Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Urbanija

Primerjava lokalizacije mobilne platforme z uporabo odometrije in vizualne informacije

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Danijel Skočaj

Ljubljana, 2012

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matej Urbanija, z vpisno številko **63080329**, sem avtor diplomskega dela z naslovom:

Primerjava lokalizacije mobilne platforme z uporabo odometrije in vizualne informacije

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Danijela Skočaja
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11. januarja 2011 Podpis avtorja:

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00280/2012
Datum: 12.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ URBANIJA**

Naslov: **PRIMERJAVA LOKALIZACIJE MOBILNE PLATFORME Z UPORABO ODOMETRIJE IN VIZUALNE INFORMACIJE**

COMPARISON OF MOBILE PLATFORM LOCALISATION USING ODOMETRY AND VISUAL INFORMATION

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Ena najosnovnejših nalog mobilnih robotov je navigacija po prostoru. Za uspešno navigacijo se mora robot pravilno lokalizirati. Poznati mora trenutni položaj, bodisi v absolutnem smislu bodisi v razmerju do prej obiskanih točk v prostoru. V postopku lokalizacije lahko uporablja različne senzorje. Samo podatki odometrije običajno ne zadoščajo za uspešno lokalizacijo na dolgi rok. V diplomski nalogi razvijte dodaten postopek lokalizacije, ki naj temelji na obdelavi vizualne informacije oz. zajetih slik stropnih oznak. Primerjajte uspešnost lokalizacije in navigacije v primeru, ko se uporabljajo samo podatki odometrije z uspešnostjo lokalizacije, ki temelji na vizualni informaciji.

Mentor:

doc. dr. Danijel Skočaj

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Zahvaljujem se mentorju doc. dr. Danijelu Skočaju, ki mi je pomagal pri izdelavi diplomske naloge.

Kazalo

Povzetek

Namen diplomske naloge je bil primerjati odometrijo mobilnega robota z lokalizacijo s pomočjo vizualne informacije. V diplomski nalogi je bil uporabljen robot iRobot Roomba serije 531. Ker je to komercialni robot, je cena takih robotov relativno nizka, zato je tudi odometrija takšnih robotov manj natančna. Lokalizacija z vizualno informacijo je bila realizirana tako, da smo na robota pritrdili barvno kamero, ki je bila usmerjena proti stropu. Na strop so bile pritrjene štiri oznake, s pomočjo katerih je bil izračunan premik robota glede na zajeto sliko stropnih oznak v začetni poziciji robota. Sistem smo ovrednotili tako, da smo mu dali neko pot po kateri se bo premikal, preko katere je bila ugotovljena napaka, ki jo tvori odometrija robota. Na koncu smo nato primerjali napako odometrije mobilnega robota z napako vizualne informacije.

Abstract

The purpose of this thesis was to compare robot odometry with localization using visual information. In this thesis we used a robot iRobot Roomba series 531. Since this is a commercial robot, the price of such robots is relatively low, therefore odometry of such robots is very inaccurate. Localization with visual information was realised such, that we fixated color camera on top of a robot, which was directed toward the ceiling. On the ceiling, we attached four markers, through which robot movement was calculated, according to the captured image of ceiling markers in the initial robot position. The system was evaluated so the robot was given a pre-known path on which it moves, through which the error of robot odometry has been found. On the same path accuracy of visual information was obtained. At the end we compared errors of visual information and robot odometry.

Poglavje 1

Uvod

Dandanes se je na tržišču začelo pojavljati veliko poceni elektronike ter robotike. Zaradi masovne proizvodnje in tekmovanja proizvajalcev med seboj za čim večjo prodajo so cene večine izdelkov zelo nizke. Zato pa so deli s katerih je robot sestavljen slabše kvalitete. Na robotu pa je tudi minimalno število senzorjev ter minimalna natančnost le teh za opravljanje podanih nalog. S tem se je poslabšala natančnost robota, kar je pomenilo, da so morali njihove funkcije narediti na ne optimalen način. Zato izvajajo svoje naloge dlje časa in manj učinkovito.

Problem robotov z nizko kvaliteto senzorjev je, da se je moč zanašati na odometrijo (odometrija je uporaba podatkov iz senzorjev za premik, za oceno položaja robota v času) le kratek čas, ker se napaka akumulira s časom. To pomeni, da dlje časa kot preteče, večja bo napaka. To pride do izraza na primer pri poziciji robota v prostoru. V diplomski nalogi smo tako primerjali kakšno natančnost ima eden izmed takih robotov, ter jo poskušali izboljšati z razširitvijo števila robotskih senzorjev, tako da smo dodali barvno kamero na robota, ter jo usmerili v strop. Na strop smo pritrdili stropne oznake preko katerih se bo robot lahko orientiral v prostoru, ter izračunal pozicijo glede na začetno točko. V našem primeru s takim sistemom ne dobimo absolutne pozicije v prostoru, temveč le premik od začetne točke. Nato smo robotu podali neko vnaprej določeno pot po kateri se je premikal. Tako smo merili

napako odometrije robota in napako lokalizacije z vizualno informacijo.

Rešitev takega problema je potrebna, če želimo izboljšati robotov algoritem lokalizacije z vizualno informacijo. S poznavanjem pozicije lahko gradimo zemljevid prostora. Ko imamo zemljevid prostora lahko robota pravilno vodimo čez prostor brez zaletavanja v predmete. Robot lahko izpolni več nalog hitreje, saj ve kje v prostoru je že bil. S tem ne ponavlja nalog, ki jih je že opravil.

Primer robotskega sesalca, ki ima dodatne senzorje in algoritme je razvil *Samsung* in sicer z oznako *SR8980 NaviBot S*. Glavna prednost tega sesalnika je, da ima vgrajeno kamero na zgornjo stran robota. Ta mu omogoča gradnjo mape prostora oziroma hiše, tako da periodično zajema slike stropa. Ta mapa mu pove kje je že čistil v določenem prostoru, ne pove pa mu kje se robot nahaja. S pomočjo mape nato določi najboljšo pot za čiščenje prostora.

V prvem poglavju bomo predstavili elemente katere smo uporabili v diplomski nalogi. Vsak element bo podrobno opisan, razloženo bo kako deluje in kam spada v celotni sliki. Opisali bomo tako strojno opremo kot je robot in kamera, kot tudi programsko opremo kot je *ROS - Robotski operacijski sistem* in programski jezik.

V drugem poglavju bomo predstavili sam algoritem, ki smo ga razvili za izračun premika robota. Predstavili bomo kako smo ga naredili, ter prikazali primer njegovega delovanja.

V tretjem poglavju bomo prikazali primerjavo rezultatov, ki smo jih dobili s testiranjem našega algoritma. V podpoglavjih bomo prikazali rezultate posameznih testiranj, kateri bodo vsebovali večje število testov, kar bo doprineslo k večji verodostojnosti. Lažje pa bo tudi razložiti zakaj so dobljeni rezultati takšni kot so.

Poglavje 2

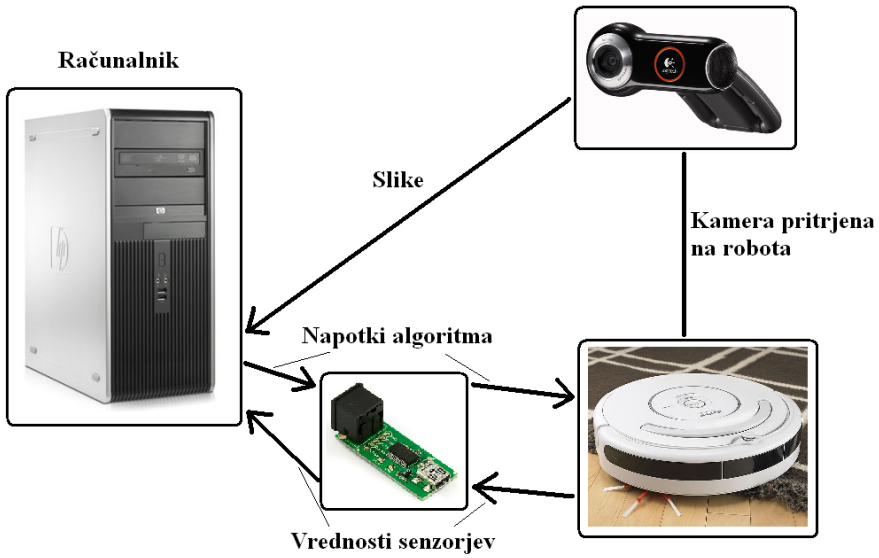
Opis sistema

Za izdelavo diplomske naloge smo uporabili robota *iRobot Roomba 531*, spletno kamero *Logitech 9000 pro* ter vmesnik med robotom in osebnim računalnikom *RooStick*. Zato, ker ima robot slabo odometrijo, smo na robota namestili kamero, katero smo usmerili proti stropu, ter tako poizkušali izboljšati slabo odometrijo robota. Preko osebnega računalnika smo zajemali slike s kamero, ki je bila usmerjena proti stropu. Na strop pa smo pritrdili stropne oznake. Zajete slike smo nato procesirali in nato robota usmerjali po prostoru. Robot pa nam je vračal pozicijo, ter po potrebi tudi vrednosti senzorjev. Shemo celotnega sistema lahko vidimo na Sliki 2.1.

2.1 Robot

V diplomski nalogi smo uporabili mobilnega robota podjetja *iRobot*. Podjetje je bilo ustanovljeno leta 1990 na *Massachusetts* institutu za tehnologijo (MIT - Massachusetts Institute of Technology) v *Cambridgeu, ZDA*. Podjetje izdeluje več vrst robotov. Izdelujejo robote za domačo uporabo, kot so robot sesalnik, robot za čiščenje bazena ter robot za čiščenje žlebov. Izdelujejo pa tudi robote za vojsko in pridobivanje informacij iz morja za znanstvenike.

V našem primeru smo si izbrali robota, ki sesa [10]. Na izbiro imajo



Slika 2.1: Shema celotnega sistema.

več kot šest vrst sesalnih robotov, ki se razlikujejo po funkcionalnostih, ki jih ponujajo. Ker mi ne bomo uporabljali robota za sesanje, smo si izbrali cenovno ugoden model *Roomba 531*. Primer osnovnega in modificiranega robota je prikazan v Slikah 2.2 in 2.3.

2.2 Primer delovanja Roombe

Ker ima *Roomba* slabo odometrijo so morali ustvarjalci narediti algoritem, ki bo posesal celotno sobo, ne da bi vedel obliko sobe ali kje se robot nahaja. To so naredili tako, da so za premikanje robota uporabili princip naključja (ang. random), ter ga primerno nadgradili. Ker se velikosti sobe zelo razlikujejo in s tem čas čiščenja, mora *Roomba* vedeti velikost prostora v katerem se nahaja. Za izračun velikosti sobe uporabi najdaljšo razdaljo v ravni črti, ki jo lahko naredi ne da bi se zaletel v oviro. Algoritem so nadgradili tudi tako, da ko *Roombo* prižgemo začne potovati spiralno navzven, dokler ne najde

Slika 2.2: Primer robotskega sesalnika Roomba 531.

Slika 2.3: Primer robotskega sesalnika Roomba z pritrjeno kamero.

Slika 2.4: Prikaz načina čiščenja robota *Roomba*.

zidu ali pa dokler premer kroga ne presega določeno vrednost. *Roomba* se tudi veliko časa drži zidov, da zagotovi večjo pokritost sobe. Če pa robot opazi mesto, kjer je večja koncentracija umazanije, se ustavi in začne spiralno navzven čistiti tisto mesto, dokler ni čisto. Primer delovanja je prikazan v sliki 2.4.

Kot je razvidno iz Slike 2.4, *Roomba* potuje naključno po sobi ali pa se drži zidu. S tem je zagotovljeno, da bo pokritost sobe zelo izboljšana.

2.3 Senzorji robota Roomba

Robot *Roomba* ima več senzorjev [1, 4], ki pomagajo pri opravljanju naloge in so prikazani na Slikah 2.5 in 2.6.

Robotu pri opravljanju nalog pomagajo naslednji senzorji:

- senzor, ki beleži odometrijo pozicije robota,
- odbijač na sprednjem delu Roombe za detekcijo ovir,

- infrardeči sprejemniki, ki locirajo navidezne zidove, ter najdejo bazo za polnjenje Roombe,
- štirje infrardeči senzorji na spodnji strani odbijača, ki merijo razdaljo od tal ter preprečujejo padec robota po stopnicah,
- dva senzorja na spodnji strani robota za detekcijo umazanije, na primer peska,
- vsako kolo ima senzor, ki zazna, če je kolo v zraku,
- infrardeč senzor na desni strani odbijača, ki zazna, če se robot pelje poleg zidu oziroma, če je zid desno od Roombe,
- šest infrardečih senzorjev, ki merijo razdaljo od odbijača do ovir. Robot uporablja te senzorje, da upočasni preden se zaleti v oviro,
- senzor za električni tok, ki pove robotu, če je pravilno postavljen na polnilni postaji,
- senzor stanja polnjenja pove ali se Roomba ne polni, polni, polni hitro, polni počasi, ali je prišlo do napake pri polnjenju,
- senzor za napetost baterije. Pove robotu napolnjenost baterije.

Roomba zgoraj naštete senzorje uporablja sebi v prid na različne načine. Odbijač na sprednjem delu robota se uporablja za to, da robot ve, da se je zaletel v oviro, naj bo to miza ali pa zid. Ker robot ne ve kje v prostoru, če kje so ovire zato odbijač deluje tudi kot vzmet, da ne poškoduje pohištva.

Navidezni zidovi delujejo kot omejevalnik robota, da ta ne zaide iz prostora, če je prostor bolj odprte narave in nima vrat. Infrardeč sprejemnik robotu pove kam ne sme iz katerega koli razloga.

Roomba ima tudi štiri infrardeče senzorje na spodnji strani robota, ki merijo razdaljo od tal. Te senzorji so predvsem za to, da preprečijo padec robota po stopnicah.

Dva senzorja na spodnji strani robota beležita stopnjo umazanije. Tam kjer robot zazna večjo stopnjo umazanije se zadrži dlje časa dokler jo ne počisti.

Vsako kolo na robotu pa ima tudi senzor, ki zazna, če je kolo v zraku. Če sta obe kolesi v zraku robot preneha z delovanjem, saj se je zgodilo nekaj nepredvidljivega in je potrebna človeška intervencija.

Infrardeč senzor na desni strani odbijača zaznava, če se robot pelje poleg zidu oziroma, če je zid desno od *Roombe*. Ta senzor robot uporablja zato, da lahko sledi zidu, ter počisti umazanijo ob robovih sobe. Uporablja pa se tudi za boljšo pokritost čiščenja sobe.

Robot ima tudi šest infrardečih senzorjev, ki merijo razdaljo od odbijača do ovir. Robot uporablja te senzorje, da upočasni preden se zaleti v oviro. To omogoča, da robot potuje z večjo hitrostjo ter preden se zaleti v oviro upočasni, da zagotovi da ne pride do poškodb robota in pohištva.

V notranjosti robota se skrivajo senzorji, ki so povezani z baterijo. Ti senzorji sporočajo robotu napolnjenost baterije, ali se baterija polni ali prazni ter hitrost polnjenja.

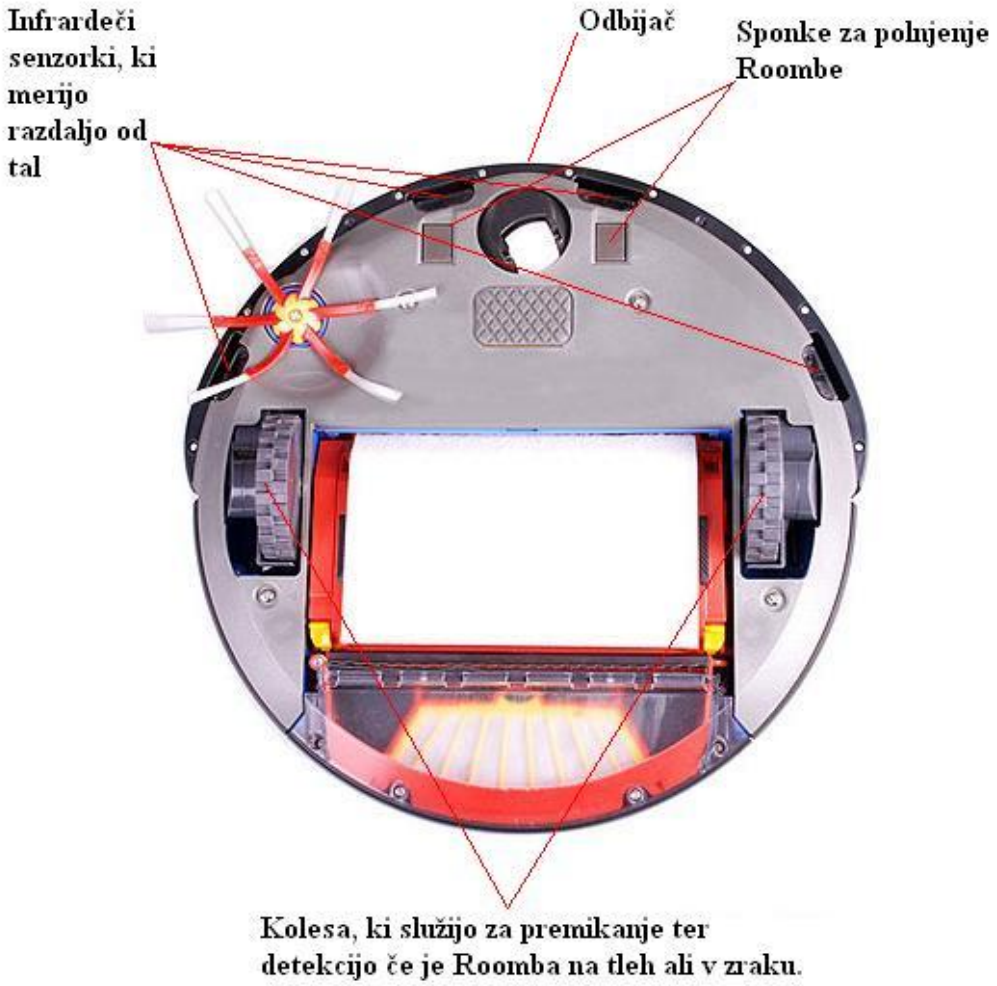
Za izdelavo diplomske naloge je bil za nas pomemben samo en senzor in to je senzor za odometrijo. Ta senzor nam je pretvarjal rotacijo koles robota v razdaljo in kot. Razdaljo je izračunal preko števila obratov koles, kot pa iz razlike razdalje obeh koles. To je nato prevedel podatke o premiku po oseh, ter kot pod katerim je robot rotiran. To smo nato lahko prebrali in primerjali z našim algoritmom ter videli razliko.

2.4 Povezava robota z računalnikom

Pri povezavi *Roombe* z osebnim računalnikom so nam pri podjetju *RoboDynamics* priskrbeli tri načine povezave robota z računalnikom. Produkti RoboDynamics so: *RooTooth* [12], *RooStick* [13] in *Roo232* [6].

RooTooth je strojna oprema, ki nam omogoča brezžični dostop do robota. To je zelo uporabno saj nam omogoča prosto premikanje po celotnem

Slika 2.5: Prikaz senzorjev na zgornji strani robota.



Slika 2.6: Prikaz senzorjev na spodnji strani robota.

Slika 2.7: Primer *RooStick* vmesnika med Roombo in računalnikom.

prostoru in nismo odvisni od dolžine žice. Vključimo ga v *SCI vhod*, ki je na zgornji strani *Roombe*.

RooStick je skoraj identičen kot *RooTooth* vendar za povezavo med računalnikom in robotom potrebuje žico. Prednost *RooSticka* je v tem, da je cenejši (30$) od *RooTootha*, ki stane 100$, ter ne potrebujemo brezžičnega usmerjevalnika(router).

Roo232 je podoben kot *RooStick* vendar za povezavo z računalnikom namesto USB potrebuje *RS232 serijski vhod*. To je predvsem uporabno za starejše računalnike, ki nimajo USB vhodov. Cena *Roo232* je pa najnižja z 23$ na kos.

Od zgoraj navedenih možnosti smo si izbrali *RooStick*, saj je bil že prisoten v laboratoriju za umetne vizualne spoznavne sisteme. RooStick smo izbrali tudi zato, ker je bila tema diplomske naloge taka, da je omogočala testiranje na majhnem prostoru in dolžina žice ni bila problem. Primer *RooSticka* je viden na Sliki 2.7.

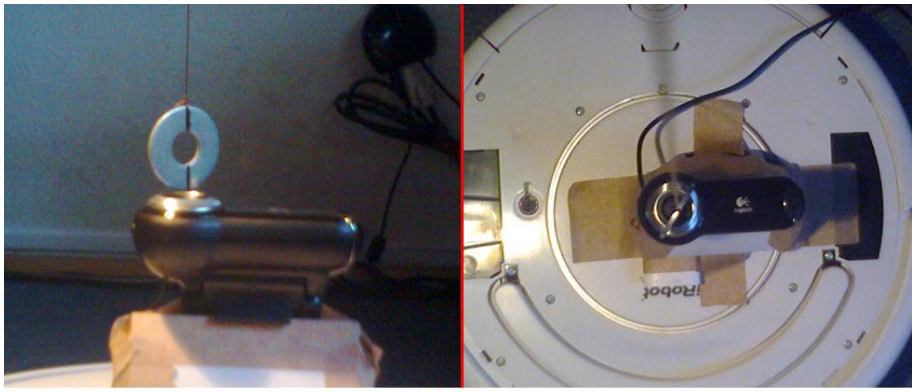
Slika 2.8: Spletna kamera *Logitech 9000 pro*.

2.5 Kamera

Za zajemanje slike, s katere smo izračunali premik robota, smo uporabili spletno kamero [3]. Za to smo se odločili, ker spletne kamere zajemajo dovolj kvalitetne posnetke zadostne ločljivosti in imajo nizko ceno. Uporabili smo kamero proizvajalca *Logitech* in sicer model *9000 pro*, kot jo vidimo na Sliki 2.8. Kamera lahko posname slike z 2 milijona slikovnimi elementi ter lahko snema posnetke z ločljivostjo 960 x 720 pri 30 sličicah na sekundo (ang. Frames per second).

2.5.1 Kalibracija lokacije kamere na robotu

Naš cilj je bil poravnati koordinatni sistem kamere in koordinatni sistem robota en z drugim. S tem bi se znebili pretvorbe med njima. Zato preden smo pritrdili kamero na robota, smo se morali prepričati, da je usmerjena v pravo smer. Najprej smo morali ugotoviti, kje je rotacijska točka robota (to je točka, ki se pri rotaciji robota ne premakne po nobeni od osi x,y,z). To smo lahko naredili sami optično, brez kakršnekoli opreme. Že s samim rotiranjem robota na mestu se je hitro videlo kje je njegova rotacijska točka. Nato smo na robota postavili kamero. Prva naloga je bila centrirati kamero z rotacijsko točko robota. To smo naredili tako, da smo uporabili nihalo. Najnižjo točko

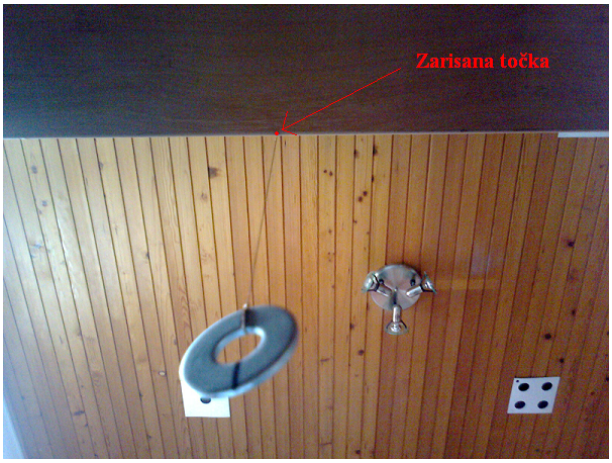


Slika 2.9: Primer kalibriranja kamere na robotu.

nihala smo centrirali z rotacijsko točko robota. Nato smo na robota postavili kamero ter jo premaknili čim bližje najnižji točki nihala. Tako smo vedeli, da je kamera na pravilnem mestu na robotu. Primer kalibracije je viden v Slikah 2.9(a) in 2.9(b).

To pa še ni bilo vse. Ker se zgornji del kamere premika, smo morali kalibrirati še rotaciji nagib in naklon, kot sta razvidni iz Slike 2.11. To smo naredili tako, da smo zarisali točko tam, kjer se je nitka nihala dotikala mize. To lahko vidimo na Sliki 2.10. Nato smo posneli dve sliki. Prvo s prvotne pozicije, drugo pa, ko smo za rotirali robota na mestu za 90 stopinj. Pozicija točke pa se ni smela premakniti. Tako smo odpravili Naklon. Sam Nagib smo pa odpravili kar iz prvotne pozicije robota. Videlo se je, če je točka bolj levo ali desno od središča slike. Tako smo ugotovili za koliko moramo popraviti Nagib.

S tem smo dobili kamero, ki je nameščena natančno v središču robota. To je pomenilo, da, ko se robot obrača, se kamera ne premika po nobeni od osi x,y in z. S tem nam ni bilo treba upoštevati translacije kamere, kar je olajšalo kompleksnost algoritma. Če tega ne bi naredili, bi morali pri izračunu premika robota pretvarjati med koordinatnima sistemoma kamere in robota. To pa bi dodalo en korak pri izračunu nove pozicije.

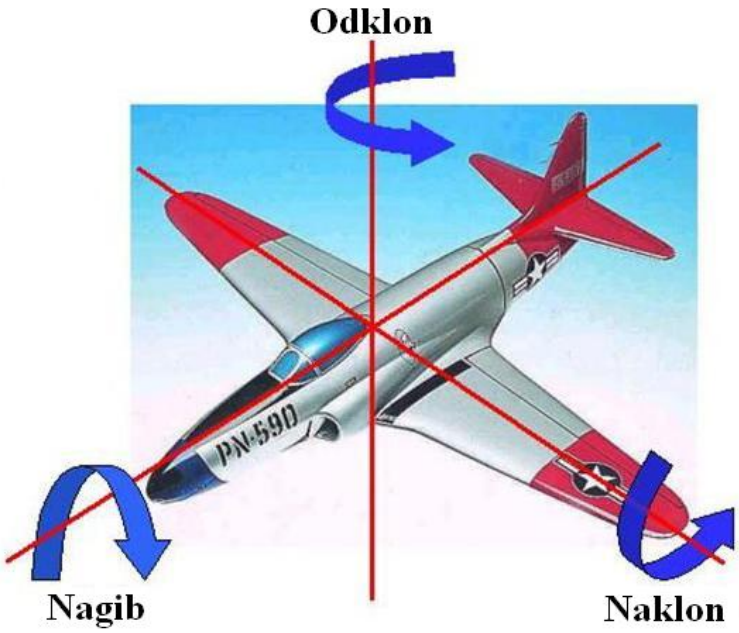


Slika 2.10: Primer zarisane točke na spodnji strani mize.

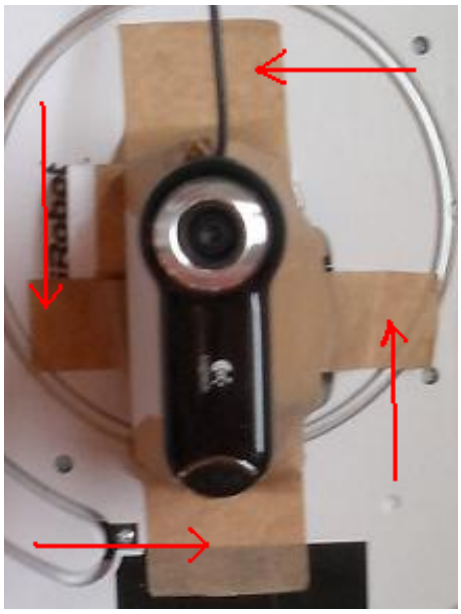
2.5.2 Pritrditev kamere na robota

Spletne kamere so namenjene za stacionarno uporabo. Zato tudi večina nima sredstev za pritrditev oziroma stojal. Problem pa je tudi teža same kamere, saj so spletne kamere zelo lahke. To pomeni, da bi med vožnjo zlahka padle z robota oziroma se premakne. Mi pa smo želeli uporabiti stacionarno kamero na premikajočem se robotu. Poleg nevarnosti, da bi kamera padala z robota, je obstajala verjetnost, da bi se kamera med vožnjo premaknila, po kateri od oseh, kot jih vidimo na Sliki 2.11. To bi pomenilo, da smo izračunali napačno vrednost premika robota.

Da se kamera ne bi premikala po osi *Odklon*, smo jo križno zalepili na robota, kot vidimo na Sliki 2.12. Za preprečitev rotacije kamere po osi *Nagib*, smo kamero podložili, kot je razvidno na Sliki 2.13. S tem, ko smo kamero podložili, smo lahko tudi kontrolirali naklon, če je bilo to potrebno. Na koncu smo kameri dodali še utež. Le ta nam služi v dva namena. Prvi je, da doda težo sistemu, da je bolj stabilen. Drugi pa je, da se utež zatakne za zatič, ki ga ima kamera, kar jo drži usmerjeno proti stropu. Brez uteži bi kamera bila usmerjena naprej in ne v strop. To vidimo na Sliki 2.13.



Slika 2.11: Osi premika nepritrjene kamere.



Slika 2.12: Lepilni trak, ki preprečuje premik po osi Odklon.

Slika 2.13: Preprečitev rotacije po osi Nagib, tako da smo podložili iz obeh strani.

2.6 ROS - Robot Operating System

Potem, ko smo si izbrali robota ter kamero, smo potrebovali še nekaj kar bo povezovalo algoritem s strojno opremo. Za to smo uporabili ROS (Robot Operating System) [11]. ROS je programska oprema, ki zagotavlja okvir za razvoj programske opreme za robotiko, ki je podoben operacijskemu sistemu. ROS je bil prvotno razvit leta 2007 pod imenom Switchyard v laboratoriju za umetno inteligenco v Stanfordu v pomoč projektu Stanford AI Robot. Od leta 2008 naprej se je razvoj nadaljeval primarno v raziskovalnem inštitutu robotike Willow Garage. Pri tem je sodelovalo več kot dvajset inštitucij v zveznem razvojnem modelu.

ROS zagotavlja standardne storitve operacijskega sistema, kot so abstrakcijo strojne opreme, nadzor naprav na nizkem nivoju (ang. low-level device control), izvajanje pogosto uporabljenih funkcionalnosti, izmenjava sporočil med procesi in upravljanje s paketi. Sistem temelji na arhitekturi grafov, kjer procesiranje poteka v vozliščih, ki lahko prejemajo in objavljajo več vrst sporočil (senzorska sporočila, nadzorna sporočila, sporočila stanja, sporočila načrtovanja, pogonska sporočila, druga sporočila) hkrati. Knjižnica je prvenstveno namenjena operacijskemu sistemu Unix oz. njegovim različicam (Ubuntu je podprt sistem, medtem, ko sta Fedora ter Mac OS X v eksperimentali fazi).

2.6.1 Prednosti sistema ROS

Velika prednost ROS-a je, da je brezplačen. Kot smo omenili v prejšnjem poglavju je sistem podprt na operacijskih sistemih Mac OS x in Linux. Katerokoli verzijo/implementacijo Linuxa si izberemo, je brezplačna. To pa pomeni, da je vsa programska oprema brezplačna.

Ker ROS bazira na vozliščih, je enostavno ustvarjanje novih vozlišč. To pomeni, da obstaja veliko število brezplačnih vozlišč, katere so naredili upo-

rabniki, ki jih lahko uporabimo. To nam zagotavlja veliko podporo strojne ter programske opreme.

Velika prednost sistema je tudi to, da vsebuje več simulatorjev, med katerimi lahko izbiramo. Na ta način nam ni treba poganjati samega robota, ampak lahko večino naredimo v simulatorju ter na koncu preizkusimo na pravem robotu. To nam prihrani veliko časa in truda.

Prednost je tudi ta, da ima ROS veliko število uporabnikov, ki so nam pripravljeni pomagati na forumu spletne strani sistema ROS. Sistem pa še vedno razvijajo tudi programerji, ki so naredili ta sistem, kar prikazuje to, da prihajajo nadgradnje (ang. updates) vsakih nekaj mesecev. Ustvarjalci skupaj z uporabniki tvorijo močno silo za razvoj, kar pomeni svetlo prihodnost sistema.

2.6.2 Slabosti sistema ROS

Največja slabost sistema, s katero smo se srečali je bila to, da spletna stran sistema ROS ni najbolj pregledna. Pri nekaterih temah manjkajo vodiči (Tutorial), ter je včasih opis zelo skop. Potrebno je veliko iskanja po spletu za odgovore.
Vozlišča pa imajo tudi slabo stran. Ponavadi poleg samih datotek vozlišča ne vsebujejo navodil za usposobitev. To je problem, če se to vozlišče navezuje na kaj drugega. Nato moramo ugotavljati, kako povezati enega ali več vozlišč skupaj. Problem je ponovno dokumentacija, ki je v veliki večini primerov avtorji ne zagotovijo.

2.6.3 Ukazi

ROS ima za lažje delo vgrajenih več ukazov, katere lahko uporabljamo preko terminala. Ukazi nam dajo veliko fleksibilnost za delo z sistemom, saj nimamo integriranega razvojnega okolja (Integrated development environment).

To pomeni, da ne moremo razhroščevati tako zlahka. Tako nam ti ukazi zelo pomagajo.

V sistemu ROS se uporablja več vrst angleških izrazov, ki smo jih razložili spodaj:

- **vozlišče** (ang. node) - vozlišče je proces, ki opravlja neko nalogo (računanje),
- **paket** - programska oprema je v ROS organizirana v pakete. Paketi lahko vsebujejo vozlišča, knjižnice, konfiguracijske datoteke ter vse kar pripomore k uporabnosti modula. Paketi so skupek datotek (ang. file) in map (ang. folder),
- **tema** (ang. topic) - teme so poimenovana vodila (ang. bus) preko katerih vozlišča izmenjujejo sporočila.

Najpogosteje uporabljeni ukazi pri delu s sistemom ROS:

- **roscore** - zažene jedro ROSa,
- **rosnode -list** - izpiše vsa aktivna vozlišča,
- **roscd** - premik v direktorij znotraj ROS strukture,
- **rosrun** - poženemo vozlišče znotraj paketa,
- **roslaunch** - zažene vozlišče, tako kot je definirano v zagonski datoteki,
- **rosmake** - prevede pakete,
- **rostopic** - dobimo informacije o specifičnih temah (topic),
- **roscreate-pkg** naredimo nov paket.

2.6.4 Branje in pisanje v podatkovni tok teme

Ko je bila programska oprema naložena v računalnik, ter preizkušena, smo se naposled osredotočili na izdelavo algoritma za detekcijo premika robota v prostoru. Prva zadeva na katero smo se osredotočili je bila pridobivanje informacij iz kamere in robota. Kot smo opisali v prejšnjih poglavjih sistem ROS objavlja informacije v formatu *teme* (topic). Vsakdo lahko bere s podatkovnega toka *teme*, če pozna njeno ime in strukturo. Vse objavljene teme lahko vidimo, če v terminal vpišemo ukaz: `rostopic list`. Primer izpisa za ukaz `rostopic list` vidimo v Sliki 2.14.

Nato smo z **rostopic echo ime teme** izpisali vsebino na zaslon, ter tako poiskali temo, ki jo potrebujemo. Sedaj, ko smo imeli imena tem, ki jih bomo uporabljali, smo samo še izvedli branje v programu. To smo naredili na naslednji način:

```
rospy.Subscriber('/usb_cam/image_raw', Image, callback)
```

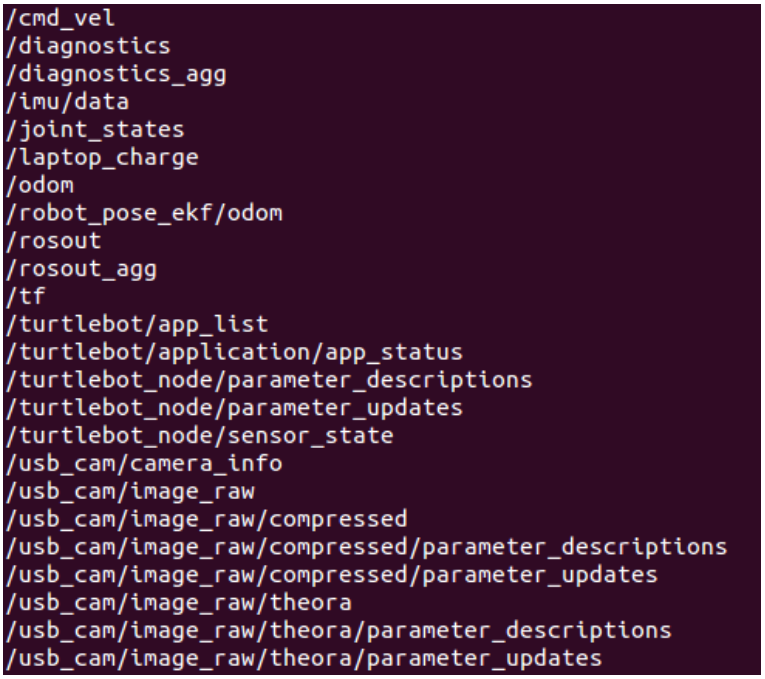
Pri tem je prvi parameter ime teme, drugi parameter tip, ki ga tema vrača in tretji parameter funkcija, ki se bo klicala, ko bo robot podal podatke v temo.

Za objavljanje podatkov v temo pa smo uporabili naslednje ukaze:

```
tpub = rospy.Publisher("cmd_vel", Twist)
t = Twist()
tpub.publish(t)
```

2.6.5 Python

Glavna programska jezika v sistemu ROS sta C++ in Python. Za oba obstajajo vodiči (ang. tutorial) na uradni spletni strani http://www.ros.org/wiki/. Potem, ko smo si ogledali vodiče za oba jezika, smo se na podlagi tega, da je bil vodič izdelan v programskem jeziku Python bolj pregleden, odločili, da bomo izdelali algoritem v Pythonu. Prednosti jezika Python v našem primeru so bile: večja preglednost kode ter lažja sintaksa.



Slika 2.14: Primer izpisa za ukaz *rostopic list*.

Python je skriptni programski jezik. Podoben je programskim jezikom Scheme, Ruby, Perl. Je splošno namenski, visoko nivojski programski jezik, ki je zasnovan tako, da je čim bolje berljiv. Python se pogosto uporablja kot skriptni jezik za spletne aplikacije.

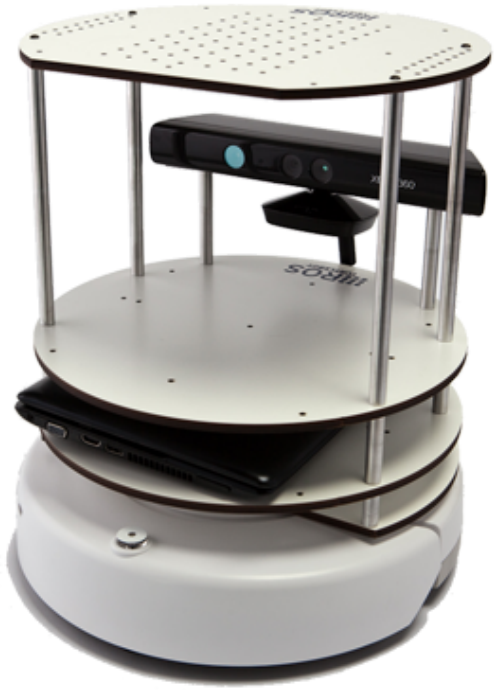
Python ima tudi veliko število knjižnic. Te nam olajšajo delo s tem, da zagotovijo funkcije, katerih nam ni potrebno pisati. V našem primeru smo uporabili dve knjižnici. Uporabili smo knjižnico PIL - Python Image Library [7]in OpenCv [5]. Python Image Library doda zmožnost obdelave slik k vašemu tolmaču (Interpreter) Pythona. Ta knjižnica podpira številne formate in omogoča zmogljivo obdelavo slik in grafičnih zmogljivosti. OpenCV (Open Source Computer Vision) je knjižnica, ki vsebuje funkcije, ki so namenjene računalniškemu vidu v realnem času.

2.7 Osrednja vozlišča diplomske naloge

2.7.1 Turtlebot

Turtlebot paket [8] lahko dobimo preko repozitorija, ki je objavljen na spletni strani ROS. Paket je v osnovi zasnovan za iRobot Create ter Kinect kot ga vidimo na Sliki 2.15. Vendar robota *Create* v Evropi ne prodajajo, zato so paket nadgradili tudi za *iRobot Roomba*. Paket nam tako omogoča vodenje robota, dostop do senzorjev robota, ter splošen nadzor nad robotom. Preden pa lahko zaženemo glavno vozlišče, ga moramo modificirati. Vozlišču moramo povedati, da bomo zagnali *Roombo* in ne *Create*, povedati moramo, da želimo, da *Roomba* objavla *TF temo*, ter povedati gonilniku, da *Roomba* nima giroskopa. Pri tem nam robot objavlja podatke o njegovi poziciji na *TF temo*.

```
<param name="robot_type" value="roomba" />
<param name="publish_tf" value="true" />
<param name="has_gyro" value="false" />
```



Slika 2.15: Robot Turtlebot , ki je sestavljen iz robota Create in kamere Kinect.

Ker pa je vozlišče narejeno za prenosni računalnik, mi pa smo uporabljali namizni računalnik, vozlišče ne bo našlo baterije, ker ne obstaja. Zato je potrebno del kode izbrisati za lepše delovanje.

```
<!-- Turtlebot Laptop Battery Diagnostics -->
<node pkg="turtlebot_node"type="laptop_battery.py"
      name="turtlebot_laptop_battery">
</node>
```

Ko smo to naredili, priklopimo robota na računalnik preko *pretvornika*

Roostick. Nato lahko zaženemo vozlišče z ukazom:

```
roslaunch turtlebot_bringup minimal.launch
```

Ko se je robot dokončno inicializiral, ga lahko upravljamo z še enim zelo pomembnim paketom, to je **Turtlebot teleop**, če zaženemo vozlišče tega paketa z ukazom:

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

Nato lahko preko tipkovnice vodimo *Roombo* po prostoru. Omejitev pa je seveda dolžina žice. Bolj elegantna rešitev pa bi bila, če bi prenosni računalnik postavili na robota. Tako nebi imeli težav z dolžino žice.

2.7.2 Usb cam

Usb cam je paket, ki povezuje ROS in kamero na robotu. Na izbiro smo imeli več paketov [2], kot so: *Probe*, *usb cam*, *uvc cam*, *gencam cu*, *logitech usb webcam*, itd. Po priporočilih s spleta smo si izbrali *Usb cam* za našo povezavo med kamero in računalnikom. Za zagon *usb cam* vozlišča smo morali narediti .launch datoteko, ki je vsebovala dodatne parametre, ki bodo uporabljeni pri zagonu vozlišča.

Datoteka vsebuje:

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node"
    output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="960" />
    <param name="image_height" value="720" />
    <param name="pixel_format" value="mjpeg" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
</launch>
```

2.7.3 Turtlebot controller

Ustvarili smo svoj paket `Turtlebot_controller`, kateri vsebuje dve pomembni vozlišči. Prvi se imenuje `pac_pic.py`. To vozlišče smo naredili zato, da zlahka zajemamo slike s kamere in jih shranjujemo na računalnik. Pridobljene slike smo uporabili za razvoj glavne aplikacije. To je pomenilo, da je bil razvoj veliko hitrejši, saj smo se lahko osredotočili samo na dan problem.

Drugi pa se imenuje diploma.py. To vozlišče vsebuje celotni programski del diplomske naloge. Za delovanje potrebuje sliko s kamere in podatke z robota. Glavni razlog zakaj smo naredili čisto svoj paket je ta, da sedaj lahko neodvisno poganjamo vozlišče. Če gre kaj narobe ga lahko ustavimo, pri tem pa nam ni potrebno ustaviti ostalih vozlišč. To je velika prednost, saj lahko naredimo majhne spremembe zelo hitro. S tem pridobimo veliko fleksibilnosti in povečanje hitrosti razvoja aplikacije.

Primer zagona vseh vozlišč, ki jih potrebujemo za izvajanje algoritma (Ukazi po vrsti):

1. `roscore`
2. roscd `usb_cam`
3. roslaunch `usb_cam-test2.launch`
4. roslaunch `turtlebot_bringup minimal.launch`
5. roslaunch `turtlebot_teleop keyboard_teleop.launch`
6. rosrun `turtlebot_controller diploma.py`

Kot vidimo zagon traja precej časa, zato je smiselno narediti svoj paket.

Na ta način zaženemo celotni sistem, ki nato zajema slike in vrača pozicijo robota na zaslon. V naslednjem poglavju bomo opisali algoritem, ki je bil razvit v okviru diplomske naloge.

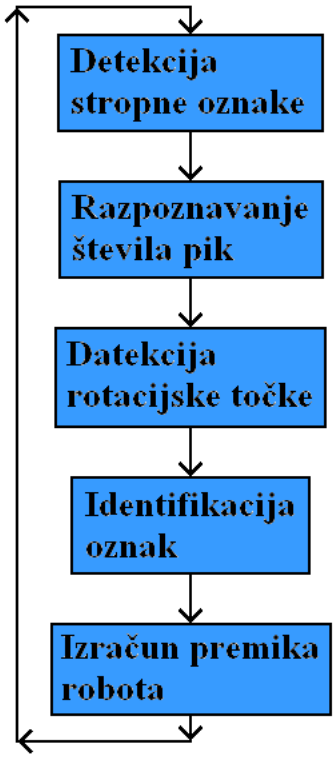
Poglavje 3

Algoritem za lokalizacijo z vizualno informacijo

3.1 Predstavitev glavnih problemov

Pri izdelavi algoritma smo morali razrešiti dva glavna problema. Eden izmed problemov je bila robustna detekcija stropnih oznak. K problemu spada tudi detekcija pozicije stropne oznake na sliki, identifikacija za katero oznako gre ter detekcija rotacije. Algoritem pa je moral detektirati oznake v manj kot pol sekunde, da je robot deloval tekoče. Robustno detekcijo smo lahko razbili na dva podproblema, ki sta bila natančnost detekcije samih oznak, ter pa zanesljivost detekcije.

Problem številka dve pa je bil najdene oznake pretvoriti v izračun nove lokacije robota. Algoritme smo razdelili na posamezne funkcije, ki so bile detekcija stropne oznake, razpoznavanje števila pik in rotacijske točke na oznakah, identifikacija oznak, izračun premika robota, kar lahko vidimo na Sliki 3.1. S tem smo si olajšali delo, saj smo celotni problem razdelili na manjše podprobleme.



Slika 3.1: Primer diagrama poteka za delovanje algoritma.

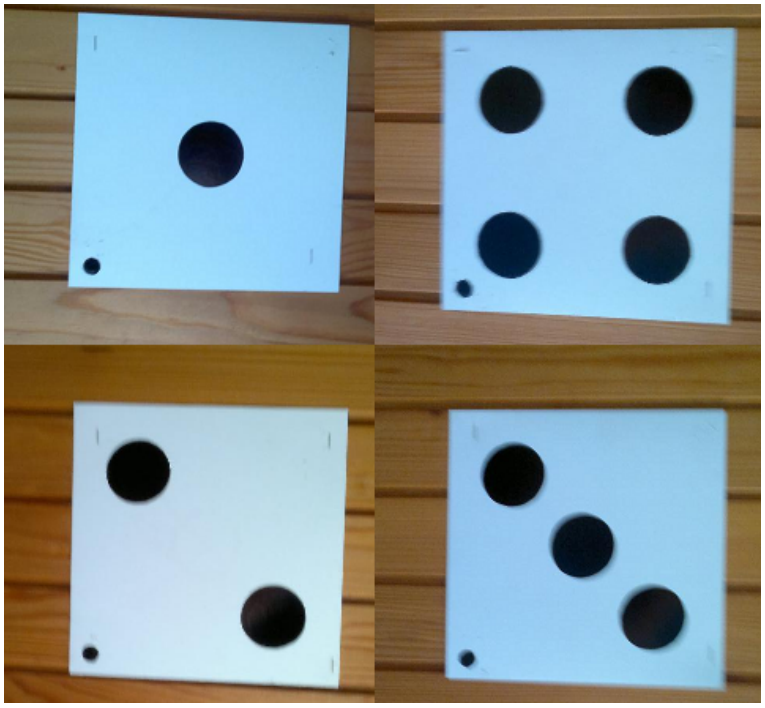
3.2 Stropne oznake

V splošnem poznamo dva principa lokalizacije z vizualno informacijo. Pri prvemu je kamera pritrjena na strop in snema robota, pri drugem pa je kamera pritrjena na robota ter snema del sobe (ponavadi strop). V prvem primeru za stropno oznako vzamemo kar samega robota. Mi pa smo vzeli drugi primer. Na robota smo pritrdili kamero ter jo usmerili proti strop. Ker na stropu ni bilo obstoječih referenčnih točk, smo sami izdelali stropne oznake po katerih bo algoritem izračunal pozicijo robota. S tem se je pojavilo več problemov kot na primer: oblika oznake, število oznak, razporeditev po stropu, identifikacija oznak, rotacija oznake, če jih naštejemo samo nekaj.

Glede oblike stropnih oznak smo hoteli zagotoviti čim večjo robustnost. Ko smo izbirali obliko stropnih oznak smo se osredotočili na osnovne geometrijske like kot so trikotniki, kvadrati, pravokotniki, itd. Ker naš strop ni vseboval nobene podobnosti s kvadratom smo se naposled odločili za to obliko stropnih oznak. Identifikacije oznak smo rešili tako, da smo na oznake narisali črne pike v taki postavitvi kot na kocki. Edina stvar, ki je še manjkala je bila zmožnost identifikacije rotacije. To smo rešili tako, da smo v kot oznake narisali majhno črno piko. Primer izdelanega stropne oznake po značilnostih, ki smo jih našteli, je na Sliki 3.2. Ker je bila soba v kateri smo izdelovali diplomsko nalogo majhna, smo se odločili za štiri stropne oznake, ki so razporejene v kvadrat. Tako se je iz vsake točke sobe videla vsaj ena oznaka. Tako smo se izognili možnosti, da robot zapelje izven detekcije oznak. V večjih prostorih pa bi morali prilagoditi število in velikost stropnih oznak, da bi algoritem deloval pravilno.

3.3 Problemi z zajemanjem slik

V kombinaciji zajemanja slik s spletno kamero in uporabo vozlišča *usb cam* smo imeli 5 do 10 okvirjev na sekundo (Frames Per Second). To je pomenilo, da smo imeli največji izpostavni čas posamezne slike 1/5 do 1/10 sekunde. Ker je bila kamera pritrjena na premikajočem se robotu, je to pomenilo, da



Slika 3.2: Primer vseh štirih stropnih oznak.

Slika 3.3: Primer slike (a) zajete med tem ko se je robot premikal in slike (b) ko je bil robot stacionaren.

smo dobili zamegljeno sliko (motion blur). To se je zgodilo zato, ker je bil čas izpostavitve svetlobi predolg. Če bi hoteli imeti kristalno čisto sliko med vožnjo bi moral biti čas izpostavitve manjši kot 1/60 sekunde. Ker tega nismo mogli doseči smo se odločili, da bomo robota ustavili vsakič, ko bomo hoteli zajeti sliko. To se je izkazalo za pravilno odločitev. Kot lahko vidimo na Sliki 3.3(a) je slika preveč zamegljena, da bi bila uporabna za robustno lokalizacijo, saj je bila zajeta med premikanjem robota. Ravno obratno pa lahko vidimo na Sliki 3.3(b), ki je bila zajeta na stacionarnem robotu. Slika je jasna in tako primerna za nadaljnjo obdelavo.

3.4 Detekcija in razpoznavanje stropnih oznak

Preden smo začeli pisati algoritem smo celotni problem razbili na več manjših problemov. To nam je olajšalo delo, saj smo se lahko osredotočili na vsak pod problem posebej. Najprej je bilo treba detektirati stropne oznake. Preden pa smo začeli detektirati, smo morali ločiti ospredje od ozadja (oznake od stropa). Kontrast med najsvetlejšim in najtemnejšim delom sobe je bil velik,

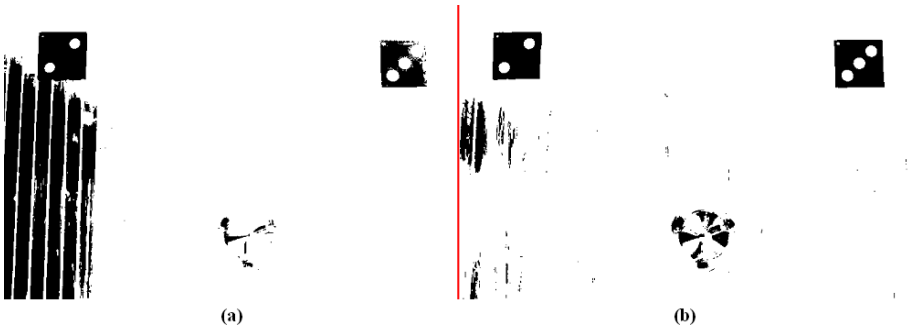
Slika 3.4: Primer slike z neenakomerno osvetlitvijo stropnih oznak.

soba je imela namreč samo eno okno, kar je pomenilo, da je bila intenziteta svetlobe blizu okna zelo velika, na drugem koncu sobe pa zelo majhna. To nam je pomagalo, da smo doumeli, da se osvetlitev zelo spreminja s prostorom v katerem smo. To nam je pomagalo k razvoju algoritma, ki bo robusten na osvetlitev v prostoru. Primer neenakomerne osvetlitve vidimo na Sliki 3.4.

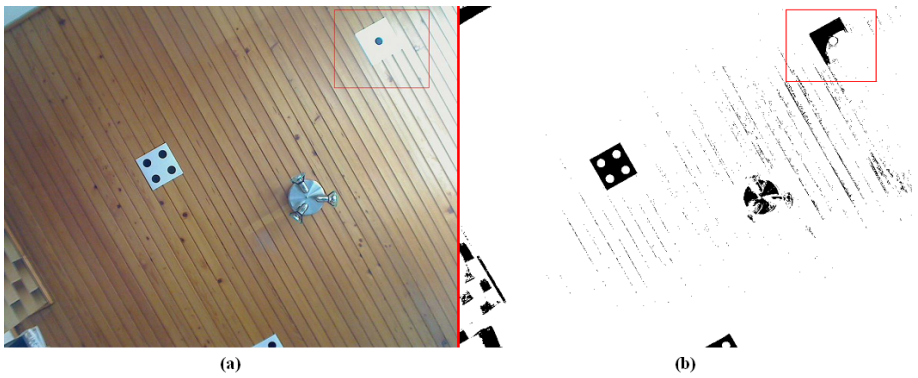
Če pretvorimo sliko v binarno s samo enim pragom (globalno upragovljenje) za celo sliko dobimo slabe rezultate tam, kjer je odstopanje svetlobe največje (veliko bolj svetlo ali temno). To lahko vidimo na Sliki 3.5(a). Če pa za sliko uporabimo več pragov (lokalno upragovljenje), pa dobimo zelo dobre rezultate, ki so vidni na Sliki 3.5(b).

Na Slikah 3.5(a) in 3.5(b) vidimo, da je velika razlika kateri postopek uporabimo, saj nam bo več pragov (lokalno upragovljenje) vedno vračalo boljšo sliko za nadaljnjo obdelavo. Nasprotno pa bo samo en prag v povprečju vračal slabše rezultate.

To, da smo uporabili lokalno upragovljenje, nam je omogočalo večjo zanesljivost pri detektirnaju. A vendarle vedno ni možno detektirati vseh oznak, če so pogoji preslabi, kot je vidno na Slikah 3.6(a) in 3.6(b). Detekcija stropne oznake v rdečem okvirju v tem primeru ni uspela, ker je bila osve-



Slika 3.5: Nad sliko (a) je bil uporabljen samo en prag = 165, nad sliko (b) pa je bilo uporabljeno lokalno upragovljenje.



Slika 3.6: Primer slabe detekcije z lokalnim upragovljenjem.

tlitev okrog stropne oznake prevelika in tako nismo detektirali kvadrata, ker nismo uspeli pravilno ločiti stropne oznake od ozadja. Detektirana regija pa ni ustrezala pravilom, ki smo jih postavili in tako smo preprečili napačno detekcijo.

Ko naposled algoritem loči stropne oznake, ki jih lahko vidimo v Sliki 3.2 od ozadja, nato na dobljeni sliki poišče vse regije. To naredi s funkcijo, ki sledi robovom in če funkciji uspe priti na mesto, kjer je začela, to pomeni, da ima element obseg in je tako regija. V regijah algoritem nato poišče vse robove, ki jih regija vsebuje. To naredi s pomočjo poligonskih krivulj. Nakar algoritem preveri njihovo velikost ter, če ustrezajo pravilom, ki so bila

določena v programu. Ker program detektira kvadrate, je smiselno določeno pravilo, da mora imeti štiri kote in, da so obsegi konveksni oziroma enostavni. Pomembna pa je bila tudi površina dobljenih likov.

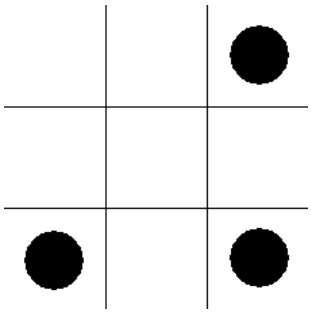
Detektirani kvadrati nato prispejo v funkcijo, ki razpozna vsako stropno oznako tako, da vrne število pik ter lokacijo rotacijske točke. Algoritem posamezno oznako prepozna tako, da vzame pod sliko v velikosti mejnega pravokotnika (ang. bounding box) okoli oznake, ter na njej izvede funkcijo, ki poišče regije. Ker pa želimo detektirati kroge in ne kvadrate je algoritem na koncu izvedel še funkcijo cvMinEnclosingCircle(), katera vrne najmanjši očrtan krog. To smo naredili zato, da iz popačenih krogov dobimo popolne kroge, ki pa so le malenkost večji od originala. Tako smo povečali robustnost algoritma. Iz tega kroga nato dobimo središče in polmer. To je zelo uporabno za razlikovanje detekcije pravih in napačnih krogov. Da smo lahko razlikovali med identifikacijskimi in rotacijskimi točkami smo jih naredili različnih velikosti. Rotacijske so bile 8% stranice oznake medtem ko identifikacijske 25%.

Ker lahko algoritem detektira manjše število točk, kot jih ima oznaka, ga lahko napačno identificira. Temu se algoritem izogne tako, da smo točke vstavi v tabelo 3x3. Preden točke vstavi v tabelo mora kvadrat poravnati, da nima rotacije. Ko točke vstavi v tabelo nato lahko predvideva, katera oznaka bi lahko bila.

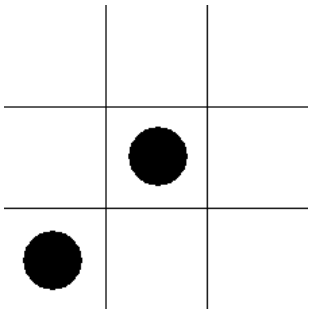
Kot vidimo v Sliki 3.7 je algoritem detektiral 3 točke, a vendar iz pozicije točk predvideva, da je kvadrat številka štiri. Algoritmu identifikacijo lajša tudi dejstvo, da imamo samo oznake z ena, dva, tri in štirimi točkami. Predvidevanje smo naredili po opazki, da vedno, ko napačno detektira število pik na oznaki je to število vedno manjše od števila pik. Le redko je število pik pravilno a pozicija pik napačna. Zato vedno algoritem poizkuša najti oznako z večjim ali enakim številom pik kot jih je detektiral.

Primer slike v kateri smo detektirali dve točki od treh vidimo na sliki 3.8.

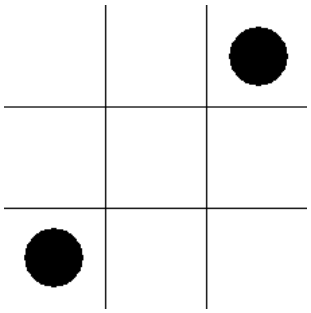
Če smo detektirali dve oznaki kot na kot Sliki 3.9 ne moremo razpoznati za katero izmed oznak gre, saj je lahko oznaka dva, tri ali štiri. V takih



Slika 3.7: Primer, ko smo detektirali tri točke vendar vemo, da ima kvadrat v resnici štiri.



Slika 3.8: Primer,ko smo detektirali dve točke vendar vemo, da ima kvadrat v resnici tri.



Slika 3.9: Primer ko iz detektiranih točk ne moremo vedeti za kateri izmed stropnih oznak gre.

primerih samo preštejemo število pik, ter iz pozicije poizkušamo prepoznati oznaki.

Kot je razvidno iz Slik 3.7 in 3.8 smo poskušali narediti identifikacijo stropnih oznak bolj robustno tako da nismo samo prešteli število pik na oznaki ampak smo tudi poskušali razpoznati oznako preko pozicije samih pik. Tako smo zelo zmanjšali število napačnih detekcij. Vendar v nekaterih primerih to ni bilo mogoče, kot vidimo v Sliki 3.9, saj iz same pozicije in števila pik ne moremo prepoznati stropne oznake in jo zato zavržemo, kot da je ne bi detektirali.

Psevdokoda algoritma:

```
DetekcijaStropnihOznak()
    1. Nad sliko naredimo lokalno upragovljenje in algoritem
    Canny za iskanje robov
    2. Poiščemo vse regije
    3. for (potujemo čez vse regije, ki smo jih našli)
    4.     V posamezni regiji poiščemo robove s pomočjo
             poligonskih funkcij
    5.     if (površina > površina_min in št. kotov == 4 in
               površina < površina_max)
    6.         if (najmanjši kot robov v kvadratu med 90 in 87)
    7.             Če ustreza vsem pravilom ga dodamo v seznam
```

```
    8. Nato iz množice detektiranih kvadratov poiščemo najboljše
    9. Identifikacije le teh

DetekcijaIndentifikacijskihTočk()
    1. for (potujemo čez vse oznake)
    2.     Najdemo mejni pravokotnik (bounding box) za oznako
    3.     Nad mejni pravokotnik naredimo lokalno upragovljenje
    4.     Poiščemo vse regije v kvadratu
    5.     for (potujemo čez vse regije)
    6.         V posamezni regiji poiščemo robove s pomočjo
                 poligonskih funkcij
    7.         Poiščemo najmanjši očrtan krog iz katerega dobimo
                sredino in premer
    8.         if (če je krog pravine velikosti za indent. točko)
    9.             dodamo v seznam točk
    10.        if (če je velikosti za rotacijsko točko)
    11.            dodamo v seznam rotacijskoh točk
    12. Pošljemo vse sezname in seznam kvadratov v
indentifikacijsko funkcijo

IdentifikacijaStropnihOznak()
    1. for (potujemo čez vse oznake)
    2.     Naredimo matriko 3x3 za hranjenje točk
    3.     V matriko vstavimo točke, ki smo jih detektirali in jih
            imamo v seznamu
    4.     Primerjamo postavitev pik v 3x3 tabeli z postavitvami pik
          na stropnih oznakah, ki imajo ena, dva, tri ali štiri pike.
    5.     if (ujema z znano postavitvijo)
    6.         v seznam dodamo število pik
    7.     else
    8.         poskušamo ugotoviti čemu je najbolj podobna postavitev
```

```
pik, ki smo jo detektirali
```

3.5 Izračun premika robota

Ko je algoritem detektiral stropne oznake ter jih identificiral, je preostal samo še izračun premika robota. Za izračun sta bili potrebni 2 sliki, ki jih nato primerja med seboj. Prvo sliko algoritem posname preden se robot začne premikati. Drugo sliko pa na neznani lokaciji. Prva težava, ki smo jo opazili je bila, če ne detektiramo isto oznako v prvi in drugi sliki. To je pomenilo, da smo morali narediti funkcijo, ki deluje na dva načina.

Preden algoritem začne premikati robota posname sliko, ki prikazuje razdalje do stropnih oznak (pozicije oznak) pri začetni poziciji. Te pozicije oznak si nato zapomni, saj jih bo potreboval pozneje. Začetno sliko bo potreboval pri obeh načinih izračuna. Ko robot posname drugo sliko nato primerja, če obe sliki vsebujeta vsaj eno isto oznako. To mu pove kateri del funkcije uporabiti, da bo dobili pravilni izračun. Izračun premika smo naredili na naslednji način.

Iz rotacijskih točk od obeh oznak smo dobili kot pod katerim sta oznaki rotirani. Oznaka, ki je bila slikana po premiku robota, nato za rotiramo za razliko v kotu od stare oznake okoli središča slike. Tako imamo pravilne koordinate točke stare in nove oznake. Nato pa samo še pravilno odštejemo točki med seboj in dobimo premik.

Če pa računamo premik robota preko oznak, ki nimata enako število pik, pa potrebujemo vedeti razdaljo med samimi oznakami. Razdalje smo zapisali v tabelo, katero prištejemo x,y koordinati. To nam vrne pravilno razdaljo, če vemo razdaljo do oznak oziroma se ta ne spreminja.

Seznam uporabljenih simbolov in spremenljivk.

- $\boldsymbol{x_1}, \boldsymbol{y_1}$ so koordinate stare stropne oznake na sliki, $\boldsymbol{\alpha_1}$ pa je rotacije le te,

- $\boldsymbol{x_2, y_2}$ so koordinate nove stropne oznake na sliki, $\boldsymbol{\alpha_2}$ pa je rotacije le te,
- $\boldsymbol{ih}$ je spremenljivka za višino slike v pikslih,
- $\boldsymbol{iw}$ je spremenljivka za širino slike v pikslih,
- $\boldsymbol{L}$ je hipotenuza, ki poteka od koordinatnega izhodišča pa do rotacijske točke oznake,
- $\boldsymbol{\gamma}$ je kot ki ga oklepa hipotenuza **L** in os X,
- $\boldsymbol{x'_2, y'_2}$ sta koordinati nove oznake brez rotacije le te,
- $\boldsymbol{Ocm}$ je velikost objekta v realnem svetu v centimetrih,
- $\boldsymbol{Opx}$ je velikost objekta na sliki v slikovnih elementih,
- $\boldsymbol{R}$ je pretvorni koeficient iz slikovnih elementov v centimetre,
- $\boldsymbol{\Delta x, \Delta y}$ sta realni premik robota v centimetrih.

Algoritem:
Vhod

$$(x_1, y_1)\alpha_1$$

$$(x_2, y_2)\alpha_2$$

1. izračun razlike orientacije,

$$\Delta\alpha = \alpha_2 - \alpha_1$$

2. izračun kota oznake po premiku robota z osjo X,

$$\mathrm{L} = \sqrt{(\frac{\mathrm{ih}}{2} - y_2)^2 + (x_2 - \frac{\mathrm{iw}}{2})^2}$$

$$\gamma = \frac{\frac{\mathrm{ih}}{2} - y_2}{\mathrm{L}}$$

3. izračun pozicije stropne oznake po premiku robota,

$$\delta = \gamma + \Delta\alpha$$

$$x_2' = \cos(\delta) * L + iw$$

$$y_2' = ih - \sin(\delta) * L$$

4. odštejemo koordinate stropnih oznak,

$$\Delta x' = x_1 - x_2'$$

$$\Delta y' = y_1 - y_2'$$

5. pretvorimo iz slikovnih elementov v centimetre,

$$\mathrm{R} = \frac{Ocm}{Opx}$$

$$\Delta x = \Delta x' * R$$

$$\Delta y = \Delta y' * R$$

Primer izračuna premika robota:

- velikost slike 3.10, je 500x500 slikovnih elementov,
- lokacija stropne oznake pred premikom je $\alpha_1(x_1, y_1) = 315(202{,}456)$,
- lokacija stropne oznake po premiku je $\alpha_2(x_2, y_2) = 315(179{,}90)$,
- nato izračunamo razliko kotov obeh stropnih oznak s formulo,

$$\Delta\alpha = \alpha_2 - \alpha_1$$

- tako dobimo kot,

$$\Delta\alpha = 315 - 315 = 0$$

- nato smo izračunali kot, ki ga oklepa premica od središča pa do stropne oznake z osjo X,

$$\gamma = \arcsin(\frac{250 - 90}{\sqrt{(250 - 90)^2 + (179 - 250)^2 = 175.045}}) = 113.928$$

- temu kotu nato prištejemo razliko kotov stropnih oznak,

$$\delta = \gamma - \Delta\alpha = 113.928 + 0 = 113.928$$

,

- nato iz dobljenega kota izračunamo nov položaj stropne oznake po formuli:

$$x = \cos(113.928) * 175.045 + 250 = 179$$

$$y = 250 - \sin(113.928) * 175.045 = 90$$

- sedaj le odštejemo in pretvorimo v centimetre

$$\Delta x' = (x_1 - x_2) = 202 - 179 = 23$$

$$\Delta y' = (y_1 - y_2) = 456 - 90 = 366$$

$$\mathrm{R} = \frac{20cm}{70px} = 0.2857$$

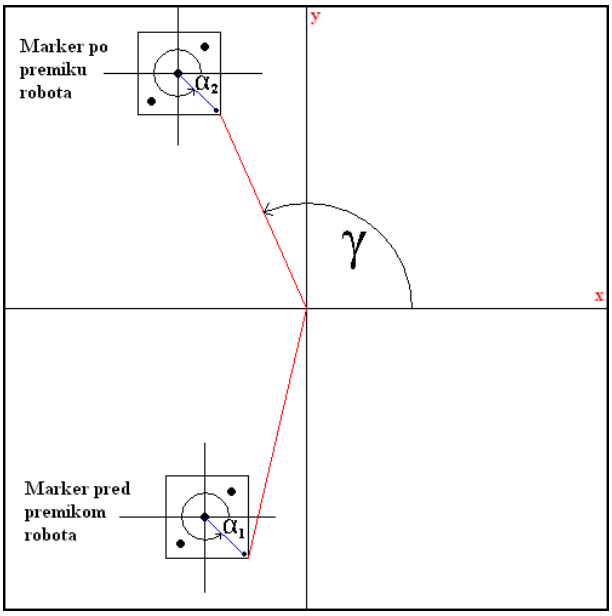
$$\Delta x = 23 * 0.2857 = 6.57$$

$$\Delta y = 366 * 0.2857 = 104.56$$

Drugi primer izračuna premika robota:

- velikost slike 3.11, je 500x500 slikovnih elementov,
- lokacija stropne oznake pred premikom je $\alpha_1(x_1, y_1) = 315(202{,}456)$,,
- lokacija stropne oznake po premiku je $\alpha_2(x_2, y_2) = 225(410{,}179)$,
- nato izračunamo razliko kotov obeh stropnih oznak s formulo,

$$\Delta\alpha = \alpha_2 - \alpha_1$$



Slika 3.10: Primer slike za izračun premika.

- tako dobimo kot,

$$\Delta\alpha = 225 - 315 = -90$$

- nato smo izračunali kot, ki ga oklepa premica od središča pa do stropne oznake po premiku z osjo X,

$$\gamma = \arcsin(\frac{250 - 179}{\sqrt{(250 - 179)^2 + (410 - 250)^2 = 175.045}}) = 23.929$$

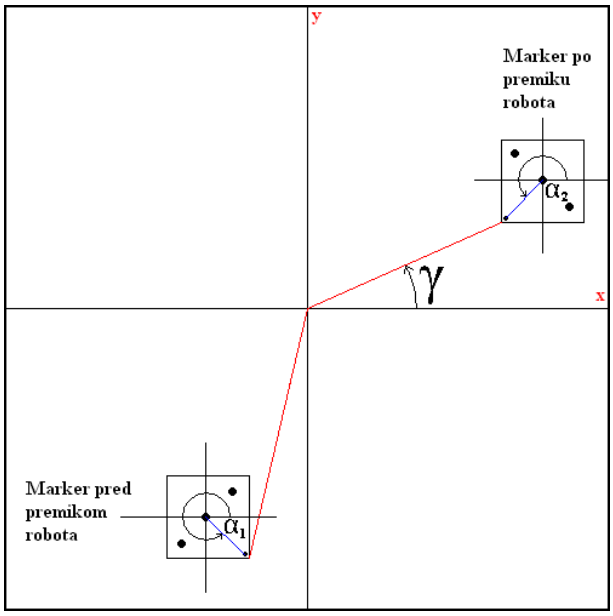
- temu kotu nato prištejemo razliko kotov stropnih oznak,

$$\delta = \gamma - \Delta\alpha = 23.929 \text{ - } -90 = 113.928$$

- nato iz dobljenega kota izračunamo nov položaj stropne oznake po formuli:

$$\Delta x' = \cos(113.928) * 175.045 + 250 = 179$$

$$\Delta y' = 250 - \sin(113.928) * 175.045 = 90$$



Slika 3.11: Primer slike za izračun premika.

- sedaj le odštejemo in pretvorimo v centimetre

$$\Delta x' = (x_1 - x_2) = 202 - 179 = 23$$

$$\Delta y' = (y_1 - y_2) = 456 - 90 = 366$$

$$\mathrm{R} = \frac{20cm}{70px} = 0.2857$$

$$\Delta x = 23 * 0.2857 = 6.57$$

$$\Delta y = 366 * 0.2857 = 104.56$$

Ko smo dobili rezultate premika, smo jih imeli v slikovnih elementih in ne v centimetrih ali metrih. Zato smo najprej morali najti način pretvorbe med obema enotama. To smo naredili tako, da smo detektirali stropno oznako ter detektirali njeno širino. Ker smo vedeli njeno velikost v realnem svetu smo z lahkoto potem izračunali pretvorbo po formuli:

$$R = \frac{\text{velikost stranice v realnem svetu v cm}}{\text{velikost stranice na sliki v pikslih}} = \frac{20}{70} = 0.2857$$

Nato smo dobljene rezultate samo množili s tem koeficientom ter dobili premik robota v realnem svetu: x = 23 * 0.2857 = 6.5, y = 366 * 0.2857 = 104.56.

Poglavje 4

Primerjava rezultatov

4.1 Odometrija robota

Vedeli smo, da odometrija robota akumulirala napako skozi čas, a nismo vedeli s kakšno hitrostjo. Preden smo se lotili primerjave odometrije robota z izdelanim algoritmom smo naredili nekaj preizkusov, da bi ugotovili natančnost odometrije. Izvedli smo dve vrsti preizkusov:

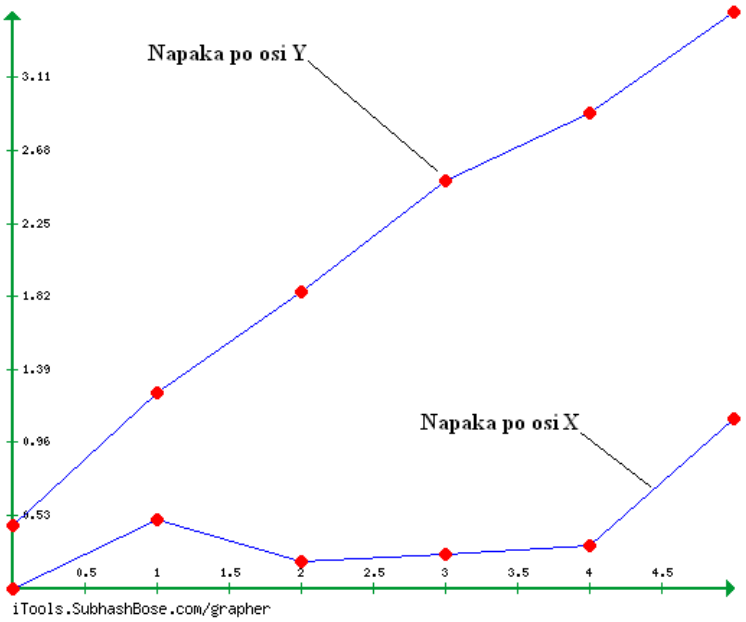
- robot se pelje v ravni črti, kjer jemljemo rezultate na nekaj deset centimetrov,
- robot se pelje v obliki kvadrata, kjer jemljemo rezultate na vsakem vogalu kvadrata.

Kot prvi preizkus smo robotu zadali naj se premika samo v smeri Y. Pri tem smo robota ustavljali vsakih nekaj sekund in pri tem jemali podatke. Napako robota v realnem svetu pa smo izmerili kar z metrom, ter videli odstopanje od točke kamor naj bi se premaknil. Tako smo dobili podatke, ki jih lahko vidite v tabeli 4.1. Kot je razvidno iz tabele se napaka odometrije povečuje s časom, kar smo tudi slutili.

Za drugi preizkus smo robotu spremenili pot iz ravne črte v kvadrat in s tem dodali še rotacijo. To je pomenilo, da je več možnosti za napako. Rezultate lahko vidimo v tabeli 4.2.

Premik robota v realnem svetu	Odometrija robota	Napaka
X = 0cm Y = 16,8cm	X = -0,1cm Y = 16,33cm	X = 0,1cm Y = 0,47cm
X = -0,5cm Y = 33,5cm	X = -0,55cm Y = 32,25cm	X = 0,5cm Y = 1,25cm
X = -1,5cm Y = 47,8cm	X = -1,24cm Y = 45,95cm	X = 0,26cm Y = 1,85cm
X = -3cm Y = 67,3cm	X = -2,7cm Y = 64,8cm	X = 0,3cm Y = 2,5cm
X = -5,5cm Y = 93,3cm	X = -5,15cm Y = 90,4cm	X = 0,35cm Y = 2,9cm
X = -9cm Y = 122,5cm	X = -7,9cm Y = 119cm	X = 1,1cm Y = 3,5cm

Tabela 4.1: Rezultati odometrije pri premikanju robota v ravni črti.



Slika 4.1: Prikaz rezultatov tabele 4.1 z grafom.

Premik robota v realnem svetu	Odometrija robota	Napaka
X = 0cm Y = 97,4cm	X = -1,2cm Y = 97,15cm	X = 1,2cm Y = 0,25cm
X = 100cm Y = 110cm	X = 101,5cm Y = 103,5cm	X = 1,5cm Y = 6,6cm
X = 133cm Y = -3cm	X = 127,3cm Y = -8,7cm	X = 5,7cm Y = 5,7cm
X = 0cm Y = -8cm	X = -7,2cm Y = -1,9cm	X = 7,2cm Y = 6,1cm

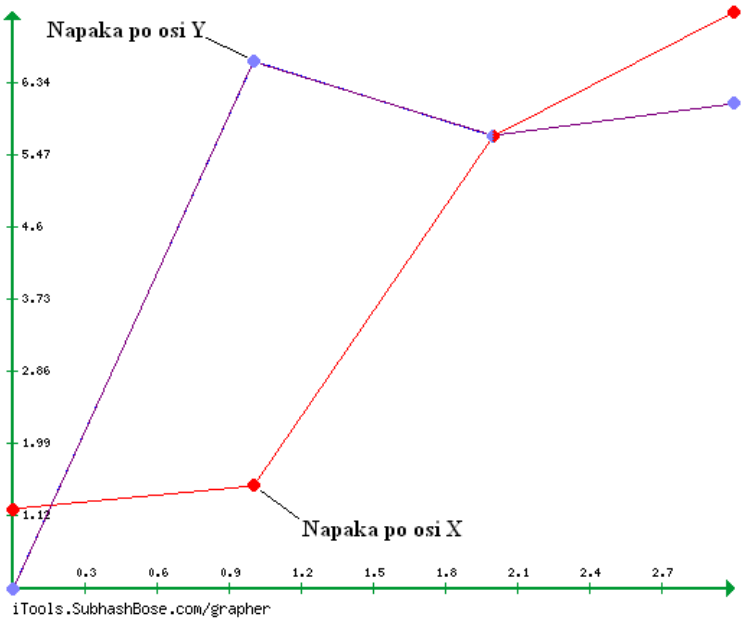
Tabela 4.2: Rezultati odometrije pri premikanju robota po poti v obliki kvadrata.

4.2 Vizualna informacija

Premik robota s pomočjo vizualne informacije smo preizkusili tako, da smo robota premikali po ravni črti. Tako smo dobili rezultate, ki jih lahko vidimo v tabeli 4.3.

4.2.1 Natančnost vizualne informacije

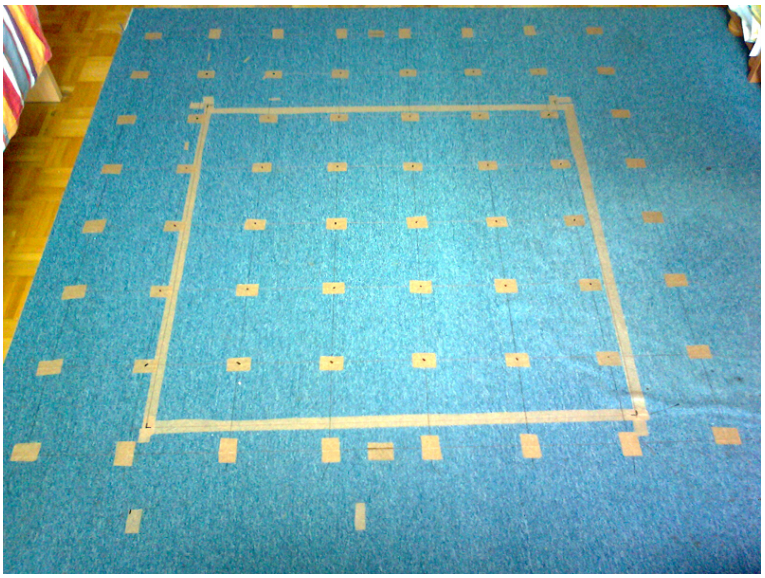
Za preizkus natančnosti vizualne informacije smo robota premikali po 8x8 mreži točk, kot je vidna na sliki 4.3.Točke so med seboj oddaljene dvajset centimetrov. V spodnjem levem kotu je bil pričetek, nato pa smo robota premikali po točkah na mreži od leve proti desni in nazaj dokler nismo dobili po 5 meritev na točko. Nato smo naredili povprečje nad meritvami, ki so bile izvedena na isti točki na mreži 8x8. Dobljene rezultate napake vizualne informacije smo nato vnesli v tabelo v takem vrstnem redu, kot smo jih dobili iz mreže točk 8x8. Rezultate lahko vidimo v tabeli 4.2.1. V tabeli 4.4 pa imamo vektorje napake, ki smo jih izračunali po formuli $R = \sqrt{X^2 + Y^2}$. Ti



Slika 4.2: Prikaz rezultatov tabele 4.2 z grafom.

Dejanski premik	Vizualna informacija	Napaka
X = 0cm Y = 22,8cm	X = 1,12cm Y = 23,24cm	X = 1,12cm Y = 0,44cm
X = -1cm Y = 42,8cm	X = -1cm Y = 38,8cm	X = 0cm Y = 4cm
X = -1cm Y = 63,5cm	X = 2,27cm Y = 59,67cm	X = 3,27cm Y = 3,83cm
X = -1cm Y = 82,8cm	X = 1,8cm Y = 83,96cm	X = 2,8cm Y = 1,16cm
X = 0cm Y = 102,6cm	X = 1,75cm Y = 105,29cm	X = 1,75cm Y = 2,69cm

Tabela 4.3: Rezultati vizualne informacije.



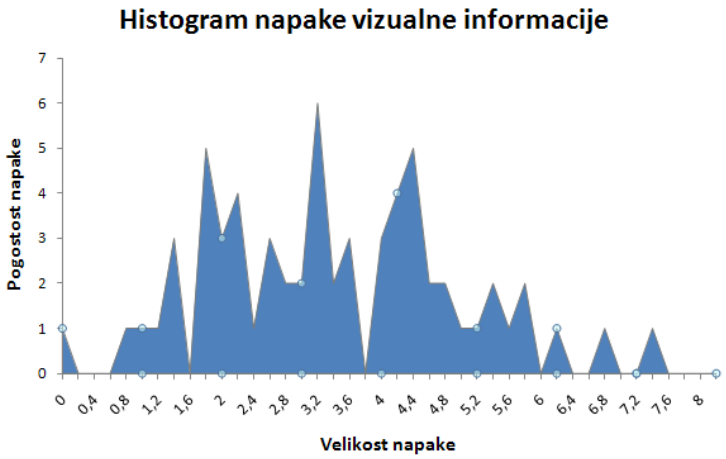
Slika 4.3: Prikaz poti robota po mreži točk 8x8.

nam povedo velikost napake ne pa smeri napake. Minimalna napaka, ki smo jo zasledili je bila -7cm maksimalna napaka pa 6cm. Kot lahko vidimo iz Slike 4.4 se napaka nahaja med 1,6cm in 5cm, kar prikazuje, da računanje pozicije z vizualno informacijo ne akumulira napake skozi čas.

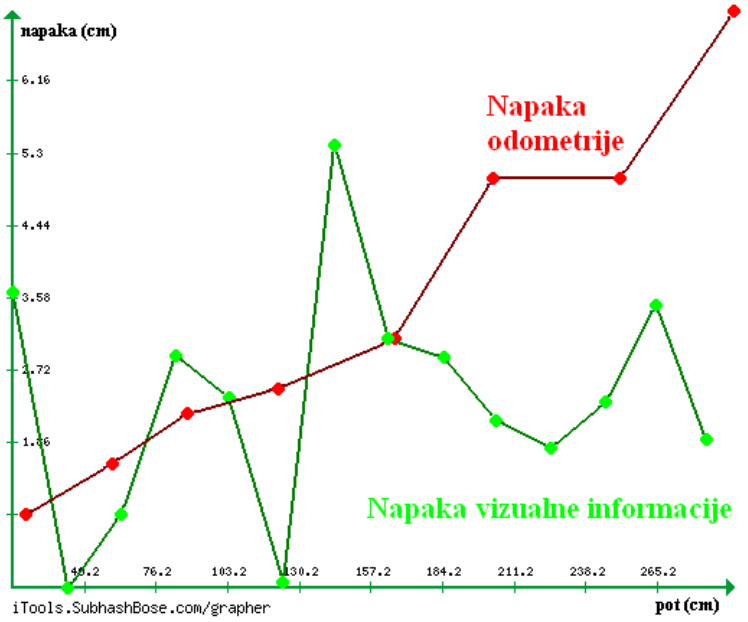
4.3 Vizualna informacija z odometrijo robota

Naposled smo primerjali algoritem vizualne informacije z odometrijo robota. To smo naredili tako, da smo robotu zadali pot v obliki kvadrata ter pot smo po osi Y (ravna črta). Pri obeh poteh smo dobili rezultate na dvajset centimetrov. Rezultate lahko vidimo v Slikah 4.5 in 4.6.

Odometrija je boljša do nekje enega metra. Potem pa napaka odometrije dohiti napako vizualne informacije in od enega metra do dveh metrov sta približno enako nenatančni obe metodi. Po dveh metrih in naprej pa je vizualna informacija boljša, kot je tudi razvidno iz zgornje Slike 4.5.



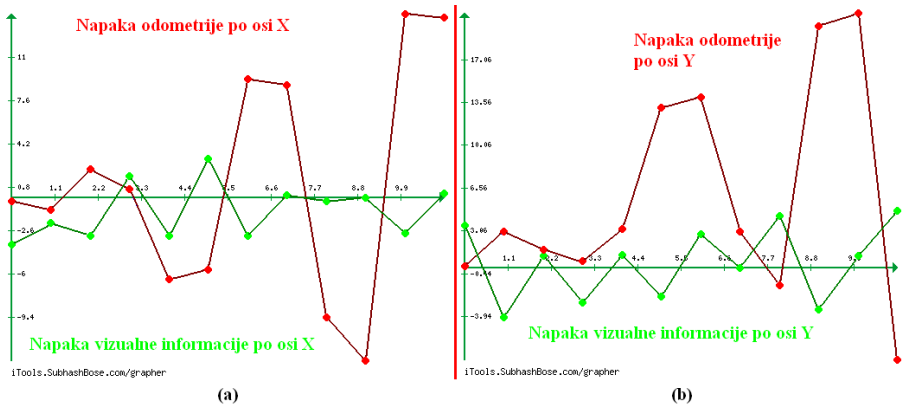
Slika 4.4: Prikaz histograma napake vizualne informacije.



Slika 4.5: Primerjava napak odometrije robota in vizualne informacije pri čem se je robot premikal smo po osi Y.

X = 0,6cm Y = 3cm	X = 0,5cm Y = -1,7cm	X = 1,7cm Y = 0,5cm	X = -2cm Y = -7cm	X = 1,7cm Y = -0,1cm	X = 1,4cm Y = 3cm	X = 1cm Y = 3cm	X = -1cm Y = -3cm
X = -0,3cm Y = 2cm	X = 3cm Y = -3cm	X = -1cm Y = -3cm	X = 2cm Y = 0cm	X = 1cm Y = -4cm	X = -2,8cm Y = 3,9cm	X = -3cm Y = 6cm	X = 1cm Y = 4cm
X = 0,5cm Y = -0,7cm	X = 1,7cm Y = -0,6cm	X = 3,7cm Y = 3,8cm	X = 4cm Y = 2,7cm	X = 4cm Y = 1,9cm	X = -1cm Y = 3cm	X = 1cm Y = 4cm	X = 1,9cm Y = 5cm
X = -4cm Y = -2,2cm	X = -1,7cm Y = -1,3cm	X = -0,3cm Y = 2,7cm	X = 0,8cm Y = 0,1cm	X = 1,4cm Y = 0,1cm	X = -1,7cm Y = -1cm	X = -3cm Y = 0cm	X = 6cm Y = 1,56cm
X = -0,37cm Y = -3cm	X = -0,9cm Y = -1,5cm	X = -0,3cm Y = -4,3cm	X = 1,1cm Y = -3,2cm	X = 1,9cm Y = -1,7cm	X = 5cm Y = 3cm	X = 0cm Y = -4cm	X = -3cm Y = -3cm
X = -1cm Y = 2,7cm	X = -0,3cm Y = 2,5cm	X = 0,3cm Y = 3,8cm	X = -0,1cm Y = 4,4cm	X = 0,56cm Y = 5,56cm	X = 1,1cm Y = 1,7cm	X = -2,3cm Y = 3,1cm	X = -5cm Y = 2,8cm
X = -0,5cm Y = -2,2cm	X = 1,5cm Y = -1,1cm	X = 1,1cm Y = 0cm	X = 2,2cm Y = 1,2cm	X = 1,9cm Y = 3,6cm	X = 1,13cm Y = 4,6cm	X = -0,9cm Y = 0,9cm	X = -2cm Y = 0,4cm
X = 0cm Y = 0cm	X = 0,8cm Y = 1cm	X = 1,7cm Y = 2cm	X = 1,14cm Y = 3,3cm	X = 1,7cm Y = 3cm	X = 3,4cm Y = 3,8cm	X = -2cm Y = 3,8cm	X = -0,6cm Y = 3,5cm

R = 3,06cm	R = 1,77cm	R = 1,77cm	R = 7,28cm	R = 1,7cm	R = 3,3cm	R = 3,16cm	R = 3,16cm
R = 2,02cm	R = 4,24cm	3,16cm	R = 2cm	R = 4,1cm	R = 4,8cm	R = 6,7cm	R = 4,1cm
R = 0,86cm	R = 1,8cm	R = 5,3cm	R = 4,83cm	R = 4,43cm	R = 3,16cm	R = 4,1cm	R = 5,34cm
R = 4,56cm	R = 2,14cm	R = 2,716cm	R = 0,8cm	R = 1,4cm	R = 1,97cm	R = 3cm	R = 6,2cm
R = 3,02cm	R = 1,75cm	R = 4,31cm	R = 3,38cm	R = 2,55cm	R = 5,8cm	R = 4cm	R = 4,24cm
R = 2,88cm	R = 2,52cm	R = 3,81cm	R = 4,4cm	R = 5,588cm	R = 2,02cm	R = 3,86cm	R = 5,73cm
R = 2,25cm	R = 1,86cm	R = 1,1cm	R = 2,5cm	R = 4,07cm	R = 4,74cm	R = 1,27cm	R = 2,04cm
R = 0cm	R = 1,28cm	R = 2,62cm	R = 3,49cm	R = 3,45cm	R = 5,1cm	R = 4,29cm	R = 3,55cm



Slika 4.6: Primerjava napak odometrije robota in vizualne informacije na sliki (a) po osi X in na sliki (b) po osi Y.

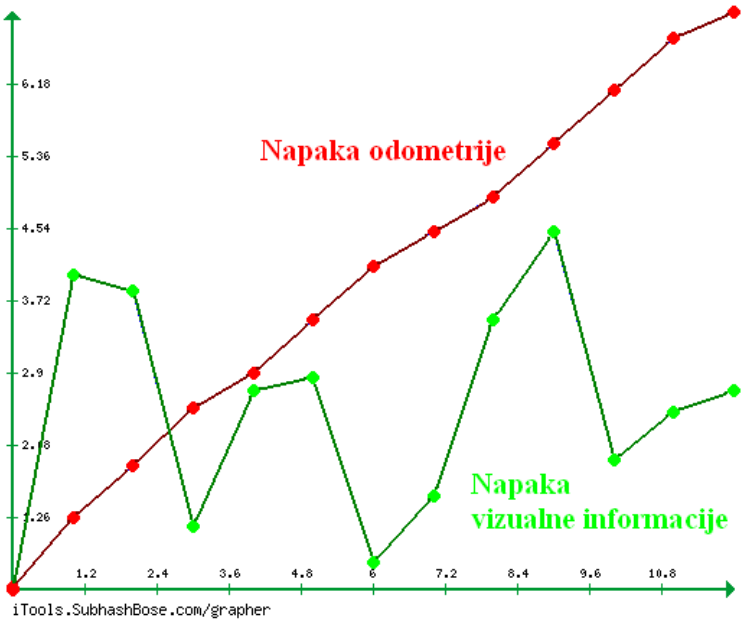
V Sliki 4.6(a) in 4.6(b) lahko vidimo napaki po oseh X in Y. Kot vidimo se približno enako akumulira napaka odometrije na obeh oseh, saj je pot robota v obliki kvadrata. Dejansko prevoženo pot pa predstavlja os X. Vse kar je nad osjo pomeni, da ne bila napaka manjša od prevožene poti, vse točke pod osjo X pa da je bila napaka večja od dejanskega premika robota.

Kot vidimo na Sliki 4.7, se že po parih meritvah napaka odometrije robota, ki se premika samo po osi Y, povzpne na 3.5cm, medtem ko napaka vizualne odometrije niha med 0 in 4cm.

4.4 Napačna detekcija

Pri detekciji ne gre vedno vse po načrtih, pa naj bo to detekcija oznake, identifikacija oznake ali pa detekcija rotacijske točke. Tako smo imeli tudi pri našem algoritmu podobne probleme z detekcijo, kar je pomenilo napačni izračun premika robota. Pri našem algoritmu tako poznamo tri vrste napačne detekcije.

- Napačno lahko detektiramo oznako. To pomeni, da detektiramo oznako tam kjer je ni. Če se to zgodi je ponavadi krivec kakšen predmet v pro-



Slika 4.7: Primerjava napak odometrije robota in vizualne informacije pri čem se je robot premikal smo po osi Y.

storu, kjer se nahajamo. Ta predmet lahko ni oblike kvadrata, vendar pod kotom pod katerim ga vidi robot, je lahko zelo podoben. Oblika predmeta ter lokalno upragovljenje pa pomenita, da čeprav predmet ni enake barve kot stropna oznaka, se detektira kot predmet ospredja in tako postane iste barve kot stropne oznake, ki so tudi del ospredja.

- Napačno prepoznava stropne oznake. To se zgodi na tri načine. Lahko detektiramo premalo število pik, da bi lahko prepoznali oznako. Nasprotno lahko detektiramo preveliko število pik in tako detektiramo oznako 3 namesto 1. Lahko pa detektiramo pravilno število pik, vendar ne na pravem mestu. V veliki večini je napaka detekcije ravno premalo število detektiranih pik na stropni oznaki.

- Napačno detektirana rotacijska točka. Rotacijska točka je krog, ki meri 1.4cm v premeru. To predstavlja le 7 % širine stropne oznake, ki je 21cm, kar pomeni, da je premer na sliki 5 slikovnih elementov. Ker je tako majhna, je velika verjetnost, da bomo detektirali večje število točk, ki imajo tako majhno površino. Tu kaj veliko ne moremo narediti, le to, da bi spremenili izgled točke ali pa povečali njeno površino.

Po testiranjih smo ugotovili, da če bi razdelili napačne detekcije na % bi videli, da je algoritem pri testiranjih napačno detektiral rotacijsko točko $\frac{\text{število napačnih detekcij}}{\text{število poizkusov}} = \frac{2}{320} = 0,625\%$. Nikoli pa algoritem ni napačno detektiral kvadrata oznake ali pa ga identificiral.

Napačno detektirana stropna oznaka ne prispeva k napačnem izračunu premika robota, saj tak oznako ne moremo identificirati ter jo tako ne uporabimo pri izračunu. Nasprotno pa, če napačno identificiramo stropno oznako, gledamo napačno razdaljo do iste oznake v začetni poziciji, ter tako dobimo napačen premik. Prav tako pa napačno detektirana rotacijska točka pripomore za napačen izračun premika. To se zgodi zato, ker dobimo napačen kot iz rotacijske točke in tako za rotiramo oznako za preveliko vrednost.

Algoritem za detekcijo smo naredili tako dobro, da deluje 99,375% brez napak od 320 meritev, ki smo jih izvedli. Težave smo predvsem imeli s premi-

kanjem kamere. Večina težav pa se pojavi zaradi napačne detekcije rotacijske točke. Če bi hoteli sistem prenesti v drug prostor, bi morali ustrezno spremeniti število in izgled stropnih oznak (predvsem velikost, če bi bil strop višji). Ker smo algoritem izdelali robustno, le tega nebi bilo potrebno spreminjati.

Poglavje 5

Sklepne ugotovitve

V diplomski nalogi smo izdelali algoritem, ki je računal premik robota v realnem svetu iz vizualne informacije. To smo naredili tako, da smo na robota pritrdili spletno kamero, preko katere smo zajemali slike stropa sobe v kateri je bil robot. Na strop smo pritrdili štiri stropne oznake preko katerih smo lahko izračunali premik robota. Nato smo to vizualno informacijo primerjali z odometrijo robota. Izkazalo se je, da se odometrija robota slabša z časom,medtem ko vizualna informacija ostaja enako natančna.

Ker odometrija poceni robotov ni zelo natančna pomeni, da tudi njihova funkcija ali ni natančna,ali pa je v večini primerov zelo potratna. To lahko odpravimo na dva načina. Prvi je, da v robota vgradimo bolj natančne senzorje, kar bi doprineslo k boljšim rezultatom, a bi pomenili znatno podražitev robota. Drugi način, ki smo ga uporabili pri naši diplomski pa je, da dodamo toliko senzorjev, kot jih je potrebno za minimalno reševanje problema. To bi bila le manjša podražitev. Problem pri robotiki je, da kaj hitro pridemo iz cenovnega razreda nekaj 100€ na nekaj 1000€ ali pa 10,000€ zato je boljše dodati več cenejših senzorjev. V našem primeru smo dodali spletno kamero, kar doprinese k bolj robustnemu sistemu.

Možne izboljšave diplomske naloge so predvsem v dveh smereh: izboljšava algoritma in izboljšava robota (ang. hardware). Diplomsko bi lahko izboljšali z boljšimi stropnimi oznakami. To bi pomenilo, da bi bila detekcija bolj

natančna. Natančnejša kamera bi pomenila, da se robotu ne bi bilo treba ustavljati za zajem slike. Če bi lahko bolje pritrdili kamero na robota, bi pomenilo, da se kamera med izvajanjem algoritma ne bi premikala, kar bi omenilo bolj natančen izračun. Če bi imeli iste oznake in istega robota, bi bilo možno več izboljšav. Lahko bi izboljšali identifikacijo stropnih oznak s samo pozicijo le teh. Naprimer, če smo detektirali oznako ena in dva ter nato detektiramo oznako, ki je točno nad oznako ena in ne vemo število pik. Potem lahko sklepamo, da je oznaka štiri, zato ker vemo kako smo jih postavili. Naslednjo stvar, ki bi jo lahko izboljšali, pa je detekcija rotacijske točke. Problem je njena velikost, zato lahko detektiramo večje število le-teh. Če bi detektirali več kot 1 rotacijsko točko, bi potem morali prepoznati katera je prava. To bi prineslo večjo pravilnost izračuna.

Metoda, ki smo si jo zamislili. je delovala zelo solidno, vendar je še veliko prostora za nadaljnjo nadgradnjo algoritma. Vizualna informacija v smislu kamere se čedalje več uporablja, saj so kamere dandanes zelo razširjena strojna oprema. So zelo poceni in iz njih lahko dobimo kritične informacije, ki pomagajo sistemu pri opravljanju dela. Kamere so tudi zelo uporabne pri lokalizaciji v prostoru ali v svetu. Algoritem je uporaben za lokalizacijo v prostoru, ki ima strop in stropne oznake. Naprimer prinašanje elementov iz sobe A v sobo B, kjer se robot orientira v prostoru po stropnih oznakah. Tako je algoritem, ki smo ga razvili zelo uporaben, saj je koristno vedeti položaj mobilne platforme in ga je možno integrirati zelo hitro, saj potrebujemo le spletno kamero in računalnik, s tem pa pridobimo zelo veliko. Torej, če poznamo premik robota v prostoru, lahko na tem zgradimo algoritem, ki bo bolj učinkovit in nam bo omogočal večje število nalog, ki jih bo robot lahko opravljal.

Literatura

[1] (2012) About-robots. Dostopno na:
http://www.about-robots.com/roomba-vacuum-robot.html.

[2] (2012) Iheartrobotics. Dostopno na:
http://www.iheartrobotics.com/2010/05/
testing-ros-usb-camera-drivers.html.

[3] (2012) Logitech. Dostopno na:
http://www.logitech.com/en-us/support/webcams/3056.

[4] (2012) Nrqm.pbworks. Dostopno na:
http://nrqm.pbworks.com/w/page/25934090/Roomba%20%E2%80%93%
20Sensors.

[5] (2012) Opencv.willowgarage. Dostopno na:
http://opencv.willowgarage.com/wiki/.

[6] (2012) Protechrobotics. Dostopno na:
http://www.protechrobotics.com/proddetail.php?prod=
DEV-00740.

[7] (2012) Pythonware. Dostopno na:
http://www.pythonware.com/products/pil/.

[8] (2012) Ros.org. Dostopno na:
http://ros.org/wiki/Robots/TurtleBot.

[9] (2012) Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Python_(programming_language).

[10] (2012) Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Roomba.

[11] (2012) Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/ROS_(Robot_Operating_System).

[12] (2012) Zagrosrobotics. Dostopno na:
http://www.zagrosrobotics.com/shop/item.aspx?itemid=678.

[13] (2012) Zagrosrobotics. Dostopno na:
http://www.zagrosrobotics.com/shop/item.aspx?itemid=679.