Razvoj upravljačkog algoritma za planiranje kretanja mobilnog robota

Ciković, Zoran

Master's thesis / Diplomski rad

2015

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:073804

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-07**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Zoran Cikoví

Zagreb, 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Bojan Jerbić, dipl. ing.

Student:

Zoran Ciković

Zagreb, 2015.

Izjavljujem da sam ovaj rad izradio samostalno, koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem se profesoru dr. sc. Bojanu Jerbiću na pruženim smjernicama u izradi ovog rada te pomoći pri radu s Pioneer 2-DX mobilnim robotom, profesoru dr. sc. Mladenu Crnekoviću na danim savjetima pri izradi vizijskog sustava i na posuđenoj opremi za rad s mobilnim robotom i Josipu Vidakoviću, mag. ing. na pomoći u radu s mobilnim robotom kao i svima ostalim članovima Katedre za projektiranje izradbenih i montažnih sustava koji su mi u bilo kojem obliku pomogli pri izradi ovog rada.

Također se želim zahvaliti svojim roditeljima, sestri, baki te ostaloj obitelji, prijateljima i kolegama na pruženoj podršci tijekom studija. Još se posebno želim zahvaliti majci na pomoći pri pisanju ovog rada.

Zoran Cikić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **Zoran Cikotić** Mat. br.: 0035174941

Naslov rada na hrvatskom jeziku: **Razvoj upravljačkog algoritma za planiranje kretanja mobilnog robota**

Naslov rada na engleskom jeziku: **Development of path-planning algorithm of a mobile robot**

Opis zadatka:

U radu je potrebno razviti upravljački program za planiranje kretanja mobilnog robota u nepoznatoj radnoj okolini s preprekama. Potrebno je razviti model percepcije radnog prostora te odgovarajući algoritam praćenja cilja s mogućnošću zaobilaženja prepreka. Za percepciju okoline robot koristi vizijski sustav i sonarne senzore. Razvijeni algoritam testirati na mobilnoj robotskoj istraživačkoj platformi Pioneer 2 u Laboratoriju za projektiranje izradbenih i montažnih sustava.

Zadatak zadan: 7. svibnja 2015.

Rok predaje rada: 9. srpnja 2015.

Predviđeni datum obrane: 15., 16. i 17. srpnja 2015.

Zadatak zadao: Prof. dr. sc. Bojan Jerbić

Predsjednik Povjerenstva: Prof. dr. sc. Franjo Cajner

SADRŽAJ

1 UVOD 1
1.1 Mobilni roboti 1
1.2 Upravljački algoritam za autonomno kretanje kroz nepoznatu okolinu 4
2 HARDVERSKA I SOFTVERSKA PLATFORMA 6
2.1 Pioneer 2Dx mobilni robot 7
2.2 MobileSim 0.7.2 9
2.3 ARIA 2.9.0 10
2.3.1 ArRobotConnector klasa 11
2.3.2 ArRobot klasa 12
2.3.3 ArRangeDevice klasa 13
2.4 OpenCV 2.4.10 14
2.4.1 Mat klasa 15
2.4.2 namedWindow funkcija 15
2.4.3 createTrackbar funkcija 16
2.4.4 VideoCapture klasa 16
2.4.5 cvtColor funkcija 17
2.4.6 inRange funkcija 19
2.4.7 Scalar funkcija 19
2.4.8 erode funkcija 19
2.4.9 dilate funkcija 20
2.4.10 getStructuringElement funkcija 20
2.4.11 imshow funkcija 21
2.4.12 waitKey funkcija 21
2.4.13 destroyAllWindows funkcija 22

2.4.14 Moments funkcija i klasa 22
2.5 Microsoft Visual Studio 2010 23
2.5.1 Podešavanje novog projekta u Visual Studio 2010 razvojnom sučelju 23
2.6 CANYON CNE-CWC1 25
3 UPRAVLJAČKI ALGORITAM 26
3.1 Metoda tangencijalnog izbjegavanja 27
3.2 Algoritam za kalibraciju parametara tražene boje 30
3.2.1 Osnovni dio algoritma 30
3.2.2 Funkcija za uklanjanje šuma ERODE_DILATE() 32
3.3 Programski kod upravljačkog algoritma 34
3.3.1 Glavni dio upravljačkog algoritma 34
3.3.2 Funkcija sonar_readings() 38
3.3.3 Funkcija forward() 39
3.3.4 Funkcija new_goal() 39
4 TESTIRANJE UPRAVLJAČKOG ALGORITMA 41
4.1 Postupak kalibracije HSV parametara 41
4.2 Testiranje upravljačkog algoritma pomoću simulacijskog modela 45
4.3 Test upravljačkoga algoritma na Pioneer 2-DX mobilnom robotu 47
5 ZAKLJUČAK 53

POPIS SLIKA

Slika 1: Sastavni dijelovi robota.... 2

Slika 2: Ciklus donošenja odluka – primjer gibanja [1].... 2

Slika 3: Spirit – autonomni rover za svemirska istraživanja [3] 3

Slika 4: Pioneer 2 mobilni robot 5

Slika 5: Dimenzije Pioneer 2 mobilnog robota [5] 7

Slika 6: Sastavni dijelovi Pioneer 2 mobilnog robota[5] 8

Slika 7: Raspored sonarnih senzora na prednjoj strani robota [5].... 8

Slika 8: Pioneer PTZ kamera 9

Slika 9: Sučelje MobileSim aplikacije 9

Slika 10: Struktura Aria programskog sučelja [10].... 11

Slika 11: Razni načini spajanja na mobilne robote iz Pioneer serije 11

Slika 12: Ostwaldov dijagram zapisa boje pomoću HSV vrijednosti [7] 19

Slika 13: Dodavanje Aria i OpenCV datoteka zaglavlja.... 23

Slika 14: Dodavanje Aria I OpenCV programskih knjižnica.... 24

Slika 15: CANYON CNE-CWC1 web kamera [13].... 25

Slika 16: Dijagram toka upravljačkog algoritma 27

Slika 17: Određivanje virtualnog cilja pomoću kuta φ [14].... 28

Slika 18: Uvedena pretpostavka o orijentaciji prepreke.... 29

Slika 19: Binarna slika prije poziva ERODE_DILATE() funkcije.... 33

Slika 20: Binarna slika nakon poziva ERODE_DILATE() funkcije 34

Slika 21: Originalna slika dohvaćena s kamere 41

Slika 22: Originalna slika transformirana u HSV način zapisa boje.... 42

Slika 23: Podešavanje klizača za filtraciju crvene boje 42

Slika 24: Binarna crno-bijela slika prije uklanjanja šuma.... 43

Slika 25: Binarna crno-bijela slika nakon uklanjanja šuma 44

Slika 26: Kamera i traženi objekt pri testiranju algoritma na simulacijskom modelu 45

Slika 27: Putanja kretanja robota pri testiranju algoritma na simulacijskom modelu..... 45

Slika 28: Pozicija stvarnog i virtualnog cilja u odnosu na središte simuliranog robota..... 46

Slika 29: Brzina kretanja robota pri testiranju algoritma na simulacijskom modelu 46

Slika 30: Testni „poligon“..... 47

Slika 31: Pioneer 2 mobilni robot s prijenosnim računalom kao upravljačkim računalom 47

Slika 32: Početna pozicija robota (lijevo), prva prepreka (desno) 48

Slika 33: Uspješno zaobilaženje prve prepreke..... 49

Slika 34: Nastavak kretanja prema drugoj prepreci 49

Slika 35: Početak zaobilaženja druge prepreke..... 50

Slika 36: Prolazak pokraj druge prepreke 50

Slika 37: Uspješno zaobilaženje druge prepreke..... 51

Slika 38: Približna pitanja kojim je robot putovao..... 51

Slika 39: : Pozicija stvarnog i virtualnog cilja u odnosu na središte robota..... 52

Slika 40: Brzina kretanja pri testiranju algoritma na Pioneer 2 mobilnom robotu 52

POPIS TABLICA

Tablica 1: Neke od važnijih metoda klase ArRobot 12

Tablica 2: Dobiveni parametri za praćenje objekata crvene boje 43

POPIS OZNAKA

Oznaka	Jedinica	Opis
BGR		Zapis boje u formatu Blue-Green-Red
HSV/HSL		Zapis boje u formatu Hue-Saturation-Value
H		Nijansa boje, eng. Hue
S		Zasićenost boje, eng. Saturation
V		Svjetlina boje, eng. Value
RGB		Red-Green-Blue
RGBA		Red-Green-Blue-Alpha
CMYK		Cyan- Magenta-Yellow-Key
PTZ		Pan-Tilt-Zoom
φ	rad	Kut između pravog i virtualnog cilja
β	rad	Kut između centra robota i najbliže prepreke
α	rad	Kut između centra robota i realnog cilja
δ	rad	Kut između centra robota i lokacije sonarnog senzora
x	pixel	Horizontalna lokacija praćenog objekta
y	pixel	Vertikalna lokacija praćenog objekta

SAŽETAK

Ovaj rad bavi se razvojem upravljačkog algoritma za autonomnu navigaciju i praćenje cilja kroz nepoznati prostor s preprekama. U prvom dijelu rada dan je osnovni pregled robotike, mobilnih robota i problema vezanih uz njihov razvoj.

Drugi dio rada opisuje hardverske i softverske platforme korištene za razvoj upravljačkog algoritma. Treći dio rada objašnjava principe rada upravljačkog algoritma, metodu izbjegavanja prepreka, i daje detaljan opis programskog koda algoritma za kalibraciju HSV parametara i projektiranog upravljačkog algoritma.

U zadnjem poglavlju dani su rezultati testiranja upravljačkoga algoritma na simulaciji i testu izvršenom pomoću Pioneer 2 mobilnog robota.

Ključne riječi: autonomna navigacija, izbjegavanje prepreka, vizijski sustav, praćenje boje, Pioneer 2-DX, OpneCV, Aria

SUMMARY

This paper deals with development of control algorithm for autonomous navigation and target tracking through unidentified space with obstacles. In the first part of the paper a basic overview of robotics, mobile robots and problems associated with their development is given.

The second part describes the hardware and software platforms used for the development of the control algorithm. The third part of the paper explains the basic working principles of the developed control algorithm, it gives a brief overview of the method used for avoiding obstacles, and then explains in detail the source code of the algorithm for calibration of HSV parameters and the source code of the developed control algorithm.

In the last chapter testing methods of the control algorithm and the obtained results are provided.

Key words: autonomous navigation, obstacle avoidance, vision system, color tracking, Pioneer 2-DX, OpneCV, Aria

1 UVOD

Robotika postaje sve češći i važniji dio ljudske svakodnevnice. Obećanja o izvršavanju raznih zadaća pomoću inteligentnih i univerzalnih strojeva koje je do sada morao obavljati isključivo čovjek primamljiva su i iz ekonomskog i iz društvenog stajališta. Za industriju razvoj robotike znači povećanje produktivnosti kroz robota kao neumornog radnika koji smanjuje troškove i omogućava fleksibilnost u ispunjavanju sve zahtjevnijeg tržišta. Za čovjeka robot može označavati oslobođenje od dosadnog i fizički teškog rada, usmjeravanje na kreativan rad i izbjegavanje radne okoline opasne za čovjekovo zdravlje.

Kako primjena industrijskih robota postaje normalna stvar, s pogleda veličine tržišta kućanstva postaju sljedeći Sveti Gral robotike. [1] Da bi se to ostvarilo glavni problem postaje mobilnost. [2] Sve veća pažnja se preusmjerava na razvoj mobilnih robota koji moraju biti u stanju savladati okolinu koji čovjek već nekoliko tisuća godina aktivno prilagođava svojim potrebama i mogućnostima.

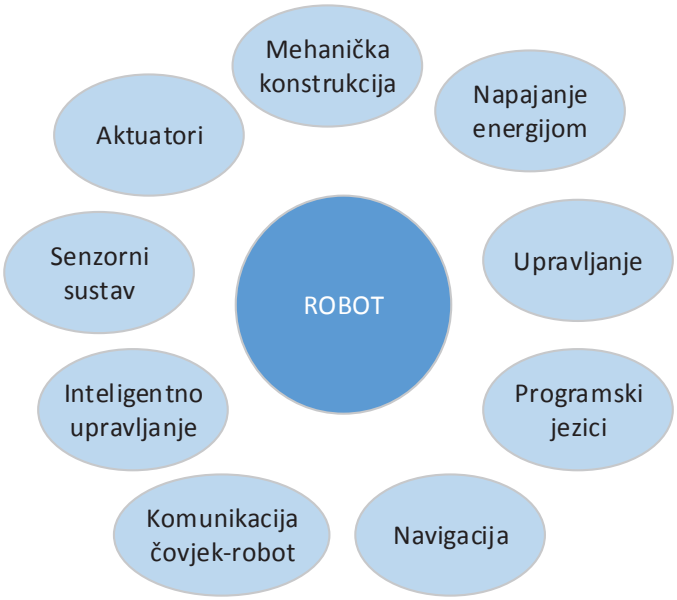
1.1 Mobilni roboti

Prema ISO 8370 definicija robota je automatski upravljani, programibilni, višenamjenski manipulator koji može izvršavati zadaće u tri ili više osi, te ovisno o primjeni može biti stacionaran ili mobilan. Ova definicija se danas više ne može primijeniti na mobilne robote jer su zahtjevi koji se stavljaju pred njih i postali daleko složeniji. [2]

Neki od zahtjeva za mobilne robote su:

- Autonomno gibanje u nepoznatom prostoru.
- Obavljanje zadatka.
- Interakcija s ljudima i/ili drugim robotima.
- Sposobnost samostalnog učenja i zaključivanja.

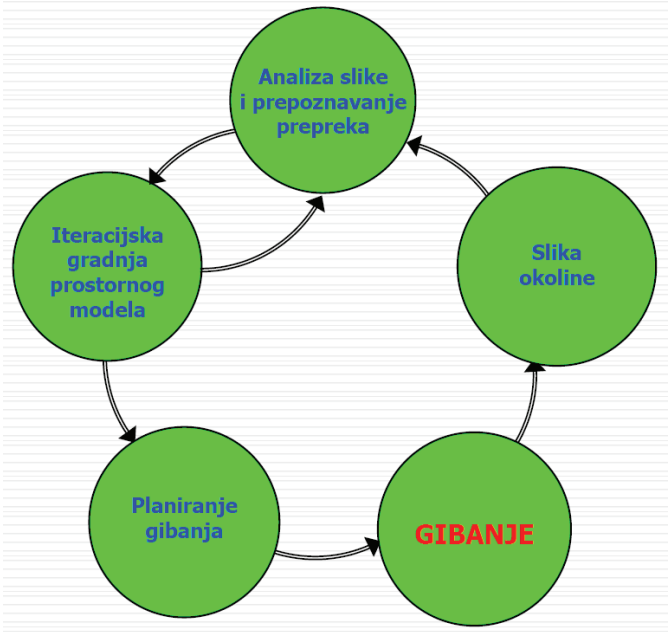
Prema tome moguća definicija mobilnog robota je: Mobilni roboti su mobilni i manipulativni fizički sustavi koji se autonomno gibaju kroz nestrukturirani prostor ostvarujući pritom interakciju s ljudskim bićima ili autonomno obavljaju neki zadatak umjetno njih. To tu tzv. uslužni roboti i oni se smatraju međukorakom između industrijskih robota današnjice i personaliziranih inteligentnih robota budućnosti koji su sposobni ponašati se kao ljudi. [2]



Slika 1: Sastavni dijelovi robota

Pri razmatranju mobilnih robota možemo ih gledati s raznih razina problema koje treba riješiti [1]:

- Primjena – industrija, kućanstva, svemir, poljoprivreda, vojne primjene
- Realizacija gibanja – noga, kotač, ravnoteža, vrste kretanja
- Interakcija s okolinom – kamere, stereo-kamere, senzori, manipulatori
- Pronalaženje putanje kretanja – poznata okolina, nepoznata okolina

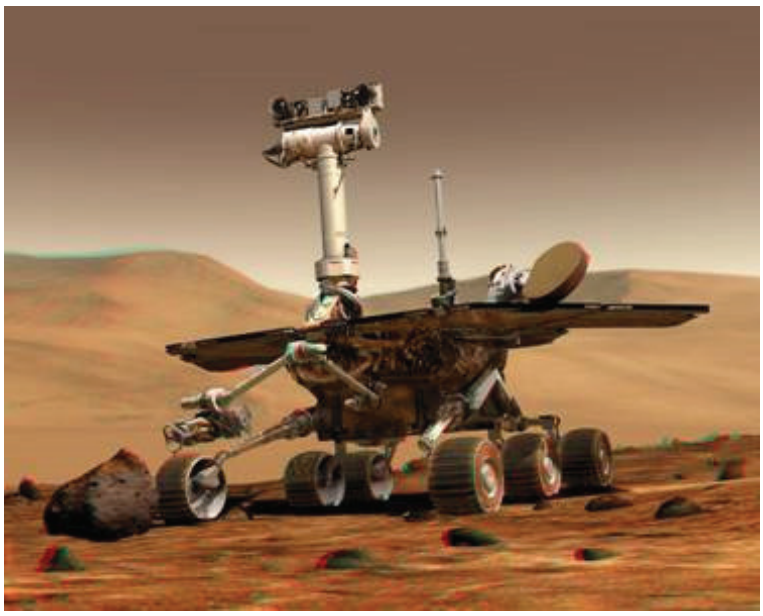


Slika 2: Ciklus donošenja odluka – primjer gibanja [1]

Još jedna česta podjela mobilnih robota je prema mediju u kojem se gibaju [2]: zračni, vodeni, svemirski i kopneni. Ostale klasifikacije mogu biti prema:

- Terenu za koji je robot predviđen
- Sustav gibanja
- Vrsta upravljanja
- Stupanj autonomnosti
- Fleksibilnost tijela robota
- Oblik tijela robota
- Način nastanka

Moguće primjene mobilnih robota su razne: prodaja potrošne robe, poljoprivredni ili šumski radovi, inspekcija rizičnih područja (mine, radijacija), rudarstvo, građevina, manipulacija proizvodnim sredstvima, istraživanje svemira i podmorja, sigurnost i zaštita, civilni prijevoz, osobna pripomoć, zabava, itd.



Slika 3: Spirit – autonomni rover za svemirska istraživanja [3]

Razvoj mobilnih robota je za mnoga spomenuta područja primjene još uvijek u povojima pa se može očekivati da će razvoj mobilnih robota u budućnosti pružiti mnogo prilika za nova znanstvena otkrića i razvoj inovativnih komercijalnih proizvoda.

1.2 Upravljački algoritam za autonomno kretanje kroz nepoznatu okolinu

Tri osnovna problema mobilne robotike su percepcija okoline, navigacija i kretanje u prostoru, te interakcija s okolinom. U ovom radu opisan je razvoj upravljačkog algoritma koji pokušava riješiti problem percepcije okoline i navigacije kroz nepoznati prostor.

Da bi riješio navedene probleme razvijeni upravljački algoritam prima podatke iz okoline na dva načina. Prima sliku prostora iz kamere i istovremeno prikuplja informacije o okolini mjerenjem udaljenosti između robota i okolnih prepreka pomoću sonarnih senzora.

Sliku iz kamere algoritam prvo pretvara u HSV sustav zapisa boja i zatim u binarni crno-bijeli zapis. Binarni zapis omogućava jednostavnije određivanje lokacije traženog objekta u odnosu na robota. No, da bi mogli doći do željenog binarnog zapisa slike potrebno je prvo pronaći optimalne HSV vrijednosti za koje binarizacija slike daje željene rezultate. U tu svrhu razvijen je pomoćni algoritam za kalibraciju HSV vrijednosti, opisan u poglavlju 3.2 i 4.1.

Lokacija objekta na binarnoj slici predstavlja cilj kojem se mobilni robot neprestano pokušava približiti, ali ukoliko se na sonarnim senzorima ispred robota očita udaljenost manja od neke zadane, prioritet nad upravljanjem robota preuzimaju očitanja sa sonarnih senzora.

U slučaju detekcije prepreke na putanji robota još uvijek se u obzir uzima lokacija realnog cilja ali se uz pomoć metode tangencijalnog izbjegavanja, opisane u poglavlju 3.1 određuje novi virtualni cilj kojeg robot prati sve dok očitanja sonarnih senzora ne pokažu da je put ispred robota bez prepreka. Nakon toga robot nastavlja slijediti realni cilj.

Platforma oko koje je izgrađen upravljači algoritam je istraživačka robotska platforma Pioneer 2 na koju je direktno vezan softver za simulaciju mobilnih robota MobileSim i paket programskih knjižnica Aria koji omogućava komunikaciju i upravljanje nad robotom.

Za razvoj dijela algoritma koji se bavi vizijskim sustavom robota iskorišten je paket programskih knjižnica OpenCV koji kroz svoje brojne, predefinirane funkcije omogućava pojednostavljeno hvatanje, obradu i prikaz slike.

Slika 4: Pioneer 2 mobilni robot

U nastavku rada u drugom poglavlju dan je detaljan pregled hardverskih i softverskih platformi korištenih u radu. To uključuje opis Pioneer 2 mobilnog robota i CANYON web kamere, opis MobileSim softvera za simulaciju rada mobilnih robota, kratak pregled Visual Studio razvojne okoline i načina na koji treba postaviti novi projekt, te pregled svih korištenih Aria i OpenCV klasa, metoda i funkcija.

U trećem poglavlju opisan je princip rada upravljačkog algoritma i metode tangencijalnog izbjegavanja prepreka te je detaljno opisan razvijeni programski kod algoritma za kalibraciju HSV parametara i upravljačkog algoritma.

U zadnjem četvrtom poglavlju prikazan je postupak kalibracije HSV parametara,, te su dani rezultati testiranja rada upravljačkog algoritma na simulacijskom modelu i na Pioneer 2 istraživačkoj platformi.

2 HARDVERSKA I SOFTVERSKA PLATFORMA

Pri izradi upravljačkog algoritma korištene su već gotove hardverske i softverske podloge koje su omogućile brži razvoj te lakše i brže testiranje razvijenog algoritma.

Glavna hardverska platforma oko koje je građen upravljački algoritma je mobilna robotska istraživačka platforma Pioneer 2-DX s kojom će na kraju rada provjeravati rad upravljačkog algoritma.

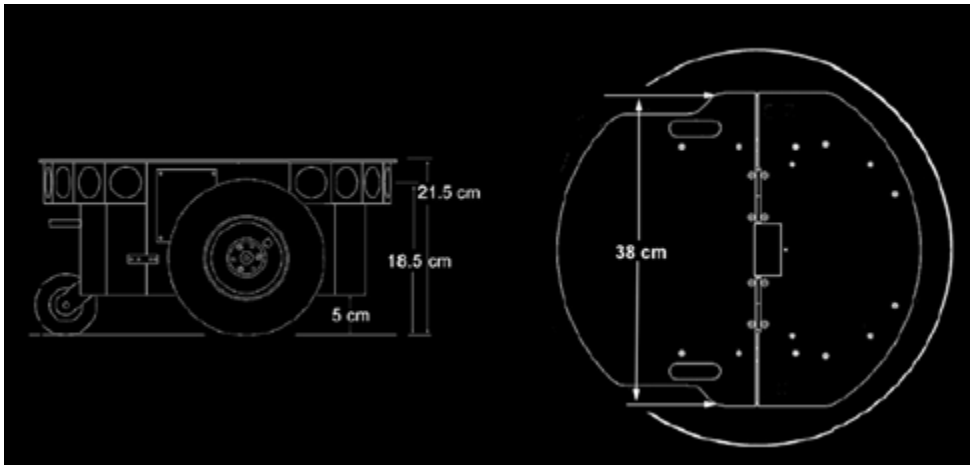
Direktno na Pioneer 2 mobilnog robota vežu se MobileSim simulacijska okolina i Aria programske knjižnice. MobileSim nudi mogućnost pokretanja simulacijskog modela Pioneer 2 mobilnog robota čime omogućava brže i jednostavnije testiranje i ispravljanje napisanog programskog koda. Aria programske knjižnice pružaju mogućnost da se korisnik Pioneer 2 istraživačke platforme fokusira na razvoj svoje ideje nudeći kroz svoje klase i metode već gotova rješenja za komunikaciju i upravljanja robotom.

Da bi upravljački algoritam mogao primati podatke o zadanom cilju iz okoline bilo je potrebno u njega ukomponirati i vizijski sustav. Kao podloga za izradu vizijskog sustava korištene su OpenCV programske knjižnice. Pomoću njih kontinuirano se hvata slika s kamere koja se potom obrađuje koristeći dostupne funkcije za pretvaranje zapisa boja, binarizaciju slike i uklanjanje šuma. OpenCV koristio se i pri razvoju algoritma koji omogućava odrađivanje HSV parametara potrebnih da bi se kod binarizacija slike prikazivala samo željena boja.

Sami programski kod pisan je i kompajliran u Microsoft Visual Studio razvojnom okruženju. Visual Studio je izabran kao razvojno okruženje jer je službeno podržan i od Aria i od OpenCV programskih knjižnica te je besplatno dostupan za sve studente tehničkih fakulteta.

2.1 Pioneer 2Dx mobilni robot

Naziv Pioneer se odnosi na obitelj mobilnih robota na dva ili četiri kotača koja radi na klijent-server arhitekturi i sadrži sve osnovne komponente potrebne za mjerenje i navigaciju kroz okolinu. Svaki robot iz Pioneer serije sadrži baterijski paket, motore, enkodere pozicije i brzine, integrirane senzore te ima mogućnost nadogradnje raznim dodacima. Svim komponentama upravlja integrirani mikrokontroler baziran na Siemens C166 mikrokontroleru te serverske aplikacije koje se na njemu izvršavaju. [5]

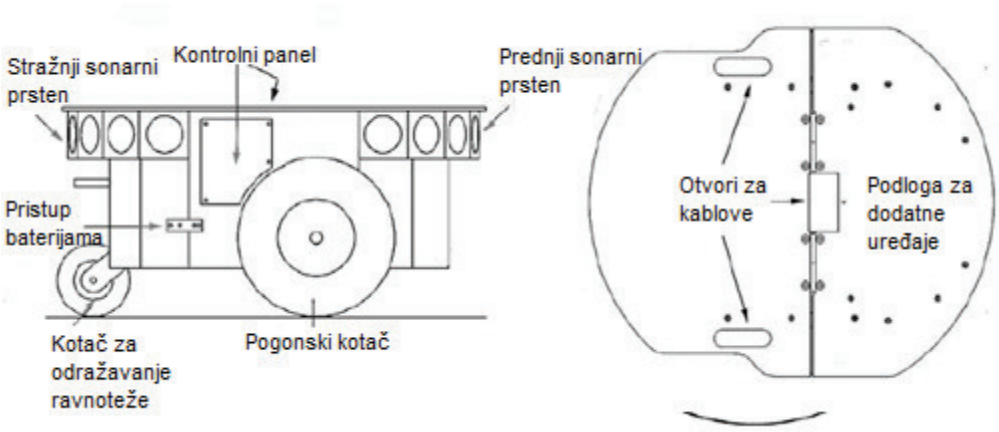


Slika 5: Dimenzije Pioneer 2 mobilnog robota [5]

Pioneer roboti dolaze sa nizom ulaznih i izlaznih portova za dodavanje do 16 dodatnih uređaja i senzora na I/O sabirnicu, dva RS-232 serijska porta, osam digitalnih I/O portova, pet analognih portova i kontrolerom za napajanja. Svim portovima se može pristupati preko zajedničkog sučelja u P2OS operativnom sustavu. [5]

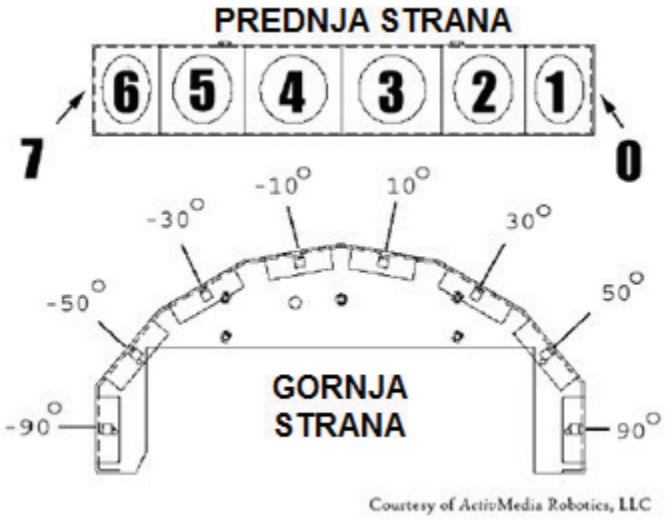
Osnovne komponente koje sačinjavaju Pioneer 2-DX robota su [5]:

- Gornja površina s korisničkim sučeljem i prihvatom za dodatne uređaje
- Osnovno tijela robota
- Nos robota iz kojeg se nalazi integrirano računalo Versa Logic EBX VSBC-6
- Niz sonarnih senzora
- Motori i enkoderi
- Baterije i sučelje za napajanje električnom energijom



Slika 6: Sastavni dijelovi Pioneer 2 mobilnog robota[5]

Važno je napomenuti i zapamtiti raspored, te način na koji su postavljeni sonarni senzori u odnosu na tijelo robota, jer se pomoću zadanih kutova za koje su zamaknuti pojedini senzori kasnije tokom programiranja može pristupati točno određenom sonarnom senzoru i saznati točnu lokaciju prepreke koje su sonarni senzori detektirali. [5]



Slika 7: Raspored sonarnih senzora na prednjoj strani robota [5]

Na robotu se nalazi još i Pioneer Pan-Tilt-Zoom PTZ kamera koja može na željeni uređaj slati standardni NTSC ili PAL video pa se kao takva može iskoristiti u razne namjene, od jednostavnog video nadzora do ulaznog sučelja za inteligentan vizijski sustav. [5]

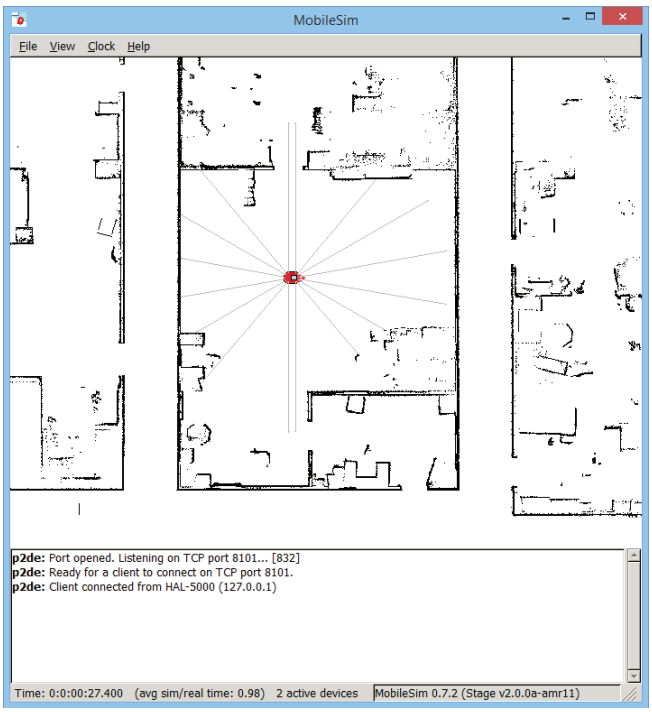
PTZ kamera nije korištena u ovom radu zbog nekompatibilnog video sučelja s prijenosnim računalom iako bi PTZ kamera predstavljala bolje rešenje u odnosu na korištene kamere zbog

mogućnosti zakretanja oko dvije osi, čime bi algoritam praćenja zadanog cilja postao manje osjetljiv na poremećaje u okolini jer bi efektivni kut vidljivosti kamere iznosio oko 250°.

Slika 8: Pioneer PTZ kamera

2.2 MobileSim 0.7.2

MobileSim je aplikacija koja služi za simulaciju rada MobileRobots mobilnih robota i njihove okoline. Može služiti kao platforma za testiranje, otklanjanje grešaka i eksperimentiranje sa Aria programskim sučeljem kao i drugim aplikacijama koje podržavaju MobileRobots platformu. MobileSim je besplatan softver koji se distributira pod GNU licencom.[6]



Slika 9: Sučelje MobileSim aplikacije

Osim rada s robotom, omogućava i pretvaranje MobileRobots prostornih karti u simulacijsku okolinu u kojoj model mobilnog robota može izvršavati svoje zadatke. Spajanje na simulirani model robota vrši se preko TCP veze slične serijskoj vezi koju koriste realni roboti. [6]

MobileSim podržava čitav niz MobileRobots robota poput Pioneer 2 DX i AT, Pioneer 3 DX i AT, PowerBot, AmigoBot, PeopleBot, PatrolBot, Seekur, Pioneer 1. Također postoji opcija definiranja vlastitih modela robota ili tipova senzora kroz Aria parametarske datoteke. [6]

2.3 ARIA 2.9.0

Aria je objektno orijentirano sučelje za razvoj algoritama i inteligentno upravljanje robotima koji su dio MobileRobots platforme. Napisana je u programskom jeziku C++ i omogućava pristup i upravljanje samim robotom kao i raznim dodacima poput senzora i manipulatora. Aria omogućava pisanje univerzalnih programa koji se mogu koristiti na raznim platformama poput Linux ili Windowsa.[7]

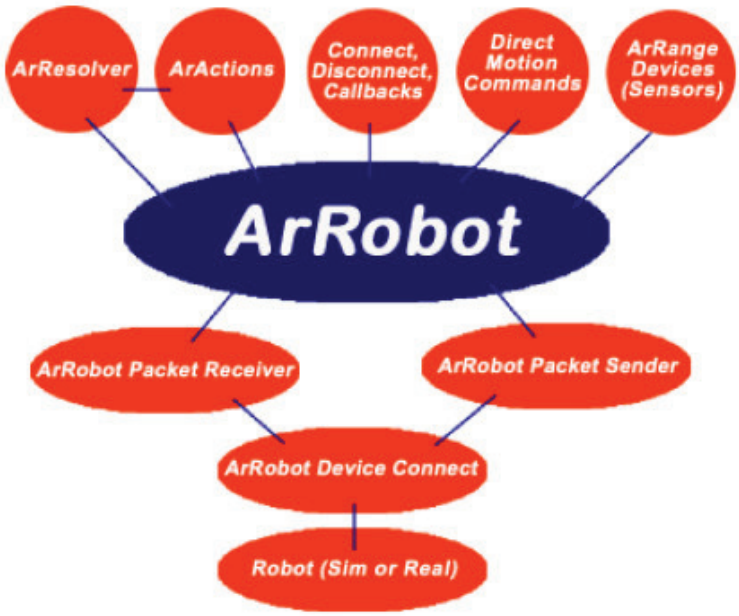
Pruža mogućnost rada na više razina, od najniže razine jednostavnog slanja naredbi pojedinim dijelovima robota, do upravljanja preko visoke razine apstrakcije kroz razne unaprijed definirane akcije. Najnovija verzija Aria sučelja se može besplatno pronaći na: http://robots.mobilerobots.com/ARIA. [7]

Od korisnika Aria-e očekuje se da budu upoznati za korištenjem osnova jezika C++ kao i s korištenjem klasa, objekata, nasljeđivanja, pokazivača, upravljanja s memorijom te procesa kompajliranja i spajanja programa. Iskustvo sa radnom na više programskim niti također može biti od velike pomoći. [7]

Za početak rada korisnik se upućuje na „…Aria/README.txt“ datoteku koja dolazi s instalacijom Arie kao i priloženom dokumentacijom u HTML ili PDF formatu. Uz dokumentaciju dostupni si još i brojni primjeri u mapi „…/Aria/examples“ kao i na raznim lokacija na internetu. [7]

U nastavku poglavlja **Error! Reference source not found.** opisane su osnovne Aria klase i funkcije koje su korištene pri izradi upravljačkog algoritma te pri testiraju rada i komunikacije s robotom.

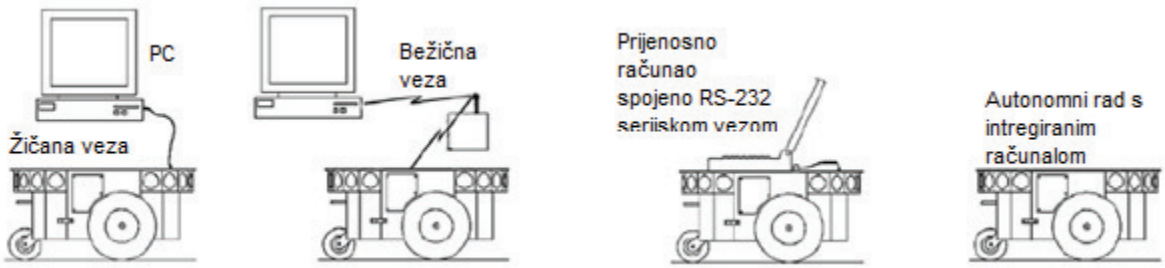
Osnovna klasa je $Aria$, čije rutine se obavezno moraju inicijalizirati prije početka slanja bilo kakvih naredbi prema robotu. Inicijalizacija Arie radi se pozivom metode $Aria::init()$.



Slika 10: Struktura Aria programskog sučelja [7]

2.3.1 ArRobotConnector *klasa*

Osnovna uloga *ArRobotConnector* klase je spajanje algoritma na robot ili simulator, ovisnosti o zadanim parametrima. *ArRobotConnector* ostvaruje vezu preko zadanog TCP porta na simulator ili na robot koji posjeduje Ethernet serijski most umjesto integriranog računala. Alternativno veza se ostvaruje preko direktne serijske veze na robot preko predefinirane port adrese. [7]



Slika 11: Razni načini spajanja na mobilne robote iz Pioneer serije

Uobičajeni automatski način rada je da se prvo pokuša uspostaviti veza preko TCP porta na simulator, no ako simulator nije pokrenut tada se pokušava ostvariti serijska veze preko porta COM1 na Windows operativnom sustavu ili preko porta /dev/ttyS0 na operativnim sustavima baziranim na Linuxu.[7]

Argumenti potrebni objektu klase $ArRobotConnector$ mogu se učitavati ručno ili se to može prepustiti objektu klase $ArArgumentParser$. U oba slučaja predaju se argumenti koje učitava glavna funkcija programa $main()$. Rasčlanjeni zadani ulazni argumenti predaju se potom objektu klase $ArRobotConnector$ koji ih koristi pri sinkronizaciji, uspostavljanju veze i zadavanju osnovnih parametara mobilnog robota. Značajne metode klase $ArRobotConnector$ su $connectRobot()$ i $parsArgs()$.

Za ručno spajanje na robota koristi se klasa $ArSerialConnect$ koja uspostavlja direktnu serijsku vezu s robotom. Važne metode su $open()$ za provjeru serijskog porta, $getPort()$ za informaciju o korištenom portu i $setPort()$ za definiranje serijskog porta preko kojeg se uspostavlja komunikacija.

2.3.2 ArRobot *klasa*

$ArRobot$ je centralna i najvažnija klasa za komunikaciju i upravljanje robotom. Koristi se za slanje naredbi i čitanje podataka s robota, te očitavanja stanja na motorima, stanja na digitalnim i analognim ulazima i izlazima, očitanja sa sonara, itd. Također se koristi za kontrolu nad svim priključenim dodacima na robotu. [7]

$ArRobot$ klasa pruža mogućnost kontrole nad kretanjem robota na tri načina. Slanjem direktnih naredbi programiranih u firmwareu robota pomoću metoda klase $ArCommands()$. Upravljanje preko funkcija koje služe za izvršavanje jednostavnih kretnji, kao što je kretanje u zadanu točku, rotacija, kretanje u zadanom smjeru, kontrola brzine oba ili svakog motora pojedinačno. Zadnji način je upravljanje preko unaprijed definiranih ili korisnički definiranih akcija pomoću klase $ArAction$. Neke od značajnijih metoda klase ArRobot navedene su u sljedećoj tablici.

Tablica 1: Neke od važnijih metoda klase ArRobot

addRangeDevice()	Dodaje objektu senzor za mjerenje udaljenosti
addSonar()	Dodaje objektu sonarne senzore
addUserTask()	Dodaje objektu zadatak definiran od strane korisnika
lock()	Zaključava objekt da ne bi došlo do promjena pri čitanju podatka
unlock()	Otključava objekt
addAction()	Dodaje objektu akciju
blockingConnection()	Uspostavlja vezu s robotom
com()	Šalje na robota naredbe na razini firmwarea

comInt()	Šalje na robota naredbe na razini firmwarea
enableMotors()	Omogućava korištenje motora
disableMotors()	Onemogućava korištenje motora
enableSonar()	Omogućava korištenje sonara
disableSonar()	Onemogućava korištenje sonara
disconnect()	Prekida vezu s robotom
isRunning()	Provjerava da li robot trenutno radi
setVel()	Postavlja brzinu motora u mm/s
setVel2()	Omogućava upravljanje s svakom motorom pojedinačno
setRotVel()	Postavlja brzinu rotacije
setHeading()	Okreće robota, u zadanom smjeru
stop()	Zaustavlja sve oblike kretanja robota
setAbsoluteMaxTransVel()	Ograničava maksimalnu dopuštenu translacijsku brzinu

2.3.3 ArRangeDevice *klasa*

Ovo je osnovna klasa za sve uređaje za mjerenje udaljenosti koji se nalaze na robotu. Održava dva memorijska spremnika. U jednom spremniku nalaze se trenutna očitanja dok drugi kumulativni spremnik pamti veći broj starijih očitanja. Maksimalni broj očitanja u svakom spremniku može se kontrolirati pri stvaranju objekta ili se veličina spremnika može naknadno prilagoditi korisničkim potrebama. [7]

Očitanja udaljenosti se najčešće reprezentiraju kao točke u dvodimenzionalnom prostoru koordinatama (x,y) na mjestima na kojima senzor detektira neki predmet u okolini. Dok neki uređaji poput lasera mogu vraćati i očitanja u neobrađenom obliku.

Postoji nekoliko podklasa poput $ArLaser$ za lasersko mjerenje udaljenosti i $ArSonarDevice$ za sonarne senzore koji se nalaze na robotu i korišteni su za očitavanje udaljenosti u sklopu ovog rada. Mogu se definirati i virtualna ograničenja poput zabranjenih zona ili virtualnih objekata u mapiranom prostoru u kojem se robot kreće.

Također, važno je zapamtiti da se uređaji za mjerenje udaljenosti često procesiraju u zasebnoj procesorskoj niti pa je važno tokom čitanja podataka koristiti metode $lock()$ i $unlock()$ kako bi čitani podaci ostali zaštićeni od mijenjanja tokom postupka čitanja. [7]

Objekt koji reprezentira uređaj za mjerenje udaljenosti obično se veže u objekt klase $ArRobot$ metodom $ArRobot::addRangeDevice()$ te se tada očitanja mogu vršiti direktno s tog objekta.

Neke od značajnijih metoda ove klase su $getCurrentBuffer()$ za očitavanje trenutnih očitanja udaljenosti i $getCumulativeBuffer()$ za pristup starijim očitanjima. Zatim, $clearCurrentReadings()$ i $clearCumulativeReadings()$ za pražnjenje memorijskih spremnika te $currentReadingPolar()$ za očitanja u određenom polarnom sektoru robota.

2.4 OpenCV 2.4.10

OpenCV (Open Source Computer Vision Library) je skupina besplatnih programskih knjižnica koje sadrže nekoliko stotina različitih algoritama za dohvaćanje i obradu videa, namijenjenih uglavnom za upotrebu u sklopu računalnih vizijskih sustava. Verzija 2.4 korištena u ovom radu namijenjena je za rad s C++ programskim jezikom. [8]

OpenCV je modularno strukturiran, a dostupni moduli za korištenje su [8]:

- **core** – modul zadužen za definiranje osnovnih struktura podataka uključujući višedimenzionalno polje tipa *Mat* koje služi za pohranu slika prije i tijekom obrade
- **imgproc** – modul zadužen za obradu slike što uključuje linearno i nelinearno filtriranje, geometrijske transformacije slike, prebacivanje između raznih oblika zapisa boja, izradu histograma i sl.
- **video** – modul za analizu videa koji uključuje procjenu smjera kretanja, uklanjanje pozadine i algoritme za praćenje objekata
- **calib3d** – sadrži osnovne algoritme vezane uz geometriju dobivenu iz više vizijskih izvora, kalibraciju pojedinačnih ili višestrukih kamera, procjenu orijentacije promatranog objekta i elemente za rekonstrukciju 3D prostora
- **objdetect** – modul za detekciju predefiniranih klasa objekata poput lica, oči, ljudi, automobila i ostalih živih bića ili predmeta
- **highgui** – sučelje za snimanje videa koje uključuje kodiranje videa, pojedinačnih slika te izradu jednostavnih korisničkih sučelja
- **gpu** – algoritmi za rad s procesorima za video obradu

Još jedna od bitnih prednosti korištenja OpenCV programskih knjižnica je automatsko upravljanje memorijskim resursima koje OpenCV pruža. Osim automatskog pražnjenja nepotrebnih objekata iz memorije OpenCV uzima u obzir i dijeljenje podataka između raznih funkcija koje korisnik poziva tako da ne dolazi do nepotrebnog kopiranja podataka, već se isti podatak na određenoj memorijskoj adresi koristi za razne svrhe. [8]

U nastavku ovog poglavlja navode se i detaljno objašnjavaju sve OpenCV funkcije koje su bile nužne za izradu upravljačkog algoritma i algoritma za kalibraciju parametara tražene boje.

2.4.1 Mat klasa

Mat klasa predstavlja n-dimenzionalni numerički niz koji se može koristiti za spremanje realnih ili kompleksnih vrijednosti u obliku vektora ili matrica za crno-bijele slike ili slike u boji, vektorska polja, polja točaka, tenzore i histograme. Struktura podataka u nekom dvodimenzionalnom nizu M je definirana preko programske metode $M.step$ tako da se adresa elemenata $(i_0, \dots, i_{M.dimanzija-1})$ gdje vrijedi $0 \leq i_k < M.size[k]$ računa kao [8]:

$$adresa(M_{i,j}) = M.data + M.step[0] * i + M.step[1] * j$$

U upravljačkom algoritmu u ovom radu Mat klasa se koristi za spremanje slika dohvaćenih s kamere te za pohranu i prikaz međukoraka i krajnjeg rezultata obrade slike na temelju kojeg se računa pozicija promatranoga objekta.

2.4.2 namedWindow funkcija

Funkcija koja stvara prozor u kojem se mogu prikazivati slike i klizači. Prozori se pozivaju po zadanom imenu. Ako već postoji prozor istog imena funkcija se ne će izvršiti.

$$void\ namedWindow(const\ string\&\ winname, int\ flags = WINDOW\_AUTOSIZE\ )$$

Parametri koje funkcija traži pri pozivanju su:

- **name** – ime prozora
- **flags** – oznaka koja definira mogućnosti manipulacije nad prozorom
    - WINDOW_NORMAL – korisnik može mijenjati veličinu prozora
    - WINDOW_AUTOSIZE – prozor poprima veličinu slike koju prikazuje i veličina prozora se ne može naknadno mijenjati
    - WINDOW_OPENGL – prozor se stvara s OpenGL podrškom

Prozori se zatvaraju pozivom funkcija $destroyWindow()$ ili $destroyAllWindows()$.

2.4.3 createTrackbar funkcija

createTrackbar je OpenCV funkcija koja stvara klizač sa zadanim imenima i rasponom vrijednosti. Slika 23 prikazuje primjer prozora s klizačima. [8]

$$int\ createTrackbar(const\ string\&\ trackbar, const\ string\&\ winname,$$
$$int*\ value, int\ count, korisnickafunkcija = 0, void*\ userdata = 0)$$

Klizaču dodjeljuje varijablu čija vrijednost se može sinkronizirati s funkcijom definiranom od strane korisnika. Na kraju funkcija pozicionira klizač u prozor zadan $winname$ parametrom. Parametri funkcije su:

- trackbar – ime stvorenog klizača
- winname – ime prozora u koji se stavlja klizač
- value – neobavezni pokazivač ili referenca na varijablu čiju vrijednost se želi mijenjati pomoću klizača
- count – maksimalna vrijednost klizača, minimalna vrijednost je uvijek nula
- onChange – pokazivač na funkciju koja će se pozvati svaki put kada se pomakne klizač. Funkcija treba biti inicijalizirana kao $\boldsymbol{void\ ime\_funkcije(int, void*)}$ pri čemu je prvi parametar trenutna pozicija klizača a drugi parametar je userdata (vidi slijedeći parametar funkcije createTrackbar) koji je definiran od strane korinsika.
- userdata – je parametar koji omogućava prenošenje željene vrijednosti pozivanoj funkciji bez korištenja globalnih varijabli

2.4.4 VideoCapture klasa

VideoCapture je klasa objekata za dohvaćanje video datoteka. Pruža programsko sučelje u programskom jeziku C++ za dohvaćanje videa iz kamera ili postojećih video datoteka. [8]

$$VideoCapture\ objekt;$$

Nakon stvaranja objekta pomoću metode $VideoCapture::open$ otvaraju se postojeće video datoteke ili kamera. Za video datoteke prosljeđuje se ime video datoteke a za kamere se prosljeđuje njihov identifikacijski broj.

$$bool\ VideoCapture::open(const\ string\&\ filename)$$

$$bool\ VideoCapture::open(int\ device)$$

Metodom $VideoCapture::isOpened$ provjerava da li je inicijalizacija metodom $VideoCapture::open$ uspijela.

Metoda $VideoCapture::set$ omogućava mijenjanje postavki objekta $VideoCapture$. Parametri koji se prosljeđuju metodom su:

- **propld** – identifikator svojstva koje se želi mijenjati. Neki od mogućih identifikatora koji se mogu podešavati su:
  - CV_CAP_PROP_FRAME_WIDTH – širina slike koja se dohvaća
  - CV_CAP_PROP_FRAME_HEIGHT - visina slike koja se dohvaća
  - CV_CAP_PROP_FPS – broj slika u sekundi
  - CV_CAP_PROP_FOURCC – četveroznamenkasti kod za odabir tipa kodiranja
  - CV_CAP_PROP_FRAME_COUNT – broj slika u video datoteci
  - CV_CAP_PROP_BRIGHTNESS – svjetlina slike (samo za kamere)
  - CV_CAP_PROP_CONTRAST – kontrast slike (samo za kamere)
  - CV_CAP_PROP_SATURATION - saturacija slike (samo za kamere)
  - CV_CAP_PROP_HUE - nijansa slike (samo za kamere)
  - CV_CAP_PROP_EXPOSURE - vrijeme ekspozicije (samo za kamere)
- **value –** vrijednost svojstva

2.4.5 cvtColor funkcija

Funkcija koja konvertira sliku iz jednog načina zapisa boje u neki drugi. Zapisi boja s tri kanala se mogu spremati u zapis s četiri kanala radi bolji performansi funkcije. [8]

$$void\ gpu::cvtColor(const\ Mat\&\ src, Mat\&\ dst, int\ code, int\ dcn = 0,$$
$$Stream\&\ stream = Stream::Null())$$

Ulazni parametri u funkciju su:

- **src** – ulazna slika
- **dst** – izlazna slika
- **code** – kod za način konvertiranja zapisa boje. Konverzija Luv i Bayer zapisa nije podržana. Neki od mogućih kodova su:
  - CV_BGR2HSV
  - CV_RGB2HSV
  - CV_HSV2BGR

- CV_HSV2RGB
- CV_BGR2HLS
- CV_HLS2BGR
- …

- **dcn** – broj kanala u izlaznoj slici, ako je vrijednost ovog parametra nula broj kanala u izlaznoj slici ostaje isti kao i u ulaznoj slici
- **stream** – niz za asinkronu verziju

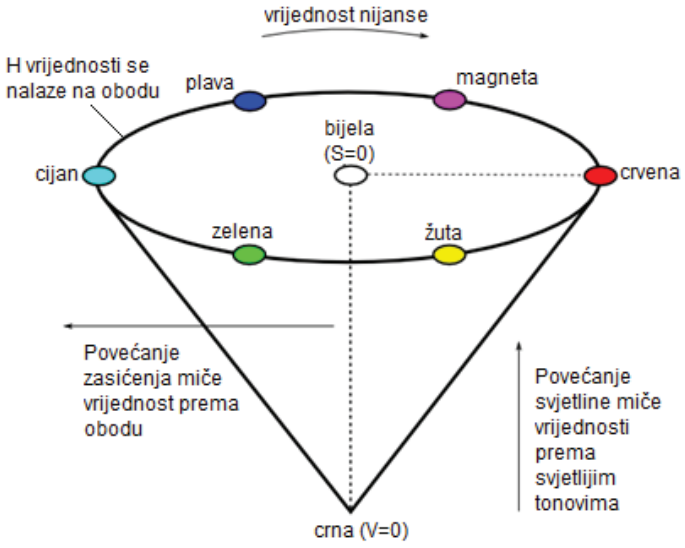
HSV sustav zapisa boja

HSV sustav zapisa boja je pogodan za korištenje pri radu s vizijskim sustavima jer za razliku od npr. najraširenijeg RGB sustava, HSV sustav odvaja informaciju o svjetlini i intenzitetu boje od informacije o kojoj se nijansi boji se radi. [9] Zbog toga su algoritmi koji koriste HSV sustav pri praćenje boje manje osjetljivi na promjene u svjetlini i intenzitetu boje koje se javljaju radi promjena u okolini.

HSV sustav diferencira boje pomoću nijanse, zasićenja i svjetline (eng. Hue, Saturation, Luminance/Value). Njih definiramo kao :

- Nijansa (eng. Hue) govori o tome kojoj čistoj boji najviše sliči neka tražena boja. Svi tonovi neke boje imaju istu vrijednost.
- Zasićenje (eng. Saturation) daje informaciju o tome koliko bijele boje sadrži neka čista boja.
- Vrijednost boje (eng. Value/Lightness) opisuje koliko je neka boja tamna.

Slika 12 prikazuje grafički prikaz HSV zapisa boje. Na obodu baze prikazanoga obrnutog stošca nalaze se čiste boje. Zasićenost vrijednosti nula se nalazi na osi stošca, pri čemu centar kružnice na bazi stošca predstavlja bijelu boju a povećanje zasićenja podrazumijeva micanje prema obodu baze. Vrijednost svjetline je nula kada se radi o crnoj boji a povećanje vrijednosti svjetline znači micanje od vrha stošca prema bazi.[11]



Slika 12: Ostwaldov dijagram zapisa boje pomoću HSV vrijednosti [11]

2.4.6 inRange funkcija

Funkcija koja provjerava da li se neki niz elemenata nalazi između elementa druga dva niza. Ulazni parametri su tri niza, originalni niz i dva niza koji predstavljaju donju i gornju granicu.

$$void\ inRange(niz, donja\_granica, gornja\_granica, izlazni\_niz)$$

Kada su, kao u ovom radu, donja i gornja granica skalari moraju se pri navođenju ograditi zagradama. [8]

2.4.7 Scalar funkcija

Funkcija koja vraća skalarni umnožak najviše četiri vektora. U ovom radu koristi se za određivanje vrijednost određene točke na slici. [8]

$$Scalar\_(\_Tp\ v0, \_Tp\ v1, \_Tp\ v2 = 0, \_Tp\ v3 = 0);$$

2.4.8 erode funkcija

Funkcija koja erodira izvornu sliku koristeći se specifičnim strukturalnim elementima koji određuju oblik skupa točaka neke slike nad kojim će se računati minimalna vrijednost. Može se izvoditi u više uzastopnih iteracija i u slučaju više-kanalnih slika izvodi se nad svakim kanalom posebno. [8]

$$void\ erode(src, dst, kernel, anchor = Point(-1,-1), int\ iterations = 1,$$
$$int\ borderType = BORDER\_CONSTANT,$$
$$const\ Scalar\&\ borderValue = morphologyDefaultBorderValue()\ )$$

Parametri koji se mogu prosljeđivati $erode$ funkciji su:

- **src** – ulazna slika
- **dst** – izlazna slika
- **element** – strukturalni element koji se koristi za erodiranje slike
- **anchor** – centar djelovanja funkcije, $(-1, -1)$ označava centar objekta koji se erodira
- **iterations** – broj iteracija erozije
- **borderType** – metoda ekstrapolacije točaka slike
- **borderValue** – vrijednost ruba u slučaju da je rub konstantan

2.4.9 dilate funkcija

Funkcija koja proširuje sliku koristeći se specifičnim strukturalnim elementima koji određuju oblik skupa točaka neke slike nad kojim će se računati maksimalna vrijednost. Može se izvodi u više uzastopnih iteracija i u slučaju više-kanalnih slika izvodi se nad svakim kanalom posebno. [8]

$$void\ dilate(src, dst, kernel, anchor = Point(-1,-1), int\ iterations = 1,$$
$$int\ borderType = BORDER\_CONSTANT,$$
$$const\ Scalar\&\ borderValue = morphologyDefaultBorderValue()\ )$$

Parametri koji se mogu prosljeđivati $erode$ funkciji su:

- **src** – ulazna slika
- **dst** – izlazna slika
- **element** – strukturalni element koji se koristi za proširivanje slike
- **anchor** – centar djelovanja funkcije, $(-1, -1)$ označava centar objekta koji se erodira
- **iterations** – broj iteracija erozije
- **borderType** – metoda ekstrapolacije točaka slike
- **borderValue** – vrijednost ruba u slučaju da je rub konstantan

2.4.10 getStructuringElement funkcija

Strukturalni element za funkcije $erode$ i $dilate$ dobiva se funkcijom $getStructuringElement()$ koja kao izlaz vraća objekt klase Mat. [8]

$$Mat\ getStructuringElement(int\ shape, ksize, anchor = Point(-1,-1))$$

Parametri koji se prosljeđuju su:

- **shape** – oblik elemnta može biti:
  - MORPH_RECT – kvadratna struktura
  - MORPH_ELLIPSE – eliptična struktura
  - MORPH_CROSS – kružna struktura
  - CV_SHAPE_CUSTOM – struktura definirana od strne korisnika
- **anchor** – centar djelovanja elementa, $(-1,-1)$ označava geometrijski centar objekta (samo za križnu strukturu)

2.4.11 imshow funkcija

Funkcija $imshow$ prikazuje sliku u zadanom prozoru. Kao parametre traži ime prozora u kojem će biti prikazana slika i ime slike koja će biti prikazana. [8]

$$void\ imshow(const\ string\&\ winname, InputArray\ mat)$$

Ako je prozor pokrenut sa $CV\_WINDOW\_AUTOSIZE$ parametrom slika će biti u originalnoj veličini ali će još uvijek biti limitirana sa rezolucijom monitora na kojem se prikazuje. Ako prozor nije pokrenut pomoću funkcije $namedWindow$ prije poziva $imshow$ funkcije automatski se pretpostavlja da će se koristiti $CV\_WINDOW\_AUTOSIZE$ parametar.

Ovisno o broju bitova koji opisuju jedan točku na slici (dubina slike), prikazana slika će biti skalirana na sljedeći način:

- slika dubine 8 bitova opisana cjelobrojnim vrijednostima ne će se skalirati
- za slike dubine 16 ili 32 bita opisane cjelobrojnim vrijednostima, vrijednosti pojedinih točaka će biti podijeljene sa 256, tj. originalni raspon $[0{,}255 * 256]$ se mapira u rasponu $[0{,}255]$.
- slike dubine 32 bita opisane realnim vrijednostima se množe sa 256, , tj. originalni raspon $[0{,}1]$ se mapira u rasponu $[0{,}255]$.

Važno je napomenuti da funkcija $waitKey$ treba slijediti odmah nakon poziva $imshow$ funkcije. U suprotnom slika ne će biti prikazana.

2.4.12 waitKey funkcija

Ova funkcija čeka na pritisak tipke beskonačno dugo ako je dani parametar ≤ 0. Ako je parametar pozitivan onda on označava čekanje na pritisak tipke u milisekundama. Budući da

treba uzeti u obzir da operativni sustava računala na kojem se izvodi algoritam ima minimalno vrijeme potrebno za prebacivanje iz jednog niza procesiranja u drugi. Stoga zadani parametar čekanja zapravo označava najmanje vrijeme čekanje. [8]

$$int\ waitKey(int\ delay = 0)$$

Funkcija vraća kod pritisnute tipke ili vrijednost -1 ako tipka nije bila pritisnuta. Također treba napomenuti da funkcija radi samo kada je aktivan jedan od prozora stvorenih pomoću OpenCV-a.

2.4.13 destroyAllWindows funkcija

Funkcija destroyWindow uklanja sve prozore stvorene pomoću OpenCV-a. [8]

$$void\ destroyAllWindows()$$

2.4.14 Moments funkcija i klasa

$Moments$ funkcija računa momente trećeg reda nekog poligona ili vektorskog ili rastriranog oblika. Rezultate vraća u obliku klase.

$$Moments\ momenti(rastr\ \_slika)$$

U sklopu ovog rada ova funkcija se koristi za računanje lokacije traženog objekta na slici. Funkcija $moments()$ računa prostorne momente u horizontalnom i vertikalnom smjeru te površinu objekta. Lokaciju objekta na slici dobiva se tako da se momenti horizontalnog i vertikalnog smjera podijele za površinom objekta.

Pri računanju lokacije objekta ovom metodom bitno je svesti šum na binarnoj crno-bijeloj slici na minimum. [8]

2.5 Microsoft Visual Studio 2010

Microsoft Visual Studio je integrirana okolina za razvoj aplikacija za osobna računala na Windows platformi, ali i za razvoj i dizajn internet aplikacije te aplikacija za mobilne uređaje.[12]

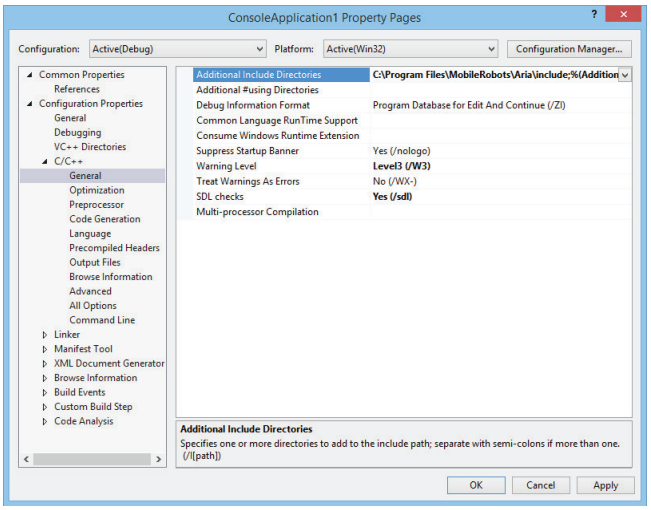
Visual Studio podržava pisanje programskog koda u više jezika, jedan od kojih je i C++ korišten u ovom radu i kojeg podržavaju Aria i OpenCV programska sučelja za upravljanje robotom i obradu slike. Osim aplikacija koje se izvršavaju u terminalu, Visual Studio podržava i jednostavno stvaranje aplikacija sa korisničkim sučeljima. [12]

Razlog odabira Visual Studia 2010 kao razvojnog sučelja je pojednostavljena upotreba Aria programskih knjižnica, budući da je Visual Studio službeno podržan od strane Adept MobileRobots platforme. Također Visual Studio je besplatno dostupan studentima tehničkih fakulteta preko Microsoftovog programa Dreamspark za akademske ustanove.

2.5.1 Podešavanje novog projekta u Visual Studio 2010 razvojnom sučelju

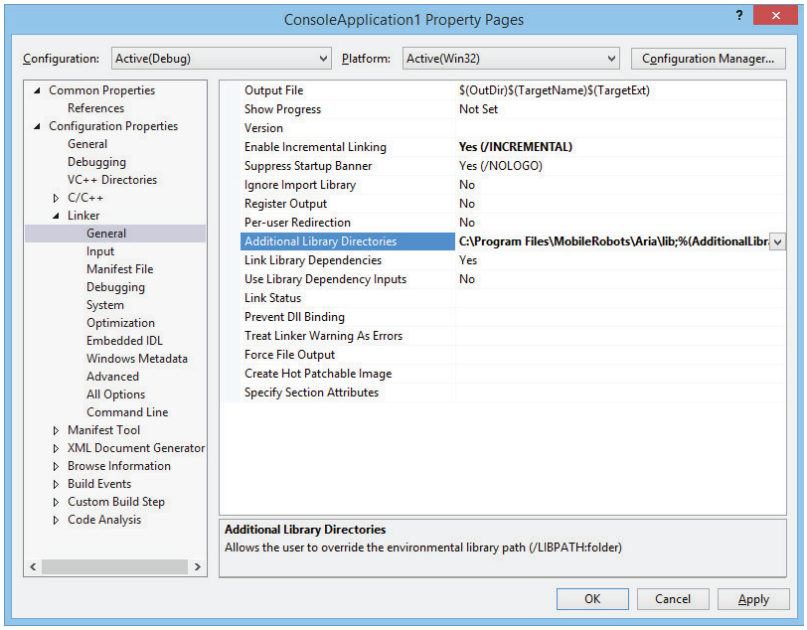
Za pravilan rad sa ARIA 2.9.0 i OpenCV 2.4.10 programskim knjižnicama bitno je pravilno podesiti postavke projekta u Visual Studiu.

Nakon otvaranja novog projekta, u ovom slučaju prazne Visual C++ Win32 Console aplikacije, u mapu Source Files dodajemo novu Source.cpp datoteku. U izbornikom PROJECT otvaramo svojstva upravo kreiranoga projekta. U izborniku C++/General pod Additional Include Directories važno je dodati putanje do ARIA i OpenCV include mapa u kojima se nalaze programska zaglavlja potrebna za pozivanje funkcija.



Slika 13: Dodavanje Aria i OpenCV datoteka zaglavlja

Potom je u izborniku Linker/General u Additional Library Directories potrebno dodati putanju do lokacija na kojima se nalaze Aria i OpenCV knjižnice.



Slika 14: Dodavanje Aria I OpenCV programskih knjižnica

Kao zadnji korak pri postavljanju novog projekta potrebno je u izborniku Linker/Input pod Additional Dependencies dopisati sve knjižnice koje će se koristiti u sklopu projekta. Knjižnice korištene u ovom radu su:

- AriaDebugVC10.lib
- opencv_calib3d2410d.lib
- opencv_core2410d.lib
- opencv_features2d2410d.lib
- opencv_highgui2410d.lib
- opencv_imgproc2410d.lib
- opencv_legacy2410d.lib
- opencv_objdetect2410d.lib
- opencv_ocl2410d.lib
- opencv_video2410d.lib

Verzija knjižnica ovisi o verziji Visual Studia koja se koristi. Ako se koristi neka druga razvojna okolina potrebno je napraviti knjižnice prilagođene specifično za tu okolinu.

Također valja napomenuti da je prije prvog pokretanja programa potrebno u mapu s generiranom .exe datotekom kopirati .dll datoteke koje se mogu naći u /bin mapi na lokacijama na kojima se nalaze instalacije Aria i OpenCV programskih knjižnica.

2.6 CANYON CNE-CWC1

CANYON CNE-CWC1 standardna je CMOS kamera. Korištena je kao ulaz vizijskog sustava za snimanje okoline i traženje zadanog objekta. Spaja se na računalo preko USB-A 2.0 sučelja. Senzor je veličine 1,3 Mpixela i može snimati maksimalno 30 slika u sekundi. Širina vidnog polja kamere je 33,5°. Postolje omogućava zakretanje kamere za 360°.

Slika 15: CANYON CNE-CWC1 web kamera [13]

3 UPRAVLJAČKI ALGORITAM

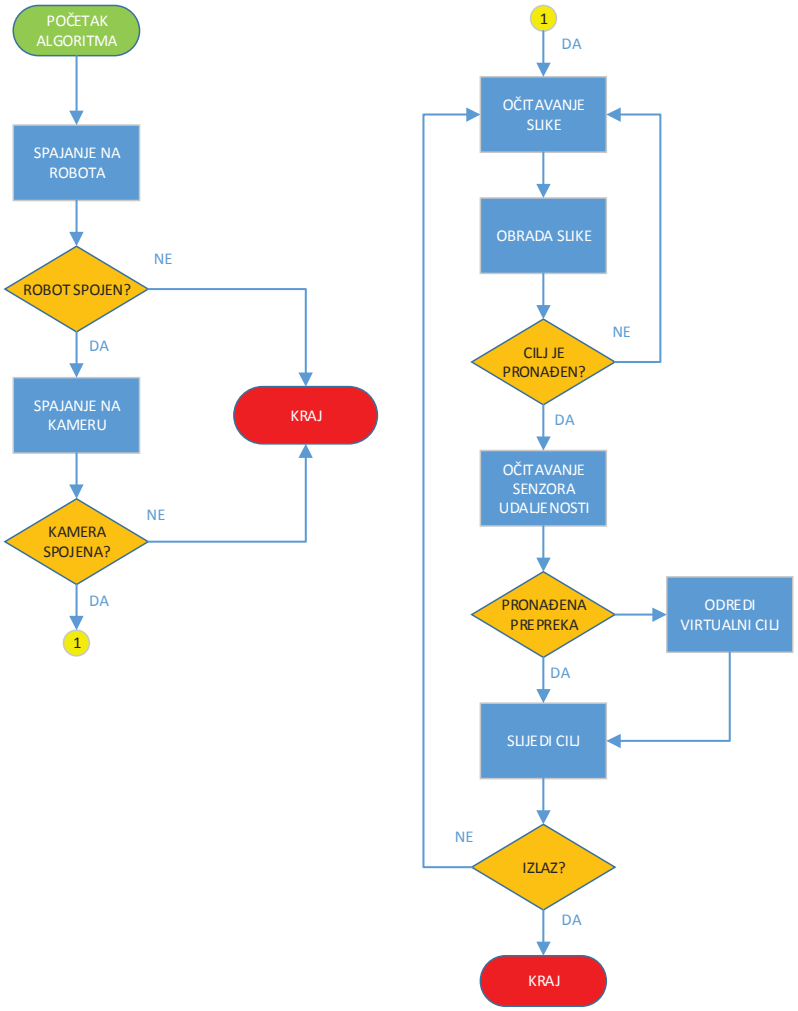
Razvijeni upravljački algoritam može se podijeliti na dva osnovna dijela. Prvi dio zadužen je za vizijski sustav koji dohvaća informacije iz okoline a drugi dio je zadužen za komunikaciju i upravljanje s robotom.

Vizijski sustav sastoji se od kamere spojene preko USB sučelja na upravljačko računalo i dijela algoritma koji koristi OpenCV funkcije za čitanje i obradu slike. On učitava sliku s kamere i prebacuje je iz RGB zapisa u HSV zapis boja. Zatim prema zadanim parametrima algoritma vrši binarizaciju slike, uklanja šum te na kraju računa lokaciju traženog objekta.

Drugi dio algoritma prima informaciju o lokaciji traženog objekta te potom modificiranom metodom tangencijalnog izbjegavanja računa smjer u kojem se robot treba gibati i na kraju pomoću Aria klasa i metoda izdaje naredbe robotu o smjeru i brzini kojom se treba gibati.

Funkcije upravljačkoga algoritma su:

1. Povezivanje s robotom i kamerom
2. Primanje podataka s kamere
3. Obrada dobivene slike
4. Određivanje lokacije traženog objekta u odnosu na robota
5. Očitavanje senzora udaljenosti
6. Donošenje odluke o kretanju
7. Kretanje



Slika 16: Dijagram toka upravljačkog algoritma

3.1 Metoda tangencijalnog izbjegavanja

Kako se robot kreće u nepoznatom prostoru pri donošenju odluke o smjeru gibanja nije moguće provesti planiranje putanje na globalnoj razini. Stoga je za donošenje odluka o načinu kretanja iskorištena modificirana metoda tangencijalnog izbjegavanja koja se temelji na stvaranju privremenog virtualnog cilja kojeg robot slijedi svaki put kada se između robota i zadanog cilja pojave prepreke.

Metoda tangencijalnog izbjegavanja tip je reaktivne metode navigacije kroz prostor koja ne zahtijeva nikakvo prethodno znanje o prostoru u kojem se robot kreće, već se kretanje robota prilagođava zatečenom stanju na putu prema cilju. Metoda je temeljena na relativno

jednostavnom i brzom algoritmu koji ne zahtjeva puno računalnog vremena pri donošenju odluka.

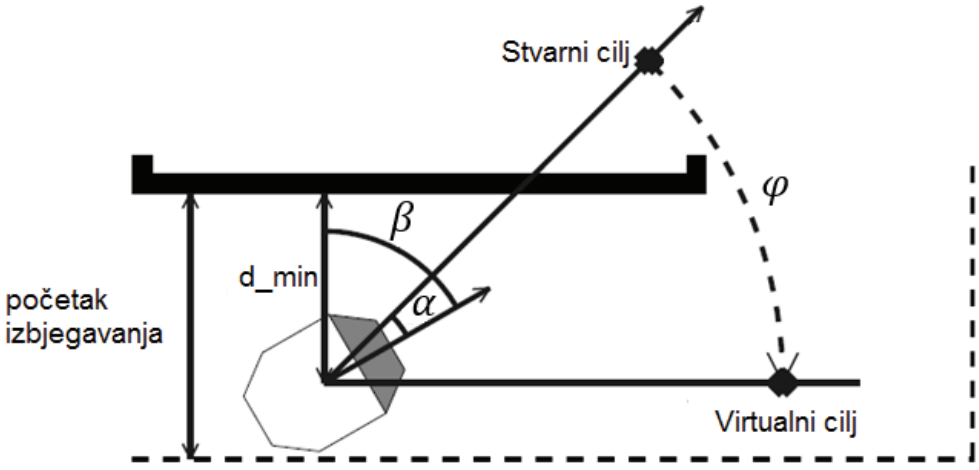
Traženje cilja može se podijeliti na dva koraka: 1. Približi se cilju. 2. Promijeni smjer kretanja svaki put kada uočiš prepreku. Navedena dva koraka se međusobno izmjenjuju ovisno o postajanju prepreke u robotovoj blizini i mogu se promatrati kao dvije ugniježdene petlje. Unutarnja petlja je zadužena za praćenje cilja kada nema detektiranih prepreka, a vanjska petlja je zadužena za stvaranje privremenog virtualnog cilja kada se detektira prepreka. [14]

Suština ove metode je biranje smjera koji je tangentan na detektiranu prepreku. To se radi određivanjem kuta φ koji se računa u odnosu na kut β koji je pak određen najbliže detektiranom preprekom i u odnosu na kut α koji je definiran u odnosu na položaj objekta koji se prati i središta robota.

Kada se detektira prepreka unutar zadane zone izbjegavanja d_{min}, određuje se kut β prema smjeru najmanje izmjerene udaljenosti u odnosu na robota i svojstvima sustava za mjerenje udaljenosti. Uz poznavanje smjera kretanja robota određuje se kut α u odnosu na lokaciju cilja te se računa se kut φ na sljedeći način:

$$\varphi = sign(\beta) * 90° - (\beta - \alpha)$$

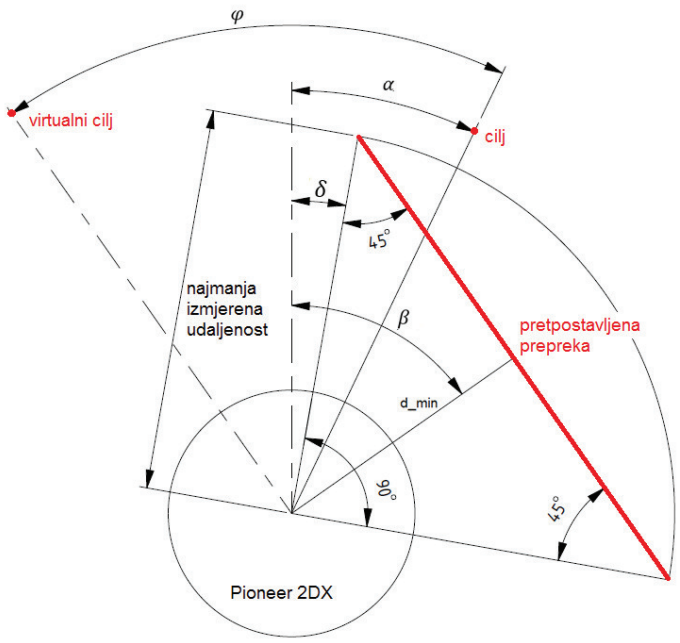
pri čemu je $\alpha > 0$ kada se cilj nalazi desno od robota, $\beta > 0$ kada se prepreka nalazi desno od robota a $sing(\beta)$ označava predznak kuta β. Slika 17 prikazuje jedan od mogućih slučajeva kada je dobiveni kut φ negativan, što znači da se robot mora rotirati u desno.



Slika 17: Određivanje virtualnog cilja pomoću kuta φ [14]

Zbog ograničenja sonarnog senzornog sustava na Pioneer 2DX mobilnom robotu koji može detektirati prepreke samo pod kutovima zadanim konstrukcijom robota i zbog činjenice da ne možemo pouzdano tvrditi da će dva senzora istovremeno detektirati prepreku manjih dimenzija, uvodi se modifikacija navedene metode.

Uvodi se pretpostavka da se brid najbliže detektirane prepreke nalazi pod kutem od 45° u odnosu na smjer očitanja δ koji je određen konstrukcijom robota i poznat je za svaki senzor.



Slika 18: Uvedena pretpostavka o orijentaciji prepreke

Nakon uvođenja pretpostavke kut β se sada račun na kao

$$\beta = 45° + \delta$$

Pa kut zakreta virtualnog cilja sada iznosi

$$\varphi = sign(\beta) * 90° - (45° - \delta - \alpha)$$

3.2 Algoritam za kalibraciju parametara tražene boje

Prije pokretanja upravljačkog algoritma potrebno je provesti kalibraciju HSV parametara pomoću kojih se pri obradi slike filtrira samo željena boja. Budući da boja nekog objekta može značajno ovisiti u uvjetima osvjetljenja okolnog prostora, kalibraciju je u pravilu potrebno provesti svaki put kada se robot nađe u novom radnom prostoru.

U tu svrhu izrađena je zasebna petlja u kojoj se pokreće samo vizijski sustav te se kreira interaktivni prozor s šest klizača pomoću kojih možemo u realnom vremenu podešavati minimalne i maksimalne HSV vrijednosti.

Algoritam za kalibraciju ugniježden je unutar $switch - case$ petlje i pokreće se odabirom druge opcije nakon pokretanja glavnog programa.

3.2.1 Osnovni dio algoritma

Osnovni dio algoritma za kalibraciju boje nalazi se ugniježden u drugom dijelu $switch - case$ i započinje nakon $case$ 2: naredbe. Odmah na početku algoritma korisniku se javlja da iz petlje može izaći držanjem tipke Ecs na tipkovnici računala.

```
cout << "Drzi ESC za izlaz iz petlje \n";
```

Nakon toga slijedi inicijalizacija objekata Mat klase koji sadrže originalnu c_image sliku dohvaćenu s kamere, transformiranu sliku $c\_HSVimage$ u HSV način zapisa boje i binarnu crno-bijelu sliku $c\_BINimage$ na kojoj su bijelom bojom označena samo područja na kojoj se nalazi praćena boja.

```
Mat c_image;
Mat c_HSVimage;
Mat c_BINimage;
```

Slijedi inicijalizacija HSV parametara koji se koriste u funkciji $inRange$ da bi se dobila binarna slika.

```
int c_H_min = 0;
int c_H_max = 179;
int c_S_min = 0;
int c_S_max = 256;
int c_V_min = 0;
int c_V_max = 256;
```

Zatim se funkcijom *namedWindow*() inicijalizira novi prozor i odmah potom klizači za podešavanje HSV parametara koji se nalaze u tom prozoru.

```
namedWindow(winS, WINDOW_AUTOSIZE);
createTrackbar( "H_MIN", winS, &c_H_min, c_H_max, slider_act);
createTrackbar( "H_MAX", winS, &c_H_max, c_H_max, slider_act);
createTrackbar( "S_MIN", winS, &c_S_min, c_S_max, slider_act);
createTrackbar( "S_MAX", winS, &c_S_max, c_S_max, slider_act);
createTrackbar( "V_MIN", winS, &c_V_min, c_V_max, slider_act);
createTrackbar( "V_MAX", winS, &c_V_max, c_V_max, slider_act);
```

Na kraju se još inicijalizira objekt klase *VideoCapture* te se odmah potom poziva metoda open() koja pokreće dohvaćanje slike s kamere vizijskog sustava. Prije pokretanja kalibracije još se metodom set() definira veličina prikaza slike s kamere.

```
VideoCapture c_cap;
c_cap.open(0);
c_cap.set(CV_CAP_PROP_FRAME_WIDTH,IMAGE_WIDTH);
c_cap.set(CV_CAP_PROP_FRAME_HEIGHT,IMAGE_HEIGHT);
```

Slijedi pokretanje beskonačne petlje za izvršavanje kalibracije.

```
while(1){
```

Prva naredba u petlji čeka sve dok se ne dohvati slika s kamere. Ovo je posebno važno pri pokretanju algoritma kada treba u obzir uzeti vrijeme koje je potrebno da bi kamera počela raditi.

```
while (!c_cap.read(c_image));
```

Slijedi konvertiranje ulazne slike s kamere koja dolazi u BGR zapisu u HSV zapis boja. Potom se vrši binarizacija slike funkcijom *inRange*() u crno-bijelu sliku pri čemu bijelom bojom ostaju označeni samo dijelovi slike na kojima je pronađena tražena boja.

```
cvtColor(c_image,c_HSVimage,COLOR_BGR2HSV);
inRange(c_HSVimage,Scalar(c_H_min,c_S_min,c_V_min),
        Scalar(c_H_max,c_S_max,c_V_max),c_BINimage);
```

U jednom od prozora prikazuje se binarna slika prije uklanjanja šuma

```
imshow(win4,c_BINimage);
```

Da se ukloni šum sa binarne slike poziva se funkcija $ERODE\_DILATE()$ koja je detaljnije opisana u 3.2.2 poglavlju.

```
ERODE_DILATE(c_BINimage);
```

Nizom poziva funkcije $imshow()$ prikazuju se sve dobivene slike u zasebnim prozorima.

```
imshow(win3,c_BINimage);
imshow(win1,c_image);
imshow(win2,c_HSVimage);
```

Na kraju beskonačne petlje nalazi se provjera o izlasku iz petlje. Iz petlje se izlazi ako je pri izvršavanju ovog upita bila pritisnuta tipka Esc. Funkcija $waitKey()$ obavlja još jednu važnu zadaću. Ona čeka za zadano vrijeme T i time daje upravljačkom računalu daje dovoljno vremena da prikaže sve tri slike na zaslonu.

```
        if (waitKey(T) == 27) break;
    }
```

Izlaskom iz beskonačne petlje gase se svi otvoreni prozori i izlazi se iz kalibracijskog dijela glavne $switch - case$ petlje.

```
        destroyAllWindows();
        break;
}
```

3.2.2 Funkcija za uklanjanje šuma ERODE_DILATE()

Zadaća ove funkcije je uklanjanje šuma s binarne crno-bijele slike.

Odmah nakon inicijalizacije same funkcije koja kao jedini ulazni parametar uzima referencu na postojeću binarni sliku, stvaraju se dva objekta klase Mat koji služe kao strukturalni elementi opisani u 2.4.8, 2.4.9 i 2.4.10. Kvaratnog su oblika. Veličina elementa za erodiranje slike je 3x3 točke a elementa za proširivanje slike 8x8 točaka.

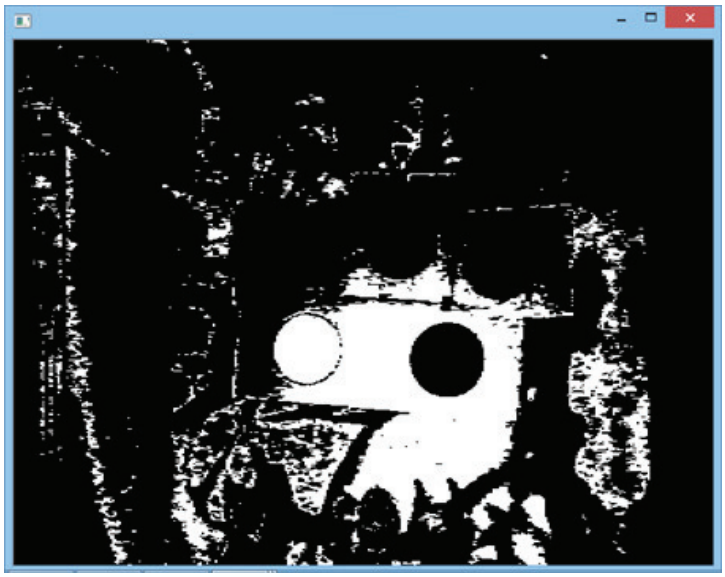
```
void ERODE_DILATE (Mat &image) {

    Mat erode_size = getStructuringElement( MORPH_RECT,Size(3,3));
    Mat dilate_size = getStructuringElement( MORPH_RECT,Size(8,8));
```

Nakon inicijalizacije strukturalnih elemenata slijedi poziv funkcija $erode()$ i $dilate()$. Svaka funkcija se poziva dva puta radi maksimalnog uklanjanja šuma.

```
        erode(image,image,erode_size);
        erode(image,image,erode_size);
        dilate(image,image,dilate_size);
        dilate(image,image,dilate_size);
}
```

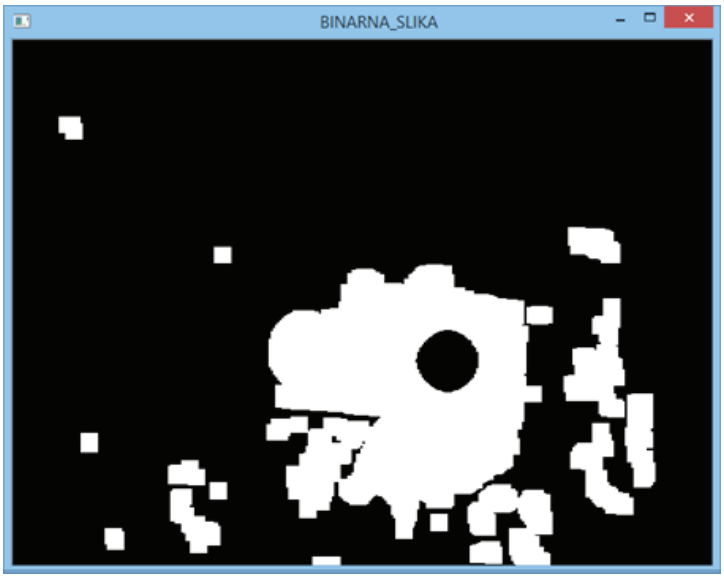
Način rada ERODE_DILATE() funkcije prikazan je na slici dohvaćenoj s kamere na kojoj se želi filtrirati samo objekt zelena boja.

Slika 19 pokazuje stanje dohvaćene slike na kojoj su filtrirane samo nijanse (Hue) zelene boje. Dok Slika 20 pokazuju stanje slike nakon poziva $ERODE\_DILATE()$ funkcije.



Slika 19: Binarna slika prije poziva ERODE_DILATE() funkcije

Vidljivo je da se nakon poziva funkcije šum znatno smanjio. Ostatak boje koja ne pripada objektu koji se želi pratiti može se filtrirati daljnjim podešavanjem parametara za zasićenost i svjetlinu boje.



Slika 20: Binarna slika nakon poziva ERODE_DILATE() funkcije

3.3 Programski kod upravljačkog algoritma

Upravljački algoritam ugniježden je unutar switch-case petlje. Pokreće se odabirom druge opcije nakon pokretanja glavnog programa.

3.3.1 Glavni dio upravljačkog algoritma

Glavni dio upravljačkog algoritma započinje inicijalizacijom Aria programskih knjižnica, objekata za upravljanje i komunikaciju s robotom i sonarnim senzorima te povezivanjem objekta sonar i robot.

```
case 1:
{       Aria::init();
        ArRobot robot;
        ArSonarDevice sonar;
        robot.addRangeDevice(&sonar);
```

Slijedi priprema potrebnih argumenata za spajanje na robota i njihovo prosljeđivanje objektu klase *ArRobotConnector*.

```
        ArArgumentParser parser(&argc, argv);
        parser.loadDefaultArguments();
        ArRobotConnector robotConnector(&parser, &robot);
        robotConnector.parseArgs();
```

Metodom $connectRobot()$ pokreće se spajanje na robota, te se istovremeno provjerava da li je spajanje bilo uspješno. Ukoliko dođe do pogreške pri spajanju prekida se izvođenje prugama.

```
if (!robotConnector.connectRobot())
{
        cout << ("Pogreska pri povezivanju na robot.\n");
        Aria::exit(1);
        return 0;
}
```

Da bi slanje naredbi i primanje očitanja sa sonarnih senzora bilo moguće, potrebno je pokrenuti asinkronu komunikaciju između robota i računala. To označava stvaranje zasebne procesne niti za komunikaciju s robotom.

```
robot.runAsync(true);
```

Slijedi inicijalizacija jednodimenzionalnog niza u kojeg se spremaju podaci očitanja s sonarnih senzora pomoću funkcije $sonar\_readings()$.

```
float son_array[14];
```

Zatim se definira objekt klase VideoCapture te se pokreće spajanje programa na kameru. Odmah poslije spajanja slijedi provjera uspješnog spajanja pomoću metode isOpened(). Pomoću metode set() postavlja se željena rezolucija snimanja, čiji izbor ovisi o korištenoj kameri i brzini glavnog procesora. OpenCV funkcijom namedWindow() još se inicijaliziraju tri prozora na kojima će se prikazivati originala slika, slika prebačena u HSV načina zapisa boje i binarna crno-bijela slika.

```
VideoCapture cap;
cap.open(0);
if(!cap.isOpened()){
        cout << "Pogreska pri povezivanju na kameru.\n";
        Aria::exit(1);
}
cap.set(CV_CAP_PROP_FRAME_WIDTH, IMAGE_WIDTH);
cap.set(CV_CAP_PROP_FRAME_HEIGHT, IMAGE_HEIGHT);
namedWindow(win1,CV_WINDOW_AUTOSIZE);
namedWindow(win2,CV_WINDOW_AUTOSIZE);
namedWindow(win3,CV_WINDOW_AUTOSIZE);
```

Još se inicijaliziraju HSV parametri potrebni za filtriranje slike i varijable x i y koje sadrže lokaciju traženog objekta na obrađenoj slici.

```
int x = -1, y = -1;;
// RED
H_min = 0;
H_max = 4;
S_min = 130;
S_max = 203;
V_min = 139;
V_max = 208;
```

Prije pokretanja beskonačne petlje u kojoj će se izvršavati upravljanje nad robotom još se na robot šalje naredba da se omogući korištenje motora, izvršava se prvo očitanje sonarnih senzora i javlja se korisniku na koji način može izaći iz beskonačne petlje.

```
robot.enableMotors();
sonar_readings(sonar, son_array, son_x, son_y);
cout << "Drzi ESC za izlaz iz petlje \n";

while(1)
{
```

Slijedi poziv funkcija za obradu slike koje su detaljno objašnjene u poglavlju 3.2.

```
Mat image;
Mat HSVimage;
Mat BINimage;
while(!cap.read(image));
cvtColor(image,HSVimage,COLOR_BGR2HSV);
inRange(HSVimage,Scalar(H_min,S_min,V_min),
                Scalar(H_max,S_max,V_max),BINimage);
ERODE_DILATE(BINimage);
```

Pomoću klase $Moments$ i funkcije $moments()$ iz dobivene binarne slike računaju se prostorni momenti u horizontalnom smjeru $BINmoments.m10$, vertikalnom smjeru $BINmoments.m01$ i dobiva se vrijednost površine koju zauzima bijela boja na binarnoj slici $BINmoments.m00$.

```
Moments BINmoments = moments(BINimage);
double BIN_M01 = BINmoments.m01;
double BIN_M10 = BINmoments.m10;
double BIN_area = BINmoments.m00;
```

Provjerava se da li je površina koju zauzima bijela boja dovoljno velika, jer se u suprotnom smatra da se na slici javlja samo šum. Ako je površina dovoljno velika, računaju se koordinate težišta objekta prikazanog bijelom bojom iz kojih se kasnije računa smjer gibanja robota.

```
if (BIN_area > 10000) {
        x = BIN_M10 / BIN_area;
        y = BIN_M01 / BIN_area;
}
else x = y = -1;
```

Ako je detektiran objekt na binarnoj slici pozivaju se funkcije $forward()$ i $new\_goal()$ koje su zadužene za pokretanje robota, praćenje objekta i izbjegavanje prepreka modificiranom metodom tangencijalnog izbjegavanja.

```
if (x != -1 ) {
        forward(robot, son_array);
        new_goal(robot, son_array, x);
}
else robot.stop();
```

Pomoću funkcije $imshow()$ prikazuju se slike u zadanim prozorima.

```
imshow(win3, BINimage);
imshow(win1, image);
imshow(win2, HSVimage);
```

Na kraju beskonačne petlje izvršava se novo očitanje sonarnih senzora i provjerava se želi li korisnik izaći iz petlje pritiskom na tipku Esc.

```
sonar_readings(sonar, son_array, son_x, son_y);
if (waitKey(T) == 27) {
        cout << "Izlaz iz petlje" << endl;
        break;
}
```

Po izlasku iz beskonačne petlje zaustavlja se robot i isključuju se motori, zatvaraju se svi prozori korišteni za prikaz videa i zaustavlja se komunikacija s robotom.

```
        }
        robot.stop();
        robot.disableMotors();
        destroyAllWindows();
        Aria::exit(1);
        break;
}
```

3.3.2 Funkcija sonar_readings()

Funkcija $sonar\_readings()$ očitava trenutna mjerenja na sonarnim senzorima. Pri pozivanju traži objekt klase $ArSonar$ i niza u kojeg sprema očitanja sa senzora.

```
void sonar_readings(ArSonarDevice &sonar, float son_array[14])
{
```

Očitavanje trenutnih mjerenja sa sonarnih senzora obavlja se pomoću metode $currentReadingPolar()$ tako da metodi proslijedimo dva kuta između kojih želimo očitati izmjerenu udaljenost. Slika 7 prikazuje detaljan raspored sonarnih senzora i pripadajućih kutova. Poslije svakog poziva za očitanje mjerenja važno je naredbom $\_sleep()$ dati robotu dovoljno vremena da odgovori na danu naredbu.

```
son_array[0] = sonar.currentReadingPolar(-95,-80) - robot_radius;
_sleep(9);
for (int i=0;i<6;i++) {
        son_array[i+1] = sonar.currentReadingPolar(-60+20*i,-40+20*i) - robot_radius;
        _sleep(9);
}
son_array[7] = sonar.currentReadingPolar(80, 95) - robot_radius;
_sleep(9);
```

Slijedi dio funkcije koji pretražuje dobivena mjerenja i na lokacijama 8,9,10,11,12,13 pamti najmanje izmjerene udaljenosti i lokacije na kojima su izmjerene.

```
son_array[8] = 6000;
son_array[9] = -1;
son_array[10] = 6000;
son_array[11] = -1;
son_array[12] = 6000;
son_array[13] = -1;
for (int i=0;i<8;i++) {
        if (son_array[i] < son_array[8]) { //minimalna udaljenost (8) i položaj (9)
                son_array[8] = son_array[i];
                son_array[9] = i;
        }
        if ((i == 2) || (i == 3) || (i == 4) || (i == 5))     //minimalna izmjerena udaljenost
                if (son_array[i] < son_array[10]) {    //ispred robota (10) i položaj (11)
                        son_array[10] = son_array[i];
                        son_array[11] = i;
                }
        if ((i == 0) || (i == 1) || (i == 6) || (i == 7))    //minimalna izmjerena bočna
                if (son_array[i] < son_array[12]) {   //udaljenost (12) i položaj (13)
                        son_array[12] = son_array[i];
                        son_array[13] = i; }}}
```

3.3.3 Funkcija forward()

Funkcija $forward()$ zadužena je za određivanje linearne brzine robota. Pri pozivu funkcija kao ulazne parametre zahtjeva objekt klase $ArRobot$ i niz u kojem se nalaze očitanja sonarnih senzora. Funkcija određuje brzinu tako da je brzina maksimalna dopuštena (u ovom slučaju $200\ mm/s$) dok se ispred robota ne detektira prepreka unutar zone od $600\ mm$. Funkcija tada počinje linearno usporavati robota i ako se detektira prepreka na $300\ mm$ brzina robota pada na nulu. Brzina kretanja robota zadaje se metodom $setVel()$.

```
void forward (ArRobot &robot, float son_array[14]) {
    float VL;
    if (son_array[10] < dist_danger*2)
        VL = ((VL_max * son_array[10]) / dist_danger) - VL_max;
    else VL = VL_max;

    robot.setVel(VL);
}
```

3.3.4 Funkcija new_goal()

Funkcija $new\_goal()$ računa lokaciju trenutnog cilja robota. Kao ulaz zahtjeva objekt klase ArRobot, niz sa očitanjima sonarnih senzora i horizontalnu lokaciju objekta koji se prati. Varijabla PHI sadrži informaciju o trenutnom cilju robota.

```
void new_goal(ArRobot &robot, float son_array[14], int x) {
        float PHI;
```

Ako je ispred robota detektirana prepreka u zoni od $600mm$ funkcija računa lokaciju cilja prema opisu danom u poglavlju 3.1.

```
    if (son_array[10] < dist_danger*2) {
        int delta;
        //određivanje kuta na kojem se nalazi najmanje očitanje
        if (abs(3.5 - son_array[11]) < 1) delta = 10;
        else if (abs(3.5 - son_array[11]) > 2) delta = 50;
        else delta = 30;
        //određivanje predznaka kuta beta
        int sign_B;
        if ((3.5 - son_array[11]) > 0) sign_B = 1;
        else sign_B = -1;
        int beta = sign_B * (45 + delta); //računanje kuta beta
        float alpha = sign_B * (((IMAGE_WIDTH / 2) - x)
                / (IMAGE_WIDTH / CAM_ANGLE));
        PHI = sign_B * 90 - (beta - alpha); //određivanje lokacije virtualnog cilja
    }
```

Ako ispred robota nije detektirana prepreka, kut φ pod kojim se nalazi praćeni objekt računa se kao razlika lokacije objekta na slici i polovice ukupne rezolucije slike. Dobivena razlika se zatim podjeli sa količnikom horizontalne širine slike i vidnog kuta kamere.

$$\varphi = \alpha = \frac{\left(\frac{Horizontalna\ Rezolucija}{2} - Horizontalna\ Lokacija\ \ Objekta\right)}{\frac{Horizontalna\ Rezolucija}{Vidni\ Kut\ Kamere}}$$

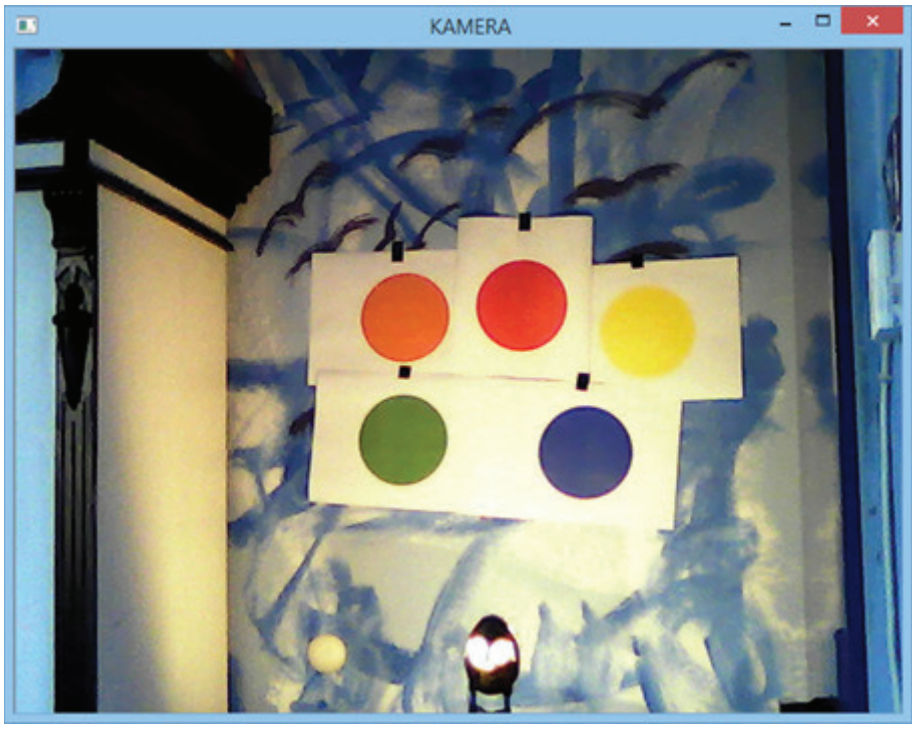
```
    else {
        if (abs((IMAGE_WIDTH / 2) - x) > 9) {
            PHI = ((IMAGE_WIDTH / 2) - x) / (IMAGE_WIDTH / CAM_ANGLE);
        }
        else PHI = 0;
    }
    robot.setDeltaHeading(PHI);
}
```

4 TESTIRANJE UPRAVLJAČKOG ALGORITMA

Testiranje upravljačkog algoritma izvršeno je na simulaciji pomoću MobileSim aplikacije i na Pioneer 2 mobilnom robotu. U oba slučaja prije testiranja izvršena je kalibracija HSV parametara da bi se dobili optimalni parametri za uvjete prostora u kojem je izvršeno testiranje.

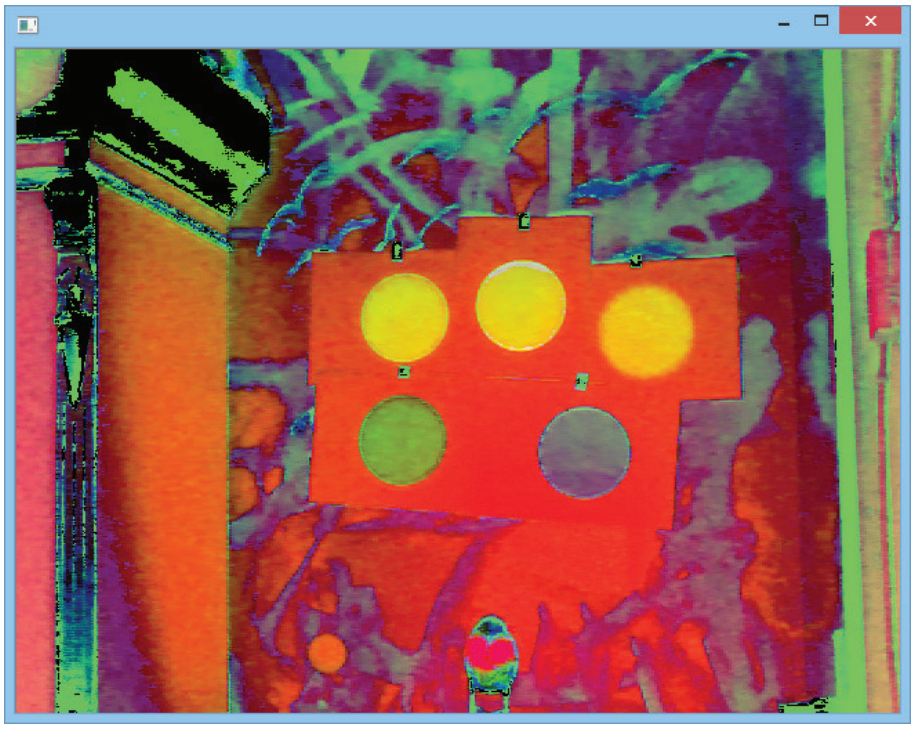
4.1 Postupak kalibracije HSV parametara

Pokretanjem algoritma za kalibraciju automatski se pokreće i dohvaćanje slike s kamere. U četiri prozora se prikazuju originalna slika dohvaćena direktno s kamere (Slika 21), originalna slika konvertirana u HSV način zapisa boja (Slika 22), binarizirana crno-bijela slika bez uklonjenog šuma (Slika 24) i binarizirana crno-bijela slika s uklonjenim šumom (Slika 25).



Slika 21: Originalna slika dohvaćena s kamere

Cilj kalibracije HSV parametara je da na finalnoj binarnoj slici bijelom bojom bude označen samo onaj dio slike ne kojem se nalazi objekt kojeg želimo pratiti. Optimalan način da se dođe do toga rezultata je da kalibraciju započnemo sa podešavanjem minimalne i maksimalne vrijednosti parametra za nijansu boje (H_MIN i H_MAX). Kada na binarnoj slici bijelom bojom ostanu označene samo nijanse tražena boje podešavanjem parametra za zasićenost (S_MIN i S_MAX) i svjetlinu (V_MIN i V_MAX) uklanjamo ostatak nepotrebne boje i šuma tako da na izlaznoj slici ostaje samo objekt kojeg želimo pratiti (Slika 25).



Slika 22: Originalna slika transformirana u HSV način zapisa boje

Slika 23: Podešavanje klizača za filtraciju crvene boje

Zbog načina na koji OpenCV definira HSV način zapisa boje klizač nijanse je definiran skalom 0…179 koja zapravo definira raspon od 0…359° dok su klizači za zasićenost i svjetlinu boje skalirani u rasponu 0...255 a definiraju raspon zasićenosti i svjetline 0…100%.

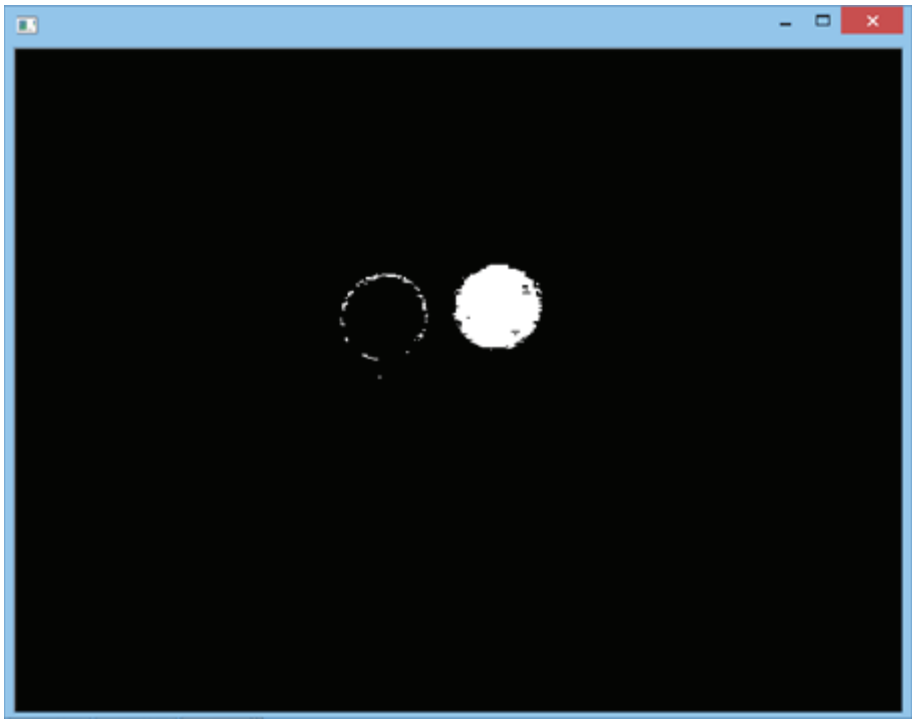
U prikazanom primjeru cilj je bio filtrirati krug crvene boje. Namještanjem parametra za nijansu boje H u raspon vrijednosti od 0 do 5 na binarnoj slici ostale su samo nijanse crvene boje. Ali budući da se crvena i narančasta boja nalaze jedna uz drugu na HSV skali, narančasti

krug uzrokovao je neželjeni šum koji nije bilo moguće ukloniti ni dodatnim podešavanjem parametra zasićenja i svjetline (Slika 24). Zato se pozivanjem funkcije *ERODE_DILATE*() na sliku prvo primjenjuje proces erozije koji djeluje na sva područja slike na kojima dolaze u doticaj crna u bijela boja. Kako je područje šuma jako usko a erozija djeluje na površinu veličine 3x3 točke, došlo je do potpunog brisanja šuma sa slike. Nakon erozije funkcija *ERODE_DILATE*() izvršila je još proces proširivanja postojećih bijelih područja kako bi povećala površinu traženog objekta i time olakšala njegovo praćenje (Slika 25).

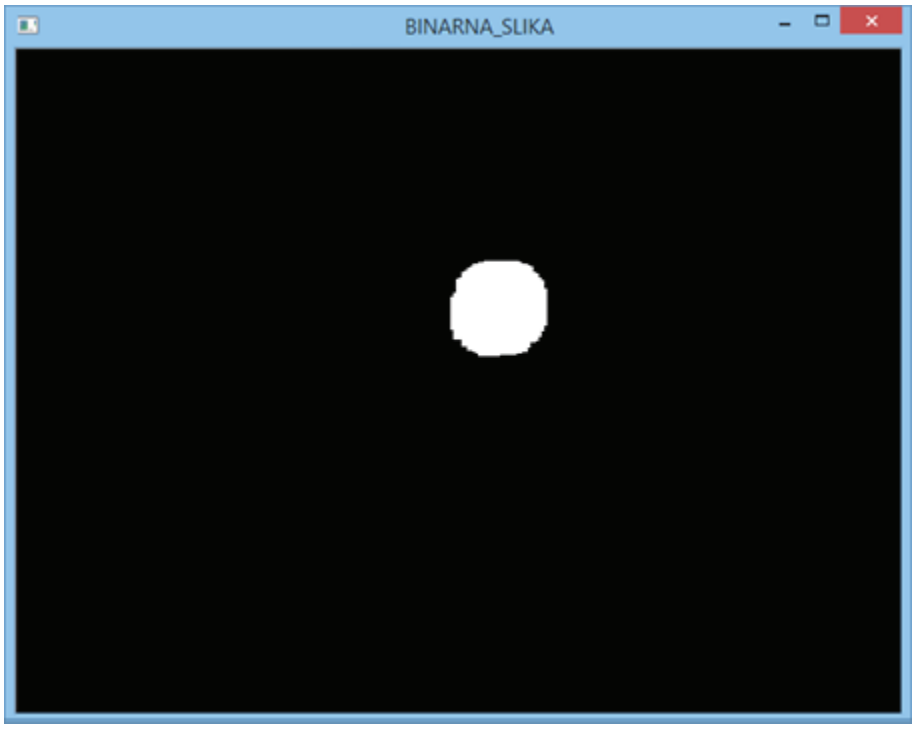
Dobiveni parametri za praćenje crvene boje su:

Tablica 2: Dobiveni parametri za praćenje objekata crvene boje

Parametar	Minimum	Maksimum
Nijansa – H	0	5
Zasićenost – S	156	228
Svjetlina – V	179	255



Slika 24: Binarna crno-bijela slika prije uklanjanja šuma



Slika 25: Binarna crno-bijela slika nakon uklanjanja šuma

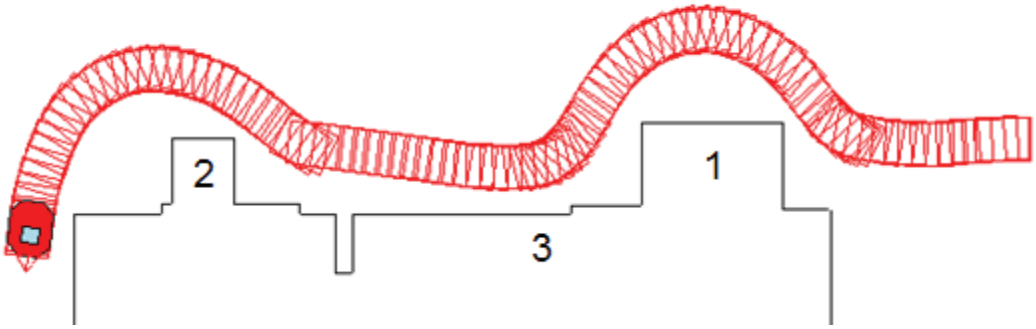
Slika 25 pokazuje krajnji rezultat kalibracije HSV parametara korištenih pri binarizacija slike. Može se zaključiti da je kalibracija parametara bila uspješna jer je na slici ostao samo čisti obris kruga crvene boje.

4.2 Testiranje upravljačkog algoritma pomoću simulacijskog modela

Testiranje na simulacijskom modelu robota izvršeno je pomoću MobileSima aplikacije. Prostor u kojem se je odvijala simulacija preuzet je iz ActiveMediaLab.map datoteke.

Pitanja robota kontroliranja je pomoću CANYON CNE-CWC1 kamere i zelenog kruga koji je služio kao praćeni objekt.

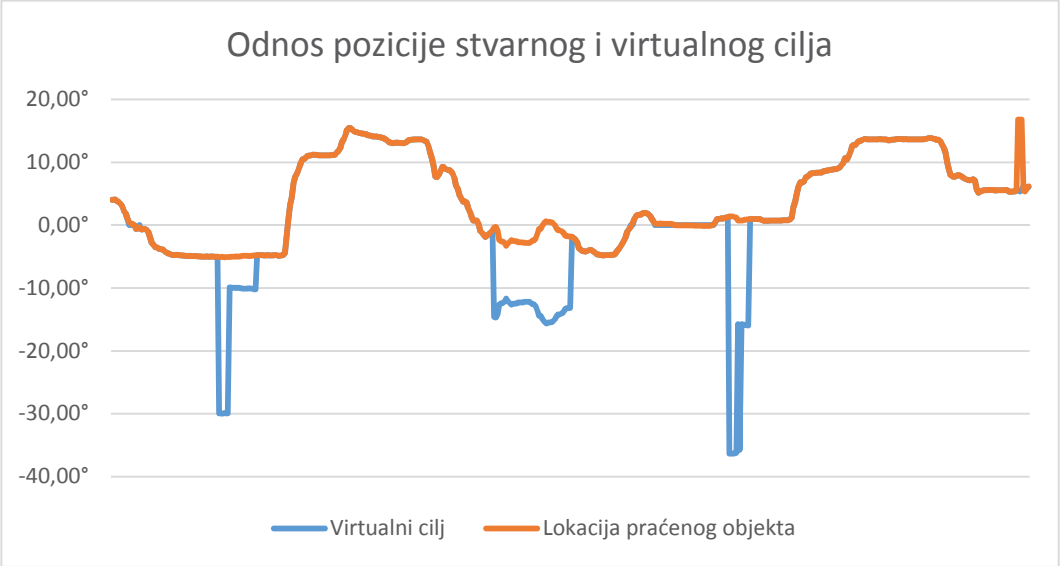
Slika 26: Kamera i traženi objekt pri testiranju algoritma na simulacijskom modelu



Slika 27: Putanja kretanja robota pri testiranju algoritma na simulacijskom modelu

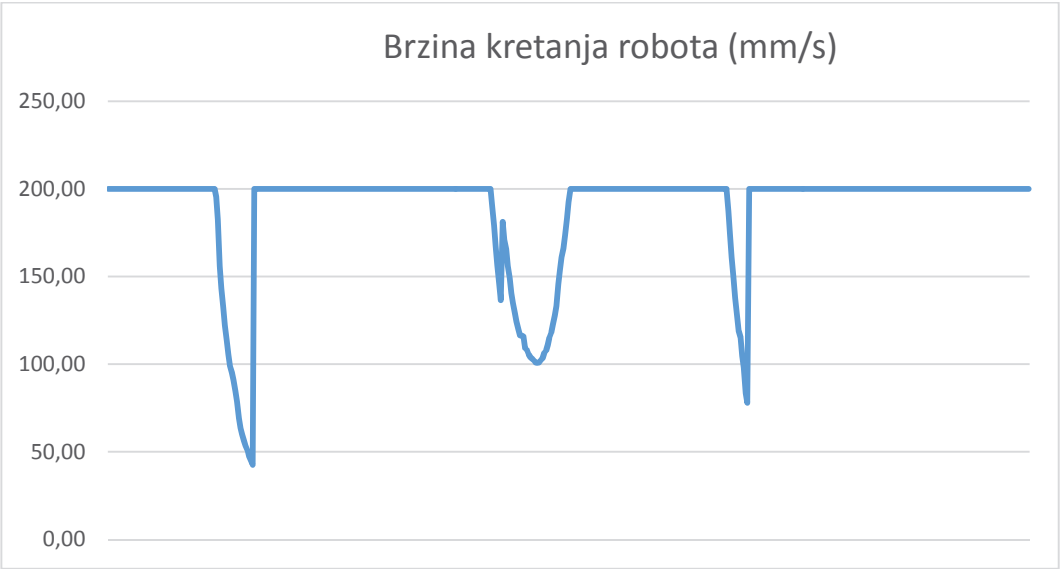
Kao što je to vidljivo iz putanje, robot je namjerno usmjeravan na izbočenja 1 i 2 te je usmjeren prema zidu na mjestu 3. Na sva tri mjesta robot je izbjegao prepreku te se nakon izbjegavanja prepreke počeo ponovno gibati prema lokaciji na kojoj je kamera detektirala zadani objekt.

Slika 28 pokazuje kut između stvarnog i virtualnog cilja u odnosu na središte robota. Vidljivo je da se na tri mjesta na kojima je robot namjerno usmjeren u prepreku stvarni i virtualni cilj razilaze i u tim slučajevima robot prati virtualni cilj da bi izbjegnuo detektiranu prepreku.



Slika 28: Pozicija stvarnog i virtualnog cilja u odnosu na središte simuliranog robota

Slika 29 prikazuje brzinu kretanja robota u mm/s. Opet se može primijetiti da na tri mjesta na kojima je robot usmjeren u prepreku brzina pada te se ponovno vraća na zadani maksimum od $200mm/s$ kad ispred robota više nema prepreke.

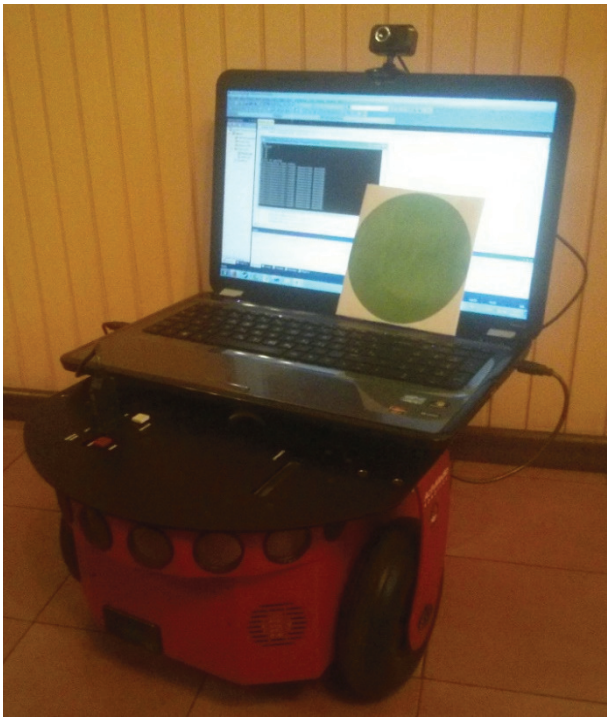


Slika 29: Brzina kretanja robota pri testiranju algoritma na simulacijskom modelu

4.3 Test upravljačkoga algoritma na Pioneer 2-DX mobilnom robotu

Test na Pioneer 2 mobilnom robotu izvršen je na izrađenom poligonu s prijenosnim računalom kao upravljačkim računalom i CANYON CNE-CWC1 kamerom kao ulazom vizijskog sustava.

Slika 30: Testni „poligon“

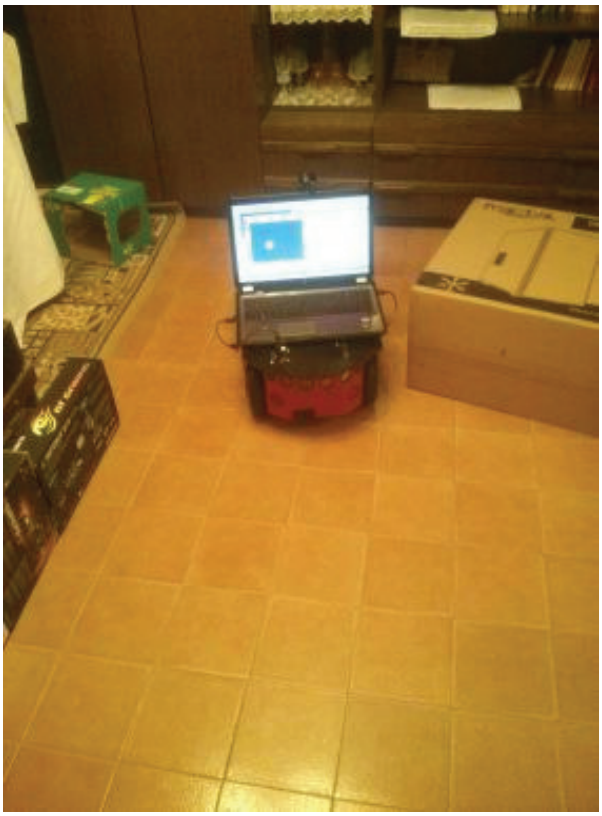


Slika 31: Pioneer 2 mobilni robot s prijenosnim računalom kao upravljačkim računalom

Robot kreće iz početnog položaja (Slika 32 lijevo). Ubrzo nailazi na prvu prepreku i počinje rotirati udesno (Slika 32 desno).

Slika 32: Početna pozicija robota (lijevo), prva prepreka (desno)

Nakon uspješnog zaobilaženja prve prepreke (Slika 33) robot nastavlja slijediti praćeni objekt koji ga vodi do druge prepreke (Slika 34). Nakon što sonarni senzori unoče prepreku unutar zadane zone od 600 mm robot počinje skretati ulijevo da bi izbjegnuo i drugu prepreku (Slika 35). Robot prolazi kraj druge prepreke (Slika 36) i nastavlja slijediti praćeni objekt (Slika 37).



Slika 33: Uspješno zaobilaženje prve prepreke

Slika 34: Nastavak kretanja prema drugoj prepreci

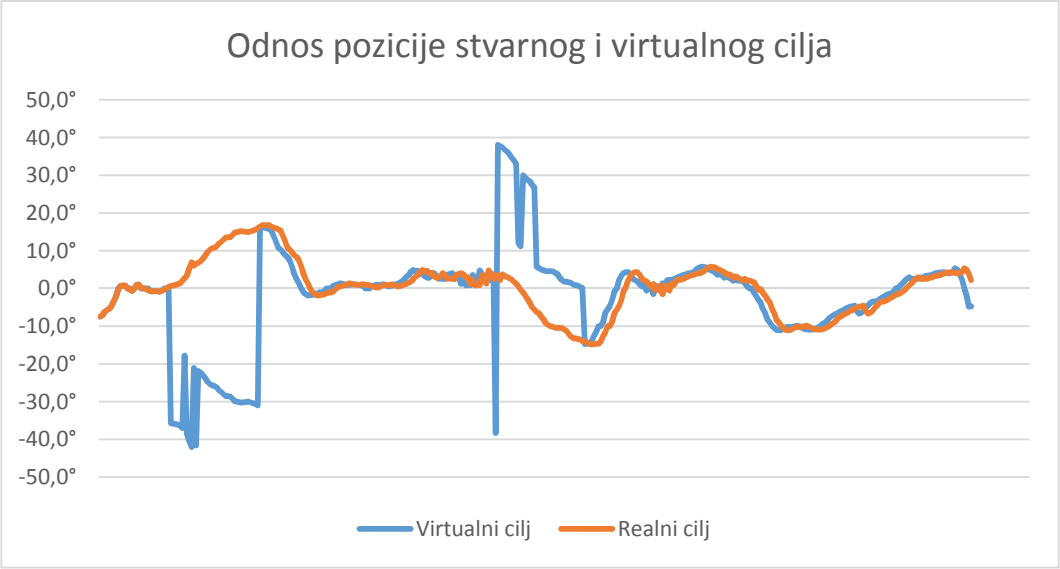
Slika 35: Početak zaobilaženja druge prepreke

Slika 36: Prolazak pokraj druge prepreke

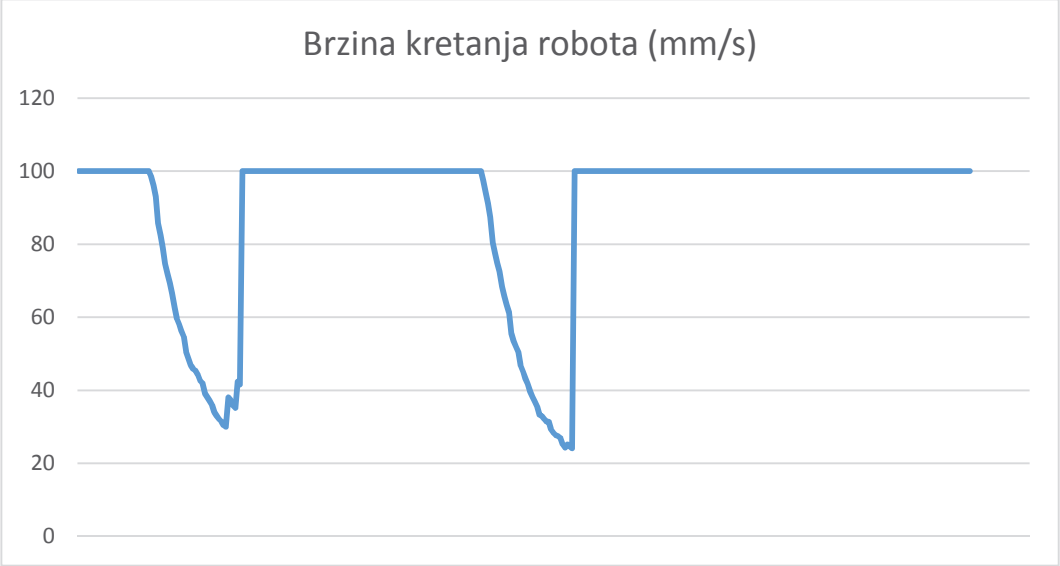
Slika 37: Uspješno zaobilaženje druge prepreke

Slika 38: Približna pitanja kojim je robot putovao

Kao što je to bio slučaj i u simulaciji, virtualni i realni cilj se poklapaju (Slika 39), osim u trenucima kada je ispred robota detektirana prepreka. U tom slučaju pada i brzina kojom se mobilni robot giba (Slika 40). Kada ispred robota više nema prepreke, robot se nastavlja kretati maksimalnom dopuštenom brzinom od 100 mm/s.



Slika 39: : Pozicija stvarnog i virtualnog cilja u odnosu na središte robota



Slika 40: Brzina kretanja pri testiranju algoritma na Pioneer 2 mobilnom robotu

Upravljači algoritam uspješno je testiran i na simulacijskom modelu i na realnom testu pomoću Pioneer 2 platforme. U oba slučaja robot je pokazao očekivano ponašanje. Kada je vizijski sustav detektirao objekt zadane boje robot ga je počeo slijediti sve dok se ispred njega nije pojavila prepreka. Nakon što je prepreka detektirana slijedilo je usporavanje brzine kretanja te je istovremeno upravljački algoritam odredio novi virtualni cilj prema kojem je robot počeo rotirati i slijediti ti ga da bi na taj način izbjegnuo prepreku koja mu se našla na putu.

5 ZAKLJUČAK

Mobilna robotika područje je neprestanog znanstvenog i komercijalnog razvoja u kojem se na svakodnevnoj bazi poduzimaju novi koraci prema novim i boljim rješenjima. Unatoč tome, put potreban za prelazak iz doba industrijskih robota koji za svoj rad zahtijevaju relativno strukturiranu okolinu u doba uslužnih i personaliziranih mobilnih robota koji će se u interakciji sa okolinom moći uspoređivati s čovjekom još je dug.

Projektirani upravljački algoritam uspješno je obavljao zamišljeni zadatak praćenja zadanog cilja i istovremenog izbjegavanja prepreka kroz nepoznatu okolinu. No, efikasnost s kojom je algoritam obavljao taj zadatak često je uvelike ovisila o uvjetima u neposrednoj okolini zbog čega je za uspješnu navigaciju mobilni robot još uvijek zahtijevao donekle strukturiranu okolinu.

Vizijski sustav bio je osjetljiv na uvjete osvjetljenja i refleksivnost materijala zbog čega je pri značajnim promjenama uvjeta rada bilo potrebno ponovno izvršiti kalibraciju parametara za prepoznavanje zadane boje. Rješenje ovog problema mogla bi biti nadogradnja vizijskog sustava mogućnošću prepoznavanja oblika čime bi se objekt kojeg robot prati definirao dodatnim parametrom koji bi smanjio utjecaj šuma na vizijski sustav te bi se omogućilo istovremeno praćenje više objekata iste boje.

Senzori za mjerenje udaljenosti još su jedno mjesto na kojem bi se mogao napraviti veliki napredak u točnosti detekcije i izbjegavanja prepreka. Povećanjem broja senzora ili korištenjem senzora koji pružaju mogućnost mjerenja veće rezolucije, npr. mjerenje udaljenosti pomoću lasera ili korištenjem stereo-vida, detektirane prepreke bi bile puno preciznije definirane što bi upravljačkom algoritmu omogućilo preciznije određivanje putanje izbjegavanja prepreka i detekciju prepreka manjih dimenzija.

Unatoč problemima s kojima su mobilni roboti danas suočeni još uvijek smatram da mobilna robotika ima svijetlu budućnost ispred sebe. Raspon zadataka koje bi mobilni roboti mogli obavljati je ogroman, od obavljanja kućanskih poslova, autonomnog javnog ili osobnog prijevoza pa sve do istraživanja svemira.

Zbog kompleksnosti područja, velikih mogućnosti napretka i raspona potencijalnih primjena, mobilna robotika je svakako područje na kojem bi volio nastaviti s radom i nakon završetka obrazovanja.

LITERATURA

[1] Crneković, M.: Predavanja iz kolegija Robotika; Fakultet Strojarstva i Brodogradnje, Zagreb, ak. god. 2014./2015.

[2] Petrović, I.: Mobilna robotika-predavanja; Fakultet elektrotehnike i računarstva, Zagreb, ak. god. 2014./2015.

[3] http://www.cs.cmu.edu/~reliability/images/MER2.jpg, zadnji pristup 4.7.2015.

[4] http://www-lar.deis.unibo.it/equipments/p2dx/images/Pioneer2DX_2.jpg, zadnji pristup 4.7.2015.

[5] Pioneer 3 & Pioneer 2 H8-Series Operations Manual, ActivMedia Robotics, 2003.

[6] MobileSim - The Adept MobileRobots Simulator Manual, Adept Technology, 2012.

[7] ARIA Developer's Reference Manual, Adept Technology, 2014.

[8] OpenCV API Reference, http://docs.opencv.org/, zadnji pristup 4.7.2015.

[9] Soulemane, M.: HSV a preferred color model in Computer Vision?, http://smoumie.blogspot.com/2013/12/hsv-preferred-color-model-of-computer.html, zadnji pristup 4.7.2015.

[10] Introduction to color theory; New Mexico Institute of Mining and Technology,

[11] http://infohost.nmt.edu/tcc/help/pubs/colortheory/web/hsv.html, zadnji pristup 4.7.2015.

[12] https://www.visualstudio.com/, zadnji pristup 4.7.2015.

[13] http://www.instar-informatika.hr/slike/velike/enhanced-3-megapixels-resolution-webcam-CNE-CWC1.jpg, , zadnji pristup 6.7.2015.

[14] Ferreira, A., Pereira, F. G., Vassallo, R. F., Filho, T. F. B., Filho, M. S.: An approach to avoid obstacles in mobile robot navigation: the tangential escape; Universidade Federal do Espirito Santo, Brazil, 2008.

[15] http://youtu.be/8HkKQ0l_EvI, zadnji pristup 6.7.2015.

[16] Hounslow, K: Tutorial: Real-Time Object Tracking Using OpenCV, 2013. https://www.youtube.com/watch?v=bSeFrPrqZ2A, zadnji pristup 4.7.2015.

[17] http://colorizer.org/, zadnji pristup 4.7.2015.

[18] Agoston, M. K.: Computer Graphics and Geometric Modelling: Implementation & Algorithms, Springer Science & Business Media, 2005.

[19] Joblove, G. H., Greenberg, D.: COLOR SPACES FOR COMPUTER GRAPHICS, Cornell University, 1978.

[20] Šribar, J., Motnik, B.: Demistificirani C++, 2. izdanje, Element, Zagreb, 2003.

PRILOZI

I. Kod upravljačkog algoritma

II. CD-R disc

PRILOG I – Programski kod upravljačkog algoritma

```
#include "StdAfx.h"

#include "Aria.h"

#include <opencv\highgui.h>
#include <opencv\cv.h>

#include <math.h>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include <string>
#include <fstream>
#include <cstdio>
#include <ctime>

using namespace cv;
using namespace std;

#define PI 3.14159265
#define robot_radius 135.0
#define VL_max 200
#define VR_max 15
#define dist_danger 300
#define T 5
#define SLEEP 10

int H_min = 0;
int H_max = 179;
int S_min = 0;
int S_max = 255;
int V_min = 0;
int V_max = 255;

ofstream rot;
ofstream v;
ofstream xpos;

const int IMAGE_WIDTH = 640;
const int IMAGE_HEIGHT = 480;
const float CAM_ANGLE = 33.5; //CANYON
//const float CAM_ANGLE = 47.3; //HP web cam

const string win1 = "KAMERA";
const string win2 = "HSV_SLIKA";
```

```
const string win3 = "BINARNA_SLIKA";
const string win4 = "BINARNA_SLIKA_BEZ_ERODE_DILATE";
const string winS = "PODESAVANJE_FILTERA";

//=============================================================
//  ERODE_DILATE
//=============================================================

void ERODE_DILATE (Mat &image) {
        Mat erode_size = getStructuringElement( MORPH_RECT,Size(3,3));
        Mat dilate_size = getStructuringElement( MORPH_RECT,Size(8,8));

        erode(image,image,erode_size);
        erode(image,image,erode_size);

        dilate(image,image,dilate_size);
        dilate(image,image,dilate_size);
}
//=============================================================/
/ SLAJDER FUNKCIJA
//=============================================================
void slider_act( int, void* ){
        //placeholder
}
//=============================================================
// SONAR_READINGS
//=============================================================
//Funkcija koja vraca ocitanja osam sonarnih senzora u obliku 1D polja
//
// Zbog hardverskog ograničenja sonarnih senzora na mjestima 0 i 7 ne mjeriti
        udaljenosti manje od 250mm (230mm)
// a na mjestima 1-6 udaljenosti manje od 200mm (180mm).
void sonar_readings(ArSonarDevice &sonar, float son_array[14],
                                        float son_x[8], float son_y[8])
{
        //ocitavanje mjerenja sa sonara i upisivanje u niz (0-7)
        son_array[0] = sonar.currentReadingPolar(-95,-80) - robot_radius;
        _sleep(SLEEP);

        for (int i=0;i<6;i++) {
                son_array[i+1] = sonar.currentReadingPolar(-60 + 20 * i, -40 + 20 * i)
                                                                        - robot_radius;
                _sleep(SLEEP);
        }

        son_array[7] = sonar.currentReadingPolar(80, 95) - robot_radius;
        _sleep(SLEEP);
```

```
        son_array[8] = 6000;
        son_array[9] = -1;
        son_array[10] = 6000;
        son_array[11] = -1;
        son_array[12] = 6000;
        son_array[13] = -1;
        for (int i=0;i<8;i++)
        {
                if (son_array[i] < son_array[8]) { //minimalna udaljenost (8) i položaj (9)
                        son_array[8] = son_array[i];
                        son_array[9] = i;
                }

                if (i>1 && i<6) //minimalna izmjerena udaljenost -
                                                        ispred robota (10) i položaj (11)
                        if (son_array[i] < son_array[10])
                        {
                                son_array[10] = son_array[i];
                                son_array[11] = i;
                        }

                if ((i == 0) || (i == 1) || (i == 6) || (i == 7)) //minimalna izmjerena bočna
                                                        udaljenost (12) i položaj (13)
                        if (son_array[i] < son_array[12])
                        {
                                son_array[12] = son_array[i];
                                son_array[13] = i;
                        }


                son_x[0] = son_x[7] = 0;
                son_y[0] = -son_array[0];
                son_y[7] = son_array[7];

                for (int i=1; i<7; i++) {
                        son_x[i] = son_array[i] * sin((20 + 20. * i)*PI/180);
                        son_y[i] = son_array[i] * cos((200 + 20. * i)*PI/180);
                }

        }
}
//==========================================================
//  FORWARD
//==========================================================
void forward (ArRobot &robot, float son_array[14]) {
        float VL;

        if (son_array[10] < dist_danger*2)
```

```
                VL = ((VL_max * son_array[10]) / dist_danger) - VL_max;
        else VL = VL_max;
        robot.setVel(VL);

        v << VL << endl;
}
//============================================================
//  NEW GOAL
//============================================================
void new_goal(ArRobot &robot, float son_array[14], int x) {

        float PHI;

        if (son_array[10] < dist_danger*2) {
                //tangent escape
                int delta;
                if (abs(3.5 - son_array[11]) < 1) delta = 10;
                else if (abs(3.5 - son_array[11]) > 2) delta = 50;
                else delta = 30;

                int sign_B;
                if ((3.5 - son_array[11]) > 0) sign_B = 1;
                else sign_B = -1;

                int beta = sign_B * (45 + delta);

                float alpha = sign_B * (((IMAGE_WIDTH / 2) - x)
                                                / (IMAGE_WIDTH / CAM_ANGLE));

                PHI = sign_B * 90 - (beta - alpha);

        }
        else {
                if (abs((IMAGE_WIDTH / 2) - x) > 9) { // nemoj rotirati za +- 9 piksela
                        PHI = ((IMAGE_WIDTH / 2) - x)
                                / (IMAGE_WIDTH / CAM_ANGLE);
                }
                else PHI = 0;
        }
        cout << x << endl;
        cout << (IMAGE_WIDTH / 2) - x << endl;
        cout << PHI << endl;

        rot << PHI << endl;

        robot.setDeltaHeading(PHI);
}
```

```
//=============================================================
==================================
// MAIN
//=============================================================
==================================

int main(int argc, char **argv)
{
        //izbor aktivnosti
        int choice = -1;
        //meni
        while (choice != 0)
        {
                cout << endl << "1 - Algoritam \n" << "2 - Kalibracija \n\n" << "0 - Izlaz
\n\n";
                cin >> choice;
                switch (choice)
                {
case 1:
{       //GLAVNA PETLJA
        //inciijalizacija Aria-e i objekta robot i sonar
        Aria::init();
        ArRobot robot;
        ArSonarDevice sonar; //definicija sonar objekta
        robot.addRangeDevice(&sonar); //spajanje robot i sonar objekata

        //spajanje na robot
        ArArgumentParser parser(&argc, argv);//objekt za
                                rasclanjivanje ulaznih parametara u funkciju main()
        parser.loadDefaultArguments();
        ArRobotConnector robotConnector(&parser, &robot); //definiranje
                                objekta za spajanje na simulator ili preko serijske veze
        robotConnector.parseArgs(); //rasclanjivanje ulaznih parametara
        if (!robotConnector.connectRobot()) //spajanje na robot –
                                        "blocking connection" (vidi dokumentaciju)
        {
                cout << ("Pogreska pri povezivanju na robot.\n");
                Aria::exit(1);
                return 0;
        }

        //pokretanje veze s robotom
        robot.runAsync(true);

        //inicijalizacija nizova za rad s sonarnim senzorima
        float son_array[14], son_x[8], son_y[8];

        //Inicijalizacija kamere
```

```
VideoCapture cap;
cap.open(0); //pokreni kameru (0 - default kamera)
if(!cap.isOpened()){  //provjera rada kamere
        cout << "Pogreska pri povezivanju na kameru.\n";
        Aria::exit(1);
}
//rezolucija slike
cap.set(CV_CAP_PROP_FRAME_WIDTH,IMAGE_WIDTH);
cap.set(CV_CAP_PROP_FRAME_HEIGHT,IMAGE_HEIGHT);

//incijalizaija prozora
namedWindow(win1,CV_WINDOW_AUTOSIZE);
namedWindow(win2,CV_WINDOW_AUTOSIZE);
namedWindow(win3,CV_WINDOW_AUTOSIZE);

//inicijalizacija varijabli lokacije objekta na slici
float PHI = 0;
int x = -1, y = -1;
int x0 = -1, y0 = -1;

//Inicijalizacija raspona vrijednosti HSV skale boja za željenu boju
//GREEN
H_min = 59;
H_max = 92;
S_min = 74;
S_max = 172;
V_min = 119;
V_max = 255;
/*// RED
H_min = 0;
H_max = 4;
S_min = 130;
S_max = 203;
V_min = 139;
V_max = 208;*/


robot.enableMotors();//uključi motore na mobilnom robotu
sonar_readings(sonar, son_array, son_x, son_y);//početno očitanje sonera

cout << "Drzi ESC za izlaz iz petlje \n";

rot.open("rot.txt");
v.open("v.txt");
xpos.open("x.txt");

//glavna petlja za real-time izvođenje algoritma upravljanja
while(1)
```

```
{
        //inicijalizacija matrica za spremanje slika
        Mat image;
        Mat HSVimage;
        Mat BINimage;

        //očitaj sliku s kamere
        while(!cap.read(image));//čekaj da kamera počne raditi
        //transformacija boja iz RGB u HSV zapis radi lakse obrade
        cvtColor(image,HSVimage,COLOR_BGR2HSV);

        //binarizacija slike za zadane HSV vrijednosti
                inRange(HSVimage,Scalar(H_min,S_min,V_min),
                                Scalar(H_max,S_max,V_max),BINimage);

        //uklananje šuma na slici
        ERODE_DILATE(BINimage);

        //određivanje lokacija pračenog objekta pomoću Moments klase
        Moments BINmoments = moments(BINimage);
        double BIN_M01 = BINmoments.m01; //x smjer
        double BIN_M10 = BINmoments.m10; //y smjer
        double BIN_area = BINmoments.m00; //površina bijele boje na binarnoj
slici

        //računanje lokacije objekta
        if (BIN_area > 10000) {
                x = BIN_M10 / BIN_area;
                y = BIN_M01 / BIN_area;
        }
        else x = y = -1;

        xpos << ((IMAGE_WIDTH/2 - x) / (IMAGE_WIDTH/CAM_ANGLE))
<< endl;

        //kretanje
        if (x != -1 ) {
                forward(robot, son_array);
                new_goal(robot, son_array, x);
        }
        else robot.stop();

        //prikaži slike na zadanim prozorima (RGB, HSV, BINARNA)
        imshow(win3, BINimage);
        imshow(win1, image);
        imshow(win2, HSVimage);

        sonar_readings(sonar, son_array, son_x, son_y);
```

```
                if (waitKey(T) == 27) {
                        cout << "Izlaz iz petlje" << endl;
                        break;
                }
        }
        rot.close();
        v.close();
        xpos.close();
        robot.stop();
        robot.disableMotors();
        destroyAllWindows();
        Aria::exit(1);
        break;
}
                case 2:
                        {
                                cout << "Drzi ESC za izlaz iz petlje \n";

                                //inicijalizacija slika za kalibraciju
                                Mat c_image;
                                Mat c_HSVimage;
                                Mat c_BINimage;

                                //inicijalizacija vrijednosti HSV slajdera
                                int c_H_min = 0;
                                int c_H_max = 179;
                                int c_S_min = 0;
                                int c_S_max = 255;
                                int c_V_min = 0;
                                int c_V_max = 255;

                                //napravi prozor s slajderima
                                namedWindow(winS,WINDOW_AUTOSIZE);
                                createTrackbar( "H_MIN", winS, &c_H_min,
                                                        c_H_max, slider_act);
                                createTrackbar( "H_MAX", winS, &c_H_max,
                                                        c_H_max, slider_act);
                                createTrackbar( "S_MIN", winS, &c_S_min,
                                                        c_S_max, slider_act);
                                createTrackbar( "S_MAX", winS, &c_S_max,
                                                        c_S_max, slider_act);
                                createTrackbar( "V_MIN", winS, &c_V_min,
                                                        c_V_max, slider_act);
                                createTrackbar( "V_MAX", winS, &c_V_max,
                                                        c_V_max, slider_act);
```

```
                    //pokreni snimanje s kamere
                    VideoCapture c_cap;
                    c_cap.open(0);

                    c_cap.set(CV_CAP_PROP_FRAME_WIDTH,
                                        IMAGE_WIDTH);

                    c_cap.set(CV_CAP_PROP_FRAME_HEIGHT,
                                        IMAGE_HEIGHT);

                    //petlja za kalibraciju
                    while(1){
                        while (!c_cap.read(c_image));

cvtColor(c_image,c_HSVimage,COLOR_BGR2HSV);

inRange(c_HSVimage,Scalar(c_H_min,c_S_min,c_V_min),
                Scalar(c_H_max,c_S_max,c_V_max),c_BINimage);

                        imshow(win4,c_BINimage);

                        ERODE_DILATE(c_BINimage);

                        imshow(win3,c_BINimage);
                        imshow(win1,c_image);
                        imshow(win2,c_HSVimage);

                        if (waitKey(T) == 27) break;
                    }
                    destroyAllWindows();
                    break;
                }
                default:
                    break;
            }
        }
        return 0;
    }
```