>();

        Image<Hsv, Byte> hsvimg = img.Convert<Hsv, Byte>();
        SetT1(a1);
        SetT2(b1);
        SetT3(c1);
        SetT4(a2);
        SetT5(b2);
        SetT6(c2);
        Hsv lowerLimit = new Hsv(a1, b1, c1);
        Hsv upperLimit = new Hsv(a2, b2, c2);
        //SetText1(lowerLimit.ToString());

        Image<Gray, byte> imageHSV = hsvimg.InRange(lowerLimit,
upperLimit);

        Image<Gray, Byte> graySoft = imageHSV.Convert<Gray,
Byte>().PyrDown().PyrUp();
        Image<Gray, Byte> gray = graySoft.SmoothGaussian(3);
        gray = gray.AddWeighted(graySoft, 1.5, -0.5, 0);

        Image<Gray, Byte> bin = gray.ThresholdBinary(new Gray(70),
new Gray(255));

        Gray cannyThreshold = new Gray(100);
        Gray cannyThresholdLinking = new Gray(100);
        Gray circleAccumulatorThreshold = new Gray(100);

```

```
        Image<Gray, Byte> cannyEdges =
gray.Canny(cannyThreshold.Intensity, cannyThresholdLinking.Intensity);

        Mat smoothedFrame = new Mat();
        Mat MedianFrame = new Mat();
        CvInvoke.GaussianBlur(frame, smoothedFrame, new Size(3, 3),
1); //filter out noises
        CvInvoke.MedianBlur(imageHSV, MedianFrame, 25);
        Image<Gray, Byte> median = MedianFrame.ToImage<Gray, Byte>();
        //frame._SmoothGaussian(3);

        CvBlobs blobs = new CvBlobs();
        _blobDetector.Detect(median, blobs);
        blobs.FilterByArea(1, int.MaxValue);

        //_tracker.Process(smoothedFrame, foregroundMask);
        maxblob = 0;
        #endregion
        if (blobs.Count != 0 && autonom == true)
        {
            Checkautonom(false);

            foreach (var pair in blobs)
            {
                if (pair.Value.Area > maxblob)
                {
                    maxblob = pair.Value.Area;
                    largestContour = pair.Value;
                }
            }
            CvBlob blo = largestContour;

            CvInvoke.Rectangle(img, blo.BoundingBox, new
MCvScalar(0.0, 50.0, 255.0), 2);
            //CvInvoke.PutText(frame, blo.Label.ToString(),
Point.Round(blo.Centroid), FontFace.HersheyDuplex, 1.0, new MCvScalar(0.0,
255.0, 255.0));

            //SetText1(blo.Centroid.ToString());
            // SetText(counter_cerc.ToString());
            // pictureBox1.Image = MedianFrame;
            //blo.imageBox2.Image = img;
            //imageBox1.Image = median;
            pictureBox1.Image = img;
            //imageBox2.Image = cannyEdges;
            // imageBox1.Image = cannyEdges;

            if (blo.Area > 4500 && blo.Area < 6500)
            {
                if (!protectMotorTemp)
                {
                    SetSpeed(0);
                    UnCheckautonom(true);
                    Checkautonom(false);
                }
            }
        }
        else
        {
            UnCheckautonom(false);
        }
    }
}
```

```

        if (blo.Area <= 5000)
        {
            if (!protectMotorTemp)
            {
                SetSpeed(65);
            }
        }
        if (blo.Area >= 6000)
        {
            if (!protectMotorTemp)
            {
                SetSpeed(-65);
            }
        }

        if (blo.Centroid.X >= 90 && blo.Centroid.X <= 140)
        {
            if (!protectMotorTemp)
            {
                SetTurn(0);
            }
        }

        if (blo.Centroid.X > 140)
        {
            if (!protectMotorTemp)
            {
                SetTurn(200);
            }
        }

        if (blo.Centroid.X < 90)
        {
            if (!protectMotorTemp)
            {
                SetTurn(-200);
            }
        }
    }

    else
    {
        Checkautonom(true);
    }
}

private void frmMain_Load(object sender, System.EventArgs e)
{
    VideoStream.Source = "http://192.168.0.75:8081/axis-
cgi/mjpg/video.cgi?resolution=320x240&req_fps=30&.mjpg";
    VideoStream.Login = "root";
    VideoStream.Password = "drrobot";

    VideoStream.Start();
}

```

```
        VideoStream1.Source = "http://192.168.0.74:8082/axis-  
cgi/mjpg/video.cgi?resolution=320x240&req_fps=30&.mjpg";  
        VideoStream1.Login = "root";  
        VideoStream1.Password = "drrobot";  
  
        VideoStream1.Start();  
    }  
  
    private void timp_brat_Tick()//object sender, EventArgs e)  
    {  
        if (checkBox2.Checked == true)  
        {  
            //timp_brat.Stop();  
            poz_J1 = false;  
  
        }  
        else  
        {  
            poz_J1 = true;  
            poz_J2 = true;  
            test = true;  
            test1 = true;  
  
            //MessageBox.Show("Am oprit bratul");  
  
        }  
  
    }  
  
    private void button7_Click(object sender, EventArgs e)  
    {  
        //for arm control  
        jointMotorData[0].jointCmd = 0;  
        jointMotorData[1].jointCmd = 0;  
        bool armCtrl2 = false;  
        bool armCtrl1 = false;  
  
        //byte[] buttons = joyState[0].GetButtons();  
        preSetButton[0] = 1;  
        preSetButton[1] = 1;  
  
        jointMotorData[1].cmdDir = -1;  
        jointMotorData[1].jointCmd = JOINT2_CMD+20;  
        if (preSetButton[1] != 7)  
        {  
            // textBox4.Text = "introduc datele de miscare";  
            jointMotorData[1].jointCmd = JOINT2_CMD+20;  
            jointMotorData[1].jointCtrl = true;  
            joyButtonCnt[1] = 0;  
            jointMotorData[1].jointJoy = true;  
  
        }  
  
        if (!jointMotorData[1].protect)  
        {  
            //textBox5.Text = "miscare jos";  
  
            //drRobotCommMot1.SendCommand("!PR 2 " +  
            jointMotorData[1].jointCmd.ToString() + "\r");  
            Arm(jointMotorData[1].jointCmd);  
  
        }  
    }
```

```

}
private void jos()
{
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;
    bool armCtrl2 = false;
    bool armCtrl1 = false;

    //byte[] buttons = joyState[0].GetButtons();
    preSetButton[0] = 1;
    preSetButton[1] = 1;

    jointMotorData[1].cmdDir = -1;
    jointMotorData[1].jointCmd = JOINT2_CMD;
    if (preSetButton[1] != 7)
    {
        // textBox4.Text = "introduc datele de miscare";
        jointMotorData[1].jointCmd = JOINT2_CMD;
        jointMotorData[1].jointCtrl = true;
        joyButtonCnt[1] = 0;
        jointMotorData[1].jointJoy = true;
    }

    if (!jointMotorData[1].protect)
    {
        //textBox5.Text = "miscare jos";

        //drRobotCommMot1.SendCommand("!PR 2 " +
        jointMotorData[1].jointCmd.ToString() + "\r");
        Arm(jointMotorData[1].jointCmd);
    }
}

private void button6_Click_1(object sender, EventArgs e)
{
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;
    bool armCtrl2 = false;
    bool armCtrl1 = false;

    //byte[] buttons = joyState[0].GetButtons();
    preSetButton[0] = 1;
    preSetButton[1] = 1;

    jointMotorData[1].cmdDir = 1;
    jointMotorData[1].jointCmd = -JOINT2_CMD-20;
    if (preSetButton[1] != 7)
    {
        //textBox4.Text = "introduc datele de miscare";

        jointMotorData[1].jointCmd = -JOINT2_CMD-20;
        jointMotorData[1].jointCtrl = true;
        joyButtonCnt[1] = 0;
        jointMotorData[1].jointJoy = true;
    }
}

```

```
        if (!jointMotorData[1].protect)
        {
            //textBox5.Text = "miscare sus";

            //drRobotCommMot1.SendCommand("!PR 2 " +
jointMotorData[1].jointCmd.ToString() + "\r");
            Arm(jointMotorData[1].jointCmd);
        }
    }

private void sus()
{
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;
    bool armCtrl2 = false;
    bool armCtrl1 = false;

    //byte[] buttons = joyState[0].GetButtons();
    preSetButton[0] = 1;
    preSetButton[1] = 1;

    jointMotorData[1].cmdDir = 1;
    jointMotorData[1].jointCmd = -JOINT2_CMD;
    if (preSetButton[1] != 7)
    {
        //textBox4.Text = "introduc datele de miscare";

        jointMotorData[1].jointCmd = -JOINT2_CMD;
        jointMotorData[1].jointCtrl = true;
        joyButtonCnt[1] = 0;
        jointMotorData[1].jointJoy = true;
    }

    if (!jointMotorData[1].protect)
    {
        //textBox5.Text = "miscare sus";

        //drRobotCommMot1.SendCommand("!PR 2 " +
jointMotorData[1].jointCmd.ToString() + "\r");
        Arm(jointMotorData[1].jointCmd);
    }
}

private void button10_Click(object sender, EventArgs e)
{
    if (button10.Text == "ESTOP")
    {
        try
        {
            drRobotCommMot1.SendCommand("!EX\r");
            button10.Text = "Resume";
        }
        catch
        {
        }
    }
    else
    {
    }
```

```

        try
        {
            drRobotCommMot1.SendCommand("!MG\r");
            button10.Text = "ESTOP";
        }
        catch
        {
        }
    }
}

private void button11_Click(object sender, EventArgs e)
{
    if (button11.Text == "ESTOP")
    {
        try
        {
            drRobotCommMot2.SendCommand("!EX\r");
            button11.Text = "Resume";
        }
        catch
        {
        }
    }
    else
    {
        try
        {
            drRobotCommMot2.SendCommand("!MG\r");
            button11.Text = "ESTOP";
        }
        catch
        {
        }
    }
}

private void button12_Click(object sender, EventArgs e)
{
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;
    bool armCtrl2 = false;
    bool armCtrl1 = false;

    //byte[] buttons = joyState[0].GetButtons();
    preSetButton[0] = 1;
    preSetButton[1] = 1;
    //move manipulator J1
    jointMotorData[0].cmdDir = 1;
    jointMotorData[0].jointCmd = -JOINT1_CMD-20;
    if (preSetButton[0] != 4)
    {
        jointMotorData[0].jointCmd = -JOINT1_CMD-20;
        jointMotorData[0].jointCtrl = true;
        joyButtonCnt[0] = 0;
        jointMotorData[0].jointJoy = true;
    }
    else

```

```
{
    joyButtonCnt[0]++;
    if (joyButtonCnt[0] == JOYDELAY)
    {
        jointMotorData[0].jointCtrl = true;
        jointMotorData[0].jointCmd = -JOINT1_CMD-20;
        jointMotorData[0].jointJoy = true;
    }
}
if (!jointMotorData[0].protect)
    drRobotCommMot1.SendCommand("!PR 1 " +
jointMotorData[0].jointCmd.ToString() + "\r");

}

private void button13_Click(object sender, EventArgs e)
{
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;
    bool armCtrl2 = false;
    bool armCtrl1 = false;

    //byte[] buttons = joyState[0].GetButtons();
    preSetButton[0] = 1;
    preSetButton[1] = 1;
    jointMotorData[0].cmdDir = -1;
    jointMotorData[0].jointCmd = JOINT1_CMD+20;
    if (preSetButton[0] != 6)
    {
        jointMotorData[0].jointCmd = JOINT1_CMD+20;
        jointMotorData[0].jointCtrl = true;
        joyButtonCnt[0] = 0;
        jointMotorData[0].jointJoy = true;
    }
    else
    {
        joyButtonCnt[0]++;
        if (joyButtonCnt[0] == JOYDELAY)
        {
            jointMotorData[0].jointCtrl = true;
            jointMotorData[0].jointCmd = JOINT1_CMD+20;
            jointMotorData[0].jointJoy = true;
        }
    }
    if (!jointMotorData[0].protect)
        drRobotCommMot1.SendCommand("!PR 1 " +
jointMotorData[0].jointCmd.ToString() + "\r");

}

private void button14_Click(object sender, EventArgs e)
{
    drRobotCommMot2.SendCommand("!G 2 0\r");
}
private void Stop_grip(object sender, EventArgs e)
{
```

```

        drRobotCommMot2.SendCommand("!G 2 0\r");
        timp_gripper.Stop();
        fanion_gripper=false;
    }

    private void Autonom(object sender, EventArgs e)
    {
        int p;
        float minim_laser = 30000;
        float minim_stanga = 30000;
        float minim_dreapta = 30000;
        if (checkBox1.Checked&&mode==2)
        {

            textBox5.Text = ((double)disData[250]).ToString();
            textBox6.Text = ((double)disData[550]).ToString();
            textBox7.Text = ((double)disData[850]).ToString();

            for (p = 490; p <= 530; p++)
            {
                if (disData[p] < minim_laser)
                {
                    minim_laser = disData[p];
                }
            }
            for (p = 0; p <= 450; p++)
            {
                if (disData[p] < minim_laser)
                {
                    minim_dreapta = disData[p];
                }
            }
            for (p = 570; p <= 1023; p++)
            {
                if (disData[p] < minim_laser)
                {
                    minim_stanga = disData[p];
                }
            }

            float fata = disData[512];
            float stanga = disData[200];
            float dreapta = disData[900];
            textBox5.Text = stanga.ToString();
            textBox6.Text = fata.ToString();
            textBox7.Text = dreapta.ToString();

            bool verificare_laterala = false;
            bool verificat = false;

            if (minim_laser<1600)//fata < 1200 || disData[570] < 1200
            || disData[454] < 1200)
            {
                trackBarForwardPower.Value = 0;
                trackBarTurnPower.Value = 0;
                verificare_laterala = true;
                textBox4.Text = "S-a oprit robotul";
            }
            else
            {
                // textBox5.Text = "";
                trackBarForwardPower.Value = 150;
            }
        }
    }

```

```
        trackBarTurnPower.Value = 0;
        textBox4.Text = "robotul CAUTA";
        verificare_laterală = false;
    }

    if (verificare_laterală)
    {
        if (stanga < dreapta)
        {
            trackBarTurnPower.Value = -500;

            verificat = true;
            verificare_laterală = false;
            //textBox4.Text = "robotul stanga";
        }
        else
        {
            trackBarTurnPower.Value = 500;

            verificat = false;
            verificare_laterală = false;
            //textBox4.Text = "robotul dreapta";
        }
        /* if (stanga <= 1000 && dreapta <= 1000)
        {
            trackBarTurnPower.Value = 480;

            for (int i = 0; i < 1340; i++)
            {
                Thread.Sleep(1);
            }
            trackBarTurnPower.Value = 0;
            sendEStopRelease();
            textBox4.Text = "robotul 180";
        }*/
    }

}

}

private void trackBar1_Scroll(object sender, EventArgs e)
{
}

private void trackBar2_Scroll(object sender, EventArgs e)
```

```

{
}

private void trackBar3_Scroll(object sender, EventArgs e)
{
}

private void checkBox2_CheckedChanged(object sender, EventArgs e)
{
}

private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
}

private void Tick_harta(object sender, EventArgs e)
{
    drawOccMap();
}

private void Start_Laser(object sender, EventArgs e)
{
    timp_laser.Stop();
    sendCommandLaser("BM");
    contor1++;
    //textBox2.Text = contor1.ToString();
    //MessageBox.Show("start laser");
    drawOccMapBackground();
    // drawOccMap();

    //btnScan.Enabled = true;
}

private void Start_Scan(object sender, EventArgs e)
{
    timp_laser2.Stop();
    sendCommandLaser(SCANCOMMAND);
    firstData = true;
    contor2++;
    //textBox3.Text = contor2.ToString();
    // MessageBox.Show("start scan");
}

private void button15_Click(object sender, EventArgs e)
{
    jointMotorData[3].jointCmd = JOINT4_CMD;
    if ((!jointMotorData[3].protect)
        || ((jointMotorData[3].protect) &&
(jointMotorData[3].preCmdDir == -1)))
    {
        jointMotorData[3].protect = false;

        if (!forceStopManipulator)
        {
            if (!jointMotorData[3].protect)
                drRobotCommMot2.SendCommand("!G 2 " +
jointMotorData[3].jointCmd.ToString() + "\r");
        }
    }
    jointMotorData[3].preCmdDir = 1;
}

```

```
        preSetButton[3] = 18000;
    }
    private void desface()
    {
        jointMotorData[3].jointCmd = JOINT4_CMD;
        if ((!jointMotorData[3].protect)
            || ((jointMotorData[3].protect) &&
                (jointMotorData[3].preCmdDir == -1)))
        {
            jointMotorData[3].protect = false;

            if (!forceStopManipulator)
            {
                if (!jointMotorData[3].protect)
                    drRobotCommMot2.SendCommand("!G 2 " +
jointMotorData[3].jointCmd.ToString() + "\r");
            }
        }
        jointMotorData[3].preCmdDir = 1;
        preSetButton[3] = 18000;
    }

    private void button16_Click(object sender, EventArgs e)
    {
        //move manipulator J4
        jointMotorData[3].jointCmd = -JOINT4_CMD;
        if ((!jointMotorData[3].protect)
            || ((jointMotorData[3].protect) &&
                (jointMotorData[3].preCmdDir == 1)))
        {
            jointMotorData[3].protect = false;
            //myJaguarArm.DcMotorVelocityNonTimeCtr(4,
(short)jointMotorData[3].jointCmd);
            if (!forceStopManipulator)
            {
                if (!jointMotorData[3].protect)
                    drRobotCommMot2.SendCommand("!G 2 " +
jointMotorData[3].jointCmd.ToString() + "\r");
            }
        }
        jointMotorData[3].preCmdDir = -1;
        preSetButton[3] = 0;
    }

    private void strange()
    {
        //move manipulator J4
        jointMotorData[3].jointCmd = -JOINT4_CMD;
        if ((!jointMotorData[3].protect)
            || ((jointMotorData[3].protect) &&
                (jointMotorData[3].preCmdDir == 1)))
        {
            jointMotorData[3].protect = false;
            //myJaguarArm.DcMotorVelocityNonTimeCtr(4,
(short)jointMotorData[3].jointCmd);
            if (!forceStopManipulator)
            {
                if (!jointMotorData[3].protect)
                    drRobotCommMot2.SendCommand("!G 2 " +
jointMotorData[3].jointCmd.ToString() + "\r");
            }
        }
        jointMotorData[3].preCmdDir = -1;
    }
}
```

```

preSetButton[3] = 0;

//tempo.Interval = 4000;
tempo.Enabled = true;
tempo.Elapsed +=new ElapsedEventHandler(tempo_Elapsed);
tempo.Start();

autonom = false;
}
void tempo_Elapsed(object sender, ElapsedEventArgs e){

    // MessageBox.Show("eveniment");
    tempo.Enabled = false;
    tempo.Stop();
    drRobotCommMot2.SendCommand("!G 2 0\r");
    terminat.Enabled = true;
    terminat.Elapsed +=new ElapsedEventHandler(terminat_Elapsed);
    terminat.Start();

}

private void terminat_Elapsed(object sender, ElapsedEventArgs e)
{
    terminat.Enabled = false;
    terminat.Stop();
    sendEStopRelease();
    // checkBox2.Checked = false;

    #region se ridica bratul
    //for arm control
    jointMotorData[0].jointCmd = 0;
    jointMotorData[1].jointCmd = 0;
    bool armCtrl2 = false;
    bool armCtrl1 = false;

    //byte[] buttons = joyState[0].GetButtons();
    preSetButton[0] = 1;
    preSetButton[1] = 1;
    jointMotorData[0].cmdDir = -1;
    jointMotorData[0].jointCmd = 450+ 20;
    if (preSetButton[0] != 6)
    {
        jointMotorData[0].jointCmd = 450+ 20;
        jointMotorData[0].jointCtrl = true;
        joyButtonCnt[0] = 0;
        jointMotorData[0].jointJoy = true;
    }
    else
    {
        joyButtonCnt[0]++;
        if (joyButtonCnt[0] == JOYDELAY)
        {
            jointMotorData[0].jointCtrl = true;
            jointMotorData[0].jointCmd = 200 + 20;
            jointMotorData[0].jointJoy = true;
        }
    }
    if (!jointMotorData[0].protect)
        drRobotCommMot1.SendCommand("!PR 1 " +
jointMotorData[0].jointCmd.ToString() + "\r");

```

```
#endregion

}

private void pictureBox2_Click(object sender, EventArgs e)
{
    trackBar1.Value = 90;
    trackBar2.Value = 107;
    trackBar3.Value = 70;
    trackBar4.Value = 255;
    trackBar5.Value = 255;
    trackBar6.Value = 255;
}

private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
    mode = 1;
    checkBox1.Checked = false;
    checkBox2.Checked = false;
    SetSpeed(0);
    SetTurn(0);
}

private void radioButton2_CheckedChanged(object sender, EventArgs e)
{
    mode = 2;
    SetSpeed(0);
    SetTurn(0);
}

private void button17_Click(object sender, EventArgs e)
{
    prindere = true;
    fanion_gripper = true;
    verificare = true;
    poz_J1 = true;
    poz_J2 = true;
    test = true;
    test1 = true;
    autonom = true;
    checkBox2.Checked = false;
    btnEstop_ev = true;
    counter_cerc = 0;
}

private void pictureBox3_Click(object sender, EventArgs e)
{
    trackBar1.Value = 0;
    trackBar2.Value = 135;
    trackBar3.Value = 135;
    trackBar4.Value = 255;
    trackBar5.Value = 255;
    trackBar6.Value = 56;
}

}
```

```

    }

}

#endregion

```

CLASA LASERSCAN.CS

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Net.Sockets;
using System.Threading;
using System.Drawing.Drawing2D;
using System.Globalization;
using System.IO.Ports;

namespace DrRobot.JaguarControl
{
    public partial class JaguarCtrl : Form
    {
        private DrRobotConnectionMethod laserCommMethod =
DrRobotConnectionMethod.WiFiComm;
        SerialPort laserSerialPort = new System.IO.Ports.SerialPort();
        bool keepSearch = true;
        bool StayConnected = true;
        byte[] decodeBuff = new byte[MAX_BUFFER_LENGTH]; //maxlength
        int lastPos = 0;
        int processLen = 0;
        int endPos = 0;
        private static byte[] recBuffer;
        private Thread receiveThread = null;

        private TcpClient clientSocketLaser = null;
        private Thread threadClientLaser = null;
        private NetworkStream cmdStreamLaser = null;
        private BinaryReader readerLaser = null;
        private BinaryWriter writerLaser = null;
        private bool receivingLaser = false;
        /*
        private int monitorCntLaser = 0;
        private int errorChkCntLaser = 0;
        private bool firstSetupCommLaser = true;
        */
        //private bool connectedLaser = false;
        private delegate void updateLaserSensorDataInfo(byte[] data);

        private const string SCANCOMMAND = "GD0000108001";
        //below variables are based on above scan command,

```

```

        private const int DATAPACKAGE_LEN = 3369;
        public const int DISDATALEN = 1081; // based on scan command, it will be 680
distance data
        private const int DATALEN = 3243; //3 byte for each distance
        private const int DATALINE = 51; // 11 data lines(each line = 64 byte data +
sum + LF,
        private const int LASTLINE_DATA_LEN = 43;
        private long[] disData = new long[DISDATALEN];
        private byte[] dataArray = new Byte[DATALEN];
        private long[] disDataLast = new long[DISDATALEN];

        private const double START_ANGLE = -45.0;
        private const double STEP_ANGLE = 0.25; // cluster 1 is from scancommand 03

//this program will get data from -120 to +120. (G04572401) 725-44 = 681;
        private const double startAng = START_ANGLE / 180 * Math.PI; //27.0 / 180 *
Math.PI;

        private const double stepAng = STEP_ANGLE / 180 * Math.PI;//
        private const double ratio = 6;
        private const int START_X = 0;
        private const int START_Y = 180;
        private const int WIDTH_SUB = 40;

        private const int DRAW_START_X = 5;
        private const int DRAW_START_Y = 5;
        private int drawCnt = 3;
        private bool firstData = true;
        private const int LASER_TIMEOUT = 5;

        private long deltaTimeStamp = 0;
        private long preTimeStamp = 0;

        private void startCommLaser()
        {
            int remotePort = 10002;
            laserCommMethod = DrRobotConnectionMethod.WiFiComm;
            String IPAddr = jaguarSetting.LaserRangeIP;
            //firstSetupComm = true;
            try
            {
                //clientSocket = new TcpClient(IPAddr, remotePort);
                clientSocketLaser = new TcpClient();
                IAsyncResult results = clientSocketLaser.BeginConnect(IPAddr, remotePort,
null, null);
                bool success = results.AsyncWaitHandle.WaitOne(500, true);

                if (!success)
                {
                    clientSocketLaser.Close();
                    clientSocketLaser = null;
                    receivingLaser = false;
                    //pictureBoxLaser.Image = imageList1.Images[1];
                }
                else
                {
                    receivingLaser = true;
                    threadClientLaser = new Thread(new ThreadStart(HandleClientLaser));
                    threadClientLaser.CurrentCulture = new CultureInfo("en-US");
                    threadClientLaser.Start();
                    //pictureBoxLaser.Image = imageList1.Images[0];
                }
            }
        }

```

```

        catch
        {
            //pictureBoxLaser.Image = imageList1.Images[1];
        }
    }

private void stopCommunicationLaser()
{
    if (laserCommMethod == DrRobotConnectionMethod.WiFiComm)
    {
        if (clientSocketLaser != null)
            clientSocketLaser.Close();

        //pictureBoxLaser.Image = imageList1.Images[1];

        receivingLaser = false;
        if (writerLaser != null)
            writerLaser.Close();
        if (readerLaser != null)
            readerLaser.Close();
        if (cmdStreamLaser != null)
            cmdStreamLaser.Close();
        if (threadClientLaser != null)
            threadClientLaser.Abort();
    }
    else
    {
        //pictureBoxLaser.Image = imageList1.Images[1];
        if (laserSerialPort != null)
        {
            try
            {
                laserSerialPort.Close();

                StayConnected = false;
                keepSearch = false;
                if (receiveThread != null)
                {
                    if (receiveThread.IsAlive)
                        receiveThread.Abort(); //terminate the receive
                }
            }
            catch
            {
            }
        }
    }
}

thread
}

private void HandleClientLaser()
{
    //here process all receive data from client
    try
    {
        cmdStreamLaser = clientSocketLaser.GetStream();
        readerLaser = new BinaryReader(cmdStreamLaser);
        writerLaser = new BinaryWriter(cmdStreamLaser);

        bool keepSearch = true;
        byte[] decodeBuff = new byte[MAX_BUFFER_LENGTH]; //maxlength
    }
}

```

```

int lastPos = 0;
int processLen = 0;
int endPos = 0;

while (receivingLaser)
{
    try
    {
        if (!cmdStreamLaser.DataAvailable)           //there is no data
        {
            //need to sleep for other thread
            Thread.Sleep(30);
        }
        else
        {
            // Reads NetworkStream into a byte buffer.

            //errorChkCntLaser = 0;
            byte[] bytes = new byte[MAX_BUFFER_LENGTH];
            // Read can return anything from 0 to numBytesToRead.
            // This method blocks until at least one byte is read.
            processLen = readerLaser.Read(bytes, 0, bytes.Length);
            if (lastPos + processLen > decodeBuff.Length)
            {
                lastPos = 0;
            }

            Array.Copy(bytes, 0, decodeBuff, lastPos, processLen);
            processLen += lastPos;
            keepSearch = true;
            endPos = 0;
            lastPos = processLen;

            while (keepSearch)
            {
                endPos = 0;
                if (processLen >= 2)
                {
                    for (int i = 0; i < processLen - 1; i++)
                    {
                        if ((decodeBuff[i] == 10) && (decodeBuff[i + 1]
== 10))
                        {
                            endPos = i;
                            break;
                        }
                    }
                }

                if (endPos > 0)
                {
                    // already find the end, cut the string to decode

                    //string recCommand =
Encoding.Unicode.GetString(decodeBuff, 0, endPos);
                    byte[] recCommand = new byte[endPos + 2];
                    Array.Copy(decodeBuff, 0, recCommand, 0, endPos + 2);
                    //process receive data here
                    processDataLaser(recCommand);
                }
            }
        }
    }
}

```

```

        if ((endPos + 2) >= processLen)
        {
            //stop search
            keepSearch = false;
            lastPos = 0;
        }
        else
        {
            // trim this package from decode buff
            int len = processLen - endPos - 2;
            byte[] temp = new byte[len];

            Array.Copy(decodeBuff, endPos + 2, temp, 0, len);
            Array.Copy(temp, 0, decodeBuff, 0, len);
            lastPos = len;
            processLen = len;
        }
    }
    else
    {
        keepSearch = false;
        //lastPos = 0;
        processLen = 0;
    }
}
}
}
catch
{
}
finally
{
}

}
readerLaser.Close();
cmdStreamLaser.Close();
}
catch
{
}
receivingLaser = false;

clientSocketLaser.Close();

}

private void processDataLaser(byte[] data)
{
    Invoke(new updateLaserSensorDataInfo(updateSensor), data);
}

private void updateSensor(byte[] msg)
{
    string rcvData = Encoding.ASCII.GetString(msg);

    StringReader dataRead = new StringReader(rcvData);

```

```

string ackData = "";
try
{
    ackData = dataRead.ReadLine();
}
catch
{
}
string status = "";
try
{
    status = dataRead.ReadLine().Remove(2, 1);
}
catch
{
}
string timeStamp = "";
try
{
    timeStamp = dataRead.ReadLine().Remove(4, 1);
}
catch
{
}
long timeData = 0;
byte[] tempData = null;
if (timeStamp.Length == 4)
{
    tempData = System.Text.Encoding.ASCII.GetBytes(timeStamp);

    timeData = (((long)(tempData[0] - 0x30)) << 18) + (((long)(tempData[1] -
0x30)) << 12)
    + (((long)(tempData[2] - 0x30)) << 6) + ((long)(tempData[3] - 0x30));
}

laserSensorData.TimeStamp = timeData;

deltaTimeStamp = timeData - preTimeStamp;
preTimeStamp = timeData;
string temp = "";
int lineCnt = 0;
if ((status == "00") && (msg.Length == DATAPACKAGE_LEN))
{
    //suppose all the data is right
    while (lineCnt < DATALINE - 1)
    {
        temp = dataRead.ReadLine();           //remove sum and LF, won't check
sum here
        temp = temp.Remove(64, 1);
        tempData = System.Text.Encoding.ASCII.GetBytes(temp);
        Array.Copy(tempData, 0, dataArray, lineCnt * 64, tempData.Length);
        lineCnt++;
    }
    if (lineCnt == DATALINE - 1)
    {
        temp = dataRead.ReadLine();
        temp = temp.Remove(LASTLINE_DATA_LEN, 1);           //the last line is 40
byte data + sum + LF + LF
        tempData = System.Text.Encoding.ASCII.GetBytes(temp);
        Array.Copy(tempData, 0, dataArray, lineCnt * 64, tempData.Length);
    }

    //transfer data to distance data

```

```

        for (int i = 0; i < DISDATALEN; i++)
        {
            disData[i] = (((long)(dataArray[3 * i] - 0x30) << 12) +
            (((long)(dataArray[3 * i + 1] - 0x30)) << 6) + (long)(dataArray[3 * i + 2] - 0x30));
            if (disData[i] < Math.Abs(LASER_OFFSET_X * 1000)) disData[i] =
            (long)(MAX_LASER_DIS * 1000);
            laserSensorData.DisArrayData[i] = ((double)disData[i]) / 1000;
        }

        laserSensorData.LaserDisLM = MAX_LASER_DIS; //there is no tilting unit,

        UpdateSensorMap();

        //drawSensor(DISDATALEN);
        if (tmrDrawing.Enabled == false)
        {
            drawCnt = 0;
            tmrDrawing.Interval = 100;
            tmrDrawing.Enabled = true;
            //btnTurnOn.Enabled = false;
        }
        drawCnt = 0;
        sendCommandLaser(SCANCOMMAND); //read data again

    }
}

private void sendCommandLaser(string s)
{
    if (laserCommMethod == DrRobotConnectionMethod.WiFiComm)
    {
        try
        {
            s = s + new string((char)13, 1);

            ASCIIEncoding ascCode = new ASCIIEncoding();
            writerLaser.Flush();
            byte[] bytes = ascCode.GetBytes(s);
            writerLaser.Write(bytes, 0, bytes.Length);

        }
        catch
        {
        }
    }
    else
    {
        s = s + new string((char)13, 1);
        byte[] cmdData = ASCIIEncoding.UTF8.GetBytes(s);
        if (cmdData != null && laserSerialPort != null && laserSerialPort.IsOpen)
        {
            int packetLength = cmdData.Length; // (2 + (cmdData.Length));

            for (int i = 0; i < packetLength; i++)
            {
                try
                {
                    laserSerialPort.Write(cmdData, i, 1);
                }
                catch
                {
                }
            }
        }
    }
}

```

```
        }
    }
}

private void tmrDrawing_Tick(object sender, EventArgs e)
{
    drawCnt++;

    if (drawCnt > LASER_TIMEOUT)
    {
        drawCnt = 0;
        sendCommandLaser(SCANCOMMAND); //read data again
    }
}

/// <summary>
/// Open a serial port if in config we choose serial communication.
/// </summary>
/// <param name="comPort"></param>
/// <param name="baudRate"></param>
/// <returns>A Ccr Port for receiving serial port data</returns>
internal bool LaserSerialOpen(int comPort, int baudRate)
{
    laserCommMethod = DrRobotConnectionMethod.SerialComm;
    recBuffer = new byte[4095];

    if (baudRate < 1200)
        baudRate = 115200;

    laserSerialPort.Close();

    string portName = "COM" +
comPort.ToString(System.Globalization.NumberFormatInfo.InvariantInfo);
    if (laserSerialPort == null)
    {
        try
        {
            laserSerialPort = new SerialPort(portName, baudRate);
        }
        catch
        {
            MessageBox.Show("Could not open " + portName, "DrRobot Jaguar
Control", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
        }
    }
    else
    {
        laserSerialPort.PortName = portName;
        laserSerialPort.BaudRate = baudRate;
    }
    laserSerialPort.Encoding = Encoding.Default;
    laserSerialPort.Parity = Parity.None;
    laserSerialPort.DataBits = 8;
    laserSerialPort.StopBits = StopBits.One;
    laserSerialPort.Handshake = Handshake.None; // Handshake.RequestToSend;
//hardware flow control
//    serialPort.WriteTimeout = 2000;
    laserSerialPort.ReadTimeout = 0;
}
```

```

        laserSerialPort.BytesToReadThreshold = 2;           //the shortest
package is ping package
        laserSerialPort.ReadBufferSize = 4096;

        //serialPort.DataReceived += new
SerialDataReceivedEventHandler(port_DataReceived);
        //serialPort.ErrorReceived += new SerialErrorReceivedEventHandler(portError);
        try
        {
            laserSerialPort.Open();

            //start a thread to receive
            //      if (receiveThread == null)
            {
                receiveThread = new Thread((dataReceive));
                receiveThread.CurrentCulture = new CultureInfo("en-US");
                receiveThread.Start();
            }

        }
        catch
        {
            MessageBox.Show("Could not open " + portName, "DrRobot Jaguar Control",
MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
            return false;
        }
        return true;
    }

    /// <summary>
    /// to decode receive package here
    /// </summary>
    private void dataReceive()
    {
        //receive data here

        StayConnected = true;

        byte[] bytes = new byte[MAX_BUFFER_LENGTH];
        while (StayConnected)           //keep running
        {
            try
            {
                // *serial

                if (laserSerialPort.BytesToRead == 0)
                {
                    Thread.Sleep(10);
                }
                else
                {
                    processLen = laserSerialPort.Read(bytes, 0, bytes.Length);

                    Array.Copy(bytes, 0, decodeBuff, lastPos, processLen);
                    processLen += lastPos;
                    keepSearch = true;
                    endPos = 0;
                    lastPos = processLen;
                    searchData();
                }
            }
        }
    }
}

```

```
        catch (Exception ed)
        {
            //need to handle some error here
            // ex = ed.Message.ToString();
            // StayConnected = false;
        }
    }
}

void searchData()
{
    while (keepSearch)
    {
        endPos = 0;
        if (processLen >= 2)
        {
            for (int i = 0; i < processLen - 1; i++)
            {
                if ((decodeBuff[i] == 10) && (decodeBuff[i + 1] == 10))
                {
                    endPos = i;
                    break;
                }
            }
        }

        if (endPos > 0)
        {
            // already find the end, cut the string to decode command

            //string recCommand = Encoding.Unicode.GetString(decodeBuff, 0,
endPos);

            byte[] recCommand = new byte[endPos + 2];
            Array.Copy(decodeBuff, 0, recCommand, 0, endPos + 2);
            //process receive data here
            processDataLaser(recCommand);

            if ((endPos + 2) >= processLen)
            {
                //stop search
                keepSearch = false;
                lastPos = 0;
            }
            else
            {
                // trim this package from decode buff
                int len = processLen - endPos - 2;
                byte[] temp = new byte[len];

                Array.Copy(decodeBuff, endPos + 2, temp, 0, len);
                Array.Copy(temp, 0, decodeBuff, 0, len);
                lastPos = len;
                processLen = len;
            }
        }
        else
        {
            keepSearch = false;
            //lastPos = 0;
            processLen = 0;
        }
    }
}
```

```

    }
    }
}

```

CLASA DRROBOTMAINCOMM

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.IO.Ports;
using System.Net;
using System.Net.Sockets;
using System.Windows.Forms;
using System.Globalization;
using System.IO;
namespace DrRobot.JaguarControl
{
    class DrRobotMainComm
    {
        private DrRobotConnectionMethod commMethod = DrRobotConnectionMethod.WiFiComm;
        //1--for wifi 2 for serial

        private StreamReader readerReply = null;
        private StreamWriter writer = null;
        private NetworkStream replyStream = null;

        private static byte[] recBuffer;

        private Thread receiveThread = null;

        SerialPort serialPort = new System.IO.Ports.SerialPort();

        private static Socket clientSocket;
        private static int localPort = 0;

        private static string lastRecvErrorMsg = String.Empty;
        private static int lastRecvError = 0;
        private static string lastSendErrorMsg = String.Empty;
        private static int socketErrorCount = 0;

        private string processStr = "";
        private string recStr = "";

        private JaguarCtrl _form;
        private string comID = "MOT1";

        public DrRobotMainComm(JaguarCtrl form, string id)
        {
            _form = form;
            comID = id;
        }
    }
}

```

```
//here is the WiFi connecting start
internal bool StartClient(string addr, int portNum)
{
    // Connect to remote server
    commMethod = DrRobotConnectionMethod.WiFiComm;
    try
    {
        IPAddress ipAddress;
        int remotePort = portNum;
        try
        {
            ipAddress = IPAddress.Parse(addr);
        }
        catch (Exception e)
        {
            lastSendErrorMsg = e.Message;
            return false;
        }

        IPEndPoint remoteEP = new IPEndPoint(ipAddress, remotePort);

        // Create a TCP socket

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

        receiveThread = new Thread((dataReceive));
        receiveThread.CurrentCulture = new CultureInfo("en-US");

        clientSocket.Connect(remoteEP);

        localPort = ((IPEndPoint)clientSocket.LocalEndPoint).Port;

        //for communication need to send a package first
        //Ping();

        receiveThread.Start();

        return true;
    }
    catch (SocketException e)
    {
        {
            socketErrorCount++;
            lastRecvError = e.ErrorCode;
            lastRecvErrorMsg = e.ToString();
            return false;
        }
    }
}

/// <summary>
/// Open a serial port if in config we choose serial communication.
/// </summary>
/// <param name="comPort"></param>
/// <param name="baudRate"></param>
/// <returns>A Ccr Port for receiving serial port data</returns>
internal bool SerialOpen(int comPort, int baudRate)
{
    commMethod = DrRobotConnectionMethod.SerialComm;
```

```

        if (recBuffer == null)
            recBuffer = new byte[4095];

        if (baudRate < 1200)
            baudRate = 115200;

        serialPort.Close();

        string portName = "COM" +
comPort.ToString(System.Globalization.NumberFormatInfo.InvariantInfo);
        if (serialPort == null)
        {
            serialPort = new SerialPort(portName, baudRate);
        }
        else
        {
            serialPort.PortName = portName;
            serialPort.BaudRate = baudRate;
        }
        serialPort.Encoding = Encoding.Default;
        serialPort.Parity = Parity.None;
        serialPort.DataBits = 8;
        serialPort.StopBits = StopBits.One;
        serialPort.Handshake = Handshake.None;           // Handshake.RequestToSend;
//hardware flow control
        // serialPort.WriteTimeout = 2000;
        serialPort.NewLine = "\r";
        serialPort.ReadTimeout = 0;
        serialPort.BytesToReadThreshold = 1;              //the shortest package is
ping package
        serialPort.ReadBufferSize = 4096;

        //serialPort.DataReceived += new
SerialDataReceivedEventHandler(port_DataReceived);
        //serialPort.ErrorReceived += new SerialErrorReceivedEventHandler(portError);
        try
        {
            serialPort.Open();

            //start a thread to receive
            // if (receiveThread == null)
            {
                receiveThread = new Thread((dataReceive));
                receiveThread.CurrentCulture = new CultureInfo("en-US");
                receiveThread.Start();
            }

        }
        catch
        {
            return false;
        }
        return true;
    }

    /// <summary>
    /// to decode receive package here
    /// </summary>
    private void dataReceive()
    {
        if (commMethod == DrRobotConnectionMethod.WiFiComm)
        {

```

```
        if (readerReply != null)
            readerReply.Close();
        if (replyStream != null)
            readerReply.Close();
        replyStream = new NetworkStream(clientSocket);

        readerReply = new StreamReader(replyStream);
    }

    //receive data here
    bool StayConnected = true;

    byte[] recs = new byte[4095];
    int scout = 0;

    while (StayConnected)           //keep running
    {
        try
        {
            // *serial
            if (commMethod == DrRobotConnectionMethod.SerialComm)
            {
                //recStr = serialPort.ReadExisting();
                if (serialPort.BytesToRead > 0)
                {
                    recStr = serialPort.ReadLine();
                    processData();
                }
            }
            else
            {
                Thread.Sleep(20);
            }
        }
        else if (commMethod == DrRobotConnectionMethod.WiFiComm)
        {
            // wifi
            if (!replyStream.DataAvailable)
            {
                Thread.Sleep(5);
            }
            else
            {
                recStr = readerReply.ReadLine();
                processData();
            }
        }
    }
    catch (Exception ed)
    {
        //need to handle some error here
    }
}

void processData()
{
    /*
    if (recStr.Length == 1)
    {
        //here is the ack message, "+", "-"
        if (recStr == "+")
```

```

        _form.UpdateAckMsg(comID + "VALID");
    else if (recStr == "-")
        _form.UpdateAckMsg(comID + "INVALID");
    }
    */
    if (recStr.Length > 0)
    {
        _form.UpdateMainRcvDataMsg(comID + recStr);
    }
}

/// <summary>
/// DO NOT USE THIS COMMAND DIRECTLY.
///
/// </summary>
/// <param name="cmd"></param>
/// <returns></returns>
internal bool SendCommand(string Cmd)
{
    /// <summary>
    Cmd += "\r\n";
    byte[] cmdData = ASCIIEncoding.UTF8.GetBytes(Cmd);

    if (commMethod == DrRobotConnectionMethod.SerialComm)
    {
        if (cmdData != null && serialPort != null && serialPort.IsOpen)
        {
            int packetLength = cmdData.Length; // (2 + (cmdData.Length));
            try
            {
                serialPort.Write(cmdData, 0, packetLength);
            }
            catch
            {
                return false;
            }
            return true;
        }
        else
            return false;
    }

    else if (commMethod == DrRobotConnectionMethod.WiFiComm)
    {
        if ((cmdData != null) && (clientSocket != null)) //&&
(clientSocket.Connected)
        {
            try
            {
                if (clientSocket.Send(cmdData) == cmdData.Length)
                {
                }
                else
                {
                    return false;
                }
            }
            catch
            {
                return false;
            }
        }
    }
}

```

```
        }

        return true;
    }
    else
        return false;
}
else
    return false;
}

/// <summary>
/// Close the connection to a serial port.
/// </summary>
public void Close()
{
    try
    {
        if (readerReply != null)
            readerReply.Close();
        if (writer != null)
            writer.Close();
    }
    catch
    {
    }

    if (serialPort != null)
    {
        if (serialPort.IsOpen)
        {
            serialPort.Close();
        }
    }
    if ((clientSocket != null))
    {
        //if (clientSocket.Connected)
        clientSocket.Close();
    }
    if (receiveThread != null)
    {
        if (receiveThread.IsAlive)
            receiveThread.Abort();    //terminate the receive thread
    }
}

/// <summary>
/// True when the serial port connection is open.
/// </summary>
public bool IsOpen
{
    get { return serialPort != null && serialPort.IsOpen; }
}

}

}
```