

MISIUNE DE EXPOLARE A UNEI ZONE CU UN ROBOT AUTONOM

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:

Gheorghe Alin Valentin

București

2016

Misiune de explorare a unei zone cu un robot autonom.

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
a studentului Gheorghe Alin Valentin 442A**

1. Titlul temei: **Misiune de explorare a unei zone folosind un robot autonom.**
2. Contribuția practică, originală a studentului va consta în:
Proiectul presupune dezvoltarea autonomiei pe un robot deja existent prin echiparea acestuia cu senzori de infraroșu care îi vor permite să detecteze și să evite obstacolele întâlnite și cu un senzor Kinect care poate fi folosit la explorarea zonei. Robotul este echipat cu un sistem de dezvoltare (Arbotix-M) pe care va fi implementat codul pentru deplasarea robotului și pentru luarea deciziilor pe baza informațiilor primite de la senzori. Performanța robotului va fi analizată în condiții reale, într-o zonă special amenajată.
3. Proiectul se bazează pe cunoștințe dobândite în principal la următoarele 3-4 discipline: Microcontrolere, Programare obiect orientată, Prelucrarea digitală a semnalelor.
4. Proprietatea intelectuală asupra proiectului aparține: Laboratorului de cercetare Speech and Dialogue (Speed), studentului Gheorghe Alin Valentin
5. Locul de desfășurare a activității: UPB - ETTI, Laboratorului de cercetare Speech and Dialogue (Speed)
6. Realizarea practică rămâne în proprietatea: Laboratorului de cercetare Speech and Dialogue (Speed), studentului Gheorghe Alin Valentin
7. Data eliberării temei: decembrie 2015

CONDUCĂTOR LUCRARE:

Ș.L. Dr. Ing. Horia Cucu
Prof. Dr. Ing. Corneliu Burileanu



STUDENT:

Gheorghe Alin Valentin



Misiune de explorare a unei zone cu un robot autonom.

DECLARAȚIE DE ONESTITATE ACADEMICĂ

Prin prezenta declar că lucrarea cu titlul ”Misiune de explorarea a unei zone cu un robot autonom” , 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, Data

Gheorghe Alin Valentin

Misiune de explorare a unei zone cu un robot autonom.

CUPRINS

Cuprins	7
Listă de Figuri	9
Listă de Tabele	11
Capitolul 1 Introducere	13
1.1 Motivația Lucrării	13
1.2 Introducere	14
1.3 Istoricul roboților hexapozi	15
1.4 Obiectivele Lucrării.....	15
1.4 Sumarul conținutului lucrării	15
Capitolul 2 Tehnologii Hardware Folosite	17
2.1 Platforma Robotică PhantomX Mark II	17
2.1.1 Controlerul robotului, Arbotix-M	18
2.1.2 Motorul Dynamixel AX-12A.....	20
2.1.3 Deplasarea robotului.	25
2.1.4 Controlul robotului.....	27
2.1.5 Acumulator LiPo.....	31
2.2 Senzorul Kinect	32
2.2.1 Camera color	32
2.2.2 Emițătorul infraroșu și senzorul de adâncime infraroșu	33
2.3 Microcalculatorul Raspberry Pi 3	35

2.4	Senzorul de distanță Sharp.....	36
Capitolul 3 Tehnologii Software Folosite.....		39
3.1	Arduino IDE.....	39
3.2	Sistemul de operare Raspian	40
3.3	Python	41
3.4	Numpy și Matplotlib	42
3.5	OpenCV	43
3.6	Protocolul I2C	44
Capitolul 4 Prelucrarea și Analiza Imaginilor		47
4.1	Introducere	47
4.2	Imagini Digitale	48
4.3	Aspecte ale procesării de imagini	51
Capitolul 5 Realizarea deplasării autonome		55
5.1	Descrierea generală a sistemului.....	56
5.1	Analiza codului sursă al robotului și conectarea senzorului Sharp.....	57
5.2	Realizarea conexiunii dintre Raspberry Pi și senzorul Kinect.....	61
5.3	Analiza imaginii obținută de senzorul Kinect.....	62
5.3.1	Caracteristicile imaginii.....	62
5.3.2	Distanța față de obiecte.....	63
5.4	Trimiterea datelor de la Raspberry Pi la Arbotix-M.....	63
5.5	Modul de funcționare al sistemului.....	64
5.6	Limitări.....	66
Capitolul 6		68
6.1	Concluzii generale.....	68
6.2	Contribuții personale.....	68
6.3	Etape viitoare	69
Referințe		70
Anexe		72

LISTĂ DE FIGURI

Figura 1	Figura 2.....	14
Figura 3	Figura 4.....	14
Figura 5: Platforma robotică PhantomX Mark II.....		17
Figura 6: Placa de dezvoltare Arbotix-M [2].....		19
Figura 7: Dynamixel AX-12A		20
Figura 8: Aria de operare a motorului.....		20
Figura 9: Partea frontală	Figura 10: Partea din spate	22
Figura 11: Atribuirea pinilor.....		22
Figura 12: Așezarea motoarelor platformei robotice		23
Figura 13: Pachetul de instrucțiuni		23
Figura 14: Pachetul de stare.....		24
Figura 15: Modul de deplasare 3+3		25
Figura 16: Modul de deplasare 4+2.		26
Figura 17: Componenta piciorului robotului.		26
Figura 18: ArbotiX Commander		28
Figura 19: Comunicatia între hexapod si maneta.		29
Figura 20: Programarea robotului prin intermediul calculatorului.		30
Figura 21: Conectarea individuală a motoarelor în vederea stabilirii ID-urilor.		30
Figura 22: Componentele senzorului Kinect.		32

Figura 23: Procesul de obținere a imaginii de adâncime.	33
Figura 24: Raspberry Pi 3.	35
Figura 25: Stabilirea distanței până la obiect.	36
Figura 26: Așezarea senzorului pentru eliminarea erorii.	37
Figura 27: Comunicația prin porturi seriale.	44
Figura 28: Protocolul SPI.	44
Figura 29: Protocolul I2C.	45
Figura 30: Tipurile de operații pe imagini.	49
Figura 31: Exemplu de vecinătate.	50
Figura 32: Figura 33: Figura 34:	50
Figura 35: Imagine binară.	52
Figura 36: Imagine cu nivele de gri.	52
Figura 37: Imagine RGB	53
Figura 38: Imagine color indexată.	53
Figura 39 : Ansamblul sistemului.	56
Figura 40 : Schema electrică ArbotiX-M – Sharp.	59
Figura 41 : Diagramă UML a modului de funcționare, cu senzorul Sharp.	60
Figura 42 : Rezoluție 640x480 Figura 43 : Rezoluție 100x100.	62
Figura 44 : Reducerea zgomotului din imagine.	62
Figura 45 : Funcția liniară de conversie, nivele de gri – milimetrii.	63
Figura 46 : Modul de funcționare al robotului.	65

LISTĂ DE TABELE

Tabel 1: Parametrii motorului Dynamixel AX-12A	21
Tabel 2: Adrese obișnuite ale motorului.	24
Tabel 3: Calculul mișcărilor.....	27
Tabel 4 : Variabile importante	28
Tabel 5 : Parametrii de configurare Xbee	31

Misiune de explorare a unei zone cu un robot autonom.

CAPITOLUL 1

INTRODUCERE

1.1 MOTIVAȚIA LUCRĂRII

Nevoia tot mai mare de utilizare a roboților nu mai poate fi ascună. Dezastrele naturale care se petrec în întreaga lume constituie unul dintre motivele pentru care roboții ar trebui dezvoltați cu scopul ajutorării oamenilor. Roboții folosiți în astfel de situații pot fi controlați fie de o persoană calificată prin intermediul unui dispozitiv de control și a unor camere prin care acesta poate vizualiza zona în care se află robotul și poate lua cele mai bune decizii, fie robotul se poate deplasa autonom pe baza informațiilor primite de la senzorii montați pe robot. Familia de roboți cu șase picioare, din care face parte și robotul folosit în această lucrare (PhantomX Mark II), este printre cele mai folosite în acest tip de situații datorită stabilității și flexibilității în timpul deplasării pe orice tip de suprafață. Deoarece robotul dispune de mai multe picioare, poate fi ușor programat să se deplaseze datorită modurilor de deplasare care pot fi utilizate, cum ar fi modul tripod, modul quadroped. Există de asemenea numeroase variante de roboți hexapod care dispun de diverse funcții și avantaje. În această lucrare, se realizează deplasarea autonomă în interiorul unei camere a unui robot hexapod. Scopul realizării unui robot hexapod cu deplasare autonomă este acela de a fi utilizat într-o zonă necunoscută fiind capabil să detecteze și să evite obiectele întâlnite.

1.2 INTRODUCERE

În societatea tehnologică de astăzi, oamenii au crescut fiind obișnuiți să folosească în fiecare zi o serie de dispozitive care să le facă viața mai ușoară, cum ar fi calculatoarele, vehiculele personale, telefoanele mobile care permit comunicarea prin intermediul internetului. Astfel, utilizarea roboților a devenit din ce în ce mai comună printre oameni. Roboții pot fi folosiți pentru realizarea de sarcini repetitive, pentru creșterea productivității într-o uzină, pentru a transporta încărcături mari și alte sarcini pe care le au de îndeplinit oamenii.

O gamă largă de vehicule terestre au fost realizate pentru a satisface diverse cerințe ale oamenilor, cum ar fi traversarea diferitelor tipuri de teren. În ceea ce privește capacitățile de traversare a diferitelor tipuri de teren, una dintre cele mai interesante platforme o constituie roboții tereștrii. Roboții pot fi găsiți pretutindeni în zilele noastre. Una dintre cele mai importante părțile ale unui robot este șasiul sau corpul robotului. Există trei tipuri de șasiuri de bază în domeniul roboticii și acestea sunt: șasiu pe bază de roți, șasiu pe bază de șenile și șasiu pe bază de picioare. Șasiul cu roți este cel mai rapid însă nu este potrivit pentru teren accidentat, în timp ce cel cu șenile este mai lent dar mult mai potrivit pentru teren accidentat. Șasiul care folosește picioare pentru a se deplasa este destul de lent și mult mai dificil de controlat însă cel mai potrivit în situații de teren accidentat și de ocolire a obstacolelor. În decursul ultimilor ani au avut loc numeroase cercetări în domeniul roboticii datorită potențialului uriaș.

Există mai multe tipuri de platforme robotice cu șase picioare diferite una de cealaltă. Diferența dintre ele este dată de construcția fizică a robotului. Există platforme cu formă dreptunghiulară, ceea ce înseamnă că poate recunoaște direcția deplasării, un exemplu fiind platforma din figura 1, altele sunt circulare, cum este platforma din figura 2. Diferențele pot fi făcute de asemenea de deplasarea picioarelor, existând roboți cu picioare asemănătoare cu cele ale insectelor, cum este cazul robotului din figura 3 sau roboți care folosesc rotirea picioarelor în jurul propriei axe, cum este robotul din figura 4.

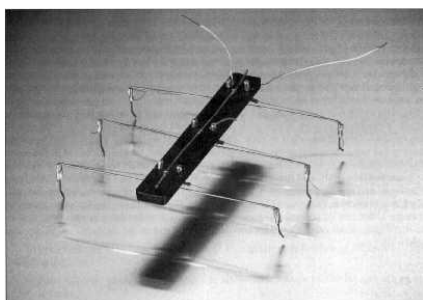


Figura 1

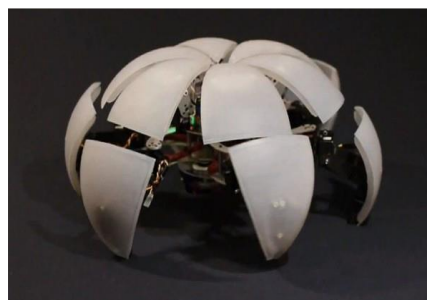


Figura 2



Figura 3



Figura 4

1.3 ISTORICUL ROBOȚILOR HEXAPOZI

Primii hexapozi au putut fi indentificați ca fiind roboți pe baza deplasării rigide de unde reiese că o adaptare la tipul de teren întâlnit nu era posibilă la acea vreme. Cercetările realizate în anii 50 au urmărit ca deplasarea roboților să fie controlată de un operator uman prin intermediul unei telecomenzi. Unul dintre cei mai de succes roboți hexapod a fost construit la Universitatea din Roma în anul 1972, ca fiind o mașinărie a cărei deplasare era controlată de un calculator. La mijlocul anilor 70, la Academia de Științe din Moscova, a fost realizat un robot cu șase picioare al cărui control de mișcare avea la bază un model matematic. În anul 1976, a fost realizat la Universitatea de Stat din Moscova un robot terestru cu șase picioare care era capabil să ocolească obstacolele întâlnite folosind un senzor de apropiere. Încetul cu încetul acest domeniu al roboticii a luat amploare ajungând astăzi să uimească omenirea cu descoperirile sale.

1.4 OBIECTIVELE LUCRĂRII

Obiectivul lucrării de față este realizarea deplasării autonome pentru o platformă robotică cu șase picioare, PhantomX Mark II, utilizând un microcalculator Raspberry Pi 3 ca și unitate centrală a ansamblului, la care este conectat un senzor Kinect. Prelucrarea datelor primite de la senzorul de adâncime de pe Kinect este realizată pe Raspberry Pi, după care rezultatele sunt trimise la placa de dezvoltare ArbotiX-M unde este stabilit algoritmul de deplasare al robotului.

1.4 SUMARUL CONȚINUTULUI LUCRĂRII

Structura acestei lucrări este organizată după cum urmează:

Capitolul 2 prezintă o descriere generală a tehnologiilor hardware folosite pentru realizarea proiectului. Capitolul în care sunt tratate doar probleme teoretice prin care cititorul capătă o privire de ansamblu asupra robotului și asupra posibilităților de care dispune utilizând acest tip de robot terestru.

Capitolul 3 descrie tehnologiile software necesare pentru realizarea proiectului, precum și a librăriilor folosite pentru stocarea și prelucrarea imaginii obținute de la senzorul Kinect.

Capitolul 4 prezintă o introducere în domeniul de prelucrare și analiză a imaginilor, pe care se bazează modul autonom de deplasare al robotului.

Capitolul 5 prezintă etapele prin care am trecut pentru realizarea modului de deplasare autonomă pentru robotul PhantomX Mark II.

Misiune de explorare a unei zone cu un robot autonom.

CAPITOLUL 2

TEHNOLOGII HARDWARE FOLOSITE

2.1 PLATFORMA ROBOTICĂ PHANTOMX MARK II



Figura 5: Platforma robotică PhantomX Mark II

Platforma robotică PhantomX Mark II [1] este un robot terestru cu șase picioare, care prezintă dimensiunile 240x120x39 mm, prezentat în Figura 5. Robotul este un kit de dezvoltare asamblat în cadrul laboratorului Speed (Speech and Dialogue Research Laboratory) și are în componența sa 18 motoare Dynamixel AX-12A, controlerul Arbotix-M, un software open-source și un dispozitiv ArbotiX Commander prin intermediul căruia robotul poate fi controlat de către utilizator.

2.1.1 Controlerul robotului, Arbotix-M

Controlerul robotului, Arbotix-M, este o placă de dezvoltare care are la bază arhitectura ARV și totodată este o placă compatibilă Arduino, realizată pentru controlul roboților de dimensiuni mici și mijlocii, în special a roboților terestrii care folosesc motoare Dynamixel AX-12.

Ca specificații hardware, Arbotix-M folosește un microcontroler din familia AVR, ATMEGA644p de 16 MHz, este dotat cu două porturi seriale dintre care un port serial este folosit pentru motoarele Dynamixel în timp ce al doilea port serial este utilizat de modulul radio XBEE. Are în componență 32 de pini digitali de intrare/ieșire, dintre care 8 pini pot fi utilizați atât ca pini digitali cât și ca pini analogici.

În continuare voi descrie o serie de caracteristici Hardware ale plăcii Arbotix-M

- Jumper-ul de selecție a alimentării poate fi văzut în Figura 6 (1). Acesta permite utilizatorului să aleagă tipul de alimentare pentru Arbotix-M. Conectând jumperul între pinul „VIN” și pinul aflat în mijloc, Arbotix-M va fi alimentat de la regulatorul de 5v, care în cazul robotului PhantomX Mark II este legat la o baterie LiPo de 11.1v. Dacă jumperul este fixat între pinul „USB” și pinul mijlociu, Arbotix-M va fi alimentat de la o sursă de 5v prin intermediul cablului FTDI, care în cele mai multe cazuri este conectat la USB-ul unui calculator.
- Portul serial FTDI poate fi văzut în Figura 6 (2). Portul FTDI poate fi utilizat ca port serial prin care se pot trimite date de la un alt dispozitiv la robot sau invers, dar este utilizat și ca port programabil prin intermediul căruia se pot încărca programele sursă dorite pe robot.
- Pinii digitali de intrare/ieșire pot fi văzuți în Figura 6 (3). Fiecare dintre acești pini are o configurație Semnal-Tensiune-Masă care oferă posibilitatea conectării pe Arbotix-M și implicit pe robot a unui număr mare de senzori utili în dezvoltarea aplicațiilor.
- Pinii digitali pentru servomotoare pot fi văzuți în Figura 6 (4). Acești pini sunt similari cu pini digitali obișnuiți având deosebirea că pinul de semnal este legat direct la microcontrolerul ATMEGA644p.
- Pinii analogici prezintă aceeași configurație Semnal-Tensiune-Masă și pot fi accesați în Arduino IDE utilizând pinii digitali 24-31. Se pot vedea în Figura 6 (5).
- Porturile Dynamixel permit conectarea motoarelor Dynamixel direct în Arbotix-M. Motoarele pot fi conectate în serie și astfel mai multe motoare pot folosi același port sau se poate folosi un „Power Hub” și astfel se mărește numărul de porturi. Aceste porturi sunt compatibile doar cu 3-Pin TTL Dynamixel și pot fi văzute în Figura 6 (6).

- Socket-ul XBEE, Figura 6 (7), permite realizarea unei comunicații wireless rapide pentru Arbotix-M prin simpla conectare a unui modul XBEE. Acesta va crea o conexiune wireless serială permițând conectarea plăcii Arbotix-M la un alt microcontroller printr-un al doilea modul XBEE.
- Pinii I2C, Figura 6 (8). Deși Arbotix-M nu are un port I2C dedicat, protocolul I2C este suportat prin intermediul pinilor 16 și 17, unde pinul 16 reprezintă pinul de sincronizare (SLC) iar pinul 17 reprezintă pinul de date (SDA)[2].

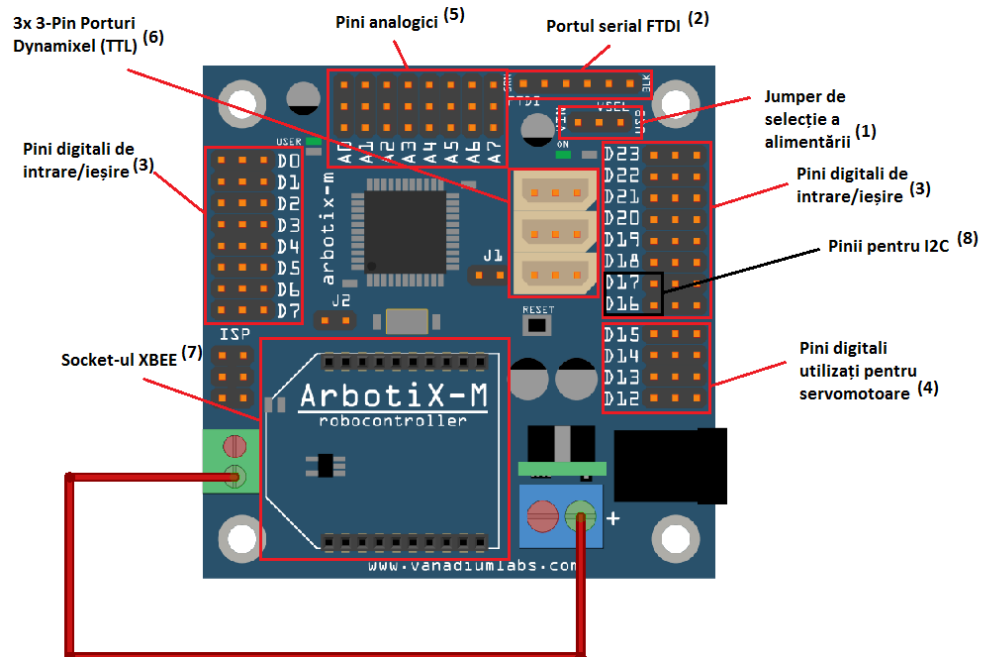


Figura 6: Placa de dezvoltare Arbotix-M [2].

2.1.2 Motorul Dynamixel AX-12A

Servo motorul Dynamixel este un motor inteligent, realizat pentru controlul mișcării roboților. Acesta nu funcționează doar ca un simplu motor, este capabil să funcționeze ca un senzor inteligent. Motorul este controlat de un microcontroler (MCU) și comunicația este realizată prin intermediul a trei fire, care sunt de asemenea folosite și la alimentarea motorului.



Figura 7: Dynamixel AX-12A

2.1.2.1 Hardware-ul motorului

Zona pe care motorul se poate deplasa este cuprinsă între 0 și 300 grade. Limitele unghiulare asigură restricționarea mișcării motorului. Dacă ambele valori ale CW și CCW sunt setate 0, motorul intră în modul „Wheel Mode” în care este capabil să se rotească continuu, spre deosebire de situația în care atât CW cât și CCW sunt diferite de 0, moment în care, motorul intră în modul „Joint Mode”, astfel robotul se deplasează în funcție de anumite unghiuri.

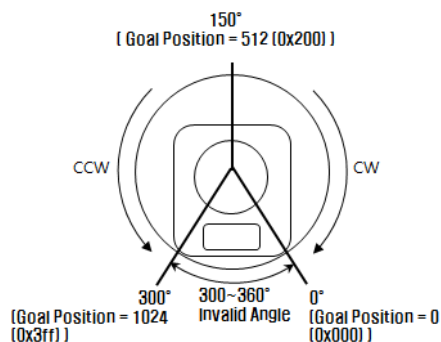


Figura 8: Aria de operare a motorului

Corpul motorului este realizat dintr-un grup de materiale plastice care poartă numele de plastic de inginerie, fiind un material foarte rezistent. În partea din față, după cum se poate vedea și în Figura 8, există două porturi pentru realizarea conexiunii, astfel motorul poate fi conectat foarte ușor prin intermediul unei magistrale seriale. Parametrii fizici ai motorului Dynamixel AX-12A sunt prezentați în Tabelul 1.

Tabel 1: Parametrii motorului Dynamixel AX-12A

Model	AX-12A
Stall Torque	1.5 N/m (12 V, 1.5 A)
Speed (RPM)	59 rpm (12 V)
Voltage	9 ~ 12 V (Recommended 11.1 V)
Stall Current Draw	1.5A
Dimensions	32mm*50mm*40mm
Weight	54,6g
Resolution	0.29 grade
Operating Angle	300
Gear Reduction	254:1
CPU	ATMega8
Position Sensor	Potentiometer
Communication Protocol	TTL
Communication Speed	7343 bps ~ 1 MBps

În interiorul motorului se află un servomotor care realizează mișcarea ansamblului de roți dințate și microcontrolerul (MCU) ATMega 8. Întregul interior al motorului este prezentat în figurile 9 și 10.

Deși este un motor mic, ca dimensiuni, este capabil să producă o tracțiune bună și poate dezvolta viteze mari, cu costul unui consum mare de energie, vezi specificațiile din Tabelul 1. Consumul de curent al unui singur motor în modul de așteptare nu trebuie să fie mai mare de 50 mA (după specificațiile producătorului), dar în timpul deplasării robotului valorile consumului de curent poate varia între 50 și 1500 mA. Această valoare depinde în special de diferența dintre poziția curentă și poziția dorită a motorului, de aceea timpul în care robotul realizează operațiile dorite este mai scurt în cazul în care este alimentat de la o baterie. Una dintre probleme, este aceea că robotul consumă energie nu doar atunci când se deplasează, ci și atunci când stă pe loc, deoarece motoarele trebuie să rămână în poziția setată ca poziție dorită. Cantitatea necesară de energie poate fi chiar mai mare, dacă toate motoarele funcționează în același timp.



Figura 9: Partea frontală

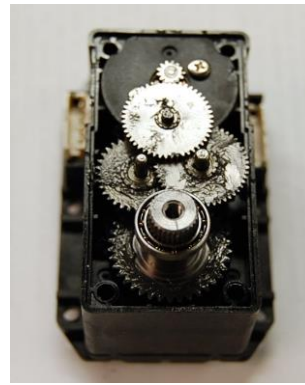


Figura 10: Partea din spate

Interiorul motorului Dynamixel AX-12A

În ceea ce privește alimentarea și comunicația, fiecare motor este conectat prin intermediul a trei cabluri. Un cablu este folosit pentru alimentarea motorului, al doilea este folosit pentru realizarea comunicației bazată pe logica tranzistor-tranzistor (TTL), iar al treilea este pinul de ground, după cum se poate vedea și în Figura 11.

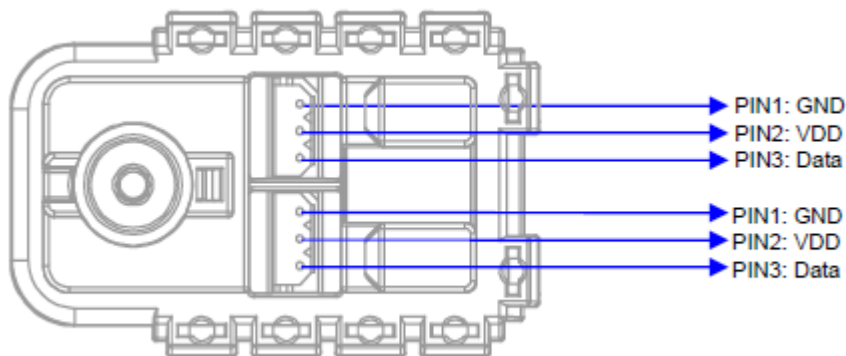


Figura 11: Atribuirea pinilor.

Robotul folosit pentru realizarea acestui proiect, PhantomX Mark II, este echipat cu 18 motoare Dynamixel AX-12A, iar așezarea lor se poate vedea în Figura 12[1].

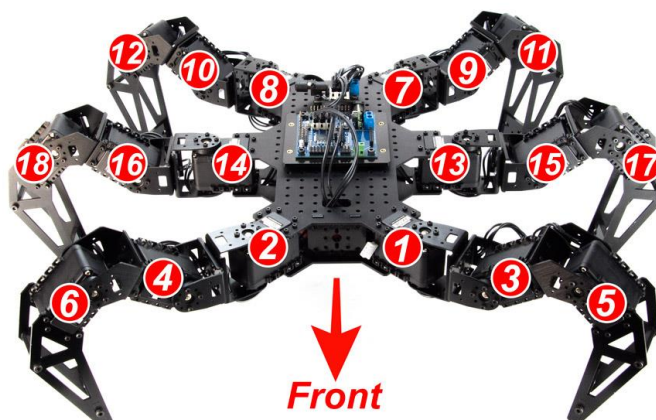


Figura 12: Așezarea motoarelor platformei robotice

2.1.2.2 Comunicația

Dynamixel AX-12A este un motor inteligent controlat prin TTL, iar unitatea de control (Arbotix-M) trebuie să suporte o comunicație serială universală asincronă de tipul receptor/transmițător (UART), în care dispozitivele nu pot trimite și primi date în același timp (half-duplex). Comunicația cu placa de dezvoltare se realizează serial prin 3 conectori și fiecare motor este identificat în cadrul schemei de comunicație printr-un identificator unic. Identificatorii motoarelor pot fi modificați, însă comenzile care se ocupă de controlul mișcării robotului sunt atribuite motoarelor în conformitate cu schema prezentată în Figura 12.

Comunicația dintre unitatea de control a robotului și motoare este realizată prin intermediul a două tipuri de pachete: 1. Pachete de instrucțiuni; 2. Pachete de stare. Fiecare pachet este divizat în două părți: partea de antet și partea de date.



Figura 13: Pachetul de instrucțiuni

Pachetul de instrucțiuni prezentat în figura 13 în care fiecare casetă reprezintă un byte. Antetul este format din primii doi bytes, folosiți pentru marcarea începutului pachetului. Partea de date este formată din ID-ul motorului, lungimea întregului pachet trimis, instrucțiunea motorului respectiv, parametrii necesari pentru instrucțiunea respectivă și parametrul checksum.

Instrucțiunea stabilește dacă parametrii sunt citiți, scriși sau scriși sincron la mai multe motoare. Lista celor mai folosite adrese este prezentată în Tabelul 2.

Tabel 2: Adrese obișnuite ale motorului.

Adresa	Componentă	Lungime (bytes)	Min	Max
3	ID	1	0	253
4	Rata de transfer	1	0	254
5	Timpul de întârziere	1	0	254
6	Unghiul Limită CW	2	0	1023
8	Unghiul Limită CCW	2	0	1023
14	Tranțiune maximă	2	0	1023
17	Led de urgență	1	0	127
18	Închidere de urgență	1	0	127
30	Poziția ideală	2	0	1023
32	Viteza de deplasare	2	0	1023

Pachetul de stare este prezentat în figura 14, în care fiecare casetă reprezintă un byte și a cărei structură este similară cu cea a pachetului de instrucțiuni. Antetul este format din primii doi bytes, care sunt folosiți la marcarea începutului pachetului, în timp ce zona de date este formată din ID-ul motorului adresat anterior, lungimea întregului pachet, posibilele erori la nivelul motorului, parametrii necesari pentru diferite instrucțiuni și byte-ul de checksum[3].

`[0xFF][0xFF][ID][LENGTH][ERROR][PARAMETER1][PARAMETER2]...[PARAMETER N][CHECK SUM]`

Figura 14: Pachetul de stare.

Motoarele Dynamixel nu sunt doar simple motoare, pot fi utilizate drept senzorii inteligenți. Când oricare din parametrii depășește limita setată pentru motorul respectiv, motorul este deconectat de către sistem. Dacă motivul este unul temporar, cum ar fi depășirea unghiului limită, motorul se poate mișca din nou în momentul în care următoarea poziție ideală se află în limitele unghiulare permise. O altă serie de probleme care pot apărea sunt cele legate de depășirea greutății maxime, care pot fi rezolvate prin resetarea motorului[3].

2.1.3 Deplasarea robotului.

Putem defini deplasarea robotului pe baza sistemului de coordonate cartezian (x , y , z) ce are originea în centrul robotului. Pentru deplasarea pe suprafețe plane în care corpul robotului se află la aceeași distanță față de suprafață, mișcarea este realizată de mișcarea fiecărui picior în parte, pe baza sistemului de coordonate.

Motivul pentru care sistemul de coordonate este cel cartezian este acela de a ușura calculele mișcării. Cum controlul poziției ideale a fiecărui motor este realizat în coordonate sferice trebuie definit un nou set de coordonate pentru fiecare picior. Acest lucru necesită definirea a trei coordonate unghiulare pentru fiecare picior, fiecare coordonată reprezentând poziția unghiulară a unui motor care este prezentată în figura 8.

Tipurile de deplasare ale roboților cu șase picioare sunt clasificate după numărul de picioare care susțin corpul. Această clasificare este formată din trei moduri principale de deplasare:

- Modul de deplasare 3+3, în care într-un moment de timp există trei picioare care susțin corpul robotului.
- Modul de deplasare 4+2, în care, într-un moment de timp există patru picioare care susțin corpul robotului.
- Modul de deplasare 5+1, în care, într-un moment de timp există cinci picioare care susțin corpul robotului.

Aceste trei moduri de deplasare reprezintă singurele posibilități de deplasare, neexistând o altă combinație de picioare care să permită deplasarea robotului. Modurile conferind robotului o stabilitate și o viteză specifică fiecăruia în parte.

Modul de deplasare 3+3

Acest mod este definit prin existența a trei picioare care suportă corpul robotului, în timp ce restul de trei picioare se află în aer și se deplasează în față realizând deplasarea robotului. Numărul de pași care trebuie realizați într-un singur ciclu de deplasare este de doi, reprezentând cel mai mic număr de pași pe care îi poate avea un mod de deplasare. Acest lucru face ca modul de deplasare trepidă să devină cel mai rapid și eficient mod de deplasare, însă dezavantajul acestui mod apare la stabilitate, fiind cel mai puțin stabil datorită faptului că robotul este susținut doar de trei picioare.

Picioarele sunt împărțite în două grupuri de trei, în așa fel încât cele trei picioare care rămân pe sol să creeze un triunghi cât mai stabil. Acest lucru înseamnă alegerea piciorului din față și a piciorului din spate pentru o parte și alegerea piciorului din mijloc de cealaltă parte, repartizare prezentă în figura 15.

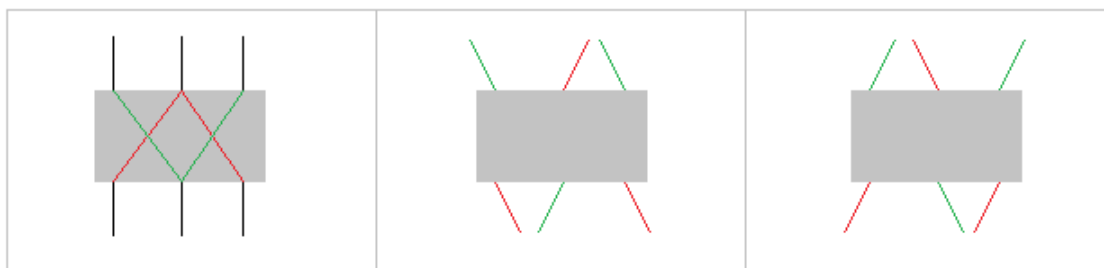


Figura 15: Modul de deplasare 3+3

Modul de deplasare 4+2

În cadrul acestui mod de deplasare, picioarele robotului sunt împărțite în trei grupuri de câte două. În oricare moment, două grupuri de picioare, reprezentând patru picioare susțin robotul în timp ce restul de două picioare se deplasează înainte. Datorită faptului că robotul este susținut de patru picioare, are o stabilitate foarte bună chiar și în cazul în care unul din picioare nu mai funcționează, doar dacă acel picior nu este un picior situat în mijloc, caz în care robotul se va dezechilibra.

Figura 16 prezintă o posibilă variantă de deplasare în modul 4+2, în care deplasarea este realizată după încheierea ciclului prezentat în figura 16, însă deplasarea robotului este încetinită.

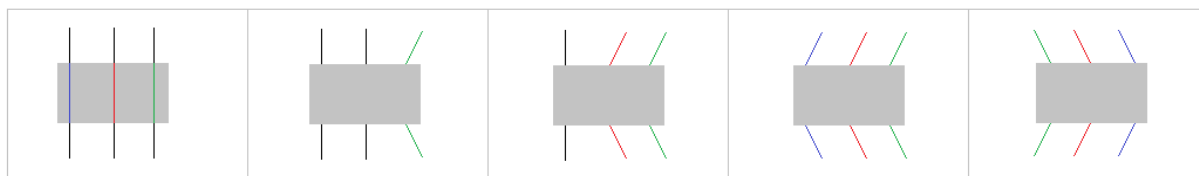


Figura 16: Modul de deplasare 4+2.

Modul de deplasare 5+1

Ultimul dintre modurile de deplasare posibile este definit prin 5 picioare care se află pe sol și un singur picior care se deplasează înainte. Picioarele se pot deplasa în orice ordine, dar în general deplasarea se realizează în sensul acelor de ceasornic sau în sens invers acelor de ceasornic. Acest mod de deplasare oferă cea mai stabilă mișcare posibilă prin care robotul se poate deplasa chiar și în cazul în care unul din picioare nu mai funcționează, însă datorită caracterului robust este și cel mai lent dintre moduri.

Cele trei moduri de deplasare prezentate reprezintă modurile uzuale folosite la deplasarea roboților cu șase picioare, însă pot fi aduse o mulțime de modificări.

Deplasarea robotului

Deplasarea unui robot cu șase picioare este realizată de unul din modurile prezentate anterior în care robotul realizează mișcări periodice. Aceste mișcări sunt definite prin sistemul de coordonate al robotului. Poziția fiecărui picior, reprezentată prin coordonatele (X, Y, Z) revine ciclic la poziția inițială (X_{init} , Y_{init} , Z_{init}), definită înainte de începerea deplasării robotului. Aceste valori fiind definite în sistemul de coordonate cartezian, care are centrul în coapsa fiecăruia din picioarele robotului, configurația a piciorului care poate fi văzută în figura 17.

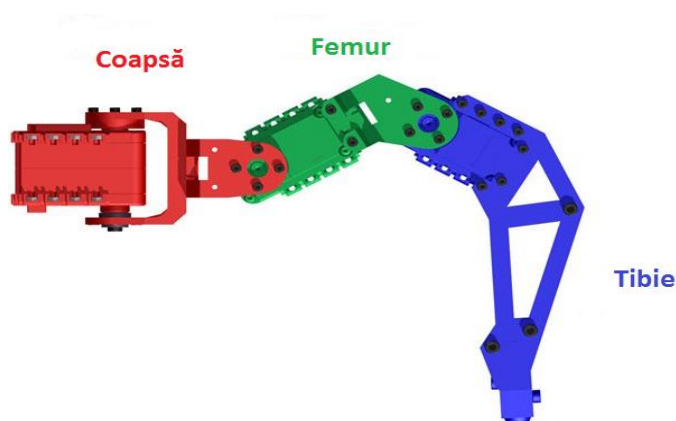


Figura 17: Componenta piciorului robotului.

Modul elementar de deplasare este realizat prin intermediul a trei pași:

- Mișcarea piciorului în sus
- Mișcarea piciorului în jos
- Mișcarea piciorului în spate

Cele trei determinând deplasarea înainte a robotului.

Calculul poziției fiecărui picior se realizează prin adugarea coordonatelor vectorilor mișcărilor elementare care coordonează poziția piciorului respectiv. Toate valorile obținute în aceste calcule fiind reprezentate în milimetrii.

Deplasarea este definită de următorii parametri:

- Vectorii de mișcare (X_v , Y_v , Z_v), care stabilesc direcția de deplasare
- Numărul de pași aparținând fiecărui ciclu ($Sciclu$)
- Numărul de pași prin care robotul se deplasează înainte ($Sfață$)
- Înălțimea până la care se ridică piciorul (l)
- Timpul unui singur ciclu (t)

Calcululele respectivelor mișcări în coordonate carteziene sunt descrise în tabelul 3.

Tabel 3: Calculul mișcărilor.

Deplasare	Sus	Down	Înapoi
X_i	X_{init}	$(X_v * Sfață * t) / 2 * Sciclu + X_{init}$	$X_{(i-1)} - (X_v * t) / 2 * Sciclu + X_{init}$
Y_i	X_{init}	$(Y_v * Sfață * t) / 2 * Sciclu + Y_{init}$	$Y_{(i-1)} - (Y_v * t) / 2 * Sciclu + Y_{init}$
Z_i	Z_{init-1}	Z_{init}	Z_{init}

Rezultatele calculelor pentru mișcare picioarelor sunt adăugate la poziția inițială a picioarelor.

- $X = X_i + X_{init}$
- $Y = Y_i + Y_{init}$
- $Z = Z_i + Z_{init}$

2.1.4 Controlul robotului

Robotul PhantomX Mark II reprezintă un kit robotic care a fost asamblat în cadrul laboratorului Speed și care poate fi controlat prin intermediul unei telecomenzi ArbotiX Commander cu ajutorul a doua module wireless Xbee.

- ArbotiX Commander

ArbotiX Commander este controler ce are la baza arhitectura AVR și care este utilizat pentru a controla robotul PhantomX Mark II în varianta lui inițială prin încărcarea programului de deplasare inițial.

Controlerul conține 2 „thumb joysticks”, 8 butoane și locul în care se conectează modulul de comunicație Xbee, controler prezentat în figura 18.

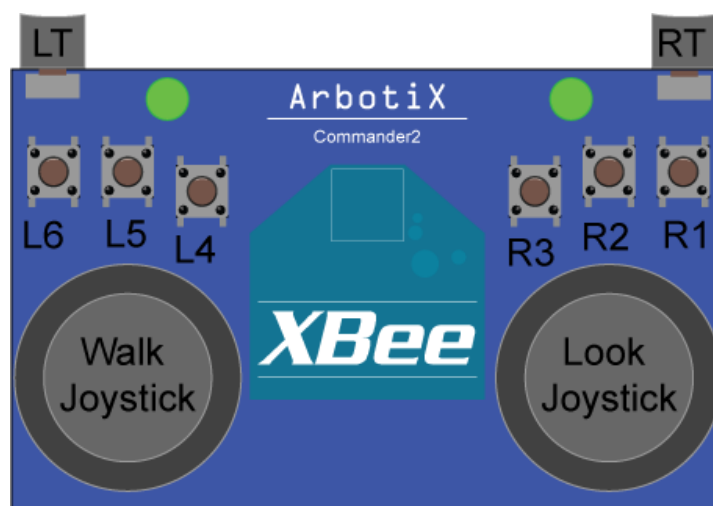


Figura 18: ArbotiX Commander

Utilizarea joystick-ului „Look”, face ca robotul să se rotească pe loc, iar folosirea joystick-ului „Walk” în față și în spate face ca robotul să se deplaseze înainte și înapoi, în timp ce mișcarea acestuia în stânga și în dreapta face ca robotul să se deplaseze în stânga sau în dreapta fără a-și modifica orientarea.

Pentru a scrie codul necesar Commander-ului ArbotiX trebuie folosită biblioteca „The Commander” prin adăugarea în codul de pe controler a următoarelor linii de cod.

```
#include <Commander.h>
Commander command=Commander();
```

Variabilele folosite în program pentru realizarea deplasării robotului sunt prezentate în tabelul 4.

Tabel 4 : Variabile importante

Variabile	Descriere
signed char walkV	Deplasarea verticală a joystick-ului „walk” în plaja de valori (-100, 100)
signed char walkH	Deplasarea orizontală a joystick-ului „walk” în plaja de valori (-100, 100)
signed char lookV	Deplasarea verticală a joystick-ului „look” în plaja de valori (-100, 100)
signed char lookH	Deplasarea orizontală a joystick-ului „look” în plaja de valori (-100, 100)

Pentru ca robotul să realizeze o anumită acțiune se face următoarea verificare

```
if(command.ReadMsgs()>0){
    if((command.buttons&BUT_LT)>0){
        // realizeaza o acțiune
```

- Modulul Xbee

Modulul Xbee este un mic dispozitiv de radio frecvență care transmite și recepționează date pe calea aerului utilizând semnale radio. Capabilitatea de funcționare fără fire este esențială în momentul în care este nevoie de senzori în locurile în care este imposibilă instalarea de cabluri.

Dispozitivul Xbee este foarte bine configurat și suportă o multitudine de protocoale, care permit alegerea tehnologiei potrivite pentru aplicația în cauză.

O serie de moduri de utilizare a dispozitivelor Xbee este enumerată mai jos:

- Controlul unui robot prin intermediul unei telecomenzi
- Crearea de dispozitive electronice care pot fi purtate de oameni sau animale fără a le incomoda deplasarea.
- Realizarea de clădiri inteligente sau cât mai receptivă la interacțiunea cu oamenii.

Aceste dispozitive oferă o comunicație simplă și sigură între microcontrolere, calculatoare sau oricare alte sisteme care dispun de porturi seriale, fiind o opțiune foarte bună în lucrul cu sisteme care folosesc Arduino[7].

- Modul de funcționare al ansamblului PhantomX Mark II – ArbotiX Commander

Funcționalitatea sistemului PhantomX Mark II – ArbotiX Commander se bazează pe transferul de date de la ArbotiX Commander la PhantomX Mark II prin intermediul celor două module Xbee plasate una pe robot și cealaltă pe manetă, după cum se poate vedea în figura 19.

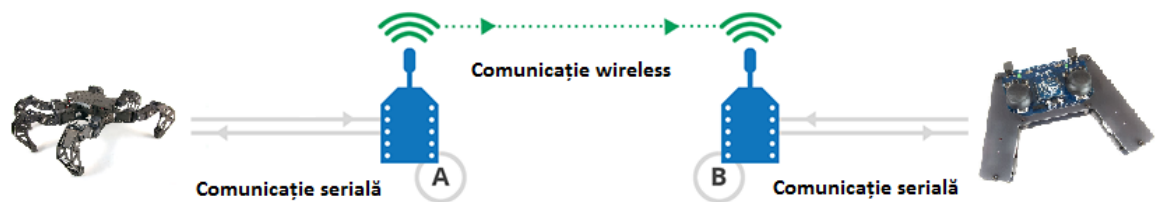


Figura 19: Comunicația între hexapod și manetă.

Atât codul sursă care se află pe robot, cât și codul sursă de pe commander-ul ArbotiX este scris în limbaj arduino, folosind software-ul Arduino 1.0.6 deoarece librariile folosite în codurile sursă specificate nu suportă versiuni mai noi ale software-ului Arduino.

Încărcarea codului sursă pe robotul PhantomX Mark II presupune parcurgerea următoarelor etape:

- Se detașează modulul Xbee al robotului, prezentat în capitolul 2.1.1.
- Se mută jumper-ul de selecție a alimentării care de asemenea este prezentat în capitolul 2.1.1 din varianta în care robotul este alimentat de la baterie, în varianta în care alimentarea se face prin intermediul calculatorului de pe care se programează.
- Se realizează legătura între calculator și robot prin intermediul unui cablu FTDI și a unui dispozitiv UartSBee care este un adaptor USB-Serial, legătură prezentată în figura 20.

- Ultima etapă fiind reprezentată de pornirea software-ului Arduino, selectarea plăcii de dezvoltare ArbotiX, a portului prin care s-a realizat conexiunea, după care se realizează încărcarea efectivă a codului sursă dorit pe robocontroller.

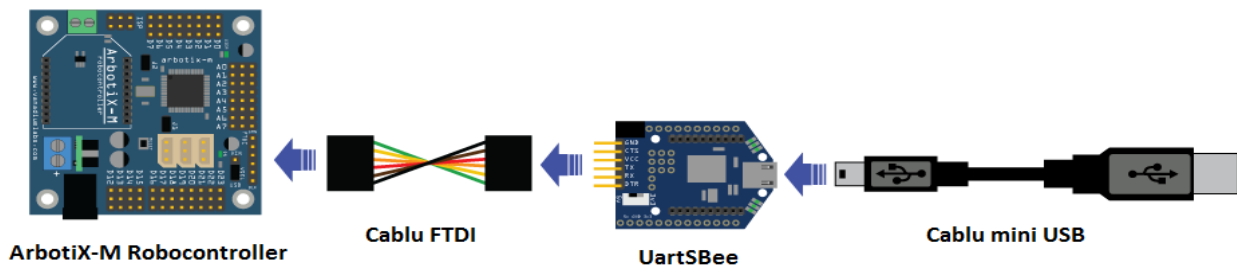


Figura 20: Programarea robotului prin intermediul calculatorului.

Încărcarea programului sursă dorit pe commander-ul ArbotiX este mai simplă fiind nevoie doar de pornirea software-ului Arduino, selectarea plăcii de dezvoltare (Arduino Duemilanove w/ Atmega 328) și încărcarea codului.

- Realizarea configurărilor

O etapă importantă în punerea în funcțiune a întregului ansamblu constă în realizarea configurărilor diferitelor dispozitive hardware folosite

- Configurarea ID-urilor pentru motoarele Dynamixel AX-12A

Stabilirea pentru fiecare dintre motoare a unui ID unic este importantă deoarece controlerul robotului identifică și trimite date motoarelor pe baze ID-ului pe care îl are fiecare dintre ele.

Setarea ID-urilor corespunzătoare este realizată prin intermediul unui software numit DynaManager care necesită conectarea la platforma robotica a fiecărui motor în parte așa cum se poate vedea în figura 21, iar stabilirea ID-urilor se face în conformitate cu figura 12.

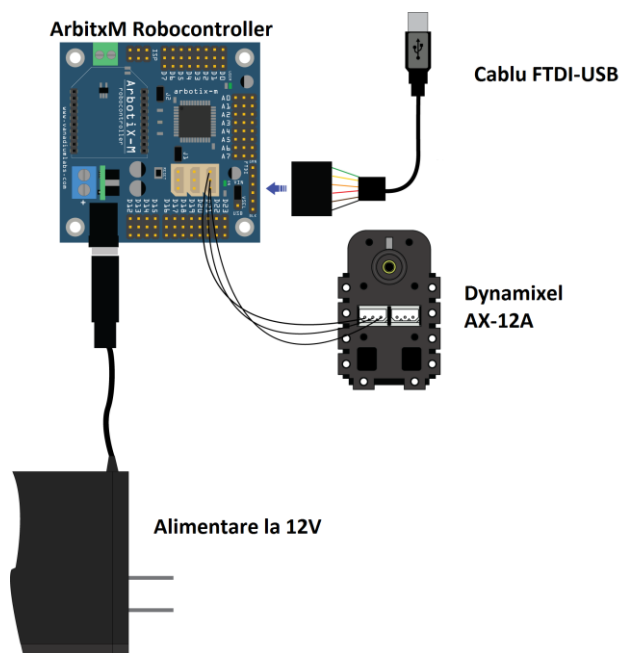


Figura 21: Conectarea individuală a motoarelor în vederea stabilirii ID-urilor.

○ Configurarea modulelor Xbee

Pentru a se realiza conexiunea wireless între ArbotiX Robocontroller și ArbotiX Commander, este necesar să fie configurate cele doua module Xbee. Pentru a fi configurate este necesară modificarea a 4 parametrii:

- PAN ID (ID) : reprezintă ID-ul rețelei personale. Pentru ca cele doua Xbee-uri sa comunice, acestea trebuie să aibă același PAN ID.
- 16-bit Source Address (MY) : acesta reprezintă ID-ul pentru modulul Xbee propriu-zis.
- Destination Address Low (DL) : este adresa sursa a ID-ului pentru Xbee-ul cu care se dorește realizarea conexiunii.
- Interface Data Rate (BD) : reprezintă rata de transfer la care vor funcționa cele două Xbee-uri.

Pentru configurare, se folosește programul X-CTU în care se introduc pe rând cele doua Xbee-uri pentru care se completeaza câmpurile necesare ca în tabelul 5 de mai jos.

Tabel 5 : Parametrii de configurare Xbee

Parametrii	Xbee #1 (pentru ArbotiX Commander)	Xbee #2 (pentru ArbotiX Robocontroller)
Pan ID	3400	3400
MY ID	1	2
DL ID	2	1
Data Rate	5	5

2.1.5 Acumulator LiPo

Bateriile Lithium Polymer, reprezintă un nou tip de baterii care în prezent sunt utilizate într-o gamă largă de dispozitive electronice. Acest tip de baterii a devenit din ce în ce mai popular în industria de radio control datorită puterii mare de care dispun și de perioada îndelungată pe care o au până în momentul descărcării.

Avantajele folosirii acumulatorilor LiPo sunt următoarele:

- Pot fi realizate în orice formă și mărime
- Sunt capabile să înmagazineze o cantitate considerabilă de putere
- Dispune de rate mari de descărcare ceea ce le face ideale pentru utilizarea în dispozitive electrice.

În cadrul proiectului curent, acumulatorul folosit la alimentarea robotului PhantomX Mark II este un acumulator LiPo de 11.1V, care este format din trei celule de 3.7V fiecare, care ofera o perioadă suficient de mare de utilizare a robotului.

2.2 SENZORUL KINECT

Senzorul Kinect era cunoscut inițial după numele „Project Natal”. Este un dispozitiv de detectare a mișcării care a fost dezvoltat pentru consolele de jocuri Xbox 360. Un avantaj foarte important care a făcut acest senzor să reziste printre celelalte dispozitive de gen este acela că nu trebuie controlat prin intermediul unei telecomenzi, ci detectează poziția corpului, mișcarea și vocea persoanei care îl utilizează. Câștigând din ce în ce mai mult popularitate datorită performanțelor obținute în domeniul jocurilor video, utilizarea senzorului Kinect a depășit granița jocurilor fiind utilizat într-o mulțime de aplicații și domenii despre care voi vorbi în continuare.

Kinect pentru Windows este un senzor special realizat pentru calculator, care ajută dezvoltatorii de software să scrie propriile lor coduri prin care se realizează aplicații bazate pe gesturile umane și pe deplasarea corpului. Odată cu lansarea dispozitivelor Kinect pentru Windows, interesul în dezvoltarea de programe bazate pe detecția mișcării a crescut foarte mult.

Senzorul Kinect este un dispozitiv orizontal care dispune de senzori de adâncime, camera color, un set de microfoane, un motor care permite înclinarea senzorului atât orizontal cât și vertical și un LED care se aprinde atunci când dispozitivul funcționează sau poate semnaliza o tensiune insuficientă atunci când are culoarea roșie. În afară de componentele enumerate anterior, senzorul Kinect are deasemenea un adaptor pentru alimentare externă și un adaptor USB care îi permite conectarea la un calculator[8].

Figura 22 prezintă diferitele componente de care dispune.

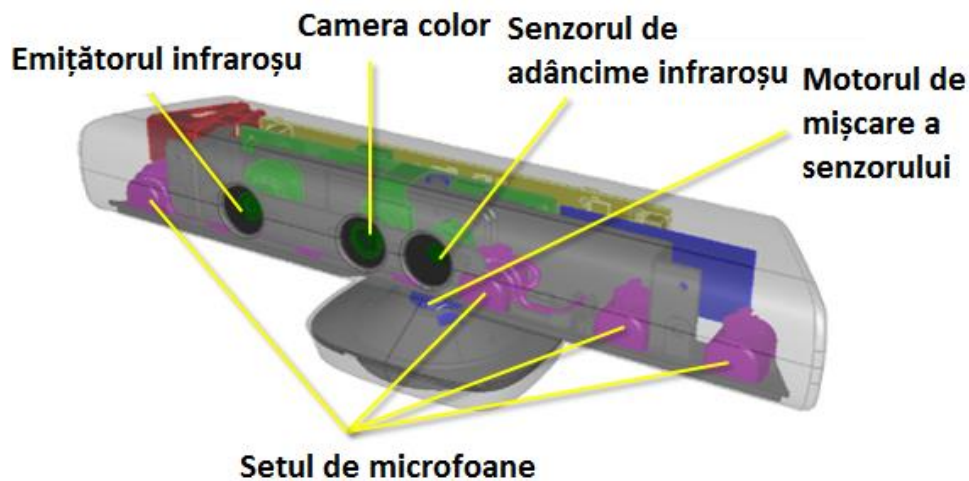


Figura 22: Componentele senzorului Kinect.

2.2.1 Camera color

Camera color se ocupă cu captura și cu stream-ul imaginii video color. Funcția sa este de a detecta culorile roșu, albastru, verde de la sursă. Stream-ul video care este primit de la camera constă într-o succesiune de poze. Stream-ul color al Kinect-ului suportă o viteză de până la 30 de frame-uri pe secundă (FPS), la o rezoluție de 640 x 480 de pixeli, ajungând până la o rezoluție maximă de 1280 x 960 de pixeli cu o viteză de 12 FPS. După cum se poate observa valoarea frame-urilor pe secundă variază invers proporțional cu rezoluția folosită pentru afișarea imaginii. Zona vizibilă prin intermediul camerei color de pe Kinect este de 43 de grade vertical și 57 de grade orizontal.

2.2.2 Emițătorul infraroșu și senzorul de adâncime infraroșu

Senzorul de adâncime al Kinect-ului este format din emițătorul de infraroșu și senzorul de adâncime infraroșu. Ambele lucrează împreună pentru a realiza imaginea de adâncime. Emițătorul infraroșu arată ca o cameră la exterior, însă în interior este un proiector infraroșu care emite constant lumină infraroșie printr-un punct pseudo-aleator care este plasat pe tot ce apare în fața senzorului. Aceste puncte sunt în mod normal invizibile pentru oameni, dar este posibilă obținerea informațiilor lor cu privire la adâncime folosind un senzor de adâncime infraroșu. Lumina punctată reflectată din diferitele obiecte din imagine este citită de senzorul de adâncime infraroșu care face conversi în informații de adâncime prin măsurarea distanței dintre senzor și obiectul de la care a fost citit punctul de infraroșu. Imaginea de adâncime a senzorului are o rezoluție de 640 x 480, 320 x 240 sau 80 x 60 pixeli, în timp ce raza de care dispune rămâne aceeași ca la camera color [8].

Următoarea figura 23, prezintă cum este realizată imaginea de adâncime:

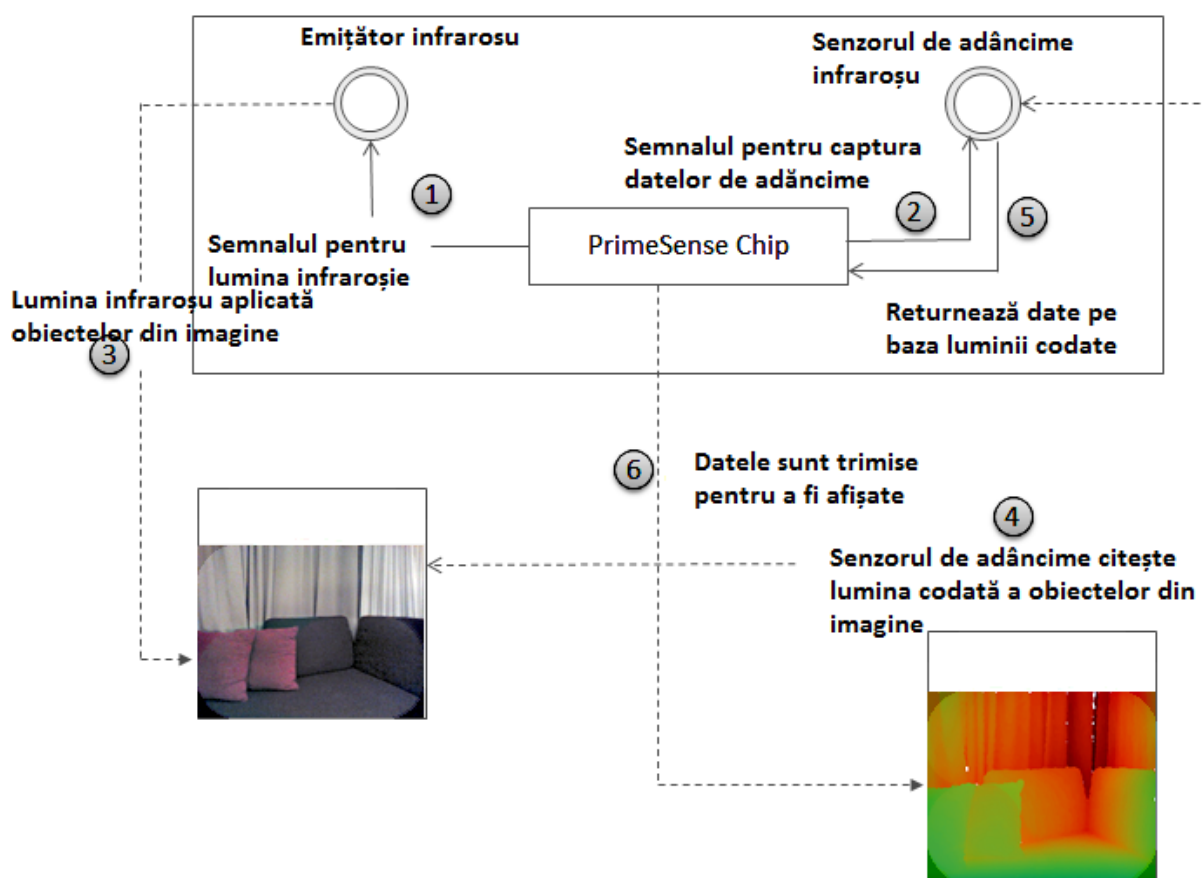


Figura 23: Procesul de obținere a imaginii de adâncime.

Imaginea anterioară este explicată în cele ce urmează:

În momentul în care se dorește obținerea de date de adâncime, cipul PrimeSense trimite un semnal către emițătorul infraroșu pentru a porni lumina infraroșu reprezentată în imagine cu (1), după care un alt semnal către senzorul de adâncime infraroșu pentru a iniția captura de date de adâncime de la poziția curentă a senzorului (2). În acest timp, emițătorul infraroșu începe trimiterea de lumină infraroșu invizibilă ochiului uman (3) către obiectul aflat în fața dispozitivului. Senzorul de adâncime infraroșu începe citirea datelor de la obiect pe baza distanței punctelor de lumină individuale reflectate (4), informații care sunt apoi trimise la cipul PrimeSense (5). Cipul PrimeSense analizează datele primite și creează imaginea de

adâncime frame cu frame, pe care o trimite la ieșirea stream-ului de adâncime ca imagine de adâncime (6).

Senzorul Kinect a deschis noi orizonturi pentru dezvoltatori de a construi o gamă din ce în ce mai mare de aplicații. O serie de aplicații posibile este prezentată în continuare:

- Captură video în timp real folosind camera color.
- Urmărește deplasarea corpului uman după care răspunde la mișcările și gesturile lui precum o interfață naturală.
- Măsoară distanța până la obiectele pe care le întâlnește.
- Este capabil de analiză 3D pe baza careia poate realiza modele 3D.
- Poate genera o hartă de adâncime a obiectelor identificate.
- Este capabil să recunoască vocea umană și pot fi dezvoltate aplicații care pot fi controlate prin intermediul vocii, acest lucru se datorează setului de microfoane de care dispune.

Următoarele exemple de folosire a senzorului Kinect vor oferi o înțelegere mai bună a dispozitivului:

- Domeniul medical: folosind senzorul kinect, se pot realiza diferite aplicații cum ar fi monitorizarea funcțiilor vitale ale pacienților, mișcările corpului uman.
- Domeniul roboticii: senzorul kinect poate fi utilizat ca un sistem de navigare pentru roboți fie prin urmărirea gesturilor umane, comenzi vocale.
- Domeniul educației: pot fi realizate aplicații pentru studenți sau copii care să îi ajute la învățarea diferitelor lecții din anul școlar, pe baza gesturilor sau comenzilor vocale.
- În sistemele de securitate: Senzorul Kinect poate fi folosit la realizarea de aplicații prin care poți detecta mișcările corpului sau fețele oamenilor după care să fie trimise notificări cu privire la identificarea diferitelor persoane.
- În realitatea virtuală: Folosind tehnologia 3D a senzorului, poate fi realizată o serie de aplicații aparținând realității virtuale.
- În domeniul militar: Kinectul poate fi folosit la construirea de roboți inteligenți care pot spiona inamicii.

2.3 MICROCALCULATORUL RASPBERRY PI 3

Plăcile Raspberry Pi au în componența lor un procesor, un cip grafic, memorie de program (RAM) și o mulțime de interfețe și conectori pentru dispozitive externe. O parte dintre aceste dispozitive sunt esențiale pentru funcționare. Rpi funcționează asemanator cu un calculator obișnuit având nevoie de o tastatură pentru scrierea comenzilor, un ecran și de alimentare.

Modelul Raspberry Pi 3 este a treia generație de Raspberry Pi. Acest calculator poate fi folosit pentru numeroase aplicații surclasând de departe modelele anterioare Raspberry Pi și Raspberry Pi 2. Având aceeași forma ca și modele precedente, acesta din urmă vine cu un procesor de 10x mai rapid decât prima generație de Raspberry Pi având totodată conectivitate Wireless LAN și Bluetooth integrate[10].

Ca și specificații, vine cu :

- Procesor de 1.2GHz Quad-Core ARM Cortex-A53
- GPU Dual Core VideoCore IV.
- Memorie de 1GB LPDDR2
- Sistemul de operare bootează de pe un card micro SD, acesta poate fii atât o versiune de Linux, cât și Windows 10 IoT
- Dimensiunile plăcii sunt 85 x 56 x 17 mm
- Pentru alimentare are nevoie de un micro USB care să produca cel puțin un curent de 1A, curent care poate ajunge până la 2.5A și o tensiune de 5V.

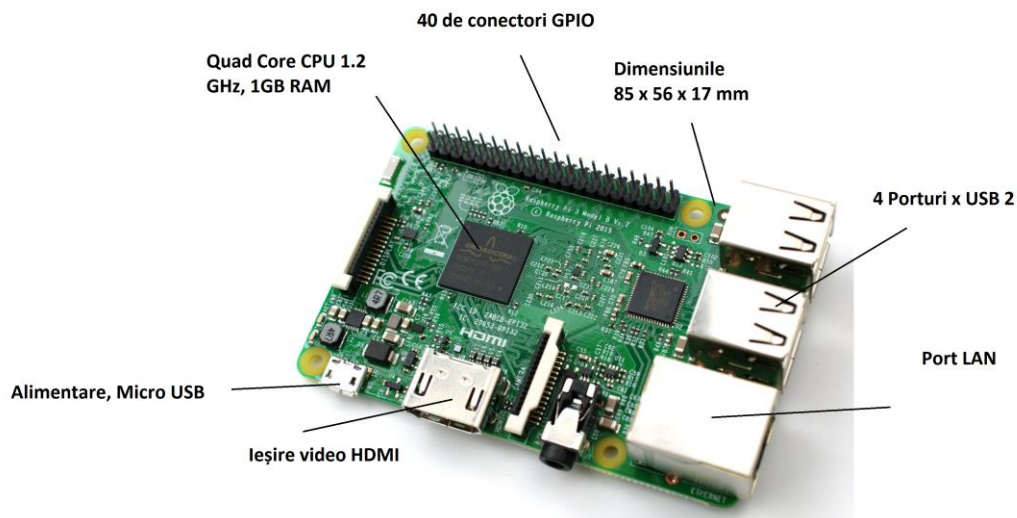


Figura 24: Raspberry Pi 3.

2.4 SENZORUL DE DISTANȚĂ SHARP

Senzorul de distanță Sharp este poate cel mai puternic senzor de distanță accesibili pe care îl poate folosi un pasionat de robotică. Are o eficiență bună, este ușor de utilizat, are un preț acceptabil, o rază suficientă pentru detecția obiecte apropiate și un consum redus de energie.

Funcționarea senzorului Sharp se bazează pe un proces de triangulație. Un impuls de lumină este emis care ulterior este reflectat înapoi sau nu se reflectă deloc. În momentul în care lumina se reflectă, această se întoarce sub un unghi de care depinde distanța până la obiectul din care s-a reflectat. Triangulația are rolul de a detecta unghiul fascicului reflectat și odată ce se cunoaște unghiul se poate determina distanța.

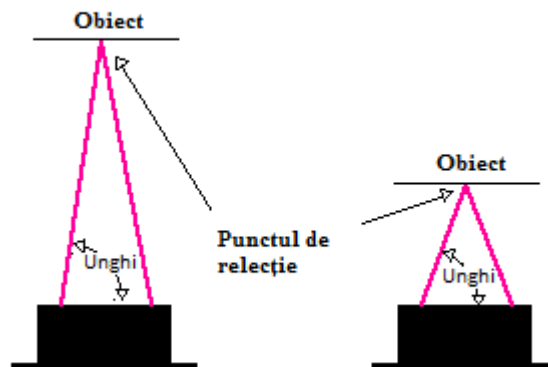


Figura 25: Stabilirea distanței până la obiect.

Receptorul senzorului IR Sharp este echipat cu lentile cu o precizie foarte bună, care transmit lumina reflectată pe o matrice liniară CCD încapsulată, bazată pe unghiul triangulației. După care matricea CCD determină unghiul și apoi determină o valoare analogică corespunzătoare unghiului, care poate fi citită de microcontroler. Mai mult, circuitele senzorului Sharp aplică o modulație de frecvență asupra fascicului infraroșu emis, ceea ce îl face aproape imun la interferențele cauzate de lumina ambientală și oferă o indiferență foarte bună la culoarea obiectului pe care îl detectează. Cu alte cuvinte, senzorul este capabil să detecteze un perete negru aflat în lumina soarelui cu zgomot aproape de zero [11].

Odată cu avantajele care au fost prezentate anterior vine și o serie de dezavantaje despre care voi discuta în continuare. O primă problemă serioasă care apare la folosirea senzorului IR Sharp este poziționarea unui obiect sub raza minimă de acțiune a senzoului. Acest lucru se întâmplă atunci când un obiect este foarte aproape și senzorul nu poate face o citire precisă, moment în care îi transmite robotului că obiectul detectat se află la o distanță mare.

O posibilă soluție la această problemă este așezarea senzorului pe robot astfel încât raza minimă a senzorului să depășească corpul robotului.

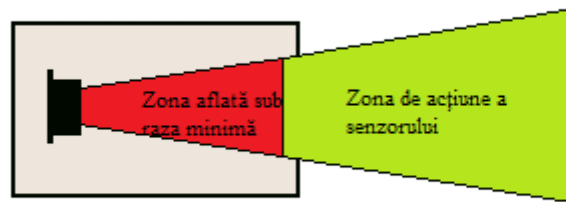


Figura 26: Așezarea senzorului pentru eliminarea erorii.

O altă problemă întâlnită la senzorul Sharp este reprezentată de fasciculul infraroșu care este foarte îngust. Pentru cazul în care este nevoie de o precizie bună, un fascicul îngust este ideal, dar problema apare când fasciculul nu este îndreptat exact pe obiect, caz în care senzorul consideră obiectul invizibil.

Misiune de explorare a unei zone cu un robot autonom.

CAPITOLUL 3

TEHNOLOGII SOFTWARE FOLOSITE

3.1 ARDUINO IDE

Arduino IDE reprezintă o unealtă foarte bună de programare hardware ușoară și rapidă. Arduino este un program open-source, care oferă o metodă simplă de a folosi cipurile Atmega. Arduino IDE este software-ul unde se poate scrie codul, care mai apoi este încărcat pe microcontrolerul Atmega. Codul este apoi executat de microcontroler. Arduino IDE folosește o variație a limbajelor de programare C și C++. Înainte de a începe scrierea de cod propriu-zis pe Arduino IDE trebuie selectate tipul de placă folosit și portul serial prin care este realizată conexiunea [4].

Pentru a începe să scriem propriul nostru cod sursă, sunt necesare câteva reguli de bază ale sintaxei limbajului de programare. Cu alte cuvinte, codul trebuie scris corespunzător astfel încât un alt programator să îl poată înțelege.

În continuare voi prezenta o serie de lucruri care trebuie reținute în momentul în care începem să scriem cod în Arduino IDE:

- Un program scris în Arduino poartă numele de sketch

- Întregul code dintr-un sketch Arduino este executat de la vârf către bază
- Sketch-urile arduino sunt divizate în 5 părți:
 1. Sketch-ul începe în mod obișnuit cu un antet în care este explicat ce face sketch-ul respectiv.
 2. Se declară variabilele globale.
 3. După ce variabilele inițiale au fost setate se începe rutina „ setup ”. În funcția „ setup ” se pun condițiile inițiale pentru variabile dacă este necesar și se rulează orice porțiune de code care se dorește a fi rulată o singură dată. Aici este locul unde se inițializează și comunicația serială care este necesară pentru folosirea monitorului serial.
 4. Urmează funcția „ loop ”, care reprezintă partea principală a sketch-ului. Aici este locul unde se află cele mai importante porțiuni din cod, iar odata ce programul a intrat în funcția „ loop ”, o execută fără oprire atâta timp cât sketch-ul continuă sa ruleze.
 5. După funcția „ loop ” se găsesc funcțiile utilizatorului care pot fi apelate atât în funcția „ main ”, cât și în funcția „ loop ”.

3.2 SISTEMUL DE OPERARE RASPBIAN

Raspbian este un sistem de operare gratuit optimizat pentru setul de instrucțiuni al arhitecturii ARMv6, pe care se bazează microcalculatorul Raspberry Pi. Acesta are la bază sistemul de operare Debian 7 „ Wheezy ”. Numele „ Raspbian ” este obținut în urma unui joc de cuvinte între Raspberry și Debian (Rasp și bian). Acest sistem de operare oferă peste 35000 de pachete software ușor de instalat, optimizate pentru a rula pe hardware-ul ARM11 al Raspberry Pi. Sistemul de operare Raspbian este un sistem de tip „ hard – float ”, adică operează în virgulă mobilă prin emulare hardware. În versiunile mai recente ale Raspbian, multe dintre elementele semnificative necesare ale Debian Wheezy au fost transferate pentru arhitectura ARMv6, deși distribuția de bază necesită ARMv7. Astfel, performanța pentru aplicațiile care folosesc operații de aritmetică în virgulă mobilă este crescută deoarece sistemele de operare precedente pentru Raspberry Pi folosesc setări de tip „ soft float ”, mai puțin eficiente, prin care operațiile aritmetice în virgulă mobilă pentru arhitectura ARMv6 sunt emulate software [9].

Sistemul de fișiere este standard pentru un sistem derivat din Linux. Acesta este de tip ierarhic, împărțit în directoare, dintre care vor fi enumerate cele mai importante.

- /bin – locul în care se instalează programele de utilizator
- /boot – aici se găsesc fișierele necesare la pornire
- /dev – conține fișiere speciale care reprezintă dispozitivele sistemului
- /etc – aici se găsesc fișierele de configurare a sistemului
- /home – directorul utilizatorului
- /lib – aici găsim driverele și modulele kernel
- /sys – este un director special pentru Raspberry Pi care conține driverele pentru dispozitivele hardware
- /usr – conține programele și datele utilizate de toți utilizatorii
- /var – conține fișiere pentru procesare și loguri de sistem

3.3 PYTHON

Python este un limbaj de programare open-source de nivel înalt care poate fi utilizat într-o gamă largă de proiecte care necesită programare. Limbajul Python a fost creat de Guido Van Rossum în anii 90, devenind un limbaj de programare din ce în ce mai cunoscut și mai utilizat de programatori. Numele limbajului a fost luat de la programul de comedie Monty Python's Flying Circus. Limbajul este folosit foarte mult în sistemele de administrație, multe dintre componentele esențiale ale sistemului de operare Linux fiind scrise în el și de asemenea este un limbaj de programare foarte utilizat de cei care doresc să învețe programare.

Python este un limbaj de programare foarte popular care oferă posibilitatea programării structurate dar și orientate pe obiecte. Este un limbaj de scriptare, ceea ce înseamnă că este interpretat și nu compilat, lucru care duce la o reducere a timpului necesar pentru procesul de dezvoltare și debugging. Limbajul aduce în zona limbajelor de programare o sintaxă foarte clară, care conține module, clase, excepții, tipuri dinamice. API-ul Python oferă module pentru o gamă foarte mare de funcționalități, de la cele de bază pentru lucrul cu șiruri și fișiere, până la cele pentru lucrul cu procese, threaduri, sockets sau serializare. Multe dintre aceste module oferă o interfață foarte similară programării la nivel de sistem din C, astfel încât funcțiile Python au același nume și aproximativ aceiași parametrii ca cei pentru funcțiile din C pentru apeluri de sistem. Pentru calcul numeric și grafice există bibliotecile NumPy, SciPy și Matplotlib[12].

Implementarea principală a Python, CPython, care este scrisă în C a fost concepută pentru a fi portabilă, funcționând pe Linux, Unix, Windows, Mac OS X. Programele pot fi executate și pe mașina virtuală Java prin folosirea compilatorului Jython, care le transformă în bytecode Java și permite folosirea de clase Java în cadrul acestora [12].

Indentarea în Python este mai mult decât parte a stilului de programare, fiind chiar parte din sintaxă. O linie nouă termină o declarație, pentru a continua o declarație pe mai multe linii, se folosește caracterul „\”. Un bloc de cod are toate liniile indentate cu același număr de spații (nu există begin și end sau {}). Instrucțiunile într-un bloc de cod sunt grupate unele sub altele, pe același nivel de indentare. Definițiile neindentate într-un program Python sunt de obicei variabile globale, definiții de clase, proceduri, funcții sau părți ale programului principal. După terminarea unei instrucțiuni, nu este necesară punerea simbolului „;”, cum ar fi necesar în alte limbaje de programare, indentarea corespunzătoare fiind suficientă. Este însă foarte important să nu fie omis simbolul „:” cu care se începe o indentare.

Exemplu de indentare corespunzătoare:

```
a = 1
while a <= 10 :
    print a,
    a += 1
print "Calculul s-a încheiat"
```

3.4 NUMPY ȘI MATPLOTLIB

Numpy este un pachet fundamental pentru calcule științifice, ce poate fi folosit în limbajul Python. Elementele cheie ale pachetului numpy sunt:

- Folosirea matricei N-dimensională
- Un număr mare de funcții
- Unelte folosite la integrarea de cod scris în C/C++ și Fortran
- Algebră liniară, transformata Fourier, capacități de randomizare.

Lăsând la o parte utilitatea științifică, numpy poate fi folosit ca o eficientă metodă de stocare de date la nivel multi-dimensional. Se pot defini tipuri de date arbitrare, care permit numpy să se integreze rapid și fără probleme într-o mulțime de baze de date.

Pachetul numpy aduce două obiecte fundamentale : o matrice N – dimensională (ndarray) și o funcție universală (ufunc). Matricea N-dimensională este o colecție omogenă de elemente indexate folosind N întregi. Există două părți esențiale care definesc o matrice N-dimensională:

1. Forma pe care o are matricea.
2. Tipul de elemente din care este formată.

Matplotlib este o librărie de reprezentare grafică 2D pentru python, care realizează imagini de calitate într-o varietate de formate. Matplotlib poate fi folosit în scripturile python, în aplicații web, în care încearcă să facă lucrurile ușoare și mai ușoare, iar lucrurile grele posibile. Cu ajutorul ei pot fi generate grafice, histogramme, grafice pentru erori, folosind doar câteva linii de cod.

Codul matplotlib este conceputal împărțit în trei părți:

- Interfața pylab reprezintă setul de funcții adus de matplotlib.pylab care permite utilizatorului sa realizeze grafice pe bază de cod asemănător ca cel necesar în Matlab
- API-ul matplotlib care reprezintă setul de clase utilizate la crearea și prelucrarea figurilor, textului, graficelor etc.
- Backends care reprezintă dispozitive de desenat dependente de dispozitiv, care transformă reprezentarea inițială într-o imagine pe suport real.

3.5 OPENCV

Proiectul OpenCV a fost început la Intel în anul 1999 de Gary Bradsky, având debutul pe piață în anul 2000. În prezent OpenCV suportă o mulțime de algoritmi ce au legătură cu Computer Vision și Machine Learning, continuându-și dezvoltarea cu fiecare zi care trece. OpenCV-ul actual suportă o varietate largă de limbaje de programare, cum ar fi C++, Python, Java și poate fi utilizat pe numeroase platforme printre care Windows, Linux, OS X, Android, iOS. De asemenea, interfețe bazate pe CUDA și OpenCL sunt în curs de dezvoltare pentru a putea fi utilizate în operații foarte rapide pe GPU-uri.

OpenCV – Python reprezintă Python API-ul OpenCV-ului, deoarece combină cele mai bune calități ale limbajelor OpenCV, C++, API și Python. Încă un avantaj este adus de folosirea librăriei Numpy despre care am discutat în subcapitolul 3.4, care reprezintă o librărie optimizată pentru operații numerice care utilizează o sintaxă asemănătoare cu cea a programului Matlab. Toate structurile matriceale alea OpenCV sunt convertite în matrice numpy, astfel operațiile care pot fi realizate în numpy pot fi combinate cu OpenCV, astfel crescând numărul de avantaje pe care îl avem la scrierea codului. Rezultă că OpenCV-Python este o alegere potrivită pentru rezolvarea rapidă a problemelor de computer vision [12].

OpenCV pentru imagini

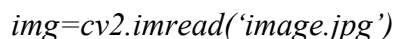
- Citirea imaginilor

Pentru a citi o imagine cu OpenCV se folosește funcția `cv2.imread()`. Imagine care se dorește a fi citită trebuie să se afle în directorul de lucru sau trebuie specificată întreaga cale a fișierului.

Al doilea argument este un fanion care specifică modul în care imaginea trebuie citită

- `cv2.IMREAD_COLOR` : încarcă o imagine color
- `cv2.IMREAD_GRAYSCALE` : încarcă o imagine cu nivele de gri
- `cv2.IMREAD_UNCHANGED`

Un exemplu de cod pentru citirea unei imagine este prezentat în continuare:

```
import numpy as np
import cv2

img=cv2.imread('image.jpg')
```

- Afișarea imaginilor

Pentru a afișa o imagine cu OpenCV se folosește funcția `cv2.imshow()`. Fereastra care afișează imaginea se adaptează la dimensiunile imaginii. Primul argument al funcției este numele ferestrei, iar al doilea argument este imaginea propriu-zisă. Pot fi create oricâte ferestre sunt necesare, dar trebuie să aibă nume diferite.

```
cv2.imshow('image',img)
```

- Salvarea imaginilor

Pentru a salva o imagine trebuie să folosim funcția `cv2.imwrite()`, în care primul argument reprezintă numele fișierului, iar al doilea este imaginea pe care vrem să o salvăm.

```
cv2.imwrite('image.png',img)
```

3.6 PROTOCOLUL I2C

I2C este un protocol în care este permisă comunicația între mai multe circuite integrate digitale „slave” și unul sau mai multe cipuri „master”. La fel ca la SPI, I2C este realizat doar pentru distanțe scurte și precum la interfețele seriale asincrone (UART) utilizează doar două fire pentru a realiza schimbul de informație.

Pentru a argumenta alegerea protocolului I2C pentru comunicație trebuie comparat cu alte opțiuni existente pentru a vedea cum diferă.

Porturile seriale

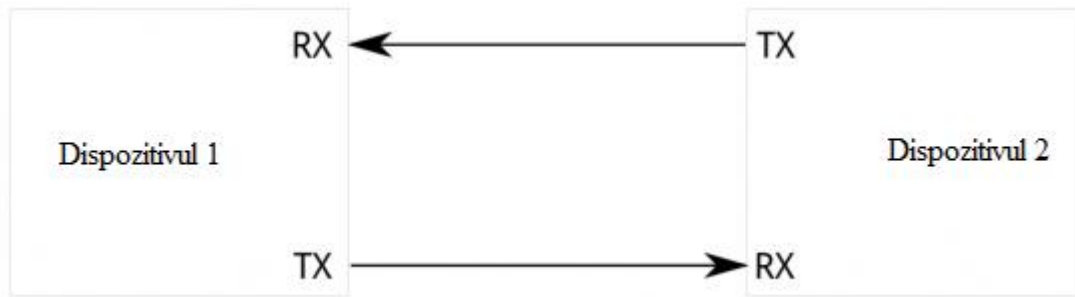


Figura 27: Comunicația prin porturi seriale.

Din cauză că porturile seriale sunt asincrone (nu sunt transmise informații cu privire la ceas), dispozitivele care le folosesc trebuie să stabilească în prealabil o rată de transfer a datelor. Porturile seriale asincrone au nevoie de hardware suplimentar la fiecare capăt al transmisiunii a cărui implementare în soft este destul de complexă și dificilă. O altă mare problemă a porturilor seriale asincrone este faptul reprezentată de comunicația care se poate realiza doar între două dispozitive.

SPI

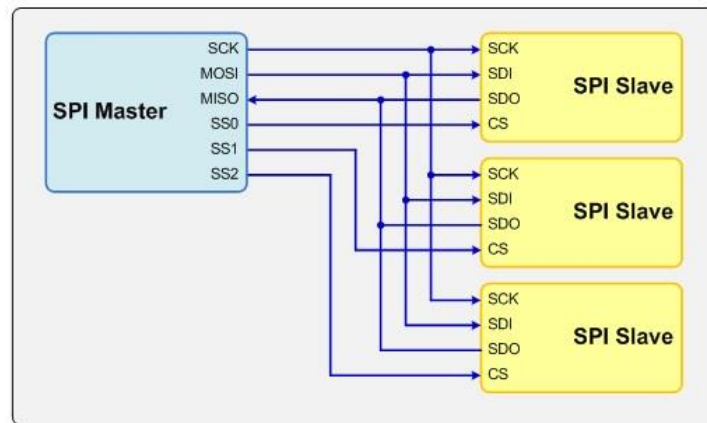


Figura 28: Protocolul SPI.

Cel mai semnificativ dezavantaj pentru SPI este reprezentat de numărul de pini necesari pentru realizarea comunicației. Conectarea unui singur master la un singur slave prin protocolul SPI necesită patru fire și fiecare nou slave pe care dorim să îl adăugăm necesită un nou pin de selecție I/O pe dispozitivul master. Protocolul SPI permite un master pe magistrală, dar suportă un număr arbitrar de dispozitive slave, care depinde doar de numărul de pini de selecție disponibili. SPI este potrivit pentru transferuri de date mari, conexiuni full-duplex, ce suportă o rată a ceasului de până la 10 MHz pentru unele dispozitive.

I2C.

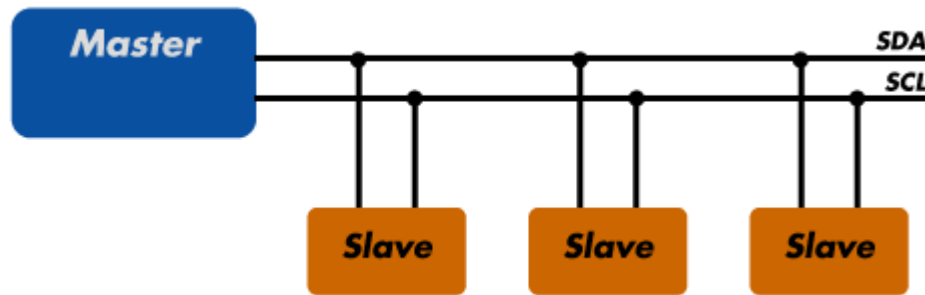


Figura 29: Protocolul I2C.

Protocolul I2C are nevoie de doar două fire, la fel ca porturile seriale asincrone însă aceste două fire sunt capabile să suporte până la 1008 dispozitive slave. De asemenea, spre deosebire de SPI, I2C este capabil să suporte mai multe dispozitive master, permițând ca mai mult de un master să comunice cu toate dispozitivele de pe magistrală, deși dispozitivele master nu pot comunica între ele iar comunicația cu restul dispozitivelor se realizează pe rând. Rata de transmitere a datelor se situează între porturile seriale asincrone și SPI, majoritatea dispozitivelor comuniând la 100kHz sau la 400kHz. Hardware-ul necesar pentru implementarea protocolului I2C este mai complex față de SPI, dar mult mai simplu ca la porturile seriale asincrone, necesitând o implementarea software ușoară [13] .

Fiecare magistrală I2C este formată din două semnale : SCL și SDA. SCL este semnalul de ceas, care este generat de master-ul ce se află pe magistrală, în timp ce SDA este semnalul de date.

Misiune de explorare a unei zone cu un robot autonom.

CAPITOLUL 4

PRELUCRAREA ȘI ANALIZA IMAGINILOR

4.1 INTRODUCERE

Procesarea de imagini implică schimbări în structura unei imaginii pentru a obține:

1. O îmbunătățire a informațiilor regăsite în imagine în raport cu interpretarea umană.
2. Face imaginea mult mai potrivită pentru a fi percepută de mașinile autonome.

Este important de realizat că cele două aspecte prezentate anterior, reprezintă aspecte ale prelucrării de imagini total opuse dar în același timp egale din punct de vedere al importanței. O metodă de prelucrare a imaginii care satisface primul aspect (1), o procedură care îmbunătățește calitatea unei imagini poate fi cea mai nepotrivită procedură care să fie utilizată pentru satisfacerea celui de-al doilea aspect discutat (2).

Oamenii preferă ca imaginile lor să fie clare, curate și pline de detalii în timp ce mașinile preferă imagini simple.

Exemple care evidențiază îmbunătățirea imaginilor în cazul (1):

- Îmbunătățirea marginilor unei imagini pentru a o face să fie mai clară, devenind o imagine mult mai plăcută decât imaginea inițială. Îmbunătățirea marginilor este o componentă vitală când vine vorba de printarea imaginilor, astfel că pentru ca imaginea să arate cât mai bine în momentul în care este printată, această practică a îmbunătățirii marginilor este frecvent utilizată.
- Înlăturarea zgomotului dintr-o imagine, zgomotul semnificând erorile aleatoare din imagine. Zgomotul este o problemă care apare foarte frecvent în transmisia de date, deoarece orice componentă electronică poate afecta datele care trec prin aceasta, rezultatele obținute fiind neplăcute. Zgomotul este întâlnit sub mai multe forme, fiecare tip de zgomot necesitând o metodă de înlăturare diferită.
- Înlăturarea neclarității datorată mișcării dintr-o imagine. În fotografiile în care obiectele se mișcă rapid, cum ar fi în cursele de mașini, problema neclarității trebuie luată în considerare și rezolvată.

Exemple care evidențiază îmbunătățirea imaginilor în cazul (2):

- Obținerea marginilor obiectelor din imagine, acțiune ce poate fi necesară pentru măsurarea obiectelor din imagine. Odată ce marginile au fost aflate, se poate măsura răspândirea obiectului și aria pe care acesta o ocupă. Se pot folosi de asemenea algoritmi de detecție a marginilor, algoritmi care rezează un prim pas în acest proces.
- Înlăturarea detaliilor din imagine. Pentru scopuri cum ar fi măsurarea sau numărarea, este posibil să fie nevoie să știm toate detalii dintr-o imagine, de exemplu pentru o mașină care verifică piesele dintr-o linie de asamblare singurul lucru care contează este forma, mărimea sau culoare obiectelor. Pentru aceste cazuri este nevoie de simplificarea imaginii.

4.2 IMAGINI DIGITALE

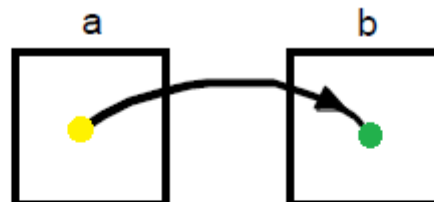
În cazul unei imagini sau al unei fotografii, în care regăsim doar alb și negru, se poate considera că imaginea în cauză este o funcție cu două dimensiuni în care valorile funcției stabilesc luminozitatea. Se poate presupune că în acest gen de imagini valorile luminozității pot fi orice număr real între 0.0 (negru) și 1.0 (alb). Gama de valori pentru coordonatele x și y va depinde de imagine, însă întotdeauna valorile pe care le pot avea vor fi cuprinse între 0.0 și 1.0.

O imagine digitală este diferită de o fotografie în care valorile lui x , y și $f(x, y)$ sunt discrete. Valorile dintr-o imagine digitală sunt doar valori întregi cuprinse între 1 și 256, valorile specifice luminozității fiind de asemenea cuprinse între 0 (negru) și 255 (alb). O imagine digitală poate fi considerată drept o matrice de puncte discrete. Aceste puncte poartă numele de elemente ale imaginii, sau mai bine spus pixeli.

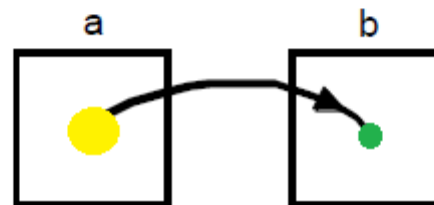
Tipurile de operații ce pot fi aplicate unei imagini digitale:

Există o mulțime de moduri în care se pot clasifica operațiile aplicate unei imagini. Tipurile de operații care pot fi aplicate pe o imagine digitală pentru a transforma imaginea inițială $a[m,n]$ în imaginea finală $b[m,n]$ pot fi împărțite în cele trei categorii prezentate în continuare:

- Operații punctuale, unde valoarea finală a unui pixel de coordonate cunoscute depinde doar de valoarea inițială a pixelului aflat la aceleași coordonate.
- Operații pe vecinătate, unde valoarea finală a unui pixel de coordonate cunoscute depinde de valorile inițiale ale pixelilor aflați în vecinătatea pixelului ce corespunde acelorași coordonate.
- Operații globale, unde valoarea finală a unui pixel de coordonate cunoscute depinde de valorile tuturor pixelilor din imaginea inițială [14].



Operație punctuală



Operație pe vecinătate



Operație globală

Figura 30: Tipurile de operații pe imagini.

Pixelii aflați în apropierea unui pixel anume, poartă numele de vecinătate a pixelului. O vecinătate poate fi caracterizată după formă în același fel ca o matrice : există vecinătăți de 3 x 3 pixeli, 5 x 7 pixeli. Cu excepția unor circumstanțe speciale, vecinătățile au un număr diferit de linii și de coloane, acest lucru asigură faptul că pixelul curent se află în centru vecinătății. Dacă o vecinătate are numărul de linii egal cu numărul de coloane, este necesar să se specifice poziția pixelului curent. Figura 31 este un exemplu de vecinătate.

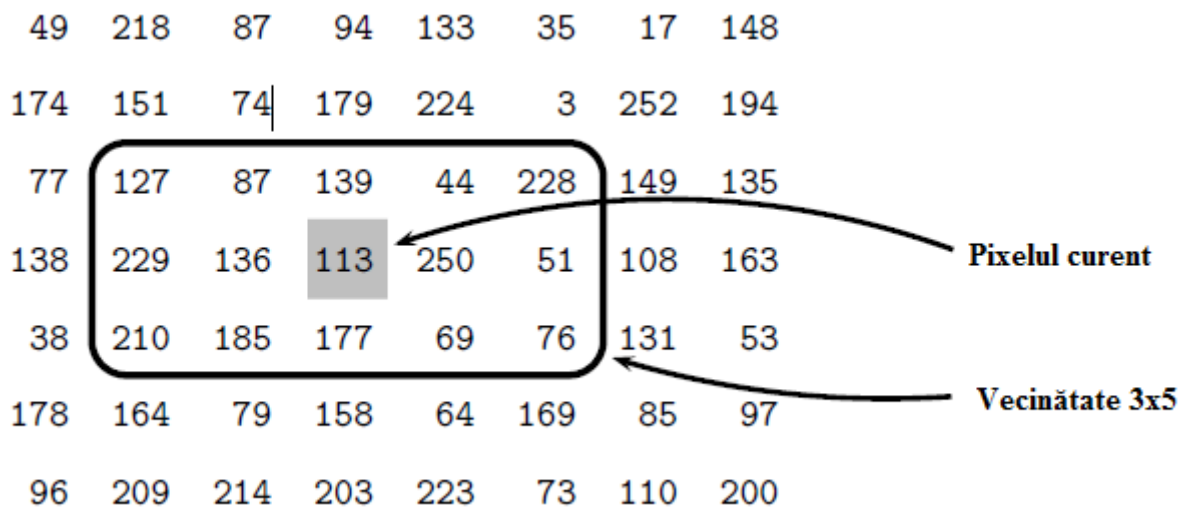


Figura 31: Exemplu de vecinătate.

Tipuri de vecinătăți existente:

Operațiile pe vecinătate au un rol important în procesarea modernă de imagini digitale. De aceea este important de înțeles cum imaginea poate fi eșantionată și cum acestea interacționează cu tipurile de vecinătăți care sunt folosite pentru procesarea imaginilor.

Eșantionare dreptunghiulară – În cele mai multe cazuri, eșantionarea se realizează prin așezarea unei ferestre dreptunghiulare peste imagine, prezentată în figurile 4a și 4b.

Eșantionare hexagonală – Reprezintă o metodă alternativă celei dreptunghiulare, prezentă în figura 4c.

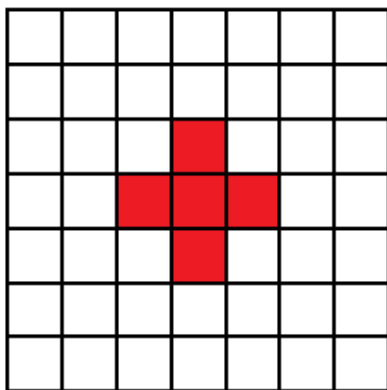


Figura 32:

Eșantionare dreptunghiulară
Dimensiune 4

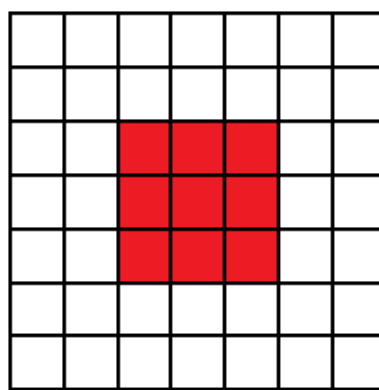


Figura 33:

Eșantionare dreptunghiulară
Dimensiune 8

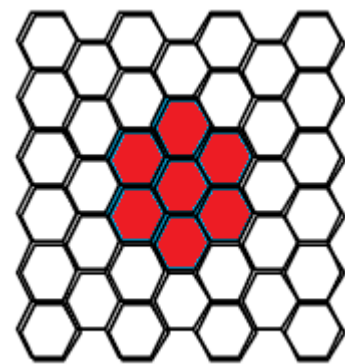


Figura 34:

Eșantionare hexagonală
Dimensiune 6

4.3 ASPECTE ALE PROCESĂRII DE IMAGINI

Procesarea de imagini are o arie de utilizare foarte mare, aproape toată zona științifică și tehnologică pot folosi metodele de procesare de imagini. În continuare este prezentată o listă a domeniilor în care se poate aplica procesarea de imagini:

1. Domeniul medical
 - Vizualizare și interpretarea imaginilor obținute de la raze-X sau MRI.
 - Analiza celulelor pe baza imaginilor obținute.
2. Domeniul agricol
 - Analiza terenului agricol cu ajutorul imaginilor obținute de diferite echipamente de zbor (drone) sau a celor de la satelit. Acestea oferă o imagine de ansamblu referitoare la porțiunea respectivă de teren cultivată, sau pentru a investiga gradul de compatibilitate a unei zone cu o anumită cultură.
 - Inspecția calității fructelor și legumelor.
3. Domeniul industrial
4. Domeniul juridic
 - Analiza amprentelor
 - Îmbunătățirea imaginilor obținute cu camera de înregistrare a vitezei.

Există de asemenea o serie de algoritmi utilizați la îndeplinirea cerințelor și rezolvarea problemelor, care se doresc clasificați după natura acestora.

1. Îmbunătățirea imaginilor. Aceasta se referă la procesarea unei imagini astfel încât rezultatul să fie mai potrivit pentru o anumită aplicație:
 - a. Eliminarea blur-ului dintr-o imagine fără focalizare,
 - b. Sublinierea marginilor,
 - c. Îmbunătățirea contrastului sau a luminozității unei imagini,
 - d. Înlăturarea zgomotului.
2. Restaurarea imaginilor. Această acțiune este cunoscută și ca reparare a defectelor aduse unei imagini de o cauză cunoscută:
 - a. Eliminarea blur-ului cauzat de deplasarea liniară,
 - b. Eliminarea distorsiunilor optice,
 - c. Eliminarea interferențelor periodice.
3. Segmentarea imaginii. Aceasta implică împărțirea unei imagini în părțile sale componente sau concentrarea pe anumite aspecte din imagine:
 - a. Găsirea de linii, cercuri sau anumite forme dorite din imagine,
 - b. Identificarea de mașini, copaci, clădiri sau drumuri.

Clasele prezentate anterior nu sunt total diferite una de cealaltă, astfel că un algoritm poate fi folosit atât pentru îmbunătățirea cât și pentru restaurarea imaginilor [16].

În continuare vor fi prezentate tipurile de imagini digitale existente:

Imaginea binară, este un tip de imagine digitală în care fiecare pixel poate fi alb sau negru. Deoarece există doar două valori posibile ce pot fi atribuite pixelilor, este nevoie de un singur bit pe pixel. Acest tip de imagine are o eficiență bună din punctul de vedere al stocării de date. Imaginile pentru care reprezentarea binară este utilizată, sunt imagini ce conțin text, amprente sau planuri. Un exemplu de imagine binară este prezentat în figura 5, în care porțiunile de contur sunt reprezentate cu alb în timp ce fundalul este reprezentat cu negru.

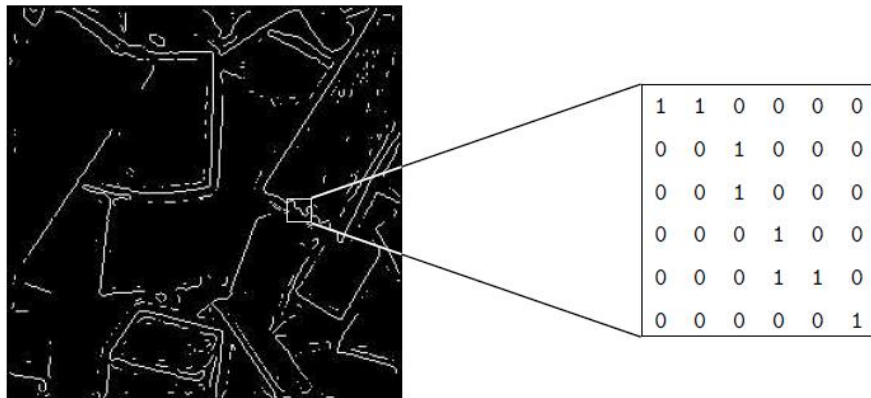


Figura 35: Imagine binară.

Imagine cu nivele de gri, în care fiecare pixel corespunde unei nuanțe de gri, de la 0 (negru) până la 255 (alb). Aceasta distanță de la 0 la 255 semnifică faptul că fiecare pixel poate fi reprezentat pe 8 biți.

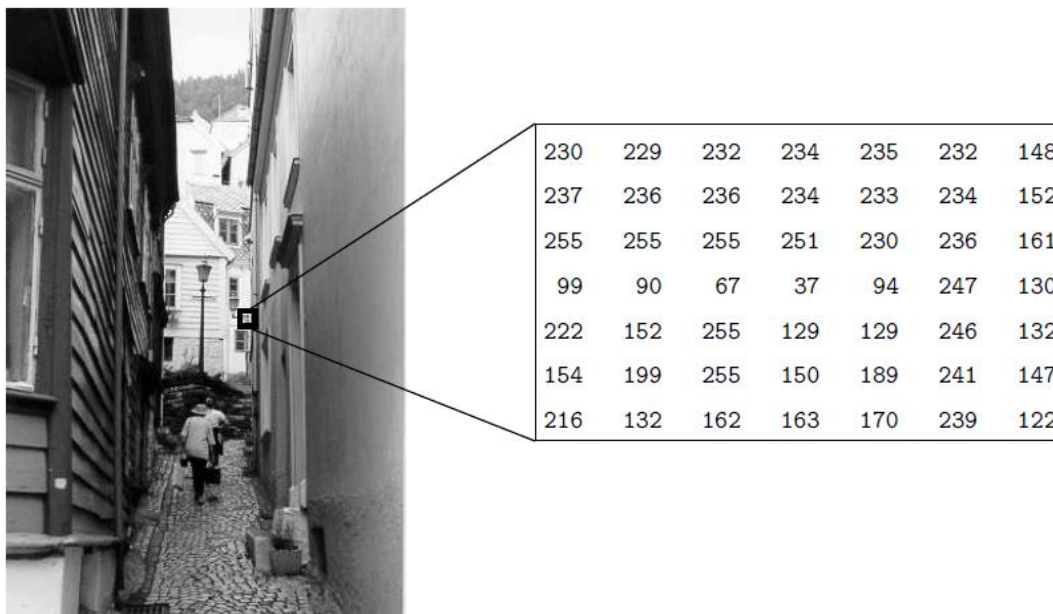


Figura 36: Imagine cu nivele de gri.

Imagini RGB, sunt imagini în care fiecare pixel reprezintă o anumită culoare, culoare care este descrisă prin cantitatea de roșu, verde și albastru de care dispune. Dacă fiecare dintre cele trei componente folosite poate lua valori între 0-255, se obține un total de $255 \times 255 \times 255 = 16.777.216$ culori diferite posibile în imagine. Deoarece numărul total de biți necesari fiecărui pixel este 24, imaginile RGB se mai numesc și „24-bit colour images”. Acest tip de imagine are la bază trei matrice, reprezentând valorile de roșu, verde și albastru pentru fiecare pixel din imagine. Ceea ce înseamnă că fiecărui pixel din imagine îi corespund trei valori.

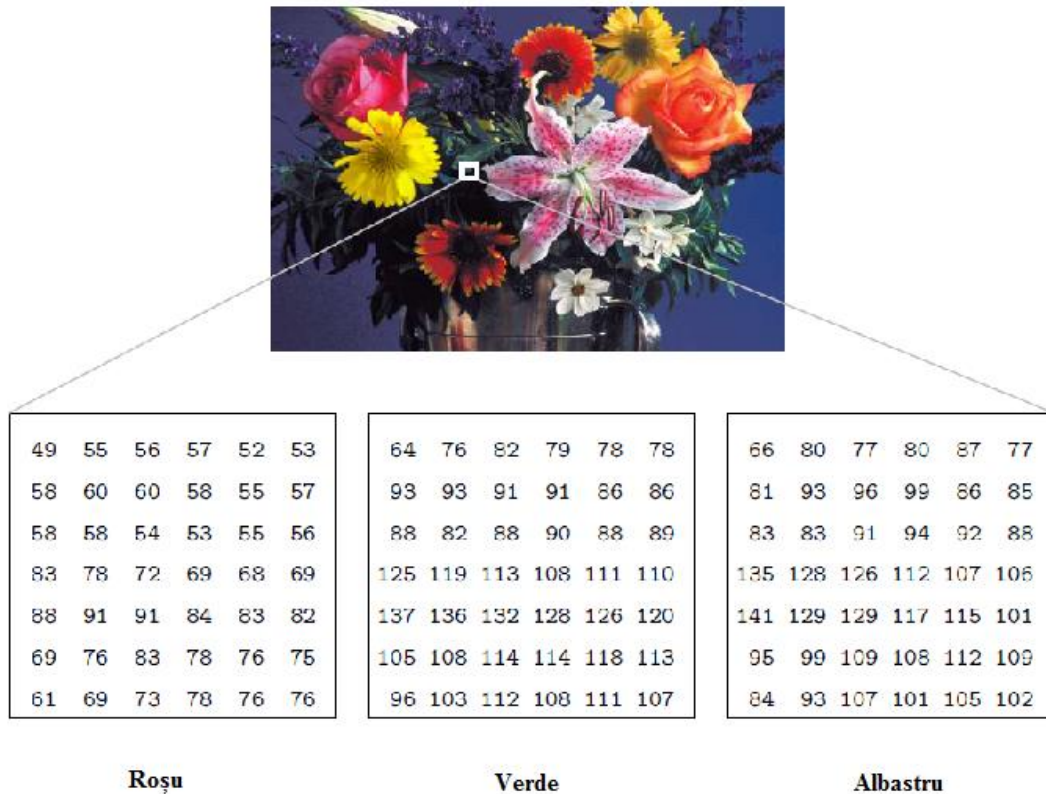


Figura 37: Imagine RGB

Imagini indexate. Cele mai multe imagini color, din motive de stocare sau de manipulare a fișierelor, au asociată o hartă de culoare, care reprezintă o listă a tuturor culorilor utilizate în imagine. Fiecărui pixel îi este dată o valoare, valoare care nu reprezintă culoare propriu-zisă a pixelului respectiv ci un index către culoarea aflată în harta de culori. În figura 38 este prezentă o imagine indexată, în care indicii în loc să fie valori de gri pentru pixeli, sunt indici în harta de culoare [16].

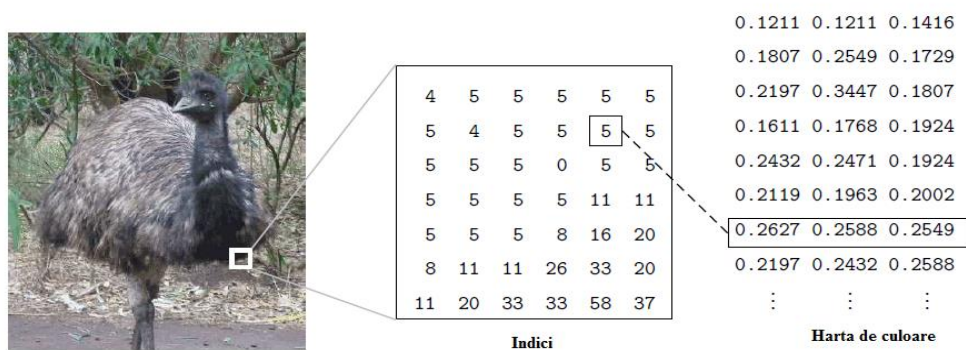


Figura 38: Imagine color indexată.

Misiune de explorare a unei zone cu un robot autonom.

CAPITOLUL 5

REALIZAREA DEPLASĂRII AUTONOME

Atât oamenii cât și roboții manipulează obiectele din jurul lor pentru a-și atinge scopurile. Oamenii își folosesc simțurile (văzul, auzul) pentru a lua contact cu mediul înconjurător. Similar oamenilor, roboții obțin informații despre împrejurimi, folosindu-se de senzori. Informația este procesată în „ creierul ” robotului, care este reprezentat de un procesor, rezultatul fiind trimiterea de semnale către motoare care vor deplasa robotul.

5.1 DESCRIEREA GENERALĂ A SISTEMULUI

Sistemul utilizat pentru realizarea proiectului de diploma „Misiune de explorare a unei zone cu un robot autonom” este format din robotul PhantomX Mark II, microcalculatorul Raspberry Pi 3 și senzorul Kinect v1, sistem prezentat în figura 39.

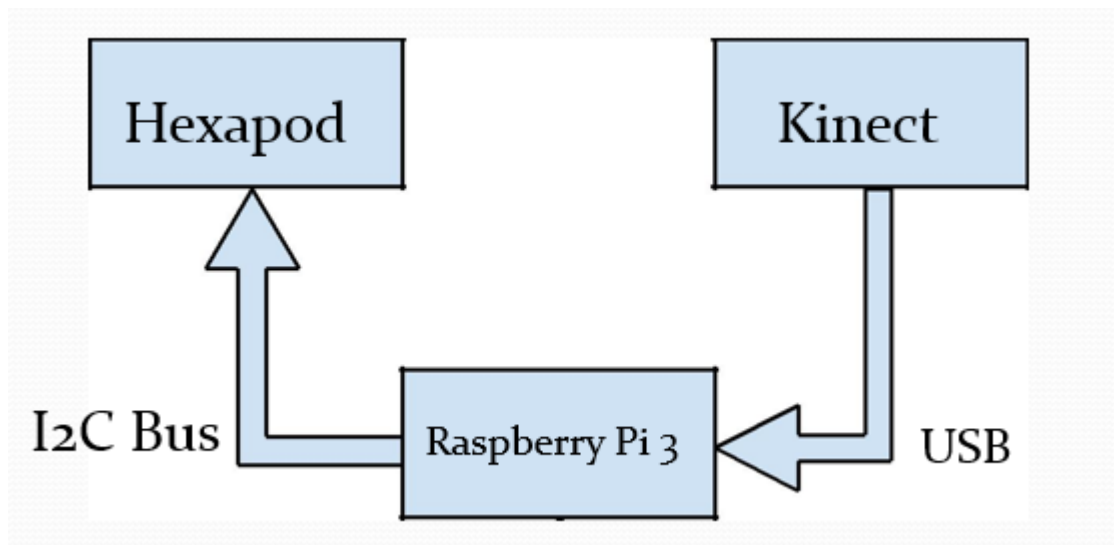


Figura 39 : Ansamblul sistemului.

Etapile prin care am trecut pentru realizarea acestui proiect vor fi prezentate în continuare, după care voi continua prin explicarea fiecărei etape în parte:

1. O primă etapă în proiect a fost parcurgerea și înțelegerea codului sursă care rulează pe robocontrollerul ArbotiX-M și a codului sursă care rulează pe commanderul ArbotiX, pentru a înțelege cum este controlat robotul prin intermediul telecomenzii.
2. După ce am înțeles cum este realizat controlul robotului, am rescris codul sursă al robotului eliminând porțiunile de cod care erau folosite în programul inițial pentru controlul prin telecomandă și am folosit informațiile de la un senzor de distanță Sharp pentru a detecta obiectele care apar în fața robotului.
3. După ce etapa a doua a fost completă, am constatat că apar probleme la detectarea și evitarea obiectelor deoarece senzorul Sharp emite un fascicul infraroșu cu o zonă de acțiune foarte îngustă astfel că dacă fasciculul nu este îndreptat exact către obiect, acesta trece nedetectat și robotul consideră că în fața lui nu se află un obiect.
4. Pentru a rezolva problema întâlnită la pasul 3 am hotărât să folosesc un senzor Kinect care are o emisie a fasciculului infraroșu mult mai extinsă, având 43 de grade deschidere verticală și 57 de grade deschidere orizontală.
5. Următoarea etapă a fost să instalez driverele necesare pentru funcționare senzorului Kinect pe microcalculatorul Raspberry Pi 3, microcalculator care se va ocupa de procesarea imaginilor primite de senzorul Kinect.
6. Ultima etapă în proiect este realizarea conexiunii între Raspberry Pi și placa de dezvoltare ArbotiX-M și astfel robotul primește informații de la Kinect cu privire la obiectele care se află în fața lui prin intermediul Raspberry Pi și așa este realizată deplasarea autonomă.

5.1 ANALIZA CODULUI SURSĂ AL ROBOTULUI ȘI CONECTAREA SENZORULUI SHARP

Codul sursă încărcat pe placa de dezvoltare ArbotiX-M în varianta lui inițială este format din patru fișiere.

- Un fișier header, *nuke.h*, folosit pentru algoritmul IK (Inverse Kinematics) și pentru algoritmul de deplasare.
- Un fișier C++, *nuke.cpp*, care conține algoritmul IK.
- Un fișier *gaits.h* care conține algoritmul de deplasare, mai exact tipurile de deplasare de care dispune robotul.
- Un fișier care conține codul sursă propriu, care poartă numele de, *Hexapod_Mark_II*.

Fișierul *nuke.cpp* conține următoarele funcții:

- Funcția *setupIK()* are scopul de a stabili pozițiile inițiale ale fiecărui picior în parte
- Funcția *radToServo(float rads)* transformă radianii în poziții de offset pentru motoare
- Funcția *bodyIK* în care se calculează unde trebuie să fie poziționat fiecare picior.
- Funcția *doIK()* calculează poziția fiecărui picior luând în considerare modul de deplasare.

Fișierul *Hexapod_Mark_II* conține funcțiile *setup()* și *loop()*.

Funcția *setup()* are în componența sa o secvență de cod care verifică dacă tensiunea din baterie este peste pragul de 10V, scăderea sub acest prag face ca robotul să nu mai funcționeze.

```
float voltage = (ax12GetRegister (1, AX_PRESENT_VOLTAGE, 1)) / 10.0;
```

```
if (voltage < 10.0)
```

```
while(1);
```

O altă secvență de cod din funcția *setup()* este aceea de inițializare a motoarele și de poziționare a robotului într-o stare de așteptare.

```
bioloid.poseSize = 18;
```

```
bioloid.readPose();
```

```
doIK();
```

```
bioloid.interpolateSetup(1000);
```

```
while(bioloid.interpolating > 0){
```

```
    bioloid.interpolateStep();
```

```
    delay(3);
```

```
}
```

În funcția *loop()* are loc întreg procesul de deplasare al robotului.

Primirea comenzilor de la controler se realizează prin:

```
if(command.ReadMsgs() > 0){
```

```
    digitalWrite(0,HIGH-digitalRead(0));
```

Dacă datele sunt trimise de către commander se intră în etapa de selecție a modului de deplasare prin apăsarea unuia din cele 6 butoane de pe telecomandă, figura 18. Odată ce unul din cele 6 butoane a fost apăsat se selectează tipul de deplasare și viteza specifică.

```
if(command.buttons&BUT_R1){
    gaitSelect(RIPPLE_SMOOTH);
    multiplier=RIPPLE_SPEED;    //variabila multiplier reprezintă valoare vitezei robotului
                                în modul RIPPLE_SPEED.
}
```

După ce modul de deplasare a fost setat, se trece la deplasarea propriu-zisă a robotului pe baza valorilor de la cele două joystickuri prezentate în figura 18. De exemplu, dacă joystickul walk este mișcat înainte, variabila *command.walkV* primește o valoare ce nu aparține intervalului [-5, 5], astfel că robotul se deplasează în față cu o viteză calculată pe baza vitezei standard a modului de deplasare ales și a valorii pe care o primește *command.walkV*.

```
if((command.walkV) > 5 || (command.walkV < -5) ){
    Xspeed = (multiplier*command.walkV)/2;
}
else
{
    Xspeed = 0;
}
```

Același mod de funcționare este și în cazul în care vrem să deplăm robotul în stânga sau în dreapta fără a-i modifica orientarea sau în cazul în care dorim ca robotul să se rotească până la o poziție dorită. Acest lucru se realizează prin înlocuirea lui *command.walkV* cu *command.walkH* și a lui *Xspeed* cu *Yspeed* pentru deplasarea stânga/dreapta și înlocuirea lui *command.walkV* cu *command.lookH* și a lui *Xspeed* cu *Rspeed* pentru rotirea corpului robotului.

Xspeed – viteza cu care robotul se deplasează înainte, dacă *Xspeed* > 0, sau viteza cu care robotul se deplasează înapoi, dacă *Xspeed* < 0. Se măsoară în milimetri/secundă.

Yspeed – viteza cu care robotul se deplasează în dreapta, dacă *Yspeed* > 0, sau viteza cu care robotul se deplasează înapoi, dacă *Yspeed* < 0. Se măsoară în milimetri/secundă.

Rspeed – viteza cu care robotul se rotește. Se măsoară în radiani/secundă.

Acum că am înțeles cum funcționează robotul voi elimina părțile de cod prin care se realizează transferul de date de la commander la controlerul robotului și voi conecta la placa de dezvoltare ArbotiX-M senzorul de distanță Sharp.

Schema electrică de conectare a senzorului de distanță Sharp la placa ArbotiX-M este prezentată în figura 40, în care conexiunea este realizată prin trei conectori, un conector pentru alimentarea senzorului, un conector de ground și un conector prin care sunt trimise datele de la senzor la ArbotiX-M.

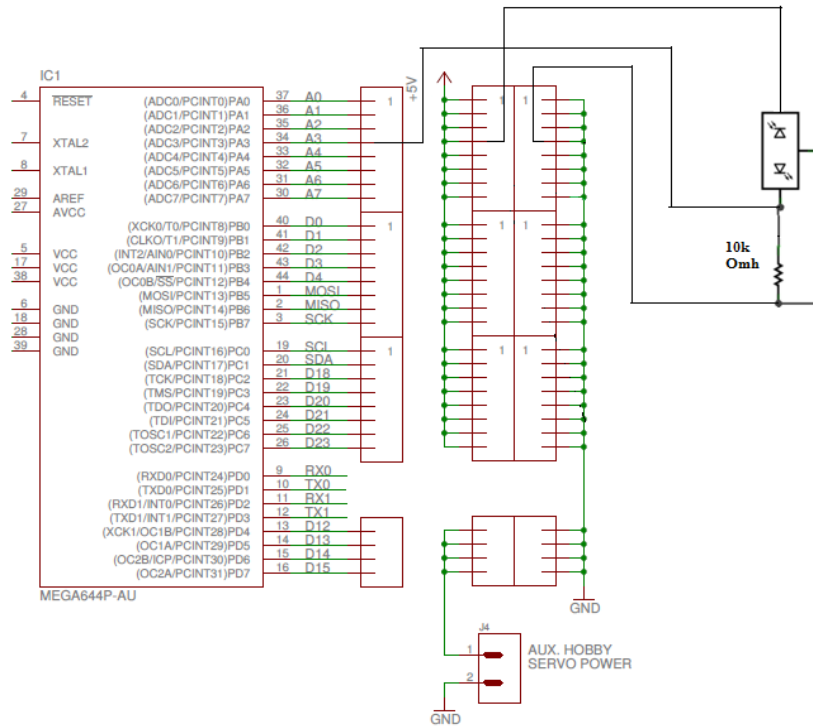


Figura 40 : Schema electrică ArbotiX-M – Sharp.

Pentru integrarea senzorului Sharp în codul robotului este nevoie de folosirea header-ului SharpIR.h și preluarea informațiilor cu privire la distanțele înregistrate. Porțiunea de cod care asigură deplasarea pe baza senzorului Sharp este prezentată în continuare, iar modul de funcționare este explicat în diagrama UML din figura 41.

```

int walkv=30;
int walkh=20;
int lookh=10;
multiplier = AMBLE_SPEED;
int dist=sharp.distance();
if (dist >= 30){
    Yspeed=0;
    Rspeed=0;
    Xspeed = (multiplier*walkv)/2;
}
else
{
    Xspeed = 0;
    if (dist>18 and dist<30){
        Xspeed=0;
        Yspeed=0;
        Rspeed = -lookh/100.0;
    }
}

```

```
else
{
  Xspeed=0;
  Rspeed=0;
  Yspeed = (multiplier*walkh)/2;
}
}
```

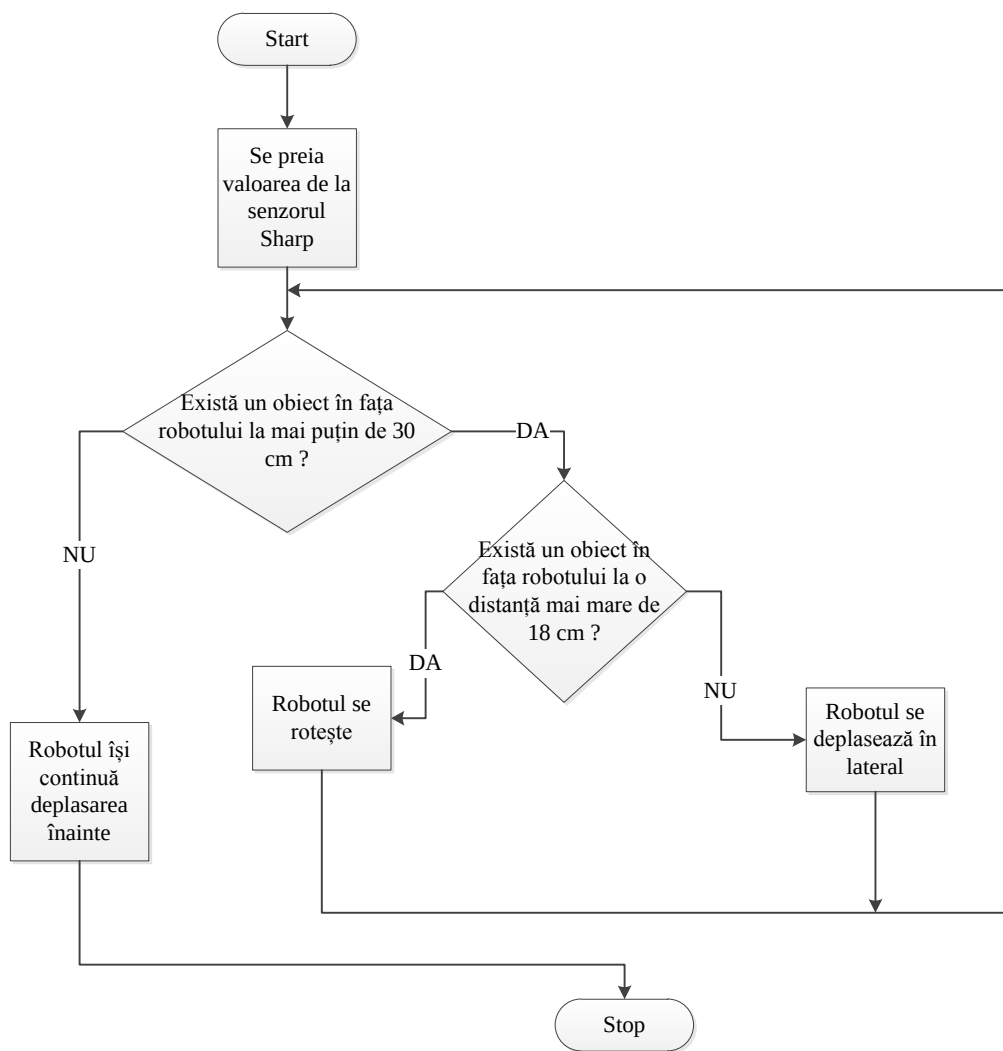


Figura 41 : Diagramă UML a modului de funcționare, cu senzorul Sharp.

5.2 REALIZAREA CONEXIUNII DINTRE RASPBERRY PI ȘI SENZORUL KINECT

Pentru a instala driverele și librăriile necesare conectării senzorului Kinect la Raspberry Pi sunt necesari următorii pași:

- Instalarea dependențelor de bază

```
sudo apt-get install automake build-essential cmake couchdb freeglut3-dev g++ git-core libboost-all-dev libcurl4-gnutls-dev libopencv-dev libusb-1.0-0-dev libxi-dev libxmu-dev pkg-config libudev-dev libxtst-dev
```

```
sudo apt-get install libxtst-dev
```

- Instalarea manuală pentru libusb

```
tar jxf libusb-1.0.19.tar.bz2
```

```
cd libusb-1.0.19
```

```
./configure
```

```
make
```

```
sudo make install
```

- Instalarea libfreenect, dar înainte de a instala libfreenect trebuie șterse versiunii anterioare pentru a evita eventualele erori.

```
sudo apt-get remove freenect
```

```
sudo apt-get remove libfreenect
```

```
sudo rm -rf /usr/local/include/libfreenect/
```

```
sudo rm -rf /usr/local/lib/libfreenect
```

```
git clone git://github.com/OpenKinect/libfreenect.git
```

```
cd libfreenect
```

```
mkdir build
```

```
cd build
```

```
#pentru a avea și suport audio trebuie realizat pasul următor
```

```
cmake -L .. -DBUILD_REDIST_PACKAGE=OFF
```

```
make
```

```
#instalarea propriu-zisă a libfreenect
```

```
sudo make install
```

```
sudo ldconfig /usr/local/lib64/
```

```
sudo ldconfig
```

```
mkdir $HOME/.libfreenect
```

```
cp audios.bin $HOME/.libfreenect/
```

```
sudo freenect-glfwview
```

```
sudo freenect-micview
```

5.3 ANALIZA IMAGINII OBTINUTĂ DE SENZORUL KINECT

5.3.1 Caracteristicile imaginii

Salvarea imaginii de la senzorul Kinect a fost realizată prin:

```
image= cv.CreateImageHeader((depth.shape[1], depth.shape[0]),cv.IPL_DEPTH_8U,1)
cv.SetData(image, depth.tostring(),depth.dtype.itemsize * depth.shape[1])
cv.SaveImage("img_kinect.jpg",image)
```

Imaginea care se prelucrează pe Raspberry Pi este imaginea de adâncime obținută cu ajutorul senzorului Kinect. Imaginea inițială are dimensiunea 640 x 480 pixeli însă din cauză resurselor limitate de care dispune Raspberry Pi a fost nevoie să îi micșorez dimensiunea la 100 x 100 pixeli păstrând ceea ce există în imagine neschimbat.

```
gray_img1 = Image.open('img_kinect.jpg').convert("L")
new_width=100
new_height=100
gray_img1=gray_img1.resize((new_width,new_height), Image.ANTIALIAS)
gray_img1.save('img_kinect1.jpg')
```



Figura 42 : Rezoluție 640x480



Figura 43 : Rezoluție 100x100

După obținerea imaginii dorite am aplicat un filtru median folosind funcția din opencv `cv2.medianBlur('imagine', dimVecinătate)`, care calculează media pixelilor aflați într-o vecinătate aleasă și înlocuiește valoarea pixelului curent cu valoarea mediei vecinătății. Acest tip de filtru este foarte util împotriva zgomotului de tip sare și piper din imagini, realizând o reducere semnificativă a zgomotului după cum se poate vedea și în imaginea 44.



Imaginea originală



Imagine după aplicarea
filtrului median.

Figura 44 : Reducerea zgomotului din imagine

Imaginea finală este scrisă sub forma unei matrice de dimensiunea imaginii (în cazul meu matricea are 100 de linii și 100 de coloane) în funcție de care se face detecția de obiecte.

5.3.2 Distanța față de obiecte.

Pentru obținerea distanței reale până la obiecte am realizat o conversie a valorilor din matricea nivelelor de gri în distanțele corespunzătoare în milimetrii. Pentru acest lucru m-am folosit de distanțele minimă și maximă de care dispune senzorul kinect (800 mm – 4000 mm) și pe baza unei formule matematice am transformat valorile pixelilor din imaginea de adâncime în distanțele respective în milimetrii.

$$\text{ValMM} = (((\text{ValPix} - \text{MinPix}) * (\text{MaxMM} - \text{MinMM})) / (\text{MaxPix} - \text{MinPix})) + \text{MinMM}.$$

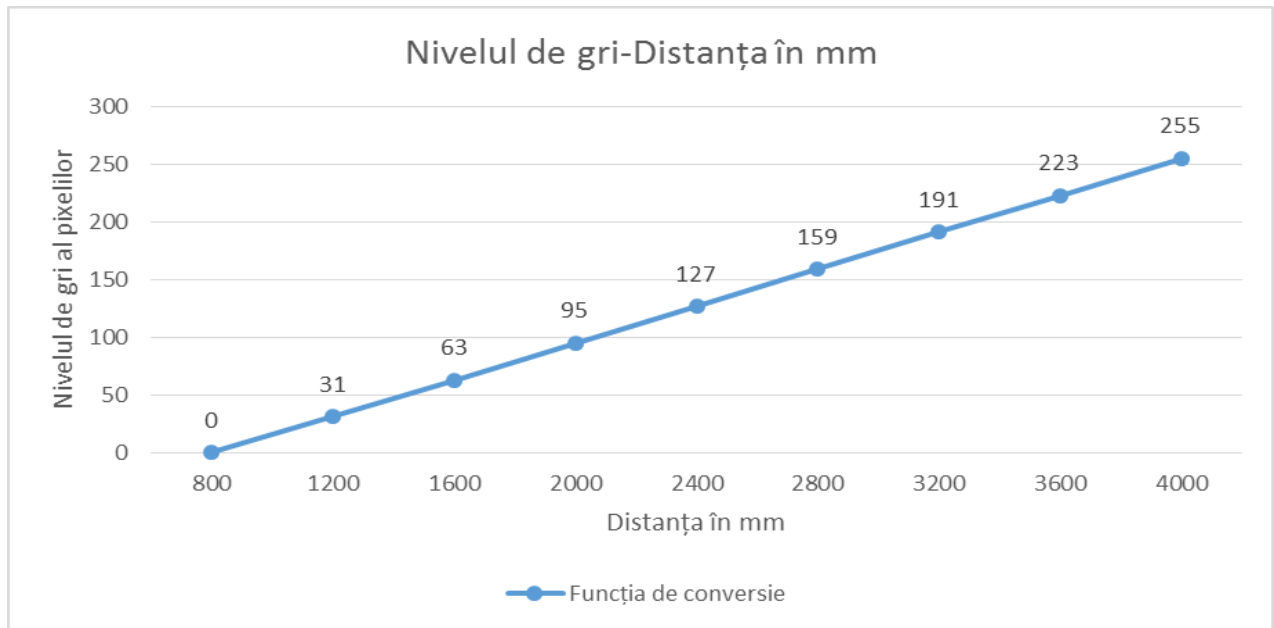


Figura 45 : Funcția liniară de conversie, nivele de gri – milimetrii.

5.4 TRIMITEREA DATELOR DE LA RASPBERRY PI LA ARBOTIX-M

După ce au fost realizate etapele necesare pentru luarea de decizii, urmează trimiterea datelor de la Raspberry Pi la ArbotiX-M prin protocolul I2C despre care am discutat în capitolul 3.

În mod normal pentru a folosi protocolul I2C între microcalculatorul Raspberry Pi, care funcționează la 3.3 V și placa de dezvoltare ArbotiX-M care funcționează la 5 V este indicată folosirea unui convertor de tensiune. În cazul proiectului meu, acest lucru nu este necesar deoarece Raspberry Pi este folosit ca „master” și ArbotiX-M este folosit ca „slave”. Motivul pentru care acest lucru este posibil se datorează faptului că ArbotiX-M nu are rezistori de pull – up însă header-ul P1 de pe Raspberry Pi are rezistori de pull-up de 1.8k ohm pentru linia de tensiune de 3.3V.

Setările necesare pentru comunicația I2C pe Raspberry Pi:

- Se elimină I2C din lista neagră a microcalculatorului

```
$ cat /etc/modprobe.d/raspi-blacklist.conf
```

```
#blacklist i2c-bcm2708
```

- Se adaugă i2c în Module File

```
$ cat /etc/modules
```

```
i2c-dev
```

- Se instalează I2C Tools

```
$ sudo apt-get install i2c-tools
```

- Se permite accesul la dispozitivele I2C pentru utilizator

```
$ sudo adduser pi i2c
```

- Se instalează Python-SMBus care oferă suport I2C pentru python

```
$ sudo apt-get install python-smbus
```

ArbotiX-M ca dispozitiv Slave pentru I2C

Pentru a transforma ArbotiX-M într-un „ slave ” trebuie stabilită o adresă pe care să trimitem datele și o funcție „callback” pentru primirea datelor. Astfel în codul sursă care se află pe robot adăugăm librăria `wire.h` (`#include <Wire.h>`) și definim adresa de „ slave ” (`#define SLAVE_ADDRESS 0x04`).

În funcția `setup()` se inițializează I2C ca slave (`Wire.begin(SLAVE_ADDRESS)`) și se definește funcția callback pentru comunicația I2C (`Wire.onReceive(receiveData)`).

În funcția `loop()` este apelată funcția `receiveData` prin care se primesc datele trimise de la Raspberry Pi prin protocolul I2C.

```
void receiveData(int byteCount){
    while(Wire.available()){
        dist=Wire.read();
    }
}
```

Raspberry Pi ca dispozitiv Master

Ca și la ArbotiX-M trebuie aleasă adresa pe care se trimit datele, (`adresa=0x04`) și este folosită librăria SMBus (`#include smbus`), care permite realizarea conexiunii între cele două dispozitive prin I2C. Funcția prin care se realizează trimiterea datelor către ArbotiX-M este afișată în continuare.

```
def writeNumber(value):
    bus.write_byte(adresa,value)
    return -1
```

5.5 MODUL DE FUNCȚIONARE AL SISTEMULUI

În forma lui finală proiect este format din kit-ul de dezvoltare PhantomX Mark II la care este conectat prin comunicație I2C microcalculatorul Raspberry Pi 3. La Raspberry Pi se conectează prin USB senzorul Kinect configurație prezentată în figura 39. Odată pornit robotul, se rulează scriptul în python de pe Raspberry Pi, script prin care imaginea de adâncime a senzorului Kinect este scrisă sub forma unei matrice de 640 x 480 pixeli pe care am transformat-o într-o matrice de 100 x 100 pixeli pentru a micșora timpul de procesare a datelor de către Raspberry Pi. Odată micșorată imaginea putem trece la modul de deplasare al robotului pe baza informațiilor din matrice. Din matricea de 100 x 100 pixeli obținută se parcurg doar ultimele 5 linii, din cauza resurselor limitate oferite de microcalculatorul Raspberry Pi, astfel că cele 5 linii sunt împărțite în trei zone: zona din stânga care cuprinde coloanele (0, 34), zona centrală care cuprinde coloanele (35, 65) și zona din dreapta imaginii care cuprinde coloanele (66, 100). După ce

sistemul a pornit, cele trei zone din matrice sunt verificate pentru a stabili direcția de deplasare a robotului. Dacă distanțele din zona centrală sunt cuprinse în intervalul (800mm – 2000mm), înseamnă că în fața robotului se află un obiect și se trece la calculul mediilor zonelor laterale pentru a determina în ce parte să înceapă robotul rotirea. Zona cu media valorilor cea mai mare este zona în care obiectele se află la o distanță suficient de mare de robot pentru ca acesta să se poată deplasa, astfel că robotul se deplasează spre zona cu media valorilor mai mare. După ce obiectul care se află inițial în zona centrală a ieșit complet din imagine, ceea ce înseamnă că se află la o distanță mai mare de 3000mm, robotul își poate continua deplasarea înainte, rutina de deplasare repetându-se la nesfârșit atâta timp cât robotul este alimentat.

Modul de funcționare al sistemului este prezentat în diagrama UML din figura 45.

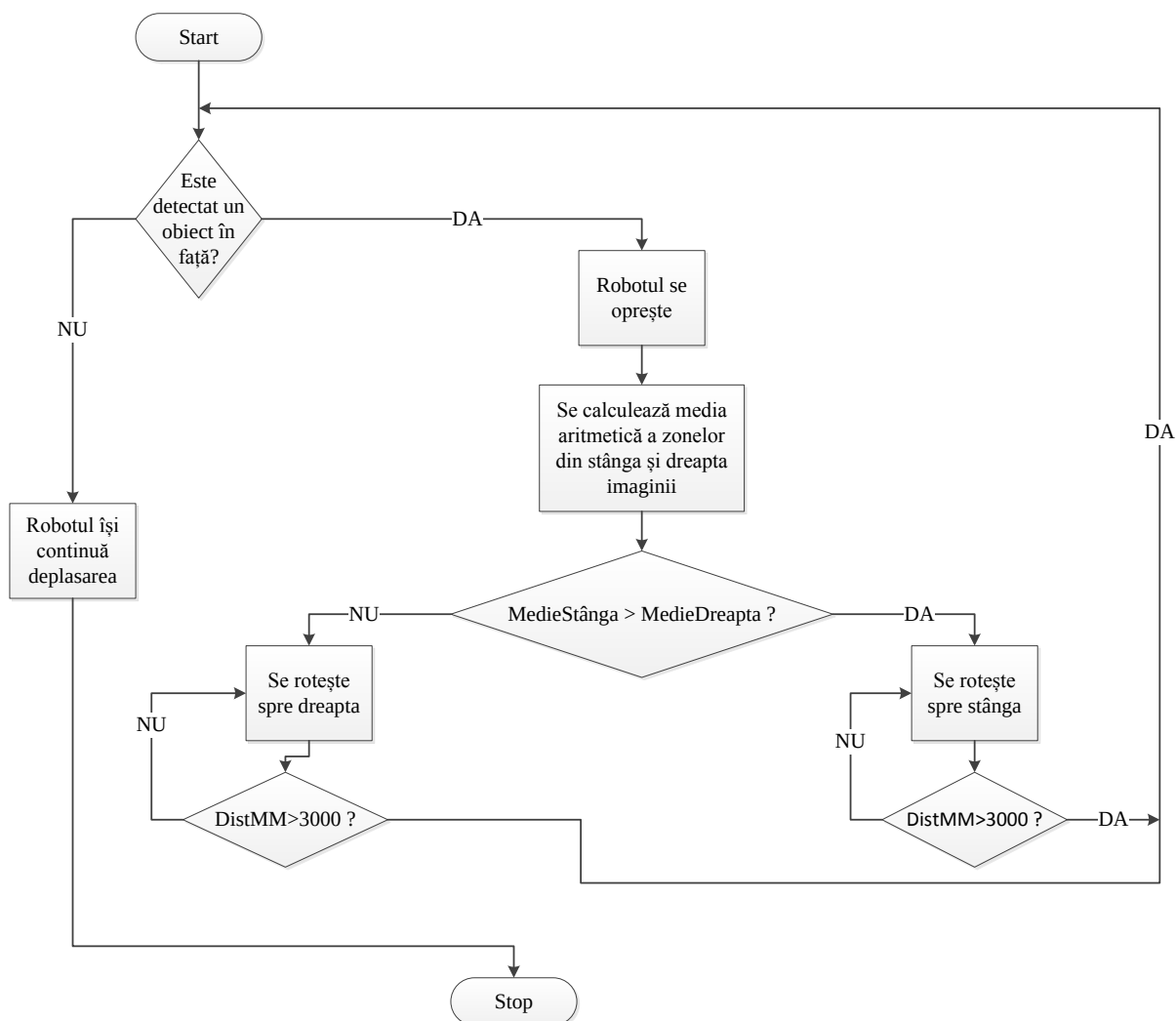


Figura 46 : Modul de funcționare al robotului.

5.6 LIMITĂRI

Limitările pe care le-am întâlnit în realizarea proiectului au fost de tip hardware.

Limitări în utilizarea microcalculatorului Raspberry Pi 3 la parcurgerea matricei imaginii obținută de senzorul Kinect. La o parcurgere a matricei de 640 x 480 pixeli timpul de parcurgere, extragerea informațiilor necesare și trecerea la următorul frame variază de la 5 până la 10 secunde în timp ce dacă este parcursă matricea transformată de 100 x 100 pixeli timpul se reduce până la 1-2 secunde. Astfel că pentru un algoritm complet în care este parcursă întreaga imagine și deciziile se bazează pe informațiile obținute din întreaga imagine este nevoie de un microcalculator mai performant ca Raspberry Pi 3.

Limitările legate de senzorul Kinect se referă la distanța minimă de la care pot fi detectate obiectele aflate în fața senzorului, problemă întâlnită și la folosirea senzorului IR Sharp. Distanța minimă la senzorul Kinect este de 0.8m astfel, dacă un obiect se află în interiorul zonei 0 – 0.8m senzorul Kinect îl consideră fundal și îl reprezintă cu alb. La fel ca și la senzorul Sharp o posibilă rezolvare a acestei problem este așezarea senzorului Kinect pe robot în așa fel încât corpul robotului să depășească distanța de 0.8m în care detecția senzorului este nulă.

CAPITOLUL 6

6.1 CONCLUZII GENERALE

În cadrul lucrării de diplomă prezentată, a fost studiat modul de realizare a deplasării autonome pentru robotul terestru PhantomX Mark II bazat pe informațiile din imaginea de adâncime furnizată de senzorul Kinect. În cadrul proiectului au fost utilizate placa de dezvoltare ArbotiX-M care asigură controlul motoarelor, microcalculatorul Raspberry Pi 3 utilizat la preluarea și procesarea imaginii de adâncime a senzorului kinect și la transmiterea rezultatelor la ArbotiX-M utilizând protocolul I2C și senzorul Kinect care reprezintă modul prin care robotul ia contact cu mediul exterior.

Obiectivul final al proiectului a fost atins, robotul fiind capabil de deplasare autonomă în interiorul unei camere având capacitatea, pe baza informațiilor obținute de la senzorul Kinect să detecteze și să evite obstacolele întâlnite în calea sa.

6.2 CONTRIBUȚII PERSONALE

Contribuțiile personale ce au fost aduse acestei lucrări sunt prezentate în Capitolul 5 și sunt enumerate după cum urmează:

- Stabilirea conexiunilor dintre dispozitivele prezentate, care au dus la obținerea rezultatului dorit.
- Preluarea stream-ului de adâncime de la senzorul Kinect și realizarea prelucrărilor necesare care au dus la determinarea distanței față de obiectele aflate în jurul robotului.

- Transmiterea datelor de la Raspberry Pi la placa de dezvoltare ArbotiX-M prin protocolul I2C.
- Realizarea algoritmului de luarea a deciziilor pentru evitarea obiectelor întâlnite.
- Modificarea codului sursă al robotului pentru a fi controlat din comenzile primite de la Raspberry Pi.

6.3 ETAPE VIITOARE

O serie de aplicații viitoare ce pot fi realizate utilizând ca punct de plecare ansamblul dispozitivelor prezentat în această lucrare sunt :

- Realizarea unui mod propriu de deplasarea a robotului
- Îmbunătățirea deplasării autonome a robotului prin realizarea unui algoritm de o complexitate mai mare față de cel folosit.
- Utilizarea senzorului Kinect pentru generarea unei hărți a camerei pe baza unui algoritm SLAM.

REFERINȚE

- [1] Kitul de dezvoltare PhantomX Mark II:
<http://www.trossenrobotics.com/phantomx-ax-hexapod.aspx>
- [2] Hardware ArbotiX-M:
<http://learn.trossenrobotics.com/arbotix/arbotix-getting-started/38-arbotix-m-hardware-overview.html#&panel1-12>
- [3] Motorul Dynamixel AX-12A:
http://support.robotis.com/en/product/dynamixel/ax_series/dxl_ax_actuator.htm
- [4] Arduino:
<http://www.instructables.com/id/Intro-to-Arduino/?ALLSTEPS>
- [5] ArbotiX Robocontroller:
<http://www.generationrobots.com/media/ArbotiX-M/arbotix-manual-draft2.pdf>
- [6] ArbotiX Commander:
<http://vanadiumlabs.com/docs/commander-manual.pdf>
- [7] Modulul Xbee:
http://www.digi.com/resources/documentation/Digidocs/90001456-13/Default.htm#concepts/c_introduction_to_xbee_modules.htm%3FTocPath%3D3
- [8] Senzorul Kinect:
<http://www.cs.upc.edu/~virtual/RVA/CourseSlides/Kinect.pdf>
- [9] Sistemul de operare Raspbian Jessie:
<https://www.raspberrypi.org/blog/raspbian-jessie-is-here/>
- [10] Detalii hardware despre modelul Raspberry Pi 3:
<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>
- [11] Senzorul Sharp:
<https://acroname.com/articles/sharp-infrared-ranger-comparison>
- [12] Alexander Mordvintsev & Abid K, OpenCV-Python Tutorials
<https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf>
- [13] Protocolul I2C :
<https://learn.sparkfun.com/tutorials/i2c/all.pdf>
- [14] M. Zamfir, Curs și laborator – Prelucrarea imaginilor

[15] C. Vertan, Curs și laborator – Analiza imaginilor

[16] Ian T. Young, Jan J. Gerbrands, Lucas J. van Vliet, “ Fundamentals of Image Processing”, Delft University of Technology.

ANEXE

Anexa 1: Fișierul PhantomX_Mark_II care conține codul sursă de deplasare a robotului.

```
#include <ax12.h>
#include <BioloidController.h>
#include <Commander.h>
#include "nuke.h"
#include <SharpIR.h>
#include <Wire.h>

#define SLAVE_ADDRESS 0x04
#define AX12_HEXAPOD

Commander command = Commander();
int multiplier;

#define RIPPLE_SPEED 1
#define AMBLE_SPEED 3
#define TRIPOD_SPEED 5
#ifdef AX12_HEXAPOD
#define TOP_SPEED 10
#endif
SharpIR sharp(3, 40, 93, 1080);
int walkv;
int rotate;
int serial;
int dist;
int walkh=0;
int lookh=0;
```

```
void setup(){
  pinMode(0,OUTPUT);
  // setup IK
  setupIK();
  gaitSelect(AMBLE_SMOOTH);
  // setup serial
  //Serial.begin(9600);

  // wait, then check the voltage (LiPO safety)
  delay (1000);
  float voltage = (ax12GetRegister (1, AX_PRESENT_VOLTAGE, 1)) / 10.0;
  Serial.print ("System Voltage: ");
  Serial.print (voltage);
  Serial.println (" volts.");
  if (voltage < 10.0)
    while(1);

  // stand up slowly
  bioloid.poseSize = 18;
  bioloid.readPose();
  doIK();
  bioloid.interpolateSetup(1000);
  while(bioloid.interpolating > 0){
    bioloid.interpolateStep();
    delay(3);
  }
  multiplier = AMBLE_SPEED;

  Wire.begin(SLAVE_ADDRESS);

  Serial.println("READY");
}

void receiveData(int byteCount){
  while(Wire.available()){
    dist=Wire.read();
    Serial.print("data received:");
    Serial.println(dist);
  }
}
```

```
void loop(){
  Wire.onReceive(receiveData);
  serial=dist;
  if(serial==30){
    walkv=60;
    rotate=0;
  }
  if(serial!=30){
    walkv=0;
  }
  if(serial == 100){
    rotate=-30;
  }
  if(serial == 10){
    rotate=30;
  }

  if(walkv > 5 || walkv < -5 ){
    Xspeed = (multiplier*walkv)/2;
  }
  else
  {
    Xspeed = 0;
  }
  if( rotate > 5 || rotate < -5 ){
    Rspeed = -(rotate)/100.0;
  }
  else
  {
    Rspeed = 0;
  }
  if(bioloid.interpolating == 0){
    doIK();
    bioloid.interpolateSetup(tranTime);
  }
  // update joints
  bioloid.interpolateStep();
}
```

Anexa 2 : Fișierul python de afișare a stream-ului.

```
#!/usr/bin/env python
import freenect
import cv
import frame_convert
import frame_convert

cv.NamedWindow('Depth')
print('Press ESC in window to stop')

def get_depth():
    return frame_convert.pretty_depth_cv(freenect.sync_get_depth()[0])

while 1:
    cv.ShowImage('Depth', get_depth())
    if cv.WaitKey(10) == 27:
        break
```

Anexa 3: Fișierul python de prelucrare a imaginii și trimitere a rezultatelor.

```
#!/usr/bin/env python
import numpy as np
import numpy as np
import cv2
import cv2.cv as cv
from matplotlib import pyplot as plt
import matplotlib.cm as cm
import Image
import PIL
import smbus
import time
from array import *
np.set_printoptions(threshold=np.nan)
bus=smbus.SMBus(1)
adresa=0x04
def writeNumber(value):
    bus.write_byte(adresa,value)
    return -1
```

```
def pretty_depth(depth):
    np.clip(depth, 0, 2**10 - 1, depth)
    depth >= 1
    depth = depth.astype(np.uint8)
    return depth

def pretty_depth_cv(depth):
    import cv
    import numpy as np
    depth = pretty_depth(depth)
    image = cv.CreateImageHeader((depth.shape[1], depth.shape[0]),cv.IPL_DEPTH_8U,1)
    cv.SetData(image, depth.tostring(),depth.dtype.itemsize * depth.shape[1])
    cv.SaveImage("test.jpg",image)
    gray_img = cv.imread('test.jpg', cv2.IMREAD_GRAYSCALE)
    gray_img_median = cv2.medianBlur(gray_img,15)
    cv2.imwrite('test_median.jpg',gray_img_median)
    gray_img1 = Image.open('test.jpg').convert("L")
    new_width=100
    new_height=100
    gray_img1=gray_img1.resize((new_width,new_height), Image.ANTIALIAS)
    gray_img1.save('test1.jpg')
    img = cv2.imread('test1.jpg', cv2.IMREAD_GRAYSCALE)
    img_median = cv2.medianBlur(img,9)
    img = Image.open('test1.jpg').convert("L")
    cv2.imwrite('test1_median.jpg',img_median)
    arr=np.asarray(img)
    arr1=np.asarray(img)
    np.savetxt('da.txt',arr,fmt='%f')
    global num,i,gol,stop,stopp,val,obj,obj1,turn,al,nr
    global count1,count2,count3,asa,sumL0,countL0,medL0,count4,count5,count6
    global sumL,sumR,sumF,countL,countR,countF,medL,medR,medF,corect,rotateL,rotateR,walkv,sumR0,countR0,medR0
    corect=0
    al=0
    asa=0
    walkv=0
    count1=0
    count2=0
    count3=0
    count4=0
    count5=0
    count6=0
    gol=0
    stop=0
    sumL=0
```

```
sumF=0
sumR=0
countF=0
countL=0
countR=0
medL=0
medF=0
medR=0
num=0
rotateL=0
rotateR=0
turn=0
t=np.zeros((100,100))
for y in xrange (0,100):
    for x in xrange ( 0, 100):
        t[y][x]=(((arr.item((y,x)) - OldMin)*(NewMax-NewMin))/(OldMax-OldMin))+NewMin
np.savetxt('da1.txt',t,fmt='%f')
for y in xrange(95, 100):
    for x in xrange(35, 65):
        if(t[y][x]<100):
            stop=stop+1
        if(stop<>0):
            for y in xrange(95, 100):
                for x in xrange(0,34):
                    sumL =sumL + t[y][x]
                    countL = countL + 1
                    medL=sumL/countL
            for y in xrange(95, 100):
                for x in xrange(66, 100):
                    sumR = sumR + t[y][x]
                    countR = countR + 1
                    medR=sumR/countR
            if(medR==medL):
                for y in xrange(95, 100):
                    for x in xrange(66,100):
                        if(t[y][x]<200):
                            count1=count1+1
            if(count1>0):
                rotateR=10
            if(medR>medL):
                for y in xrange(95, 100):
                    for x in xrange(66,100):
                        if(t[y][x]<200):
                            count2=count2+1
```

```

    if(count2>0):
        rotateR=10
    if((medR<medL):
        for y in xrange(95, 100):
            for x in xrange(0,34):
                if(t[y][x]<200):
                    count3=count3+1
            if(count3>0):
                rotateL=100
    if(stop==0):
        for y in xrange(85, 90):
            for x in xrange(0,34):
                sumL =sumL + t[y][x]
                countL = countL + 1
            medL=sumL/countL
        for y in xrange(85, 90):
            for x in xrange(66, 100):
                sumR = sumR + t[y][x]
                countR = countR + 1
            medR=sumR/countR
            if(medL>medR):
                nr=1
                for y in xrange(85,90):
                    for x in xrange(66,100):
                        if(t[y][x]<100):
                            count4=count4+1
                if(count4>0):
                    rotateL=100
                    walkv=0
                else:
                    walkv=30
                    rotateL=0
            else:
                nr=0
        if(nr==0):
            for y in xrange(85,90):
                for x in xrange(0,34):
                    if(t[y][x]<100):
                        count5=count5+1
            if(count5>0):
                rotateR=10
                walkv=0

```

```

        else:
            walkv=30
            rotateR=0

    if(medL==medR):
        al=15
    else:
        al=0
    if(al==15):
        turn=al
    if(rotateR==10):
        turn=rotateR
    if(rotateL==100):
        turn=rotateL
    if(walkv==30):
        turn=walkv

    val=tur
    print 'val',val
    writeNumber(val)
    time.sleep(1)
    return image
```