Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Jani Bizjak

Iskanje predmetov v prostoru z mobilno platformo in barvno-globinsko kamero

diplomsko delo
univerzitetni študijski program prve stopnje
računalništvo in matematika

Mentor: doc. dr. Danijel Skočaj

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo! Glej tudi sam konec Poglavja **??** na strani **??**.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jani Bizjak, z vpisno številko **63090021**, sem avtor diplomskega dela z naslovom:

Iskanje predmetov v prostoru z mobilno platformo in barvno-globinsko kamero

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, Podpis avtorja:

Zahvaljujem se vsem, ki ste mi pomagali pri izdelavi diplomske naloge.

Posebej se zahvaljujem mentorju dr. Danijelu Skočaju za nasvete in vodenje pri izdelavi naloge.

Kazalo

Povzetek

V sodobnem svetu se roboti iz tovarn in znanstvenih laboratorijev začenjajo uveljavljati v domačih gospodinjstvih, kjer pomagajo pri hišnih opravilih. V tem diplomskem delu je robotski sesalnik iRobot Roomba nadgrajen tako, da zna poiskati predmete v prostoru. Na mobilno platformo je nameščena barvno-globinska kamera Kinect, ki omogoča zajemanje 3D informacije o prostoru. Operacijski sistem robota je ROS, ki ponuja že izdelane rešitve za nekatere probleme robotike, hkrati pa skrbi za komunikacijo med strojno in programsko plastjo robota. Za prepoznavanje predmetov se uporablja že izdelan program RoboEarth. Poudarek diplomskega dela je na sistematičnem preiskovanju prostora ter povezovanju vseh komponent, da je sistem zmožen opraviti zastavljeno nalogo. Z namenom lažjega razumevanja robota je njegovo razmišljanje vizualizirano in verbalizirano. Na koncu dela so natančno izmerjene meje, pri katerih robot še vedno natančno prepoznava predmete. Nato pa je na podlagi prej izmerjenih parametrov izvedenih več testnih iskanj v prostoru.

Ključne besede

ROS, turtlebot, iskanje predmetov z mobilno platformo, robot

Abstract

Intelligent robots are nowadays making their way from laboratories to domestic homes. In this thesis, a vacuum cleaner robot iRobot Roomba is upgraded so it can search for objects in a room. Robot sensors alone are not good enough that is why a RGBD camera Kinect is added on top of the robot. For software part the robot meta operating system ROS is used. ROS connects the robot's hardware and software together and is easily upgradable with custom packages for different tasks. This thesis is divided in two parts: object recognition and systematical search of space. For object recognition, a solution made by RoboEarth is used. The focus of this thesis is on navigating the robot and bringing the whole system together. For better representation of the robot's actions, visualization and verbalization of these actions are added. Before the evaluation of the program on a real robot, exact parameters of the visual recognition system are measured and then used to create a test environment.

Keywords

ROS, turtlebot, object search with mobile platform, robot

Poglavje 1

Uvod

Ljudje so že v antiki poizkušali narediti stroje, ki bi opravljali določena opravila namesto njih. Seveda te naprave niso bile nič kaj podobne napravam, ki jih danes imenujemo roboti. Prvi zametki robotov, kot jih poznamo danes, so se pojavili ob koncu 19. stoletja. Prvi sodoben robot, ki je bil popolnoma programabilen, je bil izdelan leta 1954. Uporabljali so ga v avtomobilski industriji.

Besedo robot je prvi uporabil češki pisatelj Karel Čapek v svoji igri Rosumovi univerzalni roboti, ki je bila izdana leta 1920. Roboti v igri so bili humanoidni stroji, ki so bili sposobni lastnega inteligentnega razmišljanja. Njihov glavni namen je bil opravljanje dela namesto ljudi. Danes si večina ljudi robote predstavlja kot humanoidne stroje, ki so sposobni inteligentnega razmišljanja in samostojnega delovanja.

Sodobne robote lahko razdelimo na več vrst. Od tistih, ki jih upravljajo ljudje z radijskim vodenjem, industrijskih, ki so vnaprej programirani za točno določen gib, do avtonomnih, ki so sposobni razmišljanja in opravljanja preprostih opravil samostojno na podlagi svojih zaznav. Slednjim se bomo posvetili v tem delu.

Problem izdelave inteligentnega robota je zelo obsežen in se razprostira preko več področij. Najprej mora robot zaznati okolje, v katerem se nahaja. Tukaj se pojavi problem natančnosti senzorjev, ki jih robot uporablja.

Večina senzorjev ne bo delovala 100 procentno natančno, zato moramo naše algoritme narediti takšne, da upoštevajo šum v podatkih. Ko robot obdela podatke iz senzorjev, jih lahko uporabi. Temu sledi planiranje akcij, ki ga bodo privedle k cilju. Da bo robot lahko izpolnil nek cilj, bo moral imeti za to primerno strojno opremo, npr. če bo moral prenesti predmet iz točke A na točko B, bo potreboval kolesa in roko, s katero bo predmet prijel.

V tem diplomskem delu se bomo posvetili izdelavi mobilnega robota, ki bo samostojno navigiral po prostoru ter v njem poiskal predmete, ki mu jih bomo določili. Nalogo bomo razdelili na tri podprobleme. Poskrbeli bomo za inteligentno preiskovanje prostora z robotom iRobot Roomba. Za prepoznavanje predmetov bomo uporabili že izdelano rešitev RoboEarth. Ker senzorji na robotu niso dovolj natančni za prepoznavanje predmetov, bomo na robota namestili barvno-globinsko kamero Kinect. Vse komponente programa bomo naredili takšne, da jih bomo lahko uporabili v meta operacijskem sistemu ROS, ki bo poskrbel tudi za komunikacijo med strojno in programsko opremo.

Nalogo bomo zaključili s preizkusom in ovrednotenjem uspešnosti našega programa. Robota bomo naučili prepoznati nekaj predmetov, nato pa jih bo moral poiskati v prostoru. Potrebno bo tudi izmeriti razdalje, na katerih robot še uspešno zaznava predmete, ter glede na te parametre postaviti predmete po prostoru. Robot bo opravil več voženj, med vsako vožnjo pa bomo predmete naključno postavili.

Poglavje 2

Sistem

Namen diplomske naloge je izdelati robota, ki se bo znal samostojno premikati po prostoru ter poiskati določene predmete. Robotu bo vnaprej podan 3D model predmeta, katerega mora poiskati. Teh predmetov je lahko tudi več, različnih ali enakih. Oddaljeni morajo biti vsaj 0,5 metra drug od drugega, ustrezati pa morajo dimenzijam, katere lahko video sistem Kinect zazna. Pri preiskovanju prostora bomo privzeli omejitev, da mora biti prostor konveksen, torej brez raznoraznih udrtin ter hodnikov. Ker je problem detekcije predmetov zahteven, bomo uporabili že izdelane detektorje, mi pa se bomo posvetili izdelavi algoritmov za preiskovanje prostora ter povezovanjem vseh komponent v sistem, ki bo opravil zgoraj opisano nalogo. Za robota bomo uporabili mobilno platformo iRobot Roomba, ki bo rahlo prirejen tako, da bo podpiral sistem ROS. Nanjo bomo namestili barvno-globinsko kamero Kinect za vizualno zaznavanje okolja. Tak sistem se imenuje TurtleBot, prikazan je na Sliki 2.1.

2.1 Robot Roomba

Robotski sesalnik Roomba podjetja iRobot je popularen pri razvijalcih robotskih sistemov, saj je poceni, ponuja pa skoraj toliko kot dražji modeli robotov. Robot ima dve kolesi, ki služita kot pogonski ter krmilni sistem.

Slika 2.1: Robot turtlebot z nameščeno barvno-globinsko kamero Kinect. [4]

Smer ter hitrost robota se izračunata na podlagi podatkov s pogonskih koles. Ker kolesa na določenih podlagah drsijo, ima na sprednjem delu dodatno kroglico, ki se dotika tal. Na podlagi njenega vrtenja lahko bolj zanesljivo izračuna dolžino in smer premika. Na sprednji strani robota se nahaja odbijač, ki prekine napajanje v kolesa, ko se robot zaleti v predmet. Ostali senzorji ter sesalec se ob priklopu robota na računalnik izklopijo. Da lahko robota priklopimo na računalnik, ga moramo rahlo modificirati. Natančen vodič najdete na [5].

Za nadalno delo z robotom je pomembno, da natančno poznamo njegove omejitve torej najvišje/najnižje hitrosti. V Tabeli 2.1 so zapisane specifikacije, ki jih bomo potrebovali pri navigaciji.

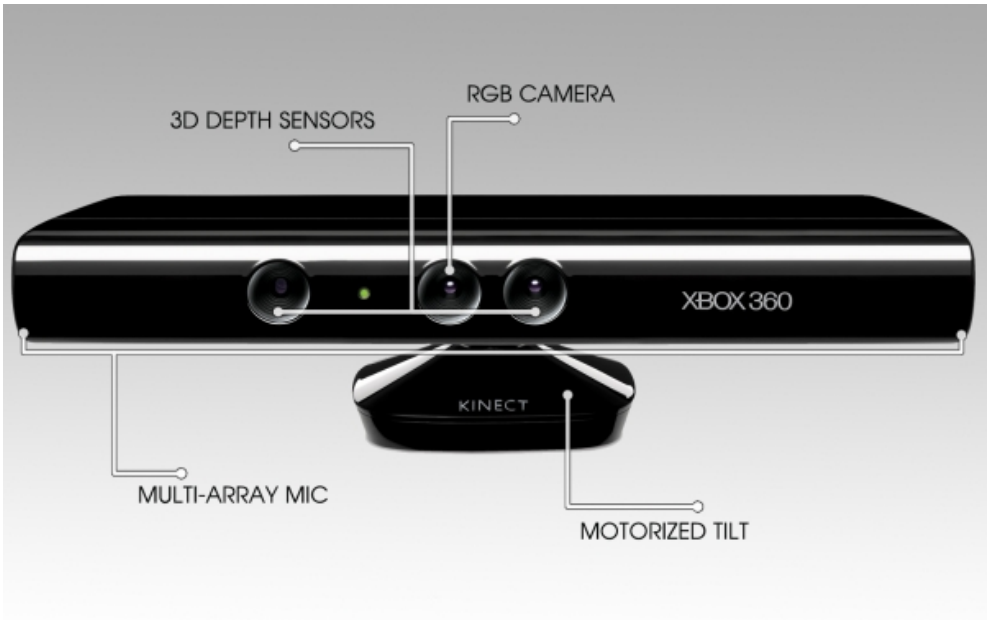
2.2 Barvno-globinska kamera Kinect

Za vizualno zaznavanje bomo na robota namestili barvno-globinsko kamero Kinect, ki je prikazana na Sliki 2.2. Kinect je sistem senzorjev za zajemanje

Tabela 2.1: Specifikacije

Premer robota	36 cm
Največji pospešek	2,5 m/s^2
Največji rotacijski poskešek	3,2 rad/s^2
Maksimalna hitrost	0,5 m/s
Minimalna hitrost	0,1 m/s
Maksimalna hitrost obračanja	1,0 rad/s
Minimalna hitrost obračanja	0,4 rad/s

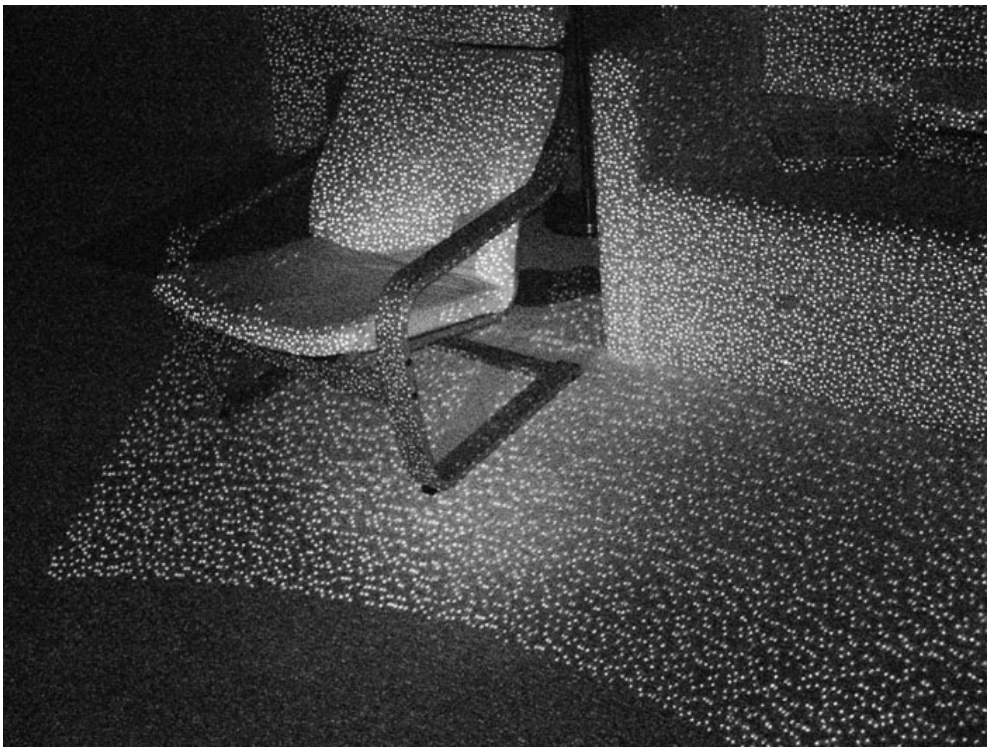
3D slike. Izdelan je bil leta 2010, namenjen, upravljanju igralne konzole Xbox 360 s kretnjami. Zaradi nizke cene je postal priljubljen pri raziskovalcih na področju robotike in umetnega zaznavanja. Sistem je sestavljen iz dveh



Slika 2.2: Barvno-globinska kamera Kinect. [20]

kamer ter infrardečega svetila. Svetilo po prostoru projicira mrežo infrardečih točk, kar lahko vidimo na Sliki 2.3, na podlagi popačenja, zaradi oddaljenosti ter oblike predmetov pa lahko nato izračuna oddaljenost posamezne točke.

Za zajemanje slike skrbita dve kameri: prva je navadna RGB (Red Green Blue) kamera z ločljivostjo 640×480 slikovnih elementov, druga pa infrardeča kamera z enako ločljivostjo, ki zaznava projicirane pike. Zaradi ločljivosti kamer sistem ne zazna dobro predmetov, ki so manjši od 5 centimetrov, najbolje pa deluje z razdalje 0,5 metra do 6,0 metra. Vidni kot je 50°. Dodatne informacije o Kinectu si lahko preberete na [6].



Slika 2.3: Projiciranje infrardečih pik v prostoru. [7]

2.3 ROS

V robotiki se soočamo z velikim problemom pomanjkanja standardov. Imamo veliko različnih senzorjev ter tipov robotov, ki jih zaradi svoje raznolikosti težko postavimo v nek skupni univerzalni program. S tem namenom je bil razvit ROS (Robot Operating System). ROS je meta operacijski sistem za

robote, ki se razprostira od strojne podpore za različne robote ter senzorje, do visokonivojskih algoritmov za navigacijo, računalniški vid in ostala opravila. Podpira tudi simulator, v katerem lahko testiramo delovanje robota brez strahu pred poškodbami resničnih sistemov.

ROS je sestavljen iz t.i. *paketov (ang. package)*, ki predstavljajo zaključen del programa za neko opravilo. Za komunikacijo med *paketi* je v ROS-u poskrbljeno s t.i. *temami (ang. topic)*. Te delujejo na podoben način kot radijske postaje v resničnem svetu. Vsaka *tema* ima svoje ime in vrsto sporočila, ki ga lahko pošilja. *Tema* lahko prejema ter pošilja samo eno vrsto sporočila, seveda pa se nanjo lahko priključi več *poslušalcev (ang. listener)* in *pošiljalcev (ang. publisher)* istočasno. Uporabna orodja (programi) se v ROS-u imenujejo *servisi (ang. services)*. *Servisi* so deli *paketov*, od katerih lahko zahtevamo neko storitev (podobno kot klicanje metod, vendar za razliko od njih niso omejeni le na razred, v katerem se nahajajo, temveč delujejo v sklopu celotnega sistema ROS). Od ostalih paketov se ločijo po tem, da nimajo metode main. Tako v *temah* kot v *servisih* so sporočila, ki jih pošiljamo definirana v *sporočilih ROS (ang. ROS messages)*, seveda pa lahko definiramo tudi svoja.

Ko govorimo o inteligentnih robotih, ki imajo neko interakcijo z okoljem, je skoraj nemogoče poustvariti enake pogoje za dva poskusa. Nemogoče je pričakovati, da se bo robot petkrat odpeljal po enaki poti ali da bo zaznal enake značilne točke kot v prejšnji sliki, saj veliko algoritmov (npr. RANSAC [13]) uporablja tudi naključnost z namenom hitrejšega izračuna dovolj dobrih približkov. Zato ROS podpira sistem, ki posname vse podatke na *temah*, nato pa jih kasneje (vse ali selektivno) v enakem zaporedju predvaja v simulatorju. S tem se bo robot vedno premikal po enaki poti in zaznaval enako sliko, kar nam omogoča lažje razhroščevanje.

Poglavje 3

Navigacija

V tem diplomskem delu smo uporabili sistem, opisan v Poglavju 2. Najprej bomo morali poskrbeti, da se bo robot Roomba lahko premikal po prostoru. Nato bomo poskrbeli za natančno navigacijo z uporabo in izgradnjo zemljevida ter predstavili problem koordinatnih sistemov v robotiki. Ker gre tukaj za splošne probleme, s katerimi se soočajo vsi robotski sistemi, so vse naslednje metode vključene v sistemu ROS [14, 15, 16]. Na koncu bomo sistem nadgradili, tako da bo sposoben preiskovanja prostora.

3.1 Načrtovanje poti

Načrtovanje poti poteka na dveh nivojih: globalno ter lokalno. Pri *globalnem načrtovalcu (ang. global planner)* se v začetku zgradi *načrt (ang. plan)* poti na podlagi vnaprej izdelanega zemljevida. Nato se *načrt* pošlje *lokalnemu načrtovalcu (ang. local planner). Lokalni načrtovalec* se poskuša držati čim bližje poti, ki jo je začrtal *globalni načrtovalec.* Za svojo navigacijo uporablja lokalni zemljevid, ki se posodablja v realnem času. Glede na podatke iz lokalnega zemljevida začrta dejansko pot robota. Ukaze za premik robota pošilja *lokalni načrtovalec*, sam ukaz pa je sestavljen iz informacije o smeri ter hitrosti. ROS nato poskrbi za dejansko komunikacijo z elektromotorji.

ROS ima implementirana dva *globalna načrtovalca* poti. Prvi je preprost,

saj *načrt* poteka v ravnem vektorju od trenutne lege robota do končnega cilja. Če se končni cilj nahaja na robotu nedostopnem področju (npr. stena), se bo v nekem radiju od cilja naključno izbral nov cilj, če tudi ta ne bo dosegljiv, se bo postopek n-krat ponovil. Ta *načrtovalec poti* je uporaben le pri lepih prostorih brez ovir, saj odpove, če se med ciljem in robotom nahaja ovira. Drugi *načrtovalec* uporablja algoritem Dijkstra.

Algoritem spremeni zemljevid v graf, kjer je vsako vozlišče uteženo glede na dostopnost. Algoritem na ta način vedno poišče najkrajšo možno pot. Več o algoritmu lahko preberete na [8].

Po izdelavi globalnega *načrta* se le-ta pošlje v *lokalni načrtovalec*. *Lokalni načrtovalec* začrta več naključnih poti v smeri globalnega *načrta*. Ker lokalni načrtovalec deluje na lokalni ravni, se tudi globalni *načrt* upošteva le v okolici nekaj decimetrov stran od robota. Nato načrtovalec vsako začrtano pot poizkuša prevoziti v simulatorju ter izbere najboljšo. Za simulacijo je potrebno, da natančno poznamo način gibanja ter parametre robota.

Oglejmo si naslednji *primer*. Imamo dva robota, prvi se lahko na mestu rotira okoli svoje osi, drugi pa kot osebni avtomobil v nekem radiju. Recimo, da je globalni *načrt* tak, da od robota zahteva zavoj za 180° ter vožnjo nazaj po enaki poti. V prvem primeru se bo robot lahko pred zavojem ustavil ter na mestu obrnil za 180° nato pa nadaljeval pot. V drugem primeru se robot ne bo ustavljal, temveč bo pot nadaljeval krožno z nekim radijem. Očitno je, da bo robot v drugem primeru prevozil daljšo pot.

3.2 Gradnja zemljevida

Da bo lahko *globalni načrtovalec* poti dobro začrtal pot, potrebuje zemljevid. Robot mora za izgradnjo zemljevida poznati svojo točno lego, po drugi strani pa za določanje lege potrebuje zemljevid. Gre za problem kokoši in jajca. Ta problem rešujemo z algoritmom SLAM (Simultaneous Localization and Mapping) [3].

Naiven način reševanja problema hkratne lokalizacije in gradnje zemlje-

vida je ta, da v začetku določimo lego robota (največkrat robota postavimo kar v koordinatno izhodišče z rotacijo 0°), nato v naslednjem koraku iteracije zgradimo zemljevid s podatki, ki jih zaznajo senzorji. Ko se robot nato premakne, spremenimo lego robota tako, da prejšnji legi prištejemo dolžino premika [dx,dy] in rotacijo α ter ponovno zgradimo zemljevid tokrat iz nove lege. Čeprav se zgoraj opisana ideja zdi dobra, pa v resničnem svetu seveda ni tako. Senzorji robota niso popolni, kar privede do šuma v podatkih. Na primer, če se robot vsakič pri rotaciji okrog svoje osi zmoti za 5°, se bo napaka pri vsakem nadaljnjem obratu seštevala in robot bo po 10 rotacijah lahko imel napako 50°. Problem je, da takih napak na senzorjih ni enostavno odpraviti. Na vrsto napake vpliva veliko dejavnikov, od vrste podlage (spodrsavanje koles), obrabe materialov iz katerih je narejen robot (eno kolo se vrti hitreje kot drugo), spreminjanje svetlobe, nezaželeni odbleski ter odsevi okolice ipd. Zaradi tega moramo naš naiven algoritem nadgraditi, da nanj ne bodo več tako močno vplivale napake ter da jih bo sam znal zaobiti ter iz danih podatkov čim bolj točno izračunati lego in zgraditi zemljevid.

Zaradi zgoraj omenjenih težav o robotovi legi nikoli ne bomo z verjetnostjo 1 prepričani, da se robot res nahaja na (x,y) koordinati z orientacijo α, ampak bomo z neko verjetnostjo npr. 0,95 verjeli, da se robot nahaja v bližnji okolici točke (x,y). Ta okolica pa bo toliko velika, kolikor natančne bodo naše metode za določanje lege robota.

Če želimo povečati verjetnost, s katero robot določa svojo lego, moramo poiskati način kako posodabljati lego robota tudi s pomočjo vizualnega zaznavanja njegove okolice. Idejo dobimo iz navigacije, ki jo je uporabljal človek pred uvedbo GPS sistema. Podobno kot so ljudje v preteklosti uporabljali zvezde za navigacijo, lahko robot v svojem okolju poišče neke statične predmete, ki so dovolj raznoliki med seboj in izstopajo iz okolja, da jih lahko z lahkoto razpoznamo. Če recimo zaznamo 3 takšne točke, lahko izmerimo oddaljenost robota od vsake od njih ter nato na podlagi triangulacije izračunamo položaj robota (podobno deluje GPS). Na podoben način deluje Kalmanov Filter (krajše KF) [3, 25, 27, 28].

KF deluje v dveh korakih, računanje približka lege ter popravljanje približka glede na izmerjene vrednosti. V koraku računanja približka lege se izračuna predvidena lega $\hat{x}_k^-$ (Enačba 3.1) in kovariančna matrika napake $\hat{P}_k^-$ (Enačba 3.2). Spremenljivka F_k je model tranzicije s prejšnjo oceno, B_k je kontrolna vrednost, ki jo uporabimo na kontrolnem vektorju u_k, dodati moramo še popravek zaradi napake, kar storimo s kovariančno matriko Q_k.

$$\hat{x}_k^- = F_k\hat{x}_{k-1} + B_k u_k \tag{3.1}$$

$$\hat{P}_k^- = F_k Pk - 1F_k^T + Q_k \tag{3.2}$$

V drugem koraku, moramo izračunati Kalmanov pribitek K_k (Enačba 3.3).

$$K_k = P_k^- H^T (H P_k^- H^T + R)^- 1 \tag{3.3}$$

Pri tem uporabimo meritve oddaljenosti značilnih točk H ter šum R, za katerega predpostavimo, da je porazdeljen normalno $N(0, R_k)$. Za problem iskanja dobre kovariančne matrike za približek šuma se uporablja algoritem ALS (Autocovariance Least-Squares) [26].

Ko imamo izračunan K_k, lahko posodobimo vrednosti $\hat{x}_k$ in $\hat{P}_k$.

$$\hat{x}_k = \hat{x}_k^- + K_k(Hx_k - H\hat{x}_k^-) \tag{3.4}$$

$$\hat{P}_k = (I - K_k H)P_k^- \tag{3.5}$$

Problem algoritma KF je, da so pri večjem številu značilnih točk kovariančne matrike zelo velike in jih ni mogoče dovolj hitro posodabljati. Ta problem rešuje algoritem FastSLAM, saj problem razdeli na več podproblemov, KF pa uporabi le na matrikah velikosti 2×2. [3].

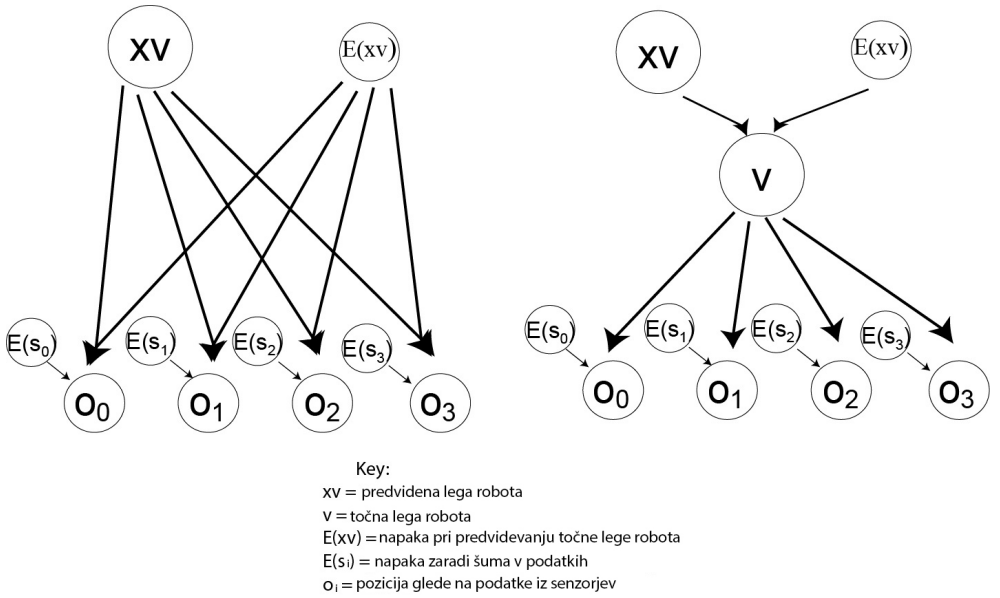
Algoritem FastSLAM

Ker se v ROS-u problem SLAM-a rešuje z algoritmom FastSLAM, bomo le tega natančneje opisali. Spodnje besedilo je povzeto po članku [3].

FastSLAM je bil razvit z namenom, da odpravi pomanjkljivosti KF. FastSLAM porazdeli problem lokalizacije in gradnje zemljevida na veliko manjših

podproblemov z uporabo pogojne neodvisnosti modela SLAM. Algoritem FastSLAM lahko torej obdela veliko več podatkov na takšen način, da se osredotoči le na najpomembnejše dele.

FastSLAM predstavi problem SLAM-a s pomočjo Bayesove verjetnosti. Pri takšnem modelu imamo slučajne spremenljivke, ki imajo določene vrednosti pri nekih verjetnostih, te pa so odvisne od drugih spremenljivk. Takšne povezave tvorijo t.i. *Bayesove mreže (ang. Bayesian Networks)* [22]. Prednosti takšnih neodvisnosti se lahko izrabljajo za pohitritev časa celotnega algoritma, saj lahko veliko spremenljivk na podlagi odvisnosti ne upoštevamo.



Slika 3.1: Graf na levi predstavlja Bayesovo mrežo, kjer robot svojo lego oceni na podlagi vhodnih podatkov z napakami. Na desni lahko vidimo, kakšno odvisnost bi imeli, če bi poznali točno lego robota. [3]

Iz Bayesovega grafa na levem delu Slike 3.1 vidimo, da nobeno vozlišče ni neodvisno med seboj. Tudi če naredimo približek lege xv_i, da bi jih s tem ločili, so ta še vedno povezana z $E(xv_i)$. Po drugi strani pa lahko na desnem delu slike vidimo, da postanejo vozlišča med seboj neodvisna, če jih

verjetnostno ločimo s tem, ko podamo točno robotovo lego V.

Če to uporabimo na primeru iz KF, ko zaznavamo značilno točko O_i, bodo vse značilne točke $O_j; (j = 0...n, j \neq i)$ odvisne od lege xv, napake $E(xv)$, ki jo naredimo, ko privzamemo to lege, ter napake $E(s_i)$, ki je storjena pri senzorju, ko ta zaznava objekt. Če bi v tem primeru točno poznali lego robota V, bi bile lege objektov odvisne le še od napake senzorja. Od tu lahko sklepamo, da se ne splača spremljati razmerij med objekti, če lahko naredimo dovolj dobre približke lege V, tako da ta verjetnostno loči vse objekte drug od drugega. Ta približek V je točna lega robota.

Če si lego robota predstavljamo kot enodimenzionalno, potem bo lega porazdeljena po Gaussovi krivulji z verjetnostjo $p(x)$. Analogno bo za večdimenzionalno premikanje robota $p(x)$ predstavljal verjetnost za vsa možna stanja njegove lege, v našem primeru (x,y).

$$p(x|u_0, ..., u_i, z_0, ..., z_i) \tag{3.6}$$

Enačba 3.6 torej predstavlja verjetnost lege robota pri pogoju u_j,z_j, kjer u_j in z_j predstavljata podatke iz odometrije ter senzorjev, ki jih je robot dobil ob kateremkoli času i. Enačbo (3.6) lahko napišemo kot

$$p(x|U_i, Z_i) = p(v, p_0, p_1, ..., p_m|U_i, Z_i).^1 \tag{3.7}$$

Sedaj predpostavimo, da imamo dve neodvisni slučajni spremenljivki A in B. Rečemo lahko, da velja $p(A, B) = p(A) * p(B)$. Če bi kasneje ugotovili, da je spremenljivka A odvisna od spremenljivke B, bi začelo veljati

$$p(A, B) = P(A) * p(B|A). \tag{3.8}$$

Ker vemo, da so značilne točke odvisne od lege robota, lahko enačbo (3.7) zapišemo kot:

$$p(v, p_0, p_1, \ldots, p_m|U_i, Z_i) = p(v|U_i, Z_i) * p(p_0, p_1, \ldots, p_m|U_i, Z_i, v). \tag{3.9}$$

[1] Zaradi kompaktnosti zapisa smo vpeljali oznako $U_i = u_0, u_1, ..., u_n$ in $Z_i = z_0, z_1, ..., z_n$

Ker pa so značilne točke med seboj pogojno neodvisne, če poznamo lego robota, katero pa seveda poznamo, oziroma poznamo vsaj njen dovolj dober približek, lahko enačbo (3.9) razbijemo v posamezne verjetnostne račune:

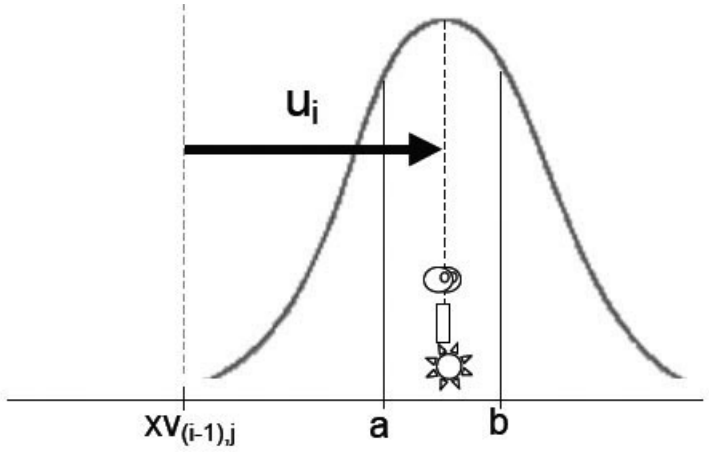
$$\begin{aligned} p(v|U_i, Z_i) * p(p_0, p_1, ..., p_m|U_i, Z_i, v) = \\ = p(v|U_i, Z_i) * p(p_0|U_i, Z_i, v) * ... * p(p_m|U_i, Z_i, v) = \\ = p(x|U_i, Z_i) * \prod_m p(p_m|U_i, Z_i, v). \quad (3.10) \end{aligned}$$

S tem smo razbili problem algoritma SLAM v $m + 1$ ločenih podproblemov, od katerih so vsi med seboj neodvisni. S tem si bomo lahko pomagali pri pohitritvi algoritma iz KF, seveda pa moramo sedaj najti način, kako hitro rešiti še vseh $m + 1$ podproblemov. Edina slabost zgornje faktorizacije je možno poslabšanje natančnosti zaradi ignoriranja korelacije značilnih točk. Če bi poznali n (kjer je n končno naravno število) približkov točne lege robota, bi lahko za vsako tako točko hranili svoj zemljevid ter ga nato primerno obtežili. S tem bi nadomestili prej omenjeno slabost ter dobili optimalno rešitev.

FastSLAM hrani veliko mogočih poti robota istočasno. Zaradi lažje predstave so avtorji v [3] malce priredili FastSLAM tako, da bomo posodabljali približke leg robota namesto celotno pot, ki jo je že obdelal. Ker imamo sedaj opravka z različnimi legami ob različnih časih, bo xv_{ij} predstavljalo lego j ob času i. Torej je $xv_i = x_{i0}, x_{i1}, ..x_{iM}$, vsak približek x_{ij} pa bomo poimenovali delec.

Delci xv_i bodo posodobljeni v skladu z verjetnostjo, to pomeni, da bomo k posodabljanju vključili nekaj naključnosti z upoštevanjem verjetnosti delca. To pa zato da ne bodo vsi delci homogeni ter da bo več dobrih približkov preživelo. Recimo, da imamo zbirko približkov xv_{i-1}, ter kontrolni vhod u_i. Zbirka približkov se ne bo posodobila, dokler ne naredimo nekaj približkov lokacije značilnih točk iz novih opazovanj na robotovi trenutni legi. Zato naredimo začasno zbirko delcev xv_t, ki bo temeljila na u_i in M delcev iz xv_{i-1}, kjer bosta število delcev M in xv_t odvisna od situacije. Z eksperimentalnimi poskusi se izkaže, da manj značilnih točk kot imamo manj delcev potrebu-

jemo. xv_t bo izbran z neko verjetnostjo iz xv_{i-1}. Na primer, da generiramo delec za xv_t, vzamemo delec $xv_{(i-1)j}$, nato uporabimo verjetnost, ki jo dobimo iz $xv_{(i-1),j} + u_i$, da izberemo nov delec za xv_t. To lahko vidimo na Sliki 3.2. Ta proces se ponavlja, dokler nimamo izbranih N delcev xv_t, kjer je



Slika 3.2: Izbiranje delca temp, glede na verjetnost podano z novo lokacijo u_i in $xv_{(i-j),j}$: $p(a < temp < b) = \int_a^b p(xv_{(i-1),j} + u_i \, \mathrm{d}x$. [3]

$N >= M$. S povečevanjem N *in/ali* M se bolje približamo optimalni rešitvi, vendar porabimo več časa, zato ju izbiramo eksperimentalno glede na dano situacijo. Ko imamo xv_t, moramo izbrati najboljše delce iz xv_t, ki jih bomo prenesli v xv_i. Tukaj upoštevamo opazovane značilne točke. Vsakemu delcu xv_t, xv_{tk} določimo normalizirano utež w_{tk}. Utež se določi tako, da se delci, ki podpirajo novo zaznavo obdržijo, tisti, ki ji nasprotujejo, pa zavržejo. To zapišemo:

$$w_{tk} = \frac{p(xv_{tk}|Z_i, U_i)}{p(xv_{tk}|Z_{i-1}, U_i)} \tag{3.11}$$

Sedaj lahko zgradimo xv_i tako, da glede na ustrezno verjetnost vzamemo M delcev iz xv_t. S tem smo izbrali delce, ki so čim bližje temu, kar smo pričakovali glede na prejšnje delce ter kontrolni vnos u_i. Torej približku, ki bi ga izbral KF. Z drugimi besedami xv_i predstavlja oceno $p(v|Z_i, U_i)$, napaka pa konvergira proti 0, ko se M približuje proti neskončno.

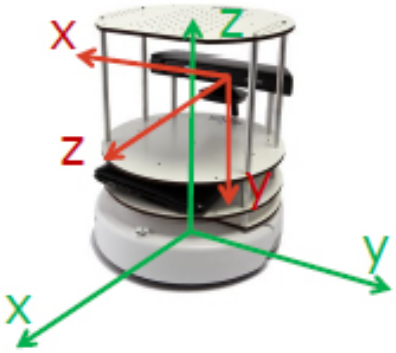
Sedaj nam ostane le še ocenjevanje pozicij značilnih točk. Potrebujemo način, kako hitro oceniti nekorelirane vrednosti značilnih točk. V ta namen uporabimo KF na vsakem delcu za vsako značilno točko, torej skupno $M * m$ filtriranj. Vsak filter bo odvisen od lege delca, moral pa bo oceniti lego značilne točke. Ker smo v 2D prostoru, lahko značilno točko določimo s smerjo (kot) in oddaljenostjo, torej bo kovariančna matrika dimenzij 2×2 (zaradi napake v smeri in oddaljenosti mora vsebovati vse 4 vrednosti).

Prej smo omenili, da je vsaka utež določena glede na nove zaznave. Ko pridemo v nov, še ne viden prostor, nimamo nič informacij o tem, kakšen naj bi ta prostor bili, zato večini uteži pripišemo enako vrednost. Po drugi strani pa lahko v že obiskanem prostoru veliko bolje določimo uteži ter s tem še dodatno zmanjšamo razpršenost delcev. Iz eksperimentalnih poskusov [3] se je izkazalo, da se je bolje večkrat vrniti na že obiskan teren, da se lege posodobijo in se s tem omogoči natančnejše približke v nadaljevanju, kot pa narediti en dolg obhod. Postopku, ko se robot vrne na že zaznan del prostora, rečemo sklenitev kroga (ang. loop closing).

3.3 Transformacije med koordinatnimi sistemi

Sedaj ko poznamo način izgradnje zemljevida, moramo najti način, kako pravilno postaviti zaznane predmete nanj. Ker je robot sestavljen iz več različnih komponent (senzorjev), je potrebno postaviti vse elemente v enoten koordinatni sistem. Na primer, če na robota dodamo kamero Kinect, bosta robot ter kamera vsak v svojem koordinatnem sistemu (Slika 3.3), zato moramo vse koordinatne sisteme prevesti na enoten globalni sistem. Za globalni sistem se največkrat vzame zemljevid, saj je statičen.

Poglejmo si primer pri človeku. Če želimo izvedeti lego prsta na roki moramo poznati rotacijo zapestja, komolca, ramena, položaj trupa ter lego samega človeka. Potrebno je poznati dolžino in lego vsake komponente, če pa se neka komponenta spremeni (človek se predkloni), to vpliva na vse komponente, ki so vezane na telo.



Slika 3.3: Koordinatni sistem Kinecta (rdeč) se razlikuje od koordinatnega sistema Roombe (zelen). [1]

Enako velja za robote. Natančno moramo specificirati povezave ter dimenzije vseh komponent ter jih v pravilnem vrstnem redu povezati med seboj. Ker se takšne transformacije v robotiki pojavljajo na vsakem koraku, je v ROS-u implementiran paket, ki poskrbi za vse zgoraj našteto. [2, 18]

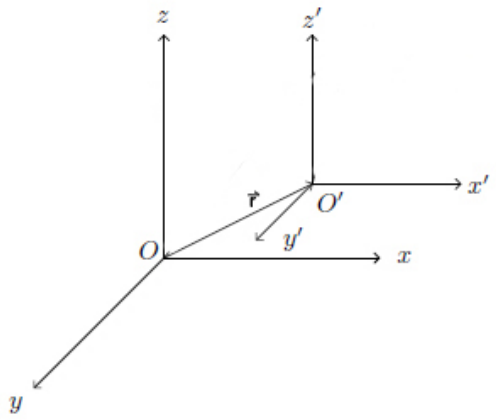
3.3.1 Kako poteka transformacija?

Začnimo z enostavnim primerom transformacije med dvema koordinatnima sistemoma, ki sta en za drugim (Slika 3.4). Trivialno bomo točki $[x, y, z]^T$, ki jo želimo preslikati prišteli razdaljo med obema koordinatnima sistemoma:

$$[x, y, z]^T + [dx, dy, dz]^T = [x', y', z']^T \tag{3.12}$$

Naslednja stopnja je takšna, kjer sta koordinatna sistema v enakem izhodišču, a drugače rotirana (Slika 3.5). Izdelali bomo rotacijsko matriko R_i (kjer indeks predstavlja rotacijo okrog določene osi) ter z njo pomnožili element.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} R_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$



Slika 3.4: Premik koordinatnega sistema za vektor $\vec{r}$. [23]

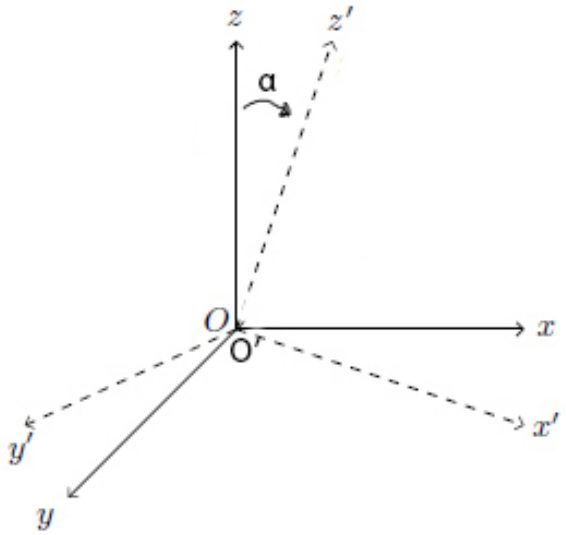
$$R_z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ko sta koordinatna sistema rotirana ter razmaknjena, bomo morali rotirati ter translirati (Slika 3.6). Pozorni moramo biti na vrstni red operacij, ki je pri množenju z matriko pomemben.

$$R_x * [x, y]^T + [dx, dy]^T = [x', y']^T \tag{3.13}$$

V izrazu 3.13 vidimo, da smo najprej množili nato seštevali. Ker želimo, da so naše operacije homogene, bomo morali matriko predelati tako, da bo vključevala rotacijo ter transformacijo. To storimo tako, da matriki dodamo novo dimenzijo, ki bo služila za seštevanje.

$$H = \begin{bmatrix} r & r & r & d \\ r & r & r & d \\ r & r & r & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Slika 3.5: Rotacija koordinatnega sistema za kot α. [23]

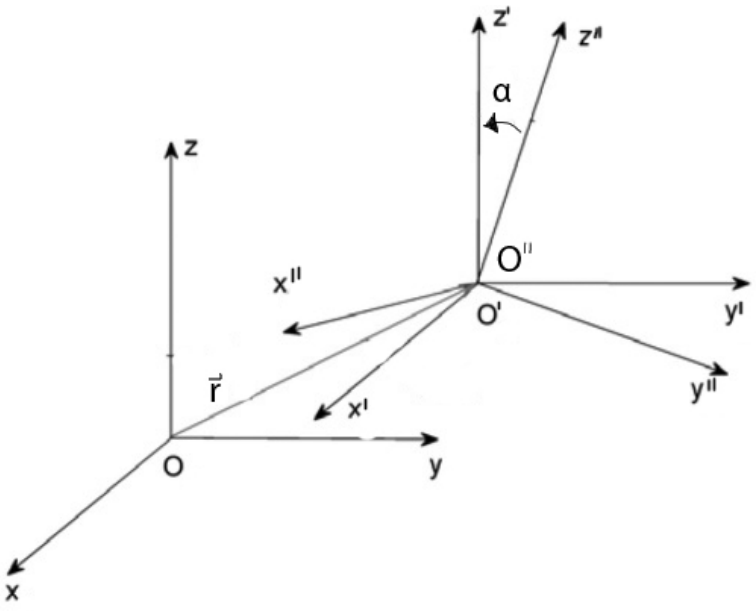
Kot vidimo, je zgornja matrika R razdeljena na rotacijski del (polja z oznako r) ter translacijski del (polja z oznako d).

Oglejmo si preprost primer. Uvesti želimo dvojno rotacijo za 90° okrog osi z in y ter translacija za $[1, 2, 3]^T$. Naredimo rotacijsko matriko:

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

Dodamo še translacijo:

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Slika 3.6: Rotacija koordinatnega sistema za kot α nato pa premik rotiranega sistema za vektor $\vec{r}$ [24]

Sedaj ko imamo transformacijsko matriko zgrajeno, lahko vse transformacije opravimo le z enim množenjem. Primer transformacija točke $[4, 5, 6]^T$:

$$X' = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 6 \\ 8 \\ 1 \end{bmatrix}$$

3.3.2 Časovni zamik

Robot in njegove komponente se veliko premikajo, hkrati pa imamo opravka z ogromnimi količinami podatkov, ki jih moramo obdelati. Teh se velikokrat ne da dovolj hitro obdelati in ko zahtevamo transformacijo neke točke, je robot že zamenjal lego, transformacija pa bi se zgodila glede na trenutni položaj. ROS ta problem rešuje tako, da hrani vse pretekle transformacije. Ko torej zahtevamo transformacijo, moramo poleg podati še čas. Čas se

podaja v milisekundah od trenutnega časa v preteklost. Torej si moramo pred začetkom operacije shraniti čas, ga po končanem obdelovanju podatkov odšteti od trenutnega, nato pa zahtevati transformacijo z razliko, ki smo jo ravnokar izračunali.

3.4 Nadgradnja načrtovanja poti

Zgoraj smo opisali sistem, ki je že implementiran v ROS. Za naš problem enostavno načrtovanje poti od točke do točke ni uporabno. Potrebujemo nov *globalni načrtovalec*, ki robota ne bo peljal do končnega cilja po najkrajši poti, temveč tako da bo robot videl čim več prostora. V nadaljevanju bomo opisali našo rešitev nadgradnje navigacijskega sistema ROS, da bo primeren za preiskovanje prostora.

3.4.1 Ideja

Če želimo, da bo naše preiskovanje prostora kar se da optimalno, bomo morali najprej prepoznatii obliko prostora iz obstoječega zemljevida. Ta problem je po težavnosti enako težek kot detekcija likov na sliki. Pravzaprav je detekcija likov na sliki točno to, kar mi potrebujemo. Iz zemljevida, ki je predstavljen kot slika s tremi barvami, kjer -1: pomeni še nevideno področje, 0: pomeni območje brez ovire, 100: pomeni oviro, moramo detektirat, kar se da velik lik, ki bo predstavljal naš konveksen prostor. Ker je večina prostorov pravokotne oblike, bomo naš detektor specializirali za takšne oblike prostorov.

Ko gradimo zemljevid, seveda ne moramo vnaprej vedeti kakšne velikosti bo, zato vedno začnemo z veliko površino, ki jo označimo kot nepoznano, znotraj nje pa začnemo barvati prosto pot ter ovire. Zaradi tega je najprej potrebno, da se določi prostor, v katerem se nahaja robot. Sedaj, ko vemo da se nahajamo v prostoru, ki ga želimo preiskati, moramo ugotoviti njegovo obliko ter dimenzije. V resničnem svetu je matematično konveksnih prostorov zelo malo, zato bomo morali poiskati čim boljši približek le temu v danem prostoru. Ker vemo, da je večina stavb zgrajenih tako, da so pro-

stori v njih pravokotne oblike, bomo naš približek naredili s pravokotnikom. To bomo storili tako, da bomo najprej včrtali eno stranico, nato pa glede na ovire in podano stranico poskušali narediti čim večji pravokotnik. To nam seveda ne zagotavlja najboljšega približka (to je pravokotnik z največjo ploščino, ki ga lahko včrtamo v prostor), zato bomo vsako stranico poskušali povečati/zmanjšati ter pri tem opazovali, kaj se dogaja s ploščino lika. Na ta način bomo hitro poiskali optimalen približek našega prostora. Pri tem približku pa bo potrebno upoštevati tudi dimenzije robota, zato bomo vse robove pomaknili za $0,25$ metra v notranjost in s tem preprečili postavitev cilja preblizu ovire.

3.4.2 Implementacija

Za izračun lege robota smo uporabili transformacijo koordinatnega sistema zemljevida v sistem robota, pri tem pa prebrali koordinate, ki pripadajo robotu. Lego torej dobimo v metrih oddaljenosti od središča zemljevida. Sam zemljevid dobimo s poslušanjem na *temi zemljevid (ang. map)*. Zemljevid, ki ga iz *teme* preberemo, je lociran v svojem koordinatnem sistemu, ki je transliran glede na koordinatni sistem statičnega zemljevida. Sami podatki so shranjeni v enodimenzionalnem polju, pri katerem vsaka celica predstavlja 5 centimetrov na zemljevidu ter ima eno izmed treh vrednosti: -1,0,100. Poleg teh podatkov dobimo tudi podatke o zamiku zemljevida iz središča koordinatnega sistema ter dimenzije zemljevida. Polje, v katerem se nahaja robot, torej dobimo na naslednji način:

$$i = (\frac{pRobotY}{0.05} - odmikY) * width + (\frac{pRobotX}{0.05} - odmikX) \qquad (3.14)$$

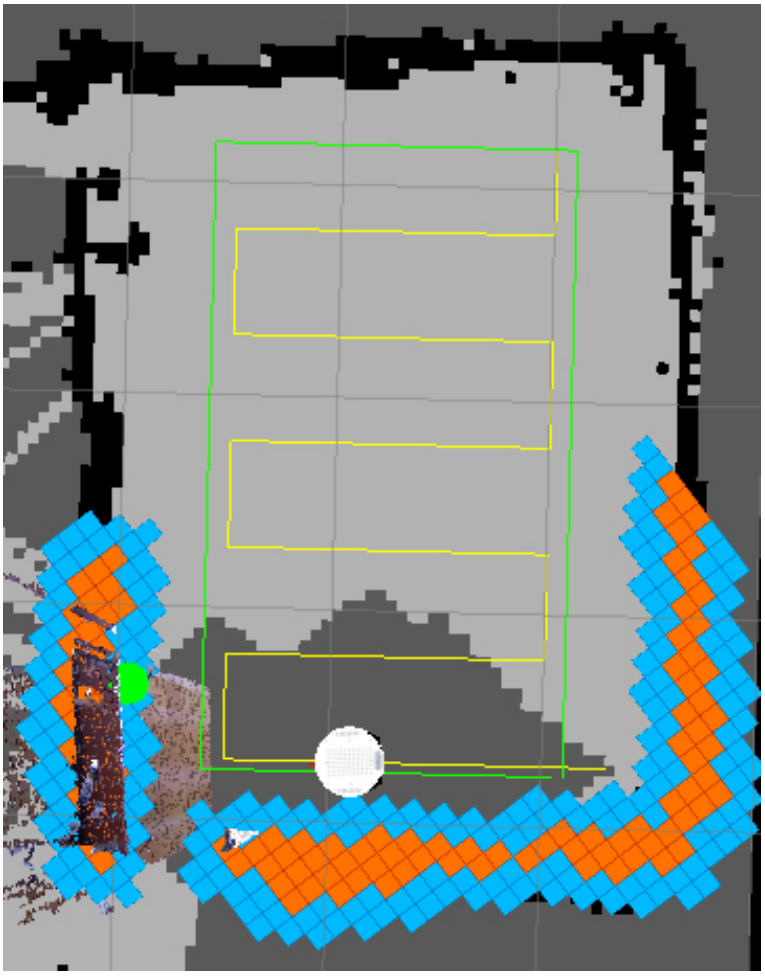
Kjer $pRobotY$ predstavlja oddaljenost robota od koordinatnega središča, $odmikY$ premik zemljevida iz koordinatnega središča, $width$ pa širino zemljevida. Ko ugotovimo polje, v katerem se nahaja robot, se moramo sprehoditi po vseh praznih poljih, ki predstavljajo ordinatno os, da v njih začrtamo, kar se da dolg vektor, ki bo predstavljal stranico našega pravokotnika. Nato se za vsako polje vektorja sprehodimo po vseh točkah vzporednih ordinatni osi ter na ta

način ugotovimo najvišjo višino lika, ki ga lahko včrtamo. S tem smo dobili pravokotnik, ki predstavlja naš prostor. Takšen pravokotnik ni optimalen približek, zato moramo zgoraj opisani postopek večkrat ponoviti in v vsaki iteraciji začetni vektor po abscisni osi zmanjšati za eno polje (5 centimetrov) (Slika 3.7).

Ko imamo izdelan konveksni približek prostora, lahko začnemo z načrtovanjem poti. Najbolj optimalno preiskovanje se zdi spiralno navznoter v prostor, vendar je detekcija predmetov med rotacijo robota zelo slaba, saj pride do zameglitve slike. Potrebno je poiskati *načrt* s čim manj zavoji, to pa bo sledenje stranicam pravokotnika po vzorcu cik-cak navzgor (Slika 3.7). Razdaljo med dvema vodoravnima črtama bomo dobili z eksperimentalnimi podatki glede na zmožnosti detektorja.

V Poglavju 3.1 smo ugotovili, da ROS vsebuje dve vrsti *načrtovalcev poti*, a žal noben od njiju ni primeren za to, kar mi potrebujemo. Imamo že izdelan globalni *načrt*, ki ga želimo podati *lokalnemu načrtovalcu*, ki naj nato skrbi za izogibanje oviram na poti. Ker so *načrtovalci poti* del paketa *move base*, *lokalnemu načrtovalcu* ne moremo enostavno poslati naše poti preko *teme*. Za takšne primere ROS podpira sistem vtičnikov. Vtičniki so paketi, ki ne vsebujejo metode main. Torej se ne morejo izvajati sami od sebe, ampak jih kličejo ter z njimi upravljajo drugi deli programa. Vtičnike je potrebno zgraditi ter registrirati v ROS-ovi knjižici *pluginlib*. Naš vtičnik bomo zgradili tako, da bo deloval kot globalni *načrtovalec poti*. ROS od vsakega *globalnega načrtovalca* poti zahteva, da vsebuje vsaj naslednji dve metodi: inicializacijsko ter tisto, ki vrne globalni *načrt*. ROS bo nato ti dve metodi klical sam v svoji niti, ko bo potreboval podatke (*načrt* poti).

Z namenom, da se izognemo morebitnim problemom po dodajanju novih vtičnikov za bodoče *načrtovalce poti*, smo se odločili, da bomo *globalni načrtovalec* naredili tako, da bo poslušal na določeni *temi* za *načrt*. Tisti del sistema, ki bo dejansko načrtoval pot, pa naredimo kot navaden ROS paket.



Slika 3.7: Zelen pravokotnik je približek prostora, pri katerem je že upoštevan odmik 0,5 metra od sten prostora. Rumena črta pa predstavlja začrtano pot, tako da bomo preiskali cel prostor. Beli krog je robot. V smeri orientacije robota je barva zemljevida temnejša, to predstavlja že videno področje.

Poglavje 4

Prepoznavanje predmetov

V Poglavju 3 smo opisali algoritme, ki našemu robotu omogočajo mobilnost ter navigacijo v prostoru. Sedaj moramo poiskati način, kako prepoznati predmete. Kot smo omenili, bomo za to nalogo uporabili RoboEarth, ki uporablja algoritem SURF (Speeded Up Robust Features) in algoritem najbližjih sosedov za dodatno klasifikacijo na podlagi 3D informacije o točkah. V naslednjih Poglavjih je natančneje opisan algoritem SURF [9]. Na koncu pa naša nadgradnja sistema, da bomo lahko uporabili podatke, ki jih dobimo, ko prepoznamo predmet.

4.1 Uvod

Veliko algoritmov za računalniški vid deluje v treh korakih. Najprej na sliki poiščemo zanimive točke. To so tiste točke, ki imajo neko značilnost, ki jo je enostavno določiti, npr. robovi, koti, pike, oglišča ipd. Če je zanimiva točka dobro izbrana, se bo pojavila na enakem mestu (na predmetu) ne glede na velikost in rotacijo predmeta. Za iskanje zanimivih točk je najbolj poznan Hessov ter Harrisov detektor. Oba detektorja temeljita na odvodu slike.

Naslednja stopnja je opisovanje zanimivih točk. Kako naj opišemo zanimivo točko, da jo bomo znali primerjati z zanimivo točko v naslednji sliki? Opisnik mora točke dovolj natančno opisati, tako da je ne zamenjamo s so-

sednjo, hkrati pa mora biti dovolj robusten na šum, da enaki točki poveže med seboj.

Tretji korak algoritmov je povezovanje zanimivih točk. Ujemanje deluje na podlagi razdalje med opisom točk. Največkrat se uporablja kar Evklidska razdalja. Ujemanje zanimivih točk predstavlja veliko časovno breme za algoritem, saj moramo primerjati vsako točko z vsako. Tako moramo velikokrat izbirati med boljšimi večdimenzionalnimi opisniki, ki bodo porabili več časa pri ujemanju, ter manj natančnimi, a hitrejšimi.

4.2 Iskanje zanimivih točk

Iskanje zanimivih točk pri algoritmu SURF temelji na približku Hessove matrike. To nam omogoča uporabo integralne slike, kar nam pomaga zmanjšati časovno zahtevnost. Uporaba integralne slike nam omogoča hitro računanje konvolucijskih filtrov. Vhod integralne slike $I_{\sum}(x)$ na lokaciji $x = [x, y]^T$ predstavlja vsoto vseh slikovnih elementov v vhodni sliki I znotraj kvadratne regije okoli slikovnega elementa x (glej enačbo 4.1). Ker seštejemo vrednosti vseh slikovnih elementov podobno kot matematični integral, temu rečemo integralna slika.

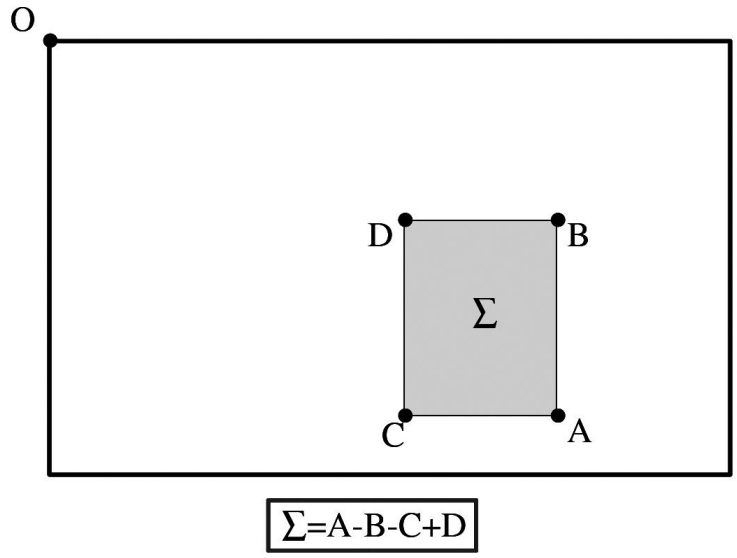
$$I_{\sum}(x) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j) \tag{4.1}$$

Ko je integralna slika izračunana, potrebujemo le tri dodatna seštevanja (Slika 4.1), da seštejemo intenziteto po kateremkoli kvadratnem območju. Torej je čas računanja neodvisen od velikosti slike in filtra.

Detektor, podobno kot Hessov, zaznava točkaste strukture, kjer je odvod maksimalen. Za dano točko $x = (x, y)$ v sliki I, je Hessova matrika $H(x, \sigma)$ v točki x za skalo σ definirana:

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix}$$

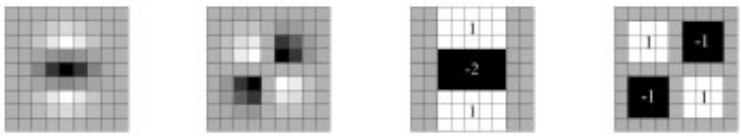
kjer $L_{xx}(x, \sigma)$ predstavlja konvolucijo Gaussovega drugega odvoda $\frac{\sigma^2}{\sigma x^2} g(\sigma)$ s sliko I v točki x, podobno velja za $L_{xy}(x, \sigma)$ in $L_{yy}(x, \sigma)$. Približek drugih



Slika 4.1: Za izračun intenzitete znotraj kateregakoli kvadratnega območja potrebujemo samo tri seštevanja. Oglišče predstavlja integralno sliko (Enačba 4.1). [9]

odvodov je zaradi priprav, ki smo jih omenili zgoraj, zelo hiter in neodvisen od velikosti filtra.

Na Sliki 4.2 je filter velikosti 9×9 s sigmo $\sigma = 1,2$ in predstavlja najmanjšo velikost za izračun pik. Označimo ga z D_{xx}, D_{yy} in D_{xy}.



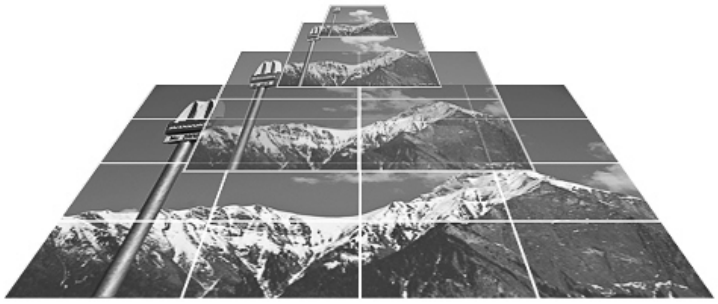
Slika 4.2: Od leve proti desni: Gaussov odvod drugega reda L_{yy} in L_{xy} ter približek D_{yy} in D_{xy} [9]

$$det(H_{approx}) = D_{xx}D_{yy} - (wD_{xy})^2 \tag{4.2}$$

Utež w, uporabljena v enačbi 4.2, je dodana z namenom boljšega približka

determinanti Hessove matrike. Po končanem filtriranju moramo odziv filtra normalizirati, saj moramo filtriranje opraviti pri različnih povečavah filtra.

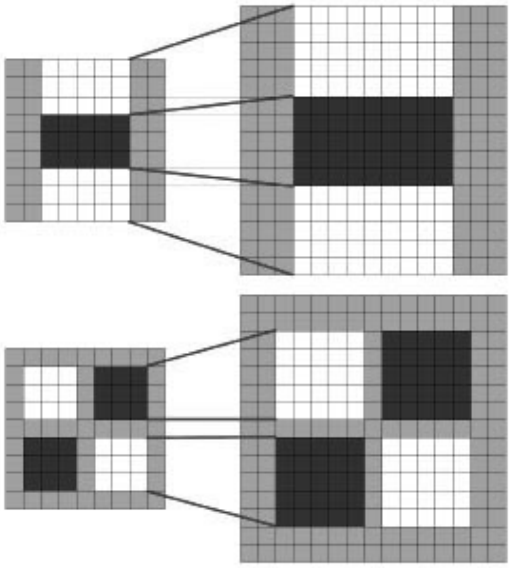
Zanimive točke moramo torej najti pri različnih velikostih slike, saj jih bomo velikokrat primerjali s sliko, ki ni v enakem merilu (robot se približuje predmetu, predmet je na sliki videti vedno večji). Različna merila slik navadno predstavljamo s piramido iz slik (Slika 4.3). Pri vsaki naslednji stopnji sliko zgladimo z Gaussovim filtrom ter jo nato pomanjšamo (vzamemo npr. 4 sosednje slikovne elemente ter izračunamo povprečje, to povprečje bo nov slikovni element na pomanjšani sliki). Ker uporabljamo kvadratne filtre in integralne slike, lahko namesto zgornjega opisanega postopka povečamo velikost filtrov na originalni sliki. S tem prihranimo na času ter se izognemo popačenju slike (ang. aliasing) pri pomanjšavi.



Slika 4.3: Piramida iz slik. Čeprav porabi več prostora kot navadna slika, pri algoritmu pridobimo na času, saj nam ni potrebno vsakič ponovno pomanjšati slike na želeno velikost. [10]

Različna merila so razdeljene na oktave. Oktava predstavlja polje odzivov ustrezno povečanega filtra. Vsaka oktava predstavlja dvakratno povečavo velikosti filtra in je razdeljena na konstantno število celic z določenim merilom. Najmanjša možna povečava je navzdol zaokrožena tretjina drugega odvoda v smeri odvajanja. Za filter velikosti 9×9 je to torej 3. Da bi lahko imeli dve zaporedni stopnji, moramo povečati masko filtra za 3 slikovne elemente na

vsaki strani (Slika 4.4). Konstrukcija oktave se začne s filtrom velikosti 9x9

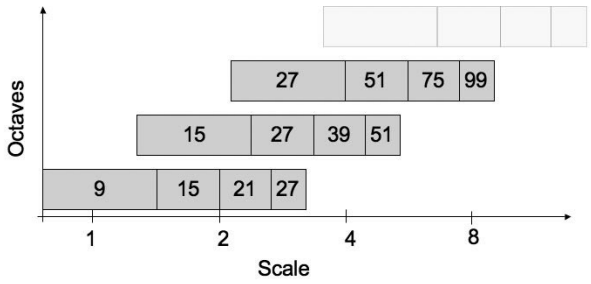


Slika 4.4: Slika prikazuje D_{yy} (zgoraj) in D_{xy} (spodaj) dve zaporedni stopnji povečave 9×9 in 15×15. Temni del slike lahko dodatno povečamo le za liho število ter s tem ohranimo 1 centralni slikovni element. [9]

(Slika 4.5) in nato nadaljuje s 15×15, 21×21 in 27×27. S tem dosežemo celo več kot 2-kratno povečavo, ta presežek pa se izniči z zatiranjem lokalnih ne-maksimumov. Podobno velja tudi za ostale oktave, kjer se velikost filtra podvoji. Tako lahko enostavno podvojimo število dodatnih slikovnih elementov, ki jih moramo dodati, in s tem zmanjšamo časovno zahtevnost algoritma.

4.3 Opisovanje zanimivih točk

Z namenom, da opisnik ne bi bil občutljiv na rotacijske spremembe, poiščemo usmerjenost zanimive točke. Za to potrebujemo izračunati Haarov val po x in y smeri, znotraj krožnega območja z radijem $6s$ okrog značilne točke, kjer je s velikost povečave, s katero je bila značilna točka zaznana. Potrebno



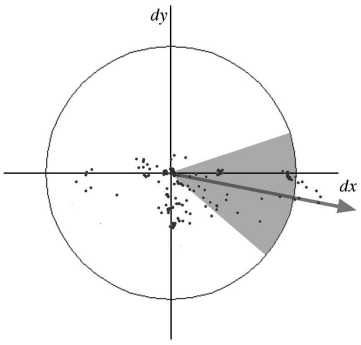
Slika 4.5: Dolžina filtrov za prve tri oktave. [9]

je pravilno raztegniti tudi Harrov val in sicer na dolžino $4s$. Uporabljamo lahko torej celotno sliko za hitro filtriranje. Uporabljeni filtri so prikazani na Sliki 4.6. Za izračun odziva je potrebnih le 6 operacij ne glede na smer ter povečavo.

Slika 4.6: Haarov valjast filter za filtriranje po x (levo) ter y (desno). Temna stran ima utež -1 svetla pa +1 [9]

Ko izračunamo in obtežimo odzive filtra z Gaussovo utežjo $\sigma = 2s$, ki je centrirana v zanimivi točki, predstavimo odzive kot točke v prostoru s horizontalnim odzivom vzdolž abscisne osi, vertikalnim pa vzdolž ordinatne. Prevladujoča usmerjenost se oceni z drsečim oknom velikosti $\frac{\pi}{3}$ (Slika 4.7). Velikost okna so avtorji v [9] določili eksperimentalno. Horizontalni in vertikalni odzivi znotraj okna se seštejejo. Oba sešteta odziva nato podajata lokalni usmerjeni vektor. Najdaljši tak vektor, znotraj celotnega območja, poda prevladujočo usmerjenost.

Za pridobivanje opisnika moramo zajeti kvadratno območje okrog zanimive točke ter ga orientirati glede na rotacijo iz prejšnje točke (Slika 4.8).

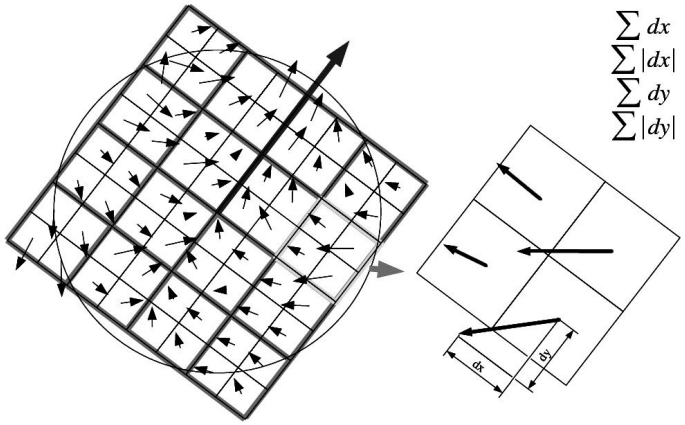


Slika 4.7: Drseče okno velikost $\frac{\pi}{3}$, ter odzivi na filtre. [9]

Slika 4.8: Prikaz obrobe zanimivih točk glede na velikost in rotacijo. [9]

Velikost okna je $20s$. Regija je nato rekurzivno razdeljena na 4 manjše kvadrate. Za vsako regijo izračunamo odziv Haarovega vala na 5×5 enakomerno porazdeljenih vzorčnih točkah (Slika 4.9). Odzivi so obteženi s σ=3,3s ter centrirani v zanimivo točko. Polariteto odziva dobimo iz absolutnih vsot. Vsaka podregija ima štiridimenzionalen vektor $\vec{v}$ (Enačba 4.3). Ko seštejemo vse podregije, dobimo opisni vektor dolžine 64.

$$v = (\sum dx, \sum dy, \sum |dx|, \sum |dy|) \tag{4.3}$$



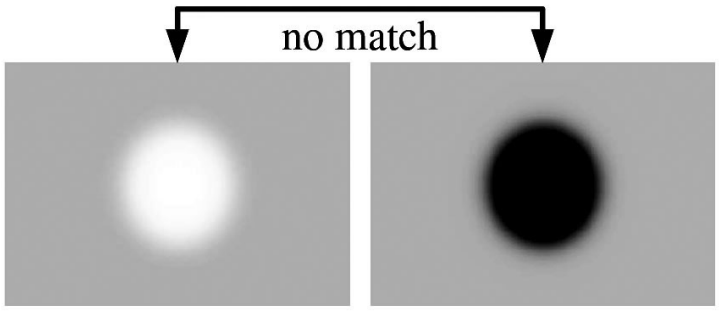
Slika 4.9: Da zgradimo opisnik, uporabimo pravilno rotiran kvadrat, ki je razdeljen na $4x4$ podregije, na zanimivi točki. Za vsak kvadratek se izmerijo odzivi Haarovega filtra. Območje velikosti 2×2 predstavlja dejanska polja opisnika. Ta se nato seštejejo v dx, $|dx|$, dy in $|dy|$. [9]

4.4 Ujemanje zanimivih točk

Za hitro indeksiranje in ujemanje točk uporabimo predznak sledi Hessove matrike. Tipično so zanimive točke najdene okrog pik na sliki. Predznak sledi razlikuje med svetlo piko na temnem ozadju in temno piko na svetlem ozadju, kar nam omogoča dodatno ločitev primerov (Slika 4.10). Pri ujemanju moramo med seboj primerjati samo tiste odzive, ki imajo enak predznak.

4.5 Prepoznavanje predmetov

V tem Poglavju smo natančno opisali potek algoritma SURF, ki je glavni algoritem za prepoznavanje predmetov pri *paketu* RoboEarth. Iz opisa algoritma je razvidno, da deluje na podlagi 2D slike. Ker uporabljamo Kinect, imamo poleg 2D slike tudi podatek o globini vsakega slikovnega elementa v sliki. Smiselno bi bilo to dodatno informacijo uporabiti pri razpoznavanju predmetov. To so avtorji algoritma RoboEarth [11] tudi storili. Najprej sliko razpoznajo na podlagi 2D informacije, nato pa klasifikacijo potrdijo



Slika 4.10: Sliki bosta imeli drugačen predznak sledi Hessove matrike, zato nam ju ni potrebno primerjati. [9]

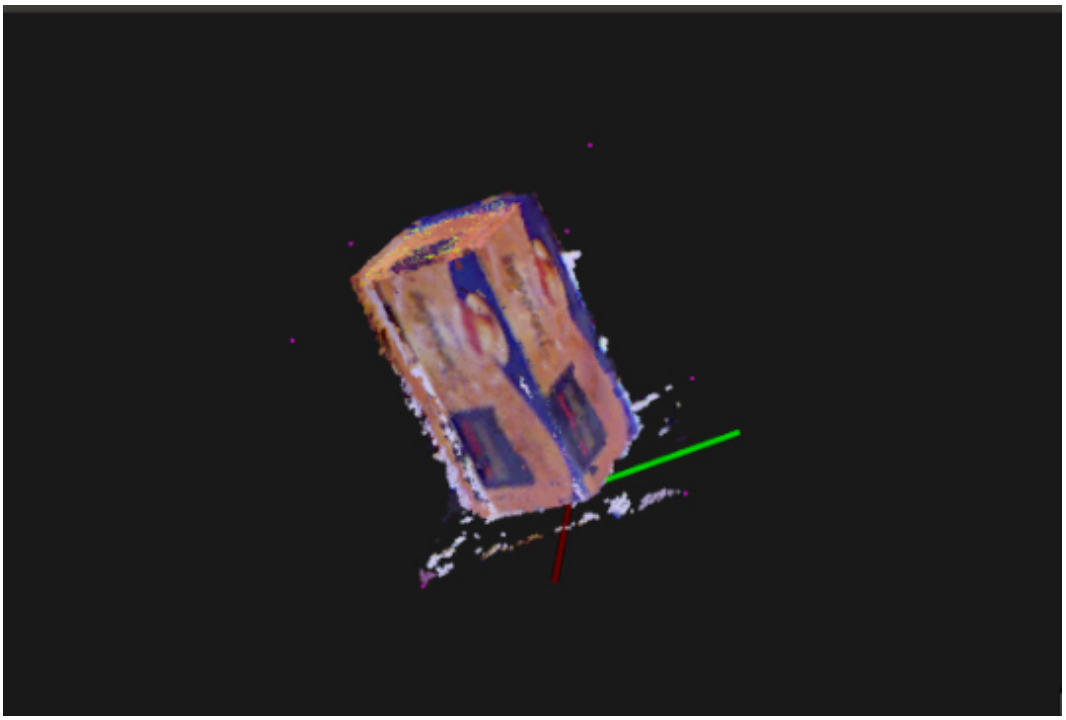
ali ovržejo na podlagi rezultatov algoritma *najbližjih sosedov (ang. nearest neighbour)*.

Mi bomo za detekcijo predmetov uporabili RoboEarth. Algoritem je v celoti izdelan, mi bomo morali le implementirati način, kako prikazati ter uporabiti podatke, ki nam jih bo program podal. V tem Poglavju bomo opisali način uporabe algoritma od izgradnje modela do trenutka, ko nam robot sporoči, da je našel predmet.

4.5.1 Ideja

Da bo naše prepoznavanje delovalo dobro ne glede na rotacijo ter oddaljenost predmeta od robota, je potrebno zgraditi celoten 3D model predmeta (ne le ene stranice). K sreči je v RoboEarth, poleg sistema za učenje predmetov, priložena tudi kombinacija oznak, ki jih natisnemo na list papirja ter položimo pod predmet. Na podlagi oznak lahko program nato natančno izračuna lego predmeta, zaradi česar lahko predmet rotiramo (skupaj z oznakami) in s tem dobimo celoten model. Sam model je shranjen kot množica 3D točk za vsako fotografijo, ki jo je kamera naredila ter korelacijo med njimi. Tako lahko iz njih kasneje sestavimo 3D model [12]. Primer modela za škatlico čaja je prikazan na Sliki 4.11.

Prepoznavanje predmetov deluje v dveh korakih. Najprej z algoritmom



Slika 4.11: Zgrajen model škatlice za čaj. Opazimo lahko, da ima model nekaj šuma.

SURF (Poglavje 4) prepoznamo predmet na podlagi 2D slike iz videosistema Kinect. Da algoritem deluje hitreje, je prag, pri katerem se predmet šteje za zaznanega, nastavljen nekoliko ohlapnejše. V drugem koraku algoritem primerja oblak točk iz Kinecta z oblakom točk, ki predstavljajo naučen predmet. Z metodo najbližjih sosedov nato določi kateremu predmetu pripada. Predmet je prepoznan, če oba koraka najdeta enak predmet.

4.5.2 Implementacija

RoboEarth je zgrajen kot *paket*, zato ga z lahkoto uporabimo v našem programu. Najprej mu moramo na določeno *temo* poslati podatke iz videosistema Kinect, nato pa poslušamo na izhodni *temi*, kjer nam sporoči najdene predmete. Najdeni predmeti so predstavljeni z ROS sporočili, ki vsebujejo

ime ter oblak točk samega predmeta.

Ker želimo predmete postaviti na zemljevid, jim moramo določiti njihovo lego. V prejetem sporočilu se sprehodimo po vseh točkah v oblaku ter izračunamo povprečni x in y koordinati predmeta, ki sta v koordinatnem sistemu kamere Kinect. Seveda to ni točno središče predmeta, a je zaradi napake pri detekciji, odometriji in vsem, kar sodi zraven, dovolj točno. Koordinate je potrebno transformirati v koordinatni sistem zemljevida. Zavedati se moramo, da je zaznavanje predmetov zelo zamudna operacija, zato bo robot velikokrat že nekaj metrov stran od točke, kjer je bila slika zajeta. Pomembno je, da si pred vsemi operacijami shranimo trenutni čas, nato pa ob vsakem klicu transformacije zahtevamo transformacijo iz preteklosti.

Zaznan predmet nato dodamo v tabelo skupaj z njegovimi koordinatami. Tako lahko na podlagi oddaljenosti predmetov med seboj določimo, ali gre za dva enaka predmeta ali za že videnega, ki se je zaradi napak pri zaznavanju ter pozicioniranju robota premaknil. Če gre za različna predmeta, ju moramo vizualizirati, kot je prikazano na Sliki 4.12. Potek vizualizacije razmišljanja robota je opisan v naslednjem Poglavju 5.1.

Slika 4.12: Ko detektor zazna predmet, ga v sliki pobarva z rdečo barvo. V *Rvizu* se na zemljevid postavi zelen stožec, na lokaciji, ki smo jo izračunali iz povprečnih x in y koordinat.

Poglavje 5

Delovanje sistema in eksperimentalni rezultati

V Poglavjih 3 in 4 smo poskrbeli, da robot preišče prostor ter razpozna predmete. Da bo robotovo početje razumljivo in si bomo lažje predstavljali vzrok za neko akcijo, moramo poskrbeti za vizualizacijo robotovega razmišljanja.

5.1 Vizualizacija delovanja sistema

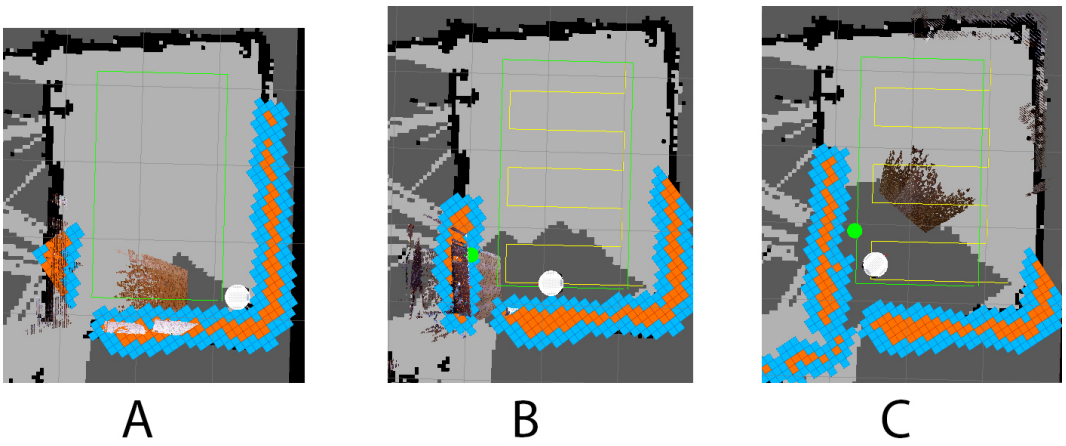
Z nekaj izpisi na konzolo o koordinatah določenega predmeta ter ostalih akcijah robota si ne moramo veliko predstavljati, zato je potrebno poiskati način, kako podatke ter delovanje robota predstaviti na intuitiven način. ROS ima *paket*, imenovan *Rviz*, ki skrbi za vizualizacijo nekaterih komponent robota.

Rviz nam izriše zemljevid ter robota na njem. Prikaže nam podatke iz videokamere ter 3D točke, ki jih zazna Kinect. Omogoča dodajanje svojih oblik in oznak. Kot vsi *paketi* v sistemu ROS tudi *Rviz* pridobi podatke preko *tem*, zato mu bomo podatke enostavno pošiljali iz naših programov preko *pošiljalcev*.

Za predstavitev načrtovanja poti se poslužimo risanja vektorjev. Vsako celico v zemljevidu, kjer bo potekala pot, dodamo v polje, ki ga nato pošljemo

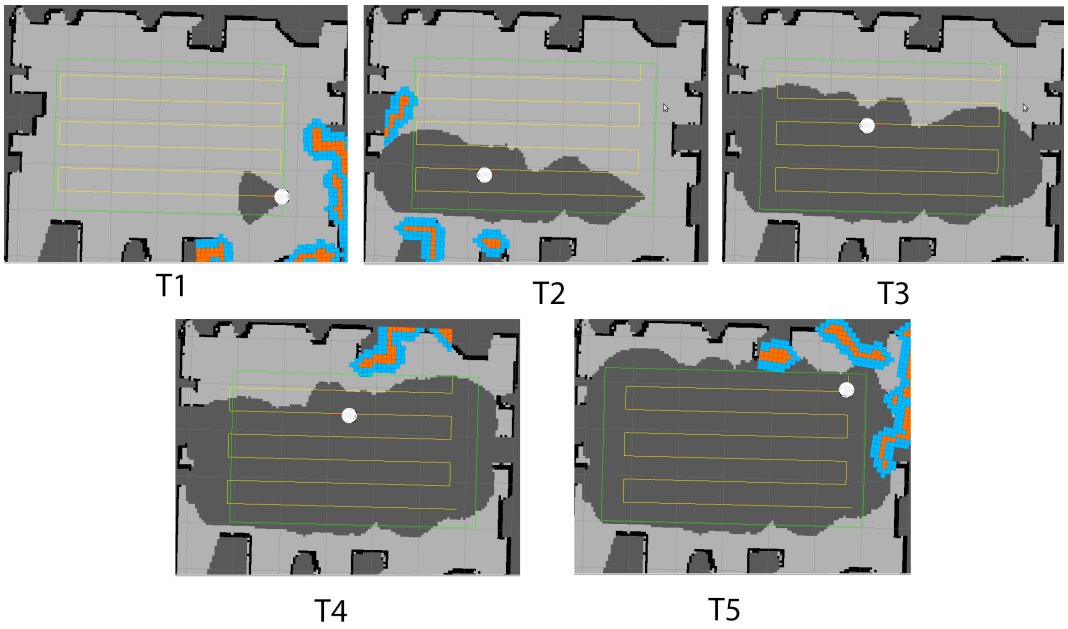
v *Rviz*, kjer bo pot predstavljena z vektorji med dvema sosednjima točkama. Na enak način lahko predstavimo detekcijo prostora.

Smiselno je označiti tudi prostor, ki ga je robot že videl na zemljevidu. Pri tem smo si pomagali s kopijo zemljevida, pri katerem smo polja, ki jih je robot videl, označili. Velikost polja, ki robot vidi pred seboj, smo določili eksperimentalno. Razteza se 1 meter pred robota z vidnim kotom 50°. Pomembno je, da smo vrednosti zemljevida spreminjali na naši kopiji zemljevida, saj bi v nasprotnem primeru uničili robotovo lokalizacijo ter načrtovanje poti. Sliki 5.1 in 5.2 predstavljata potek označevanja med preiskovanjem prostora. Slika 5.1 je posneta na robotu iRobot Roomba, Slika 5.2 pa v simulatorju.



Slika 5.1: Robot se bo premikal znotraj zelenega pravokotnika. Svetlo siva barva predstavlja še neraziskan prostor. Rdeča ter modra barva v okolici zidov predstavljajo t.i. *napihnjene ovire (ang. inflated obsticles)*. Ker je robot v sistemu predstavljen kot točka, vse zidove razširimo za radij robota (modra barva). Slika A predstavlja začetek preiskovanja, robot je videl le prostor, ki je trenutno pred njim. V sliki B je robot prepoznal predmet (zelen krog). V sliki C je ponovno prepoznal predmet iz slike B, a ga je zaradi napake pri lokalizaciji prestavil.

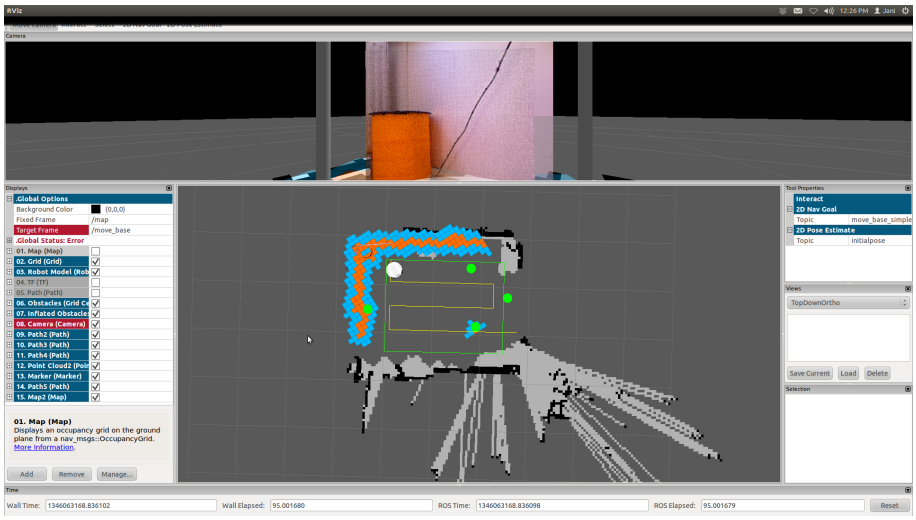
V *Rvizu* lahko izrišemo enostavne 3D objekte (kocka, valj ...). To bomo uporabili pri prikazovanju predmetov, ki smo jih zaznali. Ko bomo tak pred-



Slika 5.2: Slike predstavljajo sekvenco raziskovanja prostora v simulatorju. Ker je robot v simulatorju natančnejši (odometrija deluje pravilno z verjetnostjo 1), je zemljevid lepše zgrajen prav tako gibanje robota poteka točno po začrtani poti. Ob končanem preiskovanju vidimo, da je robot videl $99,9\%$ prostora. (Legenda je enaka kot pri Sliki 5.1.)

met zaznali, bomo v Rviz poslali njegove koordinate. Tukaj velja poudariti, da ima vsak predmet narisan v Rviz svoj id, zato je pomembno, da vsak predmet ločimo po id, tudi če gre za enak predmet. Kako to naredimo, smo opisali v Poglavju 4.5.2. Primer *Rviza* z vsemi vhodnimi podatki, ki jih izrisujemo, je na Sliki 5.3.

V želji, da bi robota še bolj približali človeku, smo mu dodali govor. Robot tako glasovno opisuje svoje trenutne aktivnosti: začetek preiskovanja, konec preiskovanja, najdbe predmetov, vrsto predmetov. To smo naredili s pomočjo programa, ki nam besedilo avtomatsko prevede v govor. Vgrajen je v operacijski sistem *Ubuntu* 10.11.



Slika 5.3: Prikaz Rviza. V zgornjem predelu vidimo sliko iz videosistema. Osrednji del predstavlja zemljevid, na katerem lahko vidimo robota (bel krog) ter prepoznane predmete (zeleni krogi). Pot je predstavljena z barvnimi vektorji. V levem delu se nahajajo *teme*, na katerih *Rviz* posluša.

5.2 Eksperimentalni rezultati

Z opisanimi metodami v prejšnjih Poglavjih smo naredili robota, ki bo kos zastavljeni nalogi. Za iskane predmete smo izbrali embalažo za čaj, DVD ter mleko (Slika 5.4). V nadaljevanju jih bomo imenovali: čaj, igra ter mleko. Da bi preverili ali robot uspešno prepozna dva enaka predmeta na različnih legah, smo uporabili dve embalaži mleka ter ju postavili vsaj 0,5 metra narazen.

Da bi lahko pot bolje načrtali, smo se odločili izmeriti razdalje, na katerih robot uspešno zazna predmet. Meritve smo izvedli tako, da smo robota postavili na neko točko, nato pa premikali predmete naravnost pred njim. Ugotovitve so zbrane v Tabeli 5.1. Kinect ne vidi predmeta le tik pred seboj, ampak ima vidno polje okrog 50°. Želeli smo se prepričati, ali robot na robovih vidnega polja enako uspešno zaznava predmete kot v sredini. Izkazalo se je, da se minimalna razdalja ne spremeni signifikantno, maksimalna pa se nekoliko zmanjša. Izmerili smo maksimalno ter minimalno razdaljo, pri

Slika 5.4: Predmeti, katere je moral poiskati robot.

kateri robot zazna predmet. Pri izvajanju meritev smo opazili, da se zanesljivost zaznavanja predmetov razlikuje med robnima razdaljama ter poziciji med njima, zato smo se odločili izmeriti to srednjo razdaljo, kjer se prepoznavanje zgodi z veliko verjetnostjo. Rezultati meritev so zbrani v tabeli 5.1 in 5.2.

Tabela 5.1: Razdalja pri kateri statičen robot zazna predmet

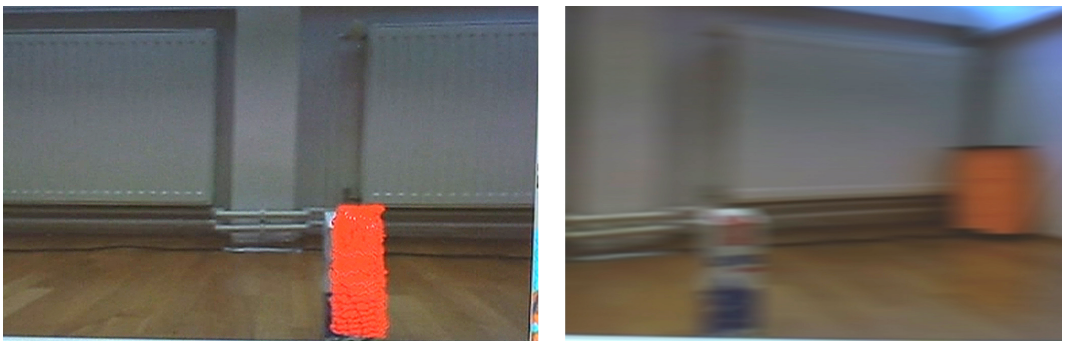
Razdalja	Mleko	Igra	Čaj
Min	0,6m	0,5m	0,6m
Zanesljivo	0,8m-1,05m	0,5m-1,4m	0,7m-0,8m
Maks	1,2m	1,6m	1,0m

Tabela 5.2: Razdalja zaznanega predmeta pri kotu 25°.

Razdalja	Mleko	Igra	Čaj
Min	0,6m	0,6m	0,7m
Zanesljivo	0,7m-0,9m	0,6m-1,0m	//
Maks	1,0m	1,2m	0,9m

Sledila je poizkusna vožnja, v kateri smo opazili, da je razdalja, pri kateri je robot zaznal predmet, veliko slabša od prej izmerjene. Ugotovili smo, da se slika zaradi premikanja robota zamegli, kar otežuje detekcijo predmeta. Da bi natančno ugotovili posledice zameglitve zaradi premikanja, smo ponovili test iz tabele 5.1. Tokrat smo predmet pustili na mestu, z robotom pa smo

se mu začeli približevati z različno hitrostjo. Ker se bo robot predmetu (iz perspektive kamere) vedno približeval, je zadostno izmeriti le maksimalno dolžino. Izkazalo se je, da se razdalja ne zmanjša veliko. Natančne meritve so zapisane v Tabeli 5.3. Večje poslabšanje detekcije je bilo pri rotaciji robota okrog svoje osi, saj je v tem primeru kotna hitrost veliko večja. Zameglitev je tako močna, da jo lahko opazimo s prostim očesom na Sliki 5.5. Preizkus smo izvedli tako, da smo predmet postavili na razdaljo 0,8 metra od robota (pri tej razdalji prepoznamo vse tri predmete najbolje), robota smo sprva obrnili tako, da predmeta ni videl, nato smo ga rotirali za 360°. Izsledki so predstavljeni v Tabeli 5.4.



Slika 5.5: Sliki, ki ju robot uporablja za detekcijo predmetov. V levi sliki je robot miroval, vidimo lahko nekaj šuma a robot predmet uspešno prepozna. Desna slika je bila posneta med rotacijo robota okrog svoje osi. Popačenje slike je veliko. Robot predmeta ne razpozna.

Tabela 5.3: Razdalja zaznanega predmeta pri različni hitrosti približevanja.

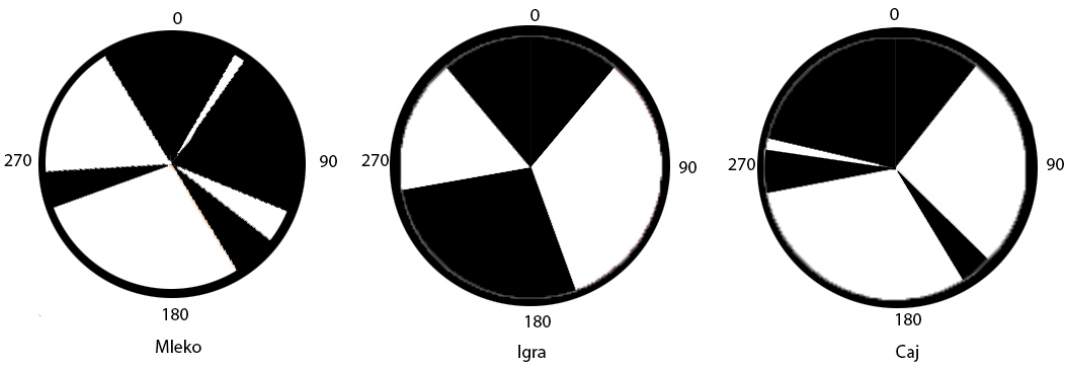
Hitrost	Mleko	Igra	Čaj
0,2 m/s	1,1m	1,3m	0,6m
0,4 m/s	1,0m	1,2m	0,6m
0,6 m/s	0,9m	1,0m	0,6m

Med nadaljnjimi testnimi vožnjami smo opazili razliko med dolžino, pri kateri je robot zaznal predmet glede na stranico predmeta, ki je bila obrnjena

Tabela 5.4: Hitrost rotacije, merjena v rad/s, pri katerih predmet še zaznamo.

Hitrost	Mleko	Igra	Čaj
1,0 rad/s	Zazna	Zazna	Zazna
1,2 rad/s	Zazna	Zazna	Zazna
1,3 rad/s	Zazna	Zazna	X
1,4 rad/s	X	Zazna	X
1,5 rad/s	X	X	X

proti robotu. Posumili smo, da model predmeta ni enakomerno zgrajen. Ob natančnem opazovanju postopka gradnje modelov se je izkazalo, da skener kontinuirano snema predmet, ko zazna oznake na podlagi, to sliko shrani v model. Med obračanjem predmeta se zato zgodi, da predmet pod nekaterimi koti ni dovolj dobro posnet. Gradnjo modela poslabša tudi neprestano vrtenje modela med učenjem, kar povzroča zameglitev zaradi premikanja. Da bi bolje ugotovili pod katerimi koti je predmet bolje prepoznan, smo predmet ponovno postavili 0,8 metra pred robota, nato pa predmet rotirali okrog svoje osi. Izkazalo se je, da je predmet najbolje prepozna v svoji začetni legi ter tam kjer ima večje ploskve, najslabše pa na ogliščih. Meritve so predstavljene na Sliki 5.6.



Slika 5.6: Rotacija predmeta pri kateri ga robot uspešno zazna. Črna barva predstavlja uspešno zaznan predmet.

Po končanih meritvah smo lahko bolje nastavili parametre robota. Vedeli smo, da mora prostor preiskati tako, da ne bo od nobene točke oddaljen več kot 1 meter. Da bi *načrtu* dodali še dodatno stopnjo zanesljivosti, smo se odločili, da bo razdalja med dvema horizontalnima črtama *načrta* 0,5 metra, zato bo robot večino predmetov videl vsaj 2-krat. Hitrost robota smo nastavili na 0,2 m/s, hitrost rotacije pa na 1 rad/s.

Za ocenjevanje uspešnosti programa smo na naključne pozicije ter z naključno rotacijo v prostor postavili 4 predmete (Slika 5.7). Nato smo pognali program ter zabeležili pravilno poiskane predmete ter napake, ki so se zgodile pri tem. Po vsaki vožnji smo predmete nekoliko premestili. Postopek smo ponovili 10-krat. Program je pravilno deloval v polovici primerov, v ostalih primerih pa je vsaj en predmet spregledal, ga narobe postavil ali napačno prepoznal. Izsledki so zapisani v Tabeli 5.5. Drugi stolpec v tabeli predstavlja število uspešno prepoznanih predmetov od 4 možnih, tretji stolpec pa napačno prepoznane ali postavljene. Na primer, če smo pravilno odkrili vse 4 predmete, nato pa napačno prepoznali dodaten predmet (npr. v steni) bomo to zapisali, kot je napisano v vožnji št. 2 v Tabeli 5.5.

Slika 5.7: Robot ter predmeti v prostoru, katerega mora raziskati.

Tabela 5.5: Preizkus sistema

Številka vožnje	Uspešno najdeni predmeti	Napačno klasificirani/ postavljeni predmeti
1	4	0
2	4	1
3	3	0
4	4	0
5	4	0
6	3	1
7	2	0
8	3	0
9	4	0
10	4	0
Popolne vožnje	5/10	//

Glede na rezultate iz Tabele 5.5 vidimo, da robot ni tako zanesljiv, kot smo si želeli. Za robota smo uporabili robotski sesalnik, ki ni bil nikoli načrtovan za natančno merjenje ter raziskovanje prostora. Poleg tega smo na robota namestili kamero Kinect, zaradi česar se je težišče robota pomaknilo nazaj, to pa je ob pospeševanju povzročilo dvigovanje sprednjega dela robota. V Poglavju 2.1 smo omenili, da ima robot na sprednjem delu kroglico, ki zaznava premike robota, da ta deluje, se mora dotikati tal. Zaradi vsega naštetega je odometrija robota slaba, kar povzroči težave pri gradnji zemljevida, s tem pa posledično tudi pri lokalizaciji robota. Zaradi teh težav je robot isti predmet zaznal na dveh različnih mestih, kar je privedlo do napačno prepoznanih predmetov.

Vizualno zaznavanje predmetov je zahteven problem v idealnih pogojih. Mi smo prepoznavanje dodatno otežili s tem, ko smo kamero namestili na mobilno platformo, kar je privedlo do zameglitve slike zaradi premikanja. Da bi ta problem omilili, smo znižali pragove, pri katerih se predmet šteje kot prepoznanega, zaradi česar smo v nekaterih primerih dobili napačno prepoznan

predmet v stenah in ostalih predmetih.

Ob upoštevanju opreme robota lahko algoritme, ki smo jih uporabili za opravljanje naloge diplomskega dela, ocenimo za uspešne. Če bi opremo zamenjali s kvalitetnejšo, bi lahko pričakovali, da se bo uspešnost preiskovanja močno povečala.

Poglavje 6

Sklep

Namen diplomskega dela je bil izdelava robota, ki bo zmožen v prostoru poiskati vnaprej določene predmete. Z namenom, da bo naš program čim bolj univerzalen in bo lahko deloval na več različnih vrstah robotov, smo ga izdelali v okolju ROS. Program je bil izdelan za robota iRobot Roomba ter barvno-globinsko kamero Kinect.

Program smo razdelili na tri podprobleme: navigacija, vizualno zaznavanje ter obdelovanje podatkov. Pri zaznavanju smo uporabili implementacijo RoboEarth, ki temelji na sodobnem algoritmu SURF in algoritmu najbližjih sosedov.

Algoritmi, ki smo jih sami implementirali, so delovali po pričakovanjih. Probleme so nam povzročali deli, ki smo jih le integrirali. Pri navigaciji je probleme povzročal lokalni načrtovalec poti, saj se je našega globalnega načrta oklepal zelo ohlapno, kar je povzročilo preskakovanje med horizontalnimi linijami načrta ter krajšanje zavojev. Problem smo uspešno rešili z implementacijo vmesnega načrtovalca, ki je celoten globalni načrt razdelil na manjše odseke, ki jih je nato podal lokalnemu načrtovalcu. Še večje probleme nam je povzročil vizualni detektor, saj ga zaradi pomanjkanja dokumentacije ni bilo mogoče prirediti glede na naše potrebe. Problem smo delno rešili tako, da smo vse ostale dele programa priredili, da so bili kar se da v pomoč prepoznavanju predmetov. V splošnem se je izkazalo, da je ves sistem ROS

zelo slabo dokumentiran, kar je povzročalo številne težave in nevšečnosti.

Kljub vsem težavam nam je program uspelo uspešno zaključiti. Robot v večini primerov najde vse predmete, ko pa pride do napake, se ta največkrat pojavi pri nezaznavanju predmeta. Predvidevamo, da bi strojno močnejši računalnik do neke mere omilil ta problem, saj bi lahko nekatere pragove pri detekciji predmetov spremenili tako, da bi s tem omogočili prepoznavanje predmetov z večje razdalje. Prav tako bi pomagala uporaba natančnejše barvno-globinske kamere.

Robotika danes postaja vse bolj popularna na številnih področjih. Le še vprašanje časa je, kdaj bodo nekatere vrste hišnih robotov postale tako nepogrešljive, kot so danes mobilni telefoni. Tudi našega robota bi lahko nadgradili tako strojno kot programsko, da bi postal resnično uporaben. Če bi mu na primer dodali robotsko roko, s katero bi lahko prijemal predmete ter mu dodali natančnejši barvno-globinski videosistem, bi kaj kmalu dobili hišnega pomočnika, ki nam poišče ključe ali prinese knjigo. Vsekakor lahko takšne in podobne sisteme pričakujemo v bližnji prihodnosti.

Literatura

[1] Danijel Skočaj, Transformacije med koordinatnimi sistemi, spletna učilnica FRI (2012), Dostopno na:
http://ucilnica.fri.uni-lj.si/mod/resource/view.php?id=17219

[2] T. Bajd, Osnove robotike, Založba FE in FRI, 2006

[3] (2012) Bradley Hiebert-Treuer, An Introduction to Robot SLAM, Dostopno na:
http://ceit.aut.ac.ir/ shiry/lecture/robotics/Robot

[4] (2012) TurtleBot, Dostopno na:
http://ros.org/wiki/Robots/TurtleBot

[5] (2012) Assembling the Roomba, Dostopno na:
http://ros.org/wiki/lse_roomba_toolbox/Tutorials/Assembling

[6] (2012) Kinect, Dostopno na:
http://en.wikipedia.org/wiki/Kinect

[7] (2012) Darwing Blanks, Kinect in infrared, Dostopno na:
http://bbzippo.wordpress.com/2010/11/28/kinect-in-infrared/

[8] (2012) Dijkstra's algorithm, Dostopno na:
http://en.wikipedia.org/wiki/Dijkstra's_algorithm

[9] H. Bay, A. Ess, T. Tuytelaars, L. V. Gool, SURF: Speeded-Up Robust Features, *Lecture Notes in Computer Science*, 2006, Dostopno na:
http://glorfindel.mavrinac.com/ aaron/school/pdf/bay06_surf.pdf

[10] (2012) Daniel Gasienica, Inside Deep Zoom Part 1: Multiscale Imaging, Dostopno na:
http://www.gasi.ch/blog/inside-deep-zoom-1/

[11] (2012) RoboEarth, Dostopano na:
http://www.roboearth.org/

[12] (2012) Creating and using RoboEarth object models, Dostopano na:
http://www.youtube.com/watch?v=5uMCa-dgtFE&feature=player_embedded

[13] Marco Zuliani, RANSAC for Dummies, Dostopno na:
http://vision.ece.ucsb.edu/ zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf

[14] (2012) ROS Move Base, Dostopno na:
http://www.ros.org/wiki/move_base

[15] (2012) ROS Gmapping SLAM, Dostopno na:
http://www.ros.org/wiki/gmapping

[16] (2012) ROS, Dostopno na:
http://www.ros.org/wiki/

[17] (2012) Robot, Dostopno na:
http://en.wikipedia.org/wiki/Robot

[18] (2012) ROS TF, Dostopno na:
http://www.ros.org/wiki/tf

[19] (2012) ROS Roboearth, Dostopno na:
http://www.ros.org/wiki/roboearth

[20] (2012) Hardik Pandya, Microsoft Kinect: Technical Introduction Dostopno na:
http://entreprene.us/2011/03/09/microsoft-kinect-technical-introduction/

[21] (2012) ROS Pluginlib, Dostopno na:
http://www.ros.org/wiki/pluginlib

[22] Ben-Gal I., Ruggeri F., Kenett R., Bayesian Networks, *Encyclopedia of Statistics in Quality & Reliability*", Willey & Sons (2007), Dostopno na:
http://www.eng.tau.ac.il/ bengal/BN.pdf

[23] (2012) Classical Mechanics, Government College Mandya, poglavje 1, Dostopno na:
http://www.gcm.ac.in/downloads/elearning/classical_mechanics.pdf

[24] Yongyin Cao, Alexander Stilgoe, Lixue Chen, Timo Niemien, Halian Rubinsztein-Dunlop, Optics Express, Vol. 20, Issue 12, Dostopno na:
http://www.opticsinfobase.org/vjbo/fulltext.cfm?uri=oe-20-12-12987&id=233614

[25] Bilgin Esme, Kalman Filter For Dummies, *A Mathematically Challanged Man's Search for Scientific Wisdom*, 2009 Dostopno na:
http://bilgin.esme.org/BitsBytes/KalmanFilterforDummies.aspx

[26] Brian Odelson, Murali Rajamani, James Rawlings, A new Autocovariance Least-Squares Method for Estimating Noise Covariances, University of Wisconsin-Madison, 2005 Dostopno na:
http://jbrwww.che.wisc.edu/tech-reports/twmcc-2003-04.pdf

[27] (2012) Kalman Filter, Dostopno na:
http://en.wikipedia.org/wiki/Kalman_filter

[28] (2012) Greg Welch, Gary Bishop, An Introduction to the Kalman Filter, University of North Carolina at Chapel Hill, 2006, Dostopno na:
http://www.cs.unc.edu/ welch/media/pdf/kalman_intro.pdf