UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Grega Pušnik

Detekcija in sledenje človeku z mobilno platformo in barvno-globinsko kamero

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Danijel Skočaj

Ljubljana, 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

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: 01811/2012
Datum: 15.03.2012

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

Kandidat: **GREGA PUŠNIK**

Naslov: **DETEKCIJA IN SLEDENJE ČLOVEKU Z MOBILNO PLATFORMO IN BARVNO-GLOBINSKO KAMERO**

HUMAN DETECTION AND TRACKING WITH MOBILE PLATFORM AND RGBD CAMERA

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Z vedno bolj pogostim vključevanjem mobilnih robotov v vsakdanje življenje narašča potreba po čim bolj naravni interakciji med robotom in človekom. Ena izmed osnovnih sposobnosti mobilne platforme v tem smislu je sledenje človeku. Mobilni robot naj bi bil sposoben zaznati človeka ter slediti njegovemu gibanju po prostoru. Izdelajte robotski sistem, ki bo znal s pomočjo barvno-globinske kamere Kinect detektirati človeka ter mu nato s pomočjo mobilne platforme IRobot Roomba slediti po prostoru z ravnimi tlemi. Pri tem uporabite in razvijte primerne algoritme za obdelavo vizualne barvne in tri-dimenzionalne informacije, ter primerne algoritme za načrtovanje poti.

Mentor:

doc. dr. Danijel Skočaj

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Grega Pušnik,

z vpisno številko 63050093,

sem avtor diplomskega dela z naslovom:

Detekcija in sledenje človeku z mobilno platformo in barvno-globinsko kamero

Human detection and tracking with mobile platform and RGBD camera

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne 10. 9. 2012 Podpis avtorja:

Zahvala

Rad bi se zahvalil vsem, ki so me kadarkoli in kakorkoli podpirali tekom študija in med pisanjem diplomskega dela. Izpostavim naj družino, ki me je skozi celotno šolanje podpirala tako finančno kot moralno. Seveda se zahvaljujem tudi punci Urški Kužnik, ki mi je stala ob strani. Za pomoč pri pisanju in izdelavi diplomske naloge bi se zahvalil tudi:

- mentorju doc. dr. Danijel Skočaju, ki mi je skozi celotno diplomsko delo svetoval in me usmerjal,
- Vojku Kužniku in Blažu Pušniku za pomoč pri izdelavi robotskega ogrodja,
- Fillipu Bassu za pomoč in svetovanje pri uporabi knjižnice *Bayes++*.

Kazalo

Seznam uporabljenih kratic in simbolov

Kratica	Angleški izraz	Slovenski izraz
RGB	Red, Green, Blue	Barvni model, ki temelji na optičnem mešanju treh osnovnih barv - rdeče, zelene in modre.
SIFT	Scale-Invariant Feature Transform	Vizualni opisnik invarianten na merilo.
HOG	Histogram of Oriented Gradients	Histogram orientiranih gradientov
RGBD	Red, Green, Blue, Depth	Model poleg barvne informacije vsebuje tudi informacijo o globini.
ROS	Robot Operating System	Robotski operacijski sistem

EOH	Edge Orientation Histogram	Histogram smeri gradientov robov
HOD	Histrogram of oriented depths	Histogram orientiranih globin
BSD	Berkley Software Distribution	Berkeleyjska programska distribucija
SVM	Support Vector Machine	Algoritem strojnega učenja z uporabo podpornih vektorjev.
URDF	Unified Robot Description Format	Poenoten format za opis robota.
XML	Extensible Markup Language	Razširljivi označevalni jezik
CAD	Computer-aided Design	Računalniško podprto oblikovanje
PCL	Point Cloud Library	Knjižnica za procesiranje oblaka točk.
GPE	Graphics Processing Unit	Grafično procesna enota
CPE	Central Processing Unit	Centralno procesna enota
RANSAC	Random Sample Consensus	Soglasje naključnih vzorcev

Povzetek

Področje računalniškega vida se že dolgo časa ukvarja s področjem detekcije (zaznavanja) in sledenja človeku, saj je to eden ključnih problemov za učinkovito delovanje avtonomnega robota med ljudmi. Zaradi različnih poz človeka, osvetljenosti, kompleksnosti ozadja in drugih spremenljivih lastnosti je to vse prej kot enostaven problem. V preteklosti so se raziskovalci večinoma posluževali cenovno ugodnejših *2D* barvnih kamer. S prihodom nizkocenovne barvno-globinske kamere Kinect pa se je pospešeno začelo tudi raziskovanje na področju, kjer imamo poleg klasične *2D* barvne informacije na voljo tudi informacijo o globini in oblaku točk.

V diplomskem delu smo reševali problem zaznave in sledenja človeka s pomočjo barvno-globinske kamere na mobilni platformi. Sistem temelji na Robotskem operacijskem sistemu - *ROS*, mobilnem robotu IRobot Roomba in barvno-globinskemu senzorju Kinect. Za zaznavo človeka smo najprej izkoristili *3D* informacijo, ki nam jo omogoča Kinect, nato pa smo za začetno klasifikacijo človeka uporabili algoritem *HOG*. Pri sledenju smo se za bolj robustno ponovno zaznavo poslužili inkrementalnega Adaboost algoritma s Haarovimi značilnicami in zoženjem okna iskanja. Za primere, ko zaznave človeka nimamo ali

smo ga izgubili, pa uporabimo napovedi lokacije Kalmanovega filtra. Tako smo celoten sistem implementirali v *ROS-u*, ki nam je med drugim tudi s pomočjo svojih knjižnic omogočal pošiljanje in prejemanje ukazov robota kot tudi navigacijo po prostoru.

Ključne besede:

zaznavanje človeka, sledenje človeku, *ROS*, mobilna platforma, Kinect, robot, IRobot Roomba, navigacija po prostoru

Abstract

For a long time detection and tracking of people has been one of the main research topics in computer vision. This is one of the key problems for efficient interaction of an autonomous robot with people. It is a complex problem due to various human poses, lighting, background complexity and other variables. In the past, researchers were mostly using *2D RGB* cameras. With the arrival of a low cost *RGBD* camera Kinect, researchers have increasingly started using not only *2D* information, but also depth information with a point cloud.
In the thesis we address the problem of a human detection and tracking on a mobile platform. System is based on *ROS* - Robot Operating System, mobile robot IRobot Roomba and *RGBD* sensor Kinect. For human detection we first use the depth information of the Kinect and *HOG* algorithm for the initial classification. For re-detection, algorithm narrows search window and then for classification, resorts to more robust online Adaboost algorithm with Haar features. For cases where we do not positively classify or lose the human we use predictions of the Kalman filter. For robot navigation we used ROS navigation stack. That is how we implemented the whole detection-tracking system for detecting, tracking and following people.

Key words:

human detection, human tracking, ROS, mobile robot, Kinect, IRobot Roomba,

robot navigation

Poglavje 1

Uvod

Vid je za ljudi ključnega pomena pri opravljanju praktično katerekoli stvari, ki se je v vsakdanjem življenju lotimo. Prepoznava prijatelja, objekta na mizi, sledenje ptice v letu. Takšne in podobne stvari se nam zdijo samoumevne. Prav zaradi tega se nam človeški vid na prvi pogled ne zdi prav nič zapleten in je zato področje računalniškega vida s strani splošne javnosti podcenjeno. Marsikdo pri tem pozabi, da se človeški vid in zmožnost prepoznavanja ter sledenja razvija praktično že od rojstva s pomočjo najzmogljivejšega računalnika - možganov.

Človek si že od nekdaj želi olajšati, poenostaviti in pohitriti marsikatera opravila. Tako so danes npr. tovarne, ki pri izdelavi produktov ne uporabljajo robotov, že prav redke. V prihodnosti bo prišel tudi čas, ko bo robot prevzel vlogo osebnega pomočnika in namesto nas opravil marsikatera dela, ki jih zdaj opravljamo še sami. Tako bo postal del našega vsakdana. Prav zaradi ključne vloge vida pri človeškem delovanju je tudi pri robotu to ena izmed pomemb-

nejših stvari za njegovo uspešno delovanje v vsakdanjem okolju. To je eden izmed glavnih razlogov, da se raziskovalci vedno več ukvarjajo z reševanjem problema računalniškega vida.

Zastavljen problem diplomskega dela je detekcija in sledenje človeku s pomočjo barvno-globinske kamere na mobilni platformi. Problem lahko razdelimo na dva dela: detekcija in sledenje. V računalniškem vidu reševanje teh dveh problemov že dolgo velja za zahtevno nalogo. Večina raziskovalcev za detekcijo uporablja le *2D* informacijo [1] ali stereo vid [2]. Nekatere metode vključujejo statistično učenje, ki bazira na lokalnih značilnicah, kot npr. *HOG* [3] in *EOH* [4]. V uporabi je tudi izvleček zanimivih točk iz slike, kot npr. *SIFT* [5] itd. Čeprav testi prikazujejo, da je lahko detekcija s pomočjo *2D* informacije zelo zanesljiva, prihaja do problemov, ko je človek v neznačilnih pozah, ali pa ko je ozadje neurejeno (*angl. cluttered*). V takih primerih zanesljivost pade in naraste računska zahtevnost. Še dodatno težavo predstavlja zaznava v notranjem okolju, saj je človek mnogokrat zastrt z različnimi objekti.

Zaznavanje ljudi s pomočjo barvno-globinskega senzorja - *RGBD* je precej novo področje, ki se še ni dodobra razvilo. Razvoj se je v zadnjih letih začel predvsem s prihodom nizko-cenovnega barvno-globinskega senzorja Kinect. Raziskovalci uporabljajo različne načine, kako dobro uporabiti še dodatno globinsko informacijo in oblak točk za zaznavo ljudi. Različni načini vključujejo različne variante *HOD-a* [6], Bottom-up Top-down detektor [7], razrez *3D* slike na *2D* kose [8], zaznavo z laserjem [9] itd. Za realizacijo diplomskega dela smo ubrali podoben pristop, kot ga v svoji magistrski nalogi opisuje g. Basso [10]. Uporabimo tako *2D* kot tudi *3D* informacijo barvno-globinske kamere in vse

skupaj združimo v enoten robotski sistem, ki temelji na ROS-u in mobilni platformi IRobot Roombi.

V prvem delu diplomskega dela pokažemo, kako je celoten sistem zgrajen in katere knjižnice so ključne za delovanje našega sistema. V drugem delu opišemo vse pomembne algoritme, ki smo jih uporabili pri reševanju problema detekcije in sledenja človeka z mobilno platformo in barvno-globinsko kamero. V tretjem delu pa opišemo, kako smo vse skupaj združili v celoten sistem, in definiramo, kako poteka zaznavanje, sledenje in navigacija robota po prostoru. Na Sliki 1.1 je prikazano delovanje celotnega sistema. V zadnjem delu pa še predstavimo prednosti in slabosti sistema in na kakšen način bi ga bilo mogoče še izboljšati.

Slika 1.1: Prikaz delovanja sistema detekcije in sledenja človeka na mobilni platformi.

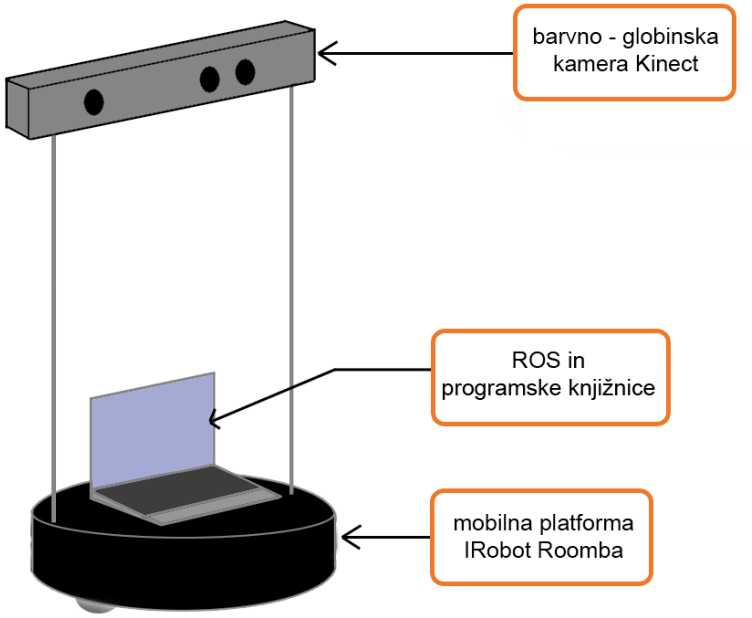
Poglavje 2

Zgradba sistema

V drugem poglavju vam bomo predstavili in opisali komponente, ki smo jih uporabili pri reševanju problema detekcije in sledenja človeka z mobilno platformo in barvno-globinsko kamero. V prvem podpoglavju vam bomo podali več informacij o mobilnem robotu IRobot Roomba. Sledi drugo podpoglavje z opisom barvno-globinske kamere Kinect. Tretje podpoglavje je namenjeno opisu in zgradbi Robotskega operacijskega sistema - *ROS*, ki mu sledi opis uporabljenih višje-nivojskih knjižnic. V zadnjem, petem podpoglavju, pa še opišemo, kako smo vse komponente povezali in implementirali enoten robotski sistem, ki nam je služil za detekcijo in sledenje človeka.

2.1 Oris sistema

Na Sliki 2.1 je grafično prikazana zgradba robotskega sistema.



Slika 2.1: Prikaz zgradbe robotskega sistema.

2.2 IRobot Roomba

IRobot Roomba je serija avtonomnih robotskih sesalnikov, ki jih prodaja podjetje IRobot. Njen prvoten namen je avtonomna navigacija po prostoru in sesanje le-tega. Javnosti je bila predstavljena leta 2002 in do februarja leta 2011 je bilo prodanih že **več kot 6 milijonov enot** [11]. Ugodna cena in skupek senzorjev sta pripeljala do tega, da je Roomba idealen robot za nizkocenovne robotsko-akademske kot tudi osebne raziskave in projekte. Roombo podpira tudi *ROS*. Roomba je **okrogle oblike, premera 34 cm in višine manj kot 9 cm**. Za zaznavanje trkov s sprednje strani ima velik odbijač. Na sprednjem delu najdemo tudi vsesmerni infrardeči senzor (*IR*) in štiri senzorje

za previse. Ima dve kolesi, ki se upravljata neodvisno in katerih ni mogoče obračati. Roombo uvrščamo med robote z **diferencialnim pogonom**, kar pomeni, da lahko pride kamor koli je robot glede na zgradbo zmožen priti. V diplomi smo uporabili IRobot Roombo 555, ki je prikazana na Sliki 2.2.

Slika 2.2: IRobot Roomba 555

2.3 Barvno-globinska kamera Kinect

2.3.1 Opis

Kinect je Microsoftova naprava, ki zaznava gibanje za konzole Xbox 360. Uporabniku omogoča interakcijo z Xbox-om brez dodatnega krmilnika, saj ga lahko

upravlja le z gibi telesa in govornimi ukazi. Uporabniku omogoča drugačno izkušnjo z igro, v katero se mora tudi sam vključiti z govorom in gibi telesa. Kinect je bil širnemu svetu predstavljen meseca novembra leta 2010. V prvih 60 dneh je bilo prodanih **8 milijonov enot.** Do meseca januarja leta 2012 je bilo prodanih že **več kot 18 milijonov enot** [12].

Kinect je zgrajen iz **dveh kamer** in enega **IR projektorja**. Senzorji objavljajo sliko s hitrostjo 30 slik na sekundo.
Kamera RGB je 8-bitna z ločljivostjo *VGA* - $640\,x\,480$ pik z Bayerjevim barvnim filtrom. Druga, monokromatska kamera, ki zaznava globino, je enake ločljivosti z 11-bitno globino. Tako omogoča 2048 stopenj občutljivosti. Vgrajeno **polje mikrofonov** vsebuje štiri mikrofone, vsak izmed njih pa je sposoben procesiranja 16-bitnega zvoka s frekvenco vzorčenja 16 kHz. Praktična razdalja od senzorjev je nekje od $0.7 - 6$ metra s horizontalnim vidnim kotom 57 stopinj in z vertikalnim kotom 43 stopinj. S pomočjo motorja za nagib je mogoč še dodaten 27 stopinjski vertikalni naklon. Zgradba je prikazana na Sliki 2.3.

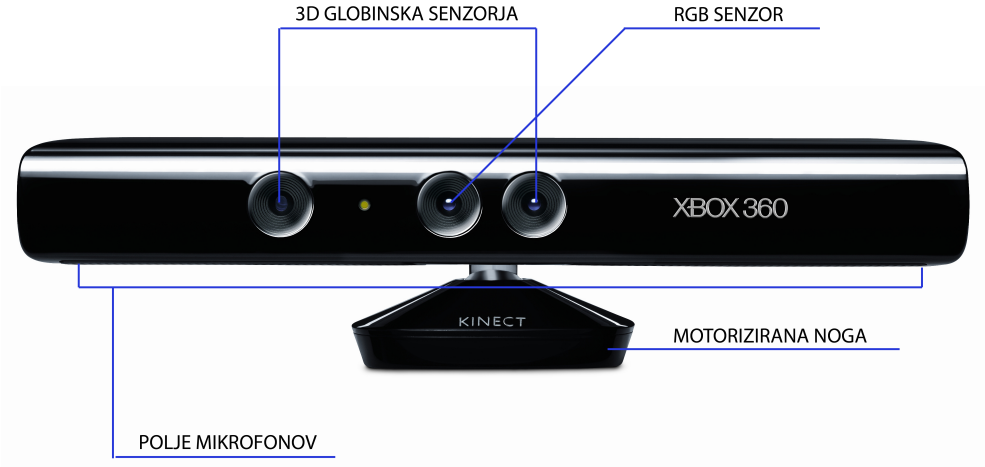
2.3.2 Gradnja 3D slike

V članku [13] J. MacCormick podrobno opiše delovanje Kinect-a. Kinect uporablja **infrardeči projektor in senzor za gradnjo 3D slike**. S pomočjo analiziranja vzorca, ki ga infrardeči laser projecira na površino, se zgradi globinski zemljevid. Sistem dela po principu strukturirane svetlobe (*angl. struc-*

tured light). Kinect združi strukturirano svetlobo s pomočjo dveh klasičnih tehnik računalniškega vida:

- **Globina iz fokusa** deluje po principu – bolj ko je objekt moten, bolj je oddaljen.
- **Globina iz sterea,** če gledaš na sceno z drugega kota, deluje po principu – bolj ko je objekt blizu, bolj je zamaknjen. Ta zamik Kinect analizira tako, da projecira infrardeči žarek z ene lokacije in opazuje z druge. Na ta način dobimo **oblak točk - globinski zemljevid točk**.

Kinect uporablja desno sučni koordinatni sistem s koordinato x za horizonzalno pozicijo, y za vertikalno in z koordinato za oddaljenost od senzorja.



Slika 2.3: Prikaz Kinecta in njegovih senzorjev.

2.3.3 Kaj je oblak točk?

Oblak točk je podatkovna struktura, ki predstavlja zbirko več dimenzijskih točk. Je pogosto uporabljen za predstavitev **tri dimenzionalnih podatkov**.

V *3D* oblaku točk je točka navadno predstavljena z x, y, z geometrično koordinato. Ko je prisotna tudi barvna informacija, postane oblak točk **3+3 dimenzionalen** (3 dimenzije za x, y, z koordinato točke in 3 dimenzije za zapis *RGB* barve točke). Oblak točk lahko dobimo iz strojnih senzorjev, kot npr. stereo kamere, 3D skenerji, lahko pa so tudi sintetično generirani s pomočjo računalnika. Oblak točk se uporablja za različne namene: za ustvarjanje modelov *CAD*, animacije, predstavitev volumetričnih podatkov v zdravstvenih namenih in nazadnje tudi za računalniško zaznavanje objektov.

2.4 ROS - robotski operacijski sistem

ROS je programsko okolje za razvoj programske opreme za različne vrste robotov. Ima podobne funkcionalnosti operacijskega sistema na heterogenih računalniških gručah.

2.4.1 Zgodovina

ROS je bil prvotno razvit leta 2007 v Stanfordskem laboratoriju za umetno inteligenco, takrat še pod imenom *»Switchyard«*. Po letu 2008 se je razvoj nadaljeval v raziskovalnem inštitutu Willow Garage. Skupaj z več kot dvajsetimi institucijami se pospešen razvoj nadaljuje. Do leta 2012 so izdali že šesto stabilno različico *ROS-a* z imenom Fuerte. Razvoj ROS-a skozi čas je prikazan v tabeli 2.1.

Različica	Datum izdaje
ROS 1.0	22. 1. 2010
Box Turtle	1. 3. 2010
C Turtle	3. 8. 2010
Diamondback	2. 3. 2011
Electric Emys	30. 8. 2011
Fuerte	23. 4. 2012

Tabela 2.1: Razvoj ROS-a skozi zgodovino.

2.4.2 Glavne značilnosti

ROS nam omogoča servise standardnega operacijskega sistema, kot so strojna abstrakcija, nizko nivojski nadzor naprav, implementacija pogosto rabljenih funkcionalnosti, pošiljanje sporočil med procesi in paketno upravljanje. *ROS* bazira na **arhitekturi grafov**, kjer procesiranje poteka v vozliščih. Vsako vozlišče lahko sprejema in pošilja različne vrste sporočil. *ROS* je namenjen Unixovim sistemov. Ubuntu različica je označena za podprto, medtem ko so preostale različice, kot sta Fedora in Mac OS X, označene le z eksperimentalno. Osnovne funkcionalnosti *ROS-a* je mogoče uporabljati tudi v operacijskem sistemu Windows, a vsekakor ni primeren za zamenjavo z Linuxovo različico. Uporabna je le za preproste primere, saj Windows nima mehanizma za rokovanje s skaliranjem kompleksnosti (*rosdeps*).
ROS temelji na dveh osnovnih idejah:

- **operacijsko-sistemski del,** kot je opisan zgoraj,
- ***ros-pkg*** – **paketi**, ki jih prispevajo uporabniki in so lahko združeni v sklade (*angl. stacks*). Tako lahko s pomočjo obstoječih paketov delamo lokalizacijo robota v prostoru, gradnjo zemljevidov, simulacijo, zaznava-

nje itd.

ROS je preprosto implementirati v druge moderne programske jezike. Trenutno je implementiran v programskih jezikih Python, C++ in Lisp, različica za Javo in Lua-o pa je še preizkusna. *ROS* je izdan pod *BSD* licenco in je odprtokodni projekt. Brezplačen je tako za komercialno kot tudi za raziskovalno rabo. Paketi in skladi paketov pa so izdani pod različnimi odprto-kodnimi licencami.

2.4.3 Arhitektura grafov

Kot je že omenjeno, *ROS* temelji na arhitekturi grafov in je implementiran **z namenom visoke modularnosti**. Iz računskega vidika je to omrežje procesov *ROS* enak z enakim, ki skupaj obdelujejo podatke, med seboj pa si izmenjujejo sporočila. Osnovni koncepti računskega grafa, ki so predstavljeni na Sliki 2.4, so:

1. **Vozlišče (*angl. node*)** – je proces, ki izvaja računanje. Celoten robotski sistem je navadno zgrajen iz več vozlišč, izmed katerih je vsak zadolžen za izvajanje svojih operacij, kot npr. nadzor motorjev koles, lokalizacije, planiranja itd.

2. **Sporočilo (*angl. message*)** – Vozlišča med seboj komunicirajo s pošiljanjem sporočil. Sporočilo je preprosta podatkovna struktura, zgrajena iz različnih tipov polj. Podprti so vsi standardni tipi (*integer, float, boolean ...*) kot tudi polja primitivnih tipov. Prav tako lahko vsebujejo gnezdene strukture in polja (podobno kot podatkovne strukture C).

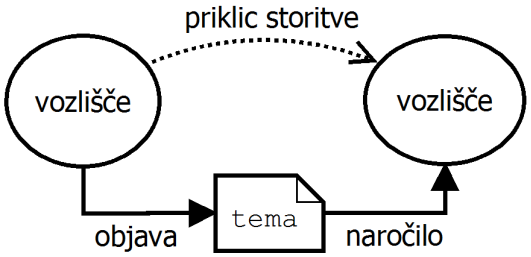
3. **Tema (*angl. topic*)** – Sporočila so usmerjena preko transportnega sistema z objavi/naroči (*angl. publish/subscribe*) semantiko. To poteka tako, da vozlišče pošlje sporočilo in ga objavi na določeni temi. Druga vozlišča lahko ta sporočila sprejemajo, tako da se naročijo na izbrano temo. Ime teme je skupna točka, kjer se prepozna vsebino in tip sporočila. Vozlišče lahko hkrati objavlja več sporočil. Prav tako je lahko na eno temo naročenih več vozlišč. V splošnem vozlišča ne vedo za druga vozlišča. Ideja je v tem, da se loči proizvajanje in koriščenje informacij.

4. **Storitev (*angl. service*)** – Objavi/naroči model je sicer zelo prilagodljiv, njegova večkratna povezava (*angl. many-to-many*) in enosmerni transport pa vendarle nista primerna za interakcijo tipa zahtevaj/odgovori (*angl. request/reply*). Zato za tako vrsto komuniciranja med vozlišči uporabimo storitve. To poteka tako, da vozlišče, ki neko storitev ponuja, objavi ime storitve, vozlišče, ki pa to storitev potrebuje, pa pošlje zahtevek in čaka na odgovor.

5. **Imenik (*angl. master*)** – deluje kot imenska storitev (*angl. name-service*), saj omogoča registracijo in iskanje imen za celoten računski graf. Je pomemben del sistema, saj si brez njega vozlišča ne bi mogla izmenjevati sporočil in klicati storitev.

6. **Parametrski strežnik (*angl. parameter server*)** – je slovar, ki omogoča, da so podatki shranjeni v centralni lokaciji s pomočjo ključev. Preko njega lahko vozlišča med delovanjem dostopajo do nastavitvenih parametrov.

7. **Vreče (*angl. bags*)** – je format za shranjevanje in predvajanje sporočil *ROS*. So pomemben mehanizem za shranjevanje podatkov (npr. senzorskih). Prav tako so pomemben del pri razvoju in testiranju algoritmov, saj lahko določene senzorske in druge podatke shranimo ter jih nato ponovno predvajamo.



Slika 2.4: Osnovni koncept komunikacije znotraj *ROS-a* (povzeto po [14]).

2.4.4 Višje-nivojski koncepti

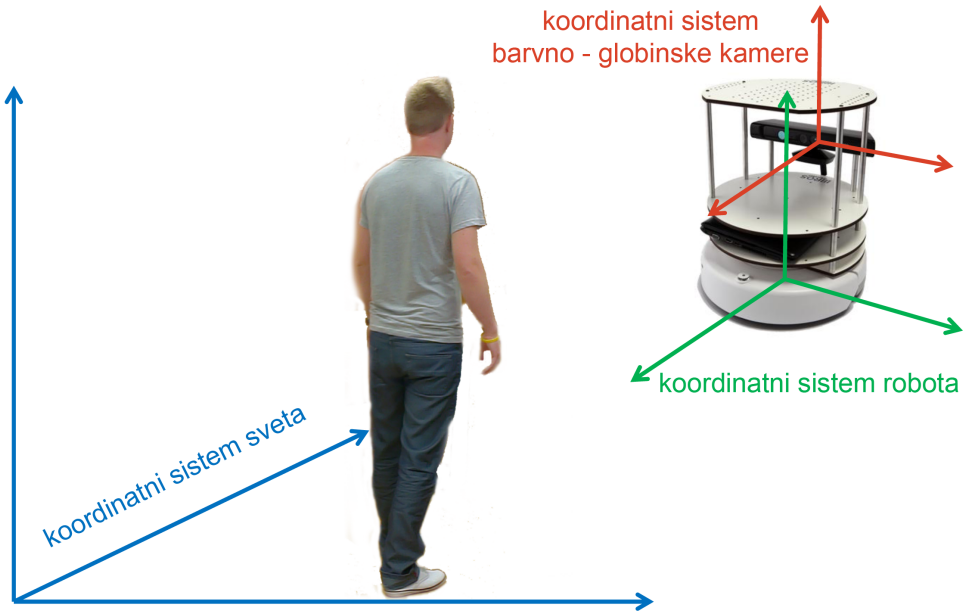
ROS vsebuje več višje-nivojskih konceptov, ki sestavljajo njegovo jedro. Vsebuje knjižnico za pretvarjanje med koordinatnimi sistemi skozi čas (***tf***), knjižnico za akcije in naloge (***actionlib***), ontologijo sporočil (***common_msgs***), vtičnike (***pluginlib***), filtre (***filters***) in knjižnico za opis robota v XML formatu (***urdf***).

Spodaj podajamo bolj podrobne opise treh knjižnic, ki smo jih v našem sistemu potrebovali najbolj pogosto.

2.4.4.1 Knjižnica Tf

Robotski sistem ima po navadi veliko *3D* koordinatnih sistemov (koordinatni sistem sveta, robota, glave ...), ki se spreminjajo skozi čas. Primer je pred-

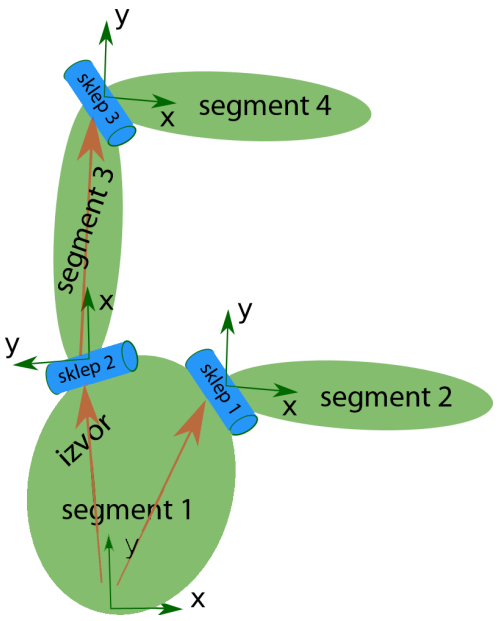
stavljen na Sliki 2.5. Paket *tf* uporabniku omogoča, da sledi več koordinatnim sistemom skozi čas in s tem obdrži razmerje med koordiniranem sistemom in drevesno strukturo. Tako lahko uporabnik sam, za katero koli točko v času, pretvori točke, vektorje ipd. med različnimi koordinatnimi sistemi. Tako lahko npr. pretvorimo lokalno (glede na pozicijo robota) zaznano točko kamere Kinect v globalni koordinatni sistem - nahajališče točke v svetu. Pretvorbo nam avtomatsko naredi knjižnica *tf*. *Tf* deluje v porazdeljenem sistemu, kar pomeni, da so informacije o koordinatnih okvirjih robota na voljo vsem komponentam ROS in vsakemu računalniku v sistemu. Ni centralnega strežnika za informacijo o transformaciji.



Slika 2.5: Primer različnih koordinatnih sistemov. (povzeto po [15])

2.4.4.2 Urdf/xacro

Enoten format za opis robota - *urdf* je format *XML* za predstavitev robotskega modela. Robot je predstavljen kot drevesna struktura, kjer so segmenti (*angl. links*) povezani s sklepi (*angl. joints*). Segmenti so namenjeni predstavitvi trdnih delov, kjer jim podamo različne lastnosti (vizualne, vztrajnostne in lastnosti trka), s sklepi pa definiramo kinematiko in dinamiko. Preprost model *urdf* je podan na Sliki 2.6. Poleg knjižnice *urdf* ima *ROS* tudi knjižnico *xacro*, ki je makro jezik *XML*, ki zna biti v primeru večjih dokumentov *XML* (npr. opisi robotov) zelo uporaben, saj lahko s pomočjo njega datoteke *urdf* znatno poenostavimo in jih naredimo lažje za razumevanje.



Slika 2.6: Prikaz preproste strukture robota *urdf,* kjer je robot predstavljen s segmenti in sklepi. (povzeto po [16])

2.4.4.3 Ontologija sporočil

Skupni sklad sporočil (*angl. common_ msgs*) nam omogoča osnovno ontologijo sporočil v robotskem sistemu. Definira več razredov sporočil, med drugimi tudi:

- sporočila, ki predstavljajo akcije - ***actionlib_ msgs,***
- sporočila za pošiljanje diagnostičnih podatkov - ***diagnostic_ msgs,***
- sporočila za predstavljanje skupnih geometričnih primitivov - ***geometry-msgs,***
- sporočila za navigacijo - ***nav_ msgs,***
- sporočila senzorjev - ***sensor_ msgs.***

2.5 Druge višje-nivojske knjižnice

Poleg osnovnih knjižnic *ROS* smo za delovanje našega sistema potrebovali še veliko drugih zunanjih knjižnic, ki so omogočale procesiranje oblaka točk in *2D* slik ter omogočale komunikacijo z robotom in s kamero *RGBD*.

2.5.1 OpenCV (Open Source Computer Vision)

OpenCV je knjižnica [17] za programiranje funkcij računalniškega vida v realnem času, ki jo je razvil Intel, zdaj pa jo podpirata Willow Garage [18] in Itseez [19]. Izdana je pod licenco *BSD* in je brezplačna tako za akademsko kot tudi za komercialno rabo. Sprva je bila napisana v jeziku C, od verzije 2.0 naprej

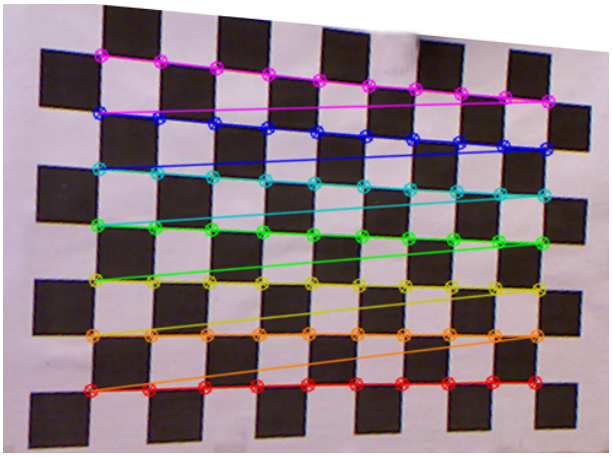
pa podpira tudi C++. Od meseca septembra leta 2010 je v razvoju in zdaj tudi že v uporabi vmesnik Cuda *GPE (grafično procesne enote)*, ki omogoča prenos procesiranja iz *CPE (centralno procesne enote)* na *GPE* in s tem tudi močno pohitritev nekaterih algoritmov. OpenCV se osredinja na mnoga področja, kot npr. interakcija človek-računalnik, mobilna robotika, prepoznava obraza, sledenje gibanja, segmentacija itd. Knjižnica je prav tako integrirana v *ROS*, ki omogoča pretvorbo slik iz formata *ROS* v format OpenCV.

2.5.2 Knjižnica openni_kinect

Knjižnica *openni_kinect* [20] omogoča integracijo Kinecta v *ROS*. Gonilnik nam omogoča zajem slike *RGB* in oblaka točk s pomočjo globinske informacije Kinecta. Za natančno ujemanje obeh slik je potrebna dodatna kalibracija Kinecta, saj osnovna tovarniška kalibracija marsikdaj ni dovolj natančna. Kalibracija poteka s pomočjo šahovnice, ki je prikazana na Sliki 2.7, in aplikacije *ROS*, ki na koncu poda in v Kinect shrani kalibrirane notranje (*angl. intrinsic*) parametre kamere Kinect.

2.5.3 Gonilniki Turtlebot

Za povezavo *ROS-a* in IRobot Roombe smo uporabili sklad Turtlebot [21] z manjšimi prilagoditvami opisne datoteke *urdf* in preostalih parametrov - hitrost prenosa podatkov, modela robota, žiroskopa itd. Na koncu smo dobili model, ki ustreza našemu robotu. Prikazan je na Sliki 2.8. S pomočjo gonilnikov lahko Roombi izdajamo različne ukaze, npr. premakni se na x, y

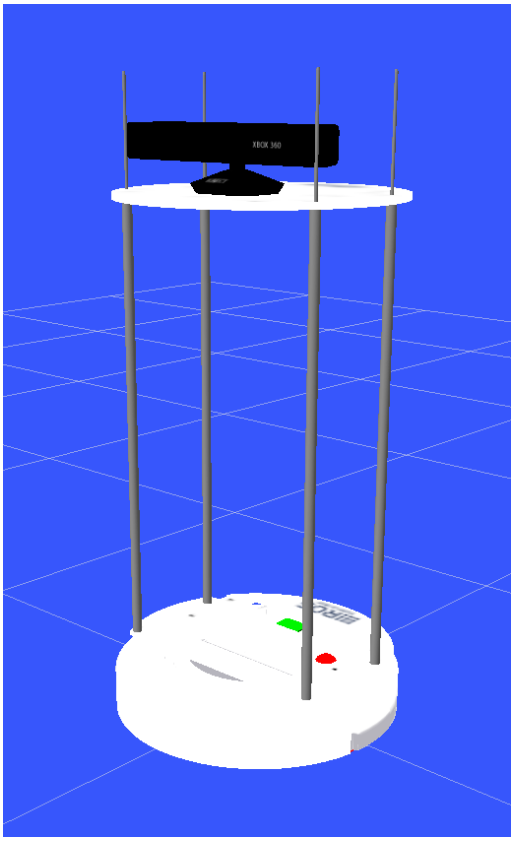


Slika 2.7: Vzorec šahovnice za kalibracijo Kinecta.

koordinato z določeno hitrostjo, in beremo podatke iz senzorjev (prednji odbijač, podatki motorjev koles itd.). Tako lahko Roombo npr. navigiramo po prostoru. Dajanje ukazov se izvrši z objavljanjem sporočila na določeno temo. Za branje s senzorjev se prijavimo na želeno temo, kjer so podatki objavljeni. Sklad vsebuje tudi model robota *urdf*, tako da lahko enostavno s pomočjo knjižnice tf pretvarjamo podatke med različnimi koordinatnimi sistemi.

2.5.4 PCL - Point Cloud Library

PCL je odprto-kodna knjižnica [22], ki omogoča procesiranje oblakov točk. Vsebuje mnoge najnovejše in najnaprednejše algoritme za filtriranje, segmentacijo, rekonstrukcijo modelov, ujemanje modelov kot tudi druga višje-nivojska orodja za modeliranje in prepoznavo objektov. Je primerna za različne platforme in je v uporabi tako na Linux, Mac, Windows in Android/iOS sistemih. Pomemben del knjižnice *PCL* je tudi njegova modularnost, saj tako omogoča uporabo tudi na sistemih z omejenimi viri. Je dobro integrirana v *ROS* in v



Slika 2.8: Model našega robota, zgrajenega s pomočjo *urdf* modela.

nekaterih primerih že kot vozlišča primerna za takojšnjo rabo. Izdana je pod licenco *BSD* in je brezplačna za komercialno in raziskovalno rabo. Denarno je podprta s strani velikih podjetjih, kot so Google [23], Toyota [24], NVidia [25], Willow Garage [18] itd.

2.6 Združitev platform za detekcijo in sledenje

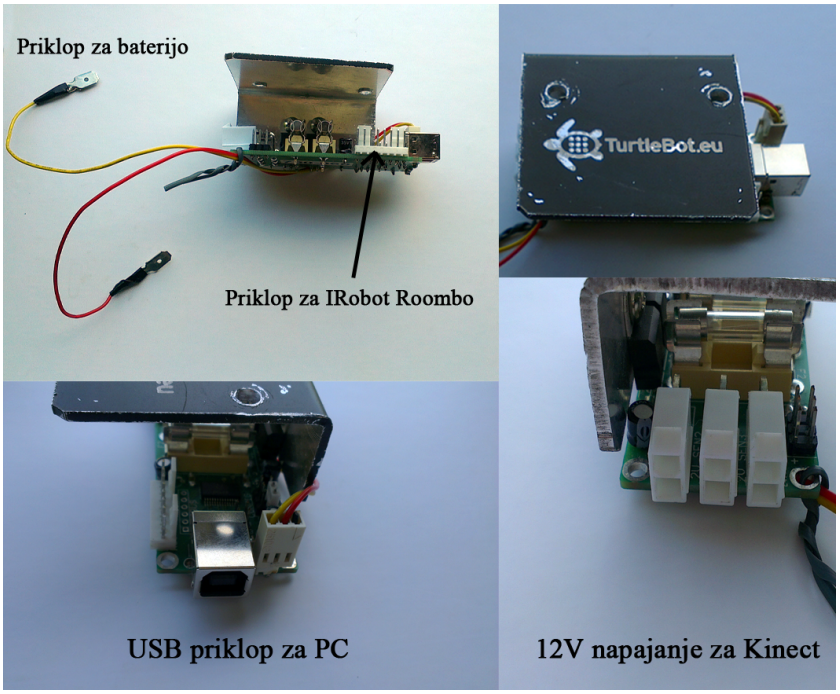
Za potrebe diplomskega dela smo vse zgoraj opisane elemente združili. IRobot Roombo smo prilagodili, tako da smo nanjo vpeli dodatna nosilca. Na zgor-

njega smo postavili Kinect, medtem ko smo na spodnjega postavili prenosni računalnik in akumulator, ki je napajal Kinect.

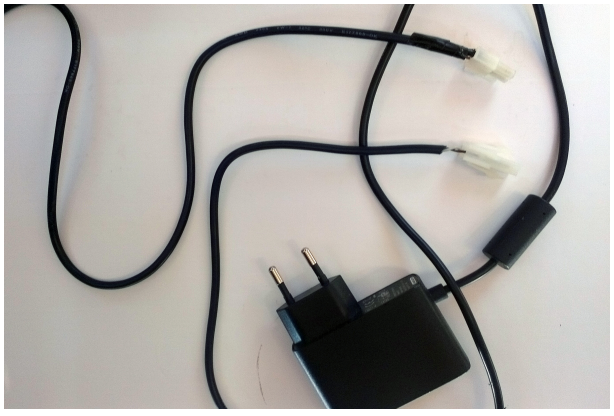
Roombo smo z računalnikom povezali preko posebnega vmesnika, ki skrbi za pretvorbo iz *7 nožnega mini-din* vmesnika v *usb*. Na vmesnik smo spojili še konektor, na katerega smo povezali 14.4 V akumulator. Za pretvorbo v 12 V, kolikor potrebuje Kinect, poskrbi vmesnik, ki je prikazan na Sliki 2.9.

Kinect je bil v računalnik priklopljen preko priključka usb, prilagoditi pa smo morali napajanje. Kot je prikazano na Sliki 2.10, smo originalni napajalni kabel, ki je primeren za 220 V vtičnice, prerezali in nanj zvezali dvonožni konektor. Tako smo ga povezali na že prej omenjeni vmesnik, ki skrbi za 12 V napajanje.

Programski del je bil v celoti implementiran v *ROS-u* v programskem jeziku *C++*. Tako smo vse elemente združili v enoten sistem in ga skozi celotno diplomsko delo uporabljali pri zaznavi in sledenju ljudem. Robotski sistem je prikazan na Sliki 2.11.



Slika 2.9: Vmesnik za komunikacijo z robotom



Slika 2.10: Prilagojeno napajanje Kinecta.

Slika 2.11: Celotni robotski sistem.

Poglavje 3

Programski del robotskega sistema

V tem poglavju vam bomo predstavili, na kakšen način smo se lotili reševanja problemov pri zaznavanju in sledenju človeka. V prvem delu so predstavljeni algoritmi, ki jih uporabljamo pri zaznavanju, medtem ko so v drugem delu predstavljeni algoritmi za sledenje.

3.1 Zaznavanje

3.1.1 Zmanjševanje ločljivosti oblaka točk

Iz barvno - globinske kamere Kinect dobimo oblak točk v polni ločljivosti. Zaradi zahteve po hitrem delovanju celotnega sistema smo ločljivosti oblaka točk zmanjšali do te mere, da je detekcija še vedno dobra, a hkrati pohitrili nadaljnje procesiranje oblaka. Za zmanjšanje števila točk smo uporabili knjižnico ***PCL* in njen *filter VoxelGrid***.

Algoritem čez celoten oblak točk, ki ga prejmemo iz Kinecta, položi *3D* mrežo

slikovnih elementov (angl. *voxels*), kjer velikost posameznega slikovnega elementa določimo sami. Nato vzame točke, ki so znotraj enega slikovnega elementa, in jih aproksimira z njihovim centroidom. Tako za vsak slikovni element dobimo eno samo točko. Prav tako filter zavrže vse neveljavne točke (*angl. nan*). To so točke, katerih pozicija zaradi napak v meritvah ni izmerljiva. Slabost uporabe filtra je, da izgubimo sosednost točk v oblaku, kar pomeni upočasnitev nadaljnjih algoritmov, kjer je sosednost pomembna, npr. pri pozneje opisanem gručenju točk.

3.1.2 Odstranjevanje tal

Tla so pri detekciji nepomembna in so moteč faktor. Zaradi njihove značilne planarne oblike in velikosti je odstranitev hitra in zelo natančna. S tem tudi pohitrimo nadaljnje procesiranje, saj se znebimo velikega dela oblaka točk. Tla zaznamo z algoritmom RANSAC.

RANSAC sta prva formalizirala Fischler in Bolles [26]. To je iterativna metoda, ki iz opazovanih točk oceni parametre matematičnega modela. Je **nedeterminističen algoritem**, zato je njegov rezultat pogojen z določeno verjetnostjo. Več iteracij kot dovolimo, bolj bo rezultat točen. Primer preprostega ujemanja modela na opazovani množici točk je podan na Sliki 3.1.
Algoritem na vhod dobi opazovano množico točk in na njej iterativno izvaja sledeč postopek:

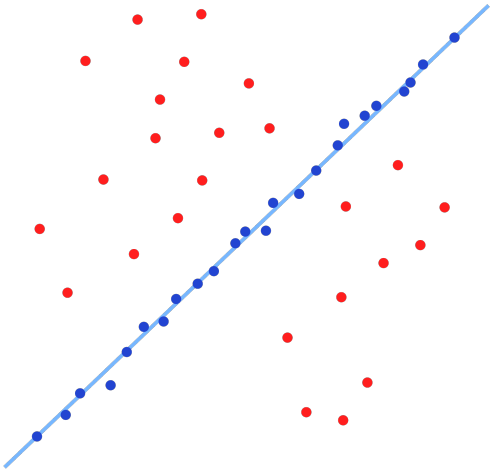
- Iz opazovane množice naključno izbere podmnožico točk, ki vsebuje vsaj n točk.

- Določi parametre modela, ki ustrezajo izbranim točkam.
- Vse preostale točke iz opazovane množice točk preizkuša, ali se prilegajo trenutnemu modelu. Torej točke, ki so znotraj praga t, pripiše trenutnemu modelu.
- V primeru, da trenutnemu modelu pripiše vsaj d točk, ga oceni še na množici vseh opazovanih točk in mu določi napako e.
- Postopek ponovi $k - krat$.

V vsakem koraku dobi model, ki je zavrnjen zaradi vsebovanja premajhnega števila točk ali sprejet z vsebujočo napako e. Po koncu vseh iteracij izbere tisti model z najmanjšo napako e.
Tako v našem primeru dobimo enačbo ravnine oblike: $ax + by + cz + d = 0$, ki jo nato lahko preprosto odstranimo iz oblaka točk. Algoritem je vgrajen v že omenjeno knjižnico PCL.

3.1.3 Gručenje

Namen gručenja je, da **podobne točke združi v gručo**. V našem primeru je primerno, da se jih združuje glede na njihovo **evklidsko razdaljo**. Za učinkovito implementacijo je v tem primeru točke najbolje urediti v tako podatkovno strukturo, ki omogoča hitro iskanje s pomočjo večdimenzionalnih ključev. Ta podatkovna struktura je **k-d drevo**. To je binarno drevo, kjer je vsako vozlišče k-dimenzionalna točka. V drevesu objekte organiziramo s pomočjo razpolavljanja prostora po dimenzijah.

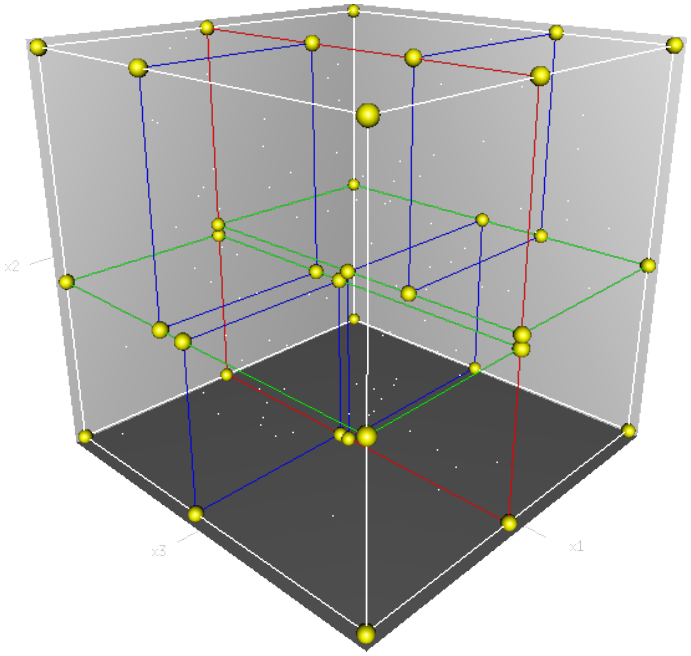


Slika 3.1: Primer ujemanja modela, v tem primeru premice, na opazovani množici točk. [27]

V našem primeru smo uporabili **3-d drevo**, saj imamo točke podane v 3 dimenzijah(x, y, z). Primer 3-d drevesa je prikazan na Sliki 3.2.
Za implementacijo gručenja smo uporabili knjižnico *PCL*, predvsem razred ***EuclideanClusterExtraction***. Razred gruči oblak točk glede na evklidsko razdaljo, ob tem pa upošteva tolerančno razdaljo, pod katero naj točke pripadajo isti gruči.

3.1.4 Ovrednotenje gruč

Ko dobimo gruče objektov, ki smo jih zaznali, lahko nekatere glede na naše predznanje izločimo. Med gručami iščemo tiste, ki naj bi bile ljudje. Vse gruče, ki ne ustrezajo človeškim dimenzijam (višina, velikost), ali pa so predaleč od robota, zavržemo. Zaradi merske napake Kinecta je detekcija objekta dlje od 6 m od robota zelo nenatančna [29].



Slika 3.2: Grafični prikaz 3-d drevesa, kjer so vozlišča predstavljena s kroglami, različne barve povezav pa predstavljajo dimenzije. [28]

Ker naš oblak točk z zmanjšano ločljivostjo nima barvne informacije in zaradi zmanjševanja ločljivosti tudi ni več urejen, dobimo *2D RGB* skupek gruče s spodaj opisanim postopkom.

Iz vsake točke iz gruče s pomočjo notranjih parametrov Kinecta in principa točkovne kamere dobimo pozicijo točke na sliki *RGB*. Za to pretvorbo uporabimo funkcijo (***project3dToPixel***) iz knjižnice OpenCV, ki naredi pretvorbo točke iz *3D* koordinat v *2D* koordinate.

Zaradi zmanjšane ločljivosti oblaka točk ne dobimo dovolj točk, da bi lahko tvorile celotni *2D* skupek. Zaradi tega dobimo med točkami praznino, ki jo je

potrebno zapolniti. To naredimo v dveh korakih:

- **dilatacija točk (*angl. dilation*)** - To storimo tako, da na sliko implici-ramo jedro določene velikosti in kvadratne oblike. Določimo še sidrno točko jedra. Jedro postavljamo čez sliko in vsakič vzamemo točko z največjo vrednostjo. Nato vrednost sidrne točke v jedru zamenjamo s to maksimalno vrednostjo. Za večji učinek ta postopek večkrat ponovimo. Tako zapolnimo vse vmesne točke, a hkrati tudi zaznano regijo preveč razširimo, tako da vsebuje še del ozadja.

- **erozija točk (*angl. erosion*) -** Da se znebimo prej omenjenega odvečnega ozadja, izvedemo obratno funkcijo - erozijo. Deluje po podobnem principu kot dilatacija, le da izračuna lokalni minimum znotraj jedra. Nato sidrno točko zamenjamo s to minimalno vrednostjo.

3.1.5 Prepoznava gruče kot človeka

Preverbo, ali je gruča človek, naredimo s klasičnim *2D* detektorjem ljudi – *HOG*, ki sta ga v svojem članku opisala Dalal in Triggs [3].
Osnovna ideja *HOG-a* je, da je navadno informacija lokalne oblike dobro opisana z **distribucijo intenzitetnih gradientov ali s smermi robov**. Opis je prav tako dober, tudi če nimamo natančno določenih lokacij robov.
Kratek opis algoritma je sledeč. Sliko razdelimo na **manjše dele - celice**, ki so lahko kvadratne (*R-HOG*) ali okrogle (*C-HOG*) oblike. Za vsako izračunamo histogram orientacij robov. Na koncu vse histograme združimo in dobimo **vektor značilnic - opisnik**, ki opisuje sliko. Z vektorji značilnic nato naučimo

algoritem nadzorovanega strojnega učenja, **linearni klasifikator *SVM***.
V splošnem je *HOG* **časovno zelo požrešen algoritem**, a se v našem primeru zelo pohitri. Navadno se *HOG* računa na celotni sliki, kar zahteva veliko časa. Pohitritev naredimo s tem, ko mu kot vhod za računanje opisnika ne podamo celotne *RGB* slike, temveč le **posamezen *RGB* skupek gruče**. Za klasifikacijo, ali je *RGB* skupek res človek, uporabimo že naučeni linearni klasifikator *SVM* iz knjižnice OpenCV.

3.2 Sledenje

Sledenje določeni zaznavi je pomemben del celotnega sistema, saj nam lahko celoten sistem dodobra pohitri in izboljša njegovo natančnost. Za dosego teh ciljev smo uporabili sledeče algoritme.

3.2.1 Adaboost

3.2.1.1 Nadzorovano strojno učenje

Algoritmi nadzorovanega strojnega učenja iščejo hipotezo, ki nam vhodne podatke, npr. vektor $x = [x_1, x_2 ..., x_n]$, klasificirajo v enega izmed razredov $y \epsilon \{c_1, c_2 ..., c_m\}$. Algoritem je najprej potrebno naučiti na množici učnih primerov, ki vsebuje že znane pare $\{(x_1, y_1), (x_2, y_2) ..., (x_L, y_L)\}$. V našem primeru imamo **binarno klasifikacijo**, saj imamo le dva klasifikacijska razreda – človek, ni človek.

3.2.1.2 Osnovni opis algoritma Adaboost

Adaboost je algoritem strojnega učenja, ki sta ga formulirala Yoav Freund in Robert Schapire [30]. Uporablja se skupaj z drugimi učnimi algoritmi. Na splošno lahko z Boosting algoritmi izboljšamo točnost učnih algoritmov. Delujejo po principu **združevanja šibkih klasifikatorjev v enega močnega**. Boosting algoritmi iterativno učijo šibke klasifikatorje in jih dodajajo k močnemu. V vsaki iteraciji se ustvari nov šibek klasifikator in posodobi distribucija uteži, ki kaže na pomembnost primerov.

Uteži se povečajo primeru, ki je bil napačno klasificiran, in znižajo primeru, ki je bil pravilno klasificiran. Tako se novi klasifikator bolj osredini na primere, ki jih je prejšnji klasifikator napačno klasificiral.

Na koncu tako dobljeno množico hipotez združimo z uteženim glasovanjem v eno močno.

Boosting algoritmi se uporabljajo v mnogih primerih, kot npr. v prepoznavi besedila, pri filtriranju besedila, učnih problemih, procesiranju naravnega jezika, zdravstveni diagnostiki itd. Definiramo dva klasifikatorja:

- **Šibek klasifikator** h^{weak} - mora klasificirati le malo boljše kot naključna izbira, da izboljša končni močni klasifikator. Tako npr. pri binarnem klasifikatorju napaka ne sme biti večja od 0.5.

- **Močan klasifikator** h^{strong} **-** je linearna kombinacija šibkih klasifikatorjev. Njegova napoved je uteženo glasovanje N šibkih klasifikatorjev, kjer je a_n glasovalna utež $n - tega$ šibkega klasifikatorja. Definiran je z enačbama (3.1) in (3.2).

$$h^{strong}(x) = sign(conf(x)) \tag{3.1}$$

$$conf(x) = \sum_{n=1}^{N} \alpha_n \cdot h_n^{weak}(x) \tag{3.2}$$

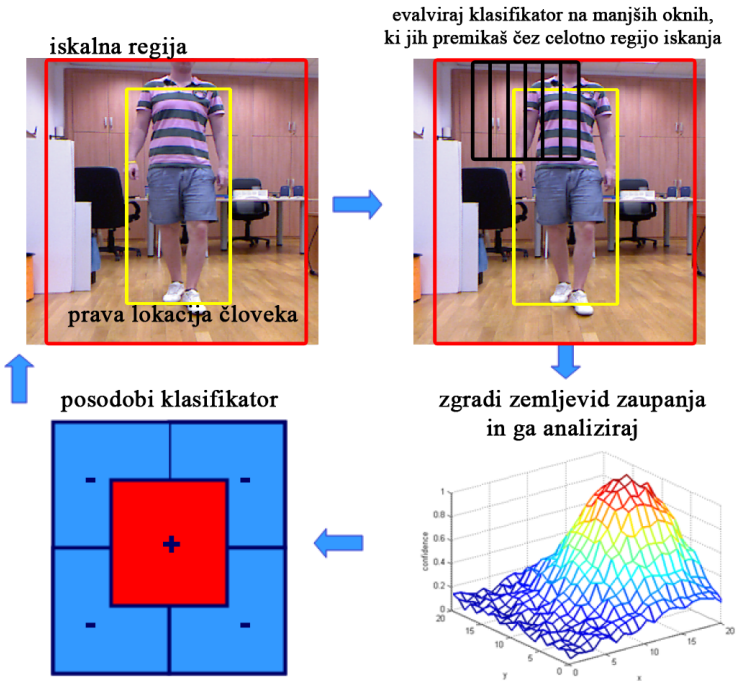
Osnovni algoritem deluje sledeče. Imamo učno množico z negativno in s pozitivno označenimi primeri (3.3)

$$X = \{(x_1, y_1), ..., (x_L, y_L) \mid x_i \epsilon \mathbb{R}^m, y_i \epsilon \{-1, +1\}\} \tag{3.3}$$

in začetno uniformno porazdelitvijo primerov $p(x_i) = \frac{1}{L}$. Na tej množici uporabimo učni algoritem, tako da pridobimo šibek klasifikator – h_n^{weak}. Glede na njegovo napako e_n mu priredimo glasovalno utež α_n, ki je podana z enačbo (3.4).

$$\alpha_n = \frac{1}{2} \cdot \ln(\frac{1 - e_n}{e_n}). \tag{3.4}$$

Porazdelitev $p(x)$ pa posodobimo tako, da verjetnost naraste pri negativno klasificiranih primerih, medtem ko pri pozitivno klasificiranih verjetnost pade. Tako se algoritem osredini na težje primere. Ta postopek iterativno ponavljamo, dokler ne dosežemo nekega ustavitvenega pogoja (npr. doseženo število naučenih šibkih klasifikatorjev). Na Sliki 3.3 je prikazan postopek posodabljanja klasifikatorja.



Slika 3.3: Prikaz delovanja Adaboost algoritma (povzeto po [31]).

3.2.1.3 Inkrementalni Adaboost

Pri klasičnih boosting algoritmih je potrebno klasifikator najprej naučiti in šele nato s pomočjo njega klasificirati primere. V našem primeru nam to preveč ne koristi, saj se tako ozadje kot tudi same poze človeka stalno spreminjajo. Prav tako spremembe osvetljenosti negativno vplivajo na klasifikacijo. Zaradi teh težav smo se v diplomskem delu odločili uporabiti inkrementalni algoritem Adaboost, ki so jih v svojih člankih opisali M.Grabner, H.Grabner in Bischof [31, 32].

Pri tem boosting algoritmu učimo klasifikator sproti, ko nove informacije – detekcije postajajo na voljo. Tak klasifikator se je zmožen prilagajati stalnim

spremembam ozadja, osvetljenosti in pozi človeka. Definiramo selektor:

- **Selektor** n **-** Iz množice šibkih klasifikatorjev s hipotezami $H^{weak} = \{h_1^{weak} ..., h_M^{weak}\}$ selektor izbere natanko enega h_m^{weak} in sicer tistega z minimalno napako e, kot je prikazano z enačbami (3.5, 3.6, 3.7).

$$arg\, min_m(e_{n,m}) \tag{3.5}$$

$$e_{n,m} = \frac{\lambda_{n,m}^{wrong}}{\lambda_{n,m}^{corr} + \lambda_{n,m}^{wrong}} \tag{3.6}$$

$$h_n^{sel}(x) = h_m^{weak}(x) \tag{3.7}$$

  Selektor lahko interpretiramo tudi kot klasifikator. Učenje selektorja pomeni, da učimo (posodobimo) vsak šibek klasifikator in izberemo tistega z najmanjšo napako. Podobno kot pri klasičnem Adaboost-u H_{weak} (selektor) ustreza značilnicam, torej generirana hipoteza bazira na odzivu značilnic. Selektor torej lahko izbira iz podmnožice vseh značilnic.

Glavna ideja je aplicirati inkrementalni boosting ne direktno na šibke klasifikatorje, temveč na selektorje. Torej vsakič, ko dobimo nov učni primer (x, y), posodobimo selektorje upoštevajoč pomembnostno utež primera - λ. Ta utež je hitrost učenja ali k-krat posodobljena Poissonova porazdelitev.
Za posodobitev šibkih klasifikatorjev in gradnjo novih hipotez lahko uporabimo kateri koli inkrementalni učni algoritem. Ta postopek naredimo za vse selektorje in na koncu z linearno kombinacijo selektorjev dobimo močni klasifikator

(3.8), ki je za razliko od klasičnega adaboost-a na voljo kadar koli.

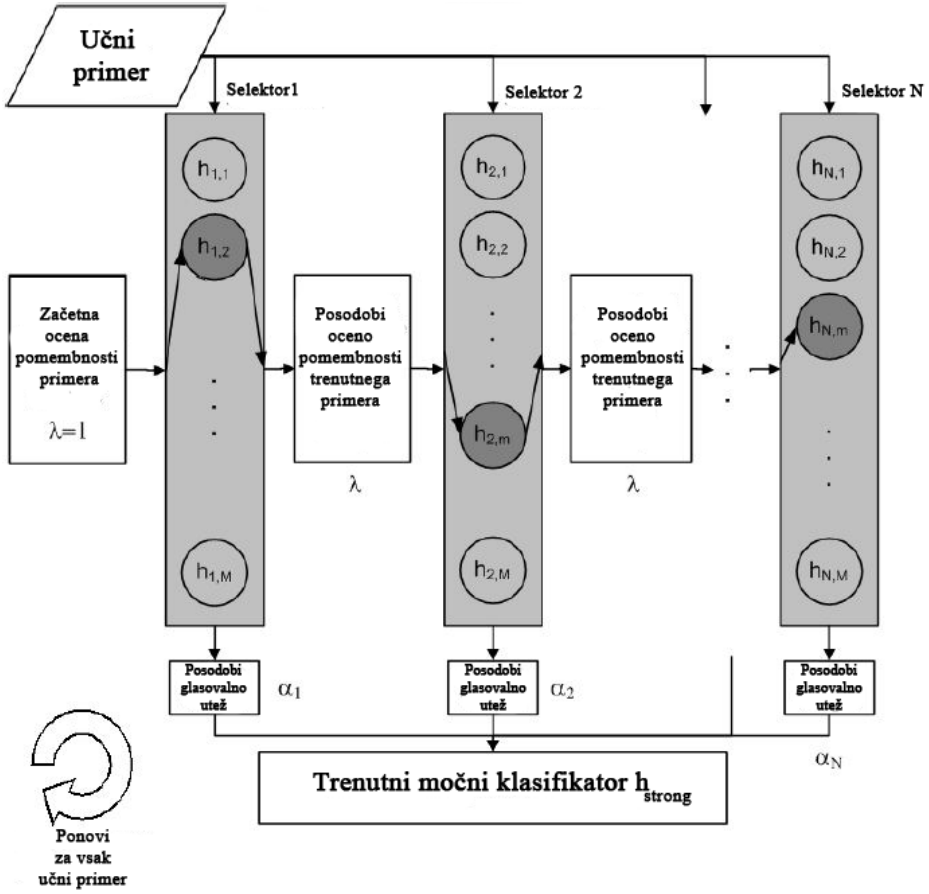
$$h^{strong}(x) = sign(\sum_{n=1}^{N} \alpha_n \cdot h_n^{sel}(x)) \tag{3.8}$$

Prednost Adaboosta je, da ni tako dovzeten za prekomerno prileganje, a je hkrati občutljiv na šumne podatke. Freund in Schapire sta dokazala, da napaka algoritma Adaboost eksponentno pada glede na število šibkih klasifikatorjev.

V sistem *ROS* smo vgradili odprtokodno knjižnico Adaboost [33], ki implementira algoritem iz že prej omenjenih člankov [31, 32]. Na Sliki 3.4 je prikazan postopek inkrementalnega algoritma Adaboost .

3.2.2 Kalmanov filter

Za sledenje se v zadnjem času uporablja filter z delci (*angl. Particle filters*) kot tudi različice Kalmanovega filtra. V znanstvenem članku [34] Bellott in Hu opisujeta prednosti in slabosti večih različic Kalmanovega in Filtra z delci. Glede na njune rezultate smo se za napovedovanje pozicije človeka odločili za uporabo različice Kalmanovega filtra, *Unscented Kalman filter* (*UKF*). Za naš primer, sledenje človeka, je *UKF* dovolj točen in robusten, a le za odtenek slabši od Filtra z delci. Kljub temu, da je Filter z delci v splošnem boljši, so za izbiro *UKF-ja* pretehtali predvsem preprostost hitrost in dovolj velika natančnost pri sledenju človeka.



Slika 3.4: Postopek inkrementalnega Adaboost algoritma (povzeto po [31]).

3.2.2.1 Kaj je Kalmanov filter?

Povzeto po članku [35] je Kalmanov filter sklad matematičnih funkcij, ki omogoča učinkovito rekurzivno ocenjevanje stanja na način, da minimizira srednjo vrednost kvadratne napake. Je zelo mogočen v aspektih napovedovanja preteklih, sedanjih in prihodnih stanj. Kalmanov filter je sestavljen iz dveh korakov:

- **Opazovanje -** Skozi čas opazuje serijo meritev. Meritve lahko vsebujejo tudi šum in druge nenatančnosti.

- **Napoved -** Na podlagi modela gibanja, preteklih pozicij in hitrosti Kalmanov filter poda napoved oz. približek neznanke v naslednjem časovnem koraku, upoštevajoč šum. V našem primeru je ta neznanka lokacija človeka.

Zaradi iterativne narave algoritma lahko teče v realnem času, poleg tega pa potrebuje le trenutno meritev in prejšnje preračunano stanje. Tako ne potrebujemo nobene pretekle informacije.

Kalmanov filter se uporablja v več tehnologijah in primerih, npr. v navigaciji, nadzoru vozil, kot so letala, vesoljska plovila, procesiranje signalov itd.

3.2.2.2 Dinamičen model filtra

Ker je osnovni Kalmanov filter omejen na linearne napovedi, smo se glede na rezultate v članku [34] odločili za *UKF*. Tako lahko dobro napovedujemo tudi bolj kompleksne modele gibanja, kot je človeško. Kot je omenjeno v članku [34], je za opis človeškega gibanja dovolj dober **konstanten model gibanja**. Spodnji model gibanja človeka je povzet po [10].

Skozi opazovanje nas zanimata pozicija človeka (x, y) kot tudi njegova hitrost $(\dot{x}, \dot{y})$. Torej tako lahko definiramo vektor x_k (3.9), ki opisuje, kakšno $k-to$ stanje bo filter ocenil.

$$x_k = (x,\, y,\, \dot{x},\, \dot{y}) \tag{3.9}$$

Vsakič, ko prejmemo novo zaznavo, moramo filter posodobiti. Torej je naslednji vektor, ki ga moramo definirati, **vektor opazovanja** (3.10).

$$z_k = (x,\, y)^T \tag{3.10}$$

Kot smo že omenili, privzamemo konstanten model gibanja človeka. Iz te predpostavke sklepamo, da se med dvema zaznavama človek v obeh smereh giblje s konstantnim, z normalno porazdeljenim pospeškom, s srednjo vrednostjo 0 in z varianco σ_a^2.
Tako lahko stanje x_k definiramo tudi z enačbama (3.11) in (3.12), kjer je:

- x_{k-1} - prejšnje stanje,
- F - matrika, ki pove, kako iz prejšnjega stanja pridemo v novo,
- G in a_k - sta matriki, ki sta vhodna podatka,
- Δt - sprememba časa med dvema Kalmanovima napovedima,
- $\ddot{x}-$pospešek v smeri koordinate x,
- $\ddot{y}-$pospešek v smeri koordinate y.

$$x_k = F \cdot x_{k-1} + G \cdot a_k \tag{3.11}$$

$$F = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},\ G = \begin{pmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix},\ a_k = \begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix} \tag{3.12}$$

x_k lahko zapišemo tudi z enačbo (3.13), kjer je šum w_k porazdeljen z normalno multivariantno porazdelitvijo z ničelno srednjo vrednostjo in s kovarianco. $Q\,(w_k \sim N\,(0,\,Q))$. Q je definiran z enačbo (3.14)

$$x_k = F \cdot x_{k-1} + w_k \tag{3.13}$$

$$Q = G^T G \cdot \Sigma_a = \begin{pmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix} \cdot \begin{pmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix}^T \cdot \begin{pmatrix} \sigma_{ax}^2 \\ \sigma_{ay}^2 \end{pmatrix} \tag{3.14}$$

Zdaj, ko imamo definiran model, ki nam opisuje trenutno stanje, definiramo še **model opazovanja sistema** z enačbo (3.15) in (3.16), kjer je:

- H matrika, ki nam opisuje, kaj merimo, torej pozicijo človeka (x, y),

- R_k kovariančna matrika šuma meritev, kjer je σ_v^2 **kvantizacijska napaka** zaradi zmanjševanja ločljivosti oblaka točk in je definirana z enačbo (3.17), σ_d^2 je definiran kot **napaka globinskega senzorja** Kinecta, ki je glede na članek [36] enaka kvadratu razdalje. Napaka je prikazana na Sliki 3.5.

$$z_k = H \cdot x_k + v_k \tag{3.15}$$

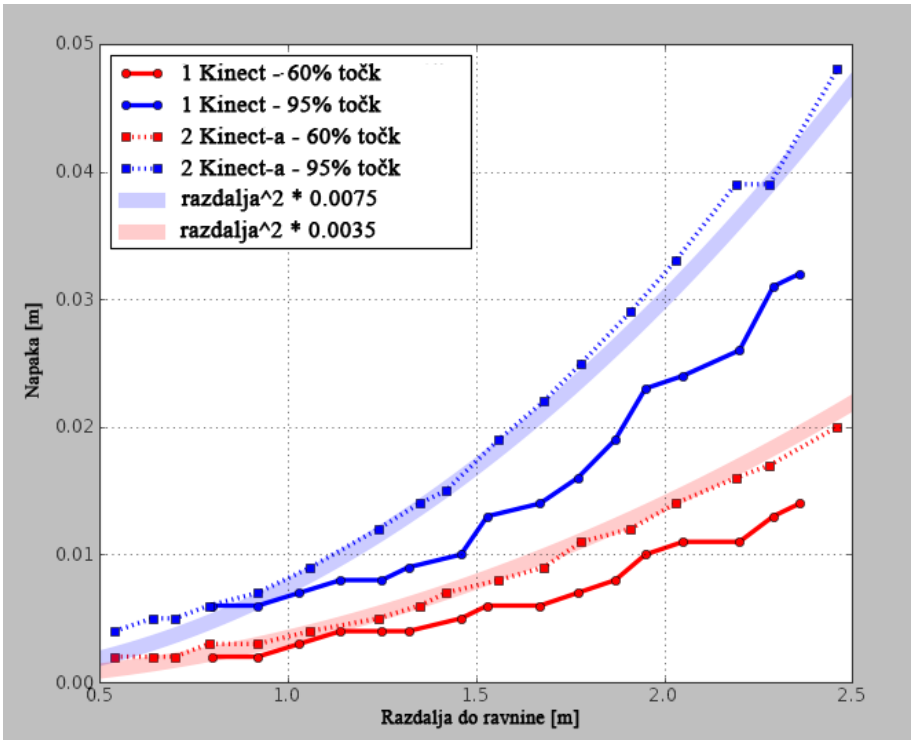
$$\begin{cases} v_k \sim N(0,\, R_k) \\ H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \\ R_k = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot (\sigma_v^2 + \sigma_d^2) \end{cases} \tag{3.16}$$

$$\sigma_v^2 = \frac{velikost\_voksla^2}{12} \tag{3.17}$$

Tako na koncu, ko združimo model opazovanja in model trenutnega stanja, dobimo **celoten dinamičen model filtra (3.18).**

$$\begin{cases} x_k = F \cdot x_{k-1} + w_k \\ z_k = H \cdot x_k + v_k \end{cases} \tag{3.18}$$

Za realizacijo *UKF-ja* smo uporabili odprtokodno knjižnico Bayesovih filtrov - Bayes++ [37], ki smo jo integrirali v naš ROS sistem.



Slika 3.5: Prikaz napake barvno-globinskega senzorja Kinect, ki je enaka kvadratu razdalje (povzeto po [36]).

Poglavje 4

Opis delovanja sistema

V tem poglavju bomo opisali, kako smo se iz analize problema iz prejšnjih poglavij lotili reševanja problema detekcije in zaznave človeka na mobilni platformi z barvno-globinsko kamero. Opisane posamezne dele in algoritme iz prejšnjega poglavja smo uporabili in jih povezali v enoten sistem, ki nam omogoča reševanje zastavljenega problema diplomskega dela.

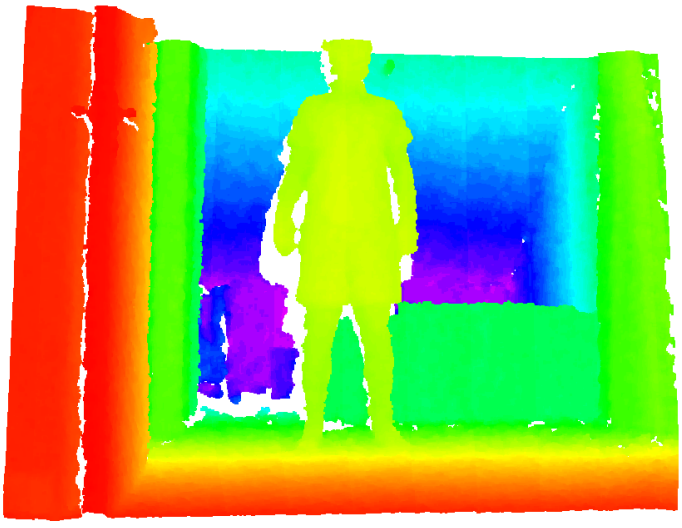
V prvem delu bomo opisali, kako algoritem človeka zazna, v drugem, kako zaznanemu človeku sledi, in v zadnjem, tretjem delu, kako s pomočjo *ROS*-ovega navigacijskega sklada robot sledi človeku skozi prostor.

4.1 Detekcija

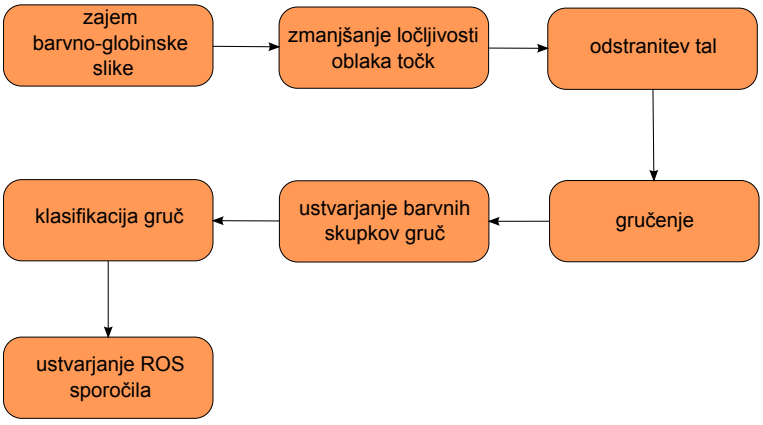
Za detekcijo smo v *ROS*-u implementirali paket *Detection*, katerega naloga je, da iz Kinecta zajema barvno sliko (prikazano na Sliki 4.1), oblak točk (primer prikazan na Sliki 4.2) in podatke o kameri. Te podatke ustrezno obdela in na

temi *Detections* objavi zaznane detekcije. Postopek je prikazan na Sliki 4.3.

Slika 4.1: Prikaz barvne slike, ki jo zajema Kinect.

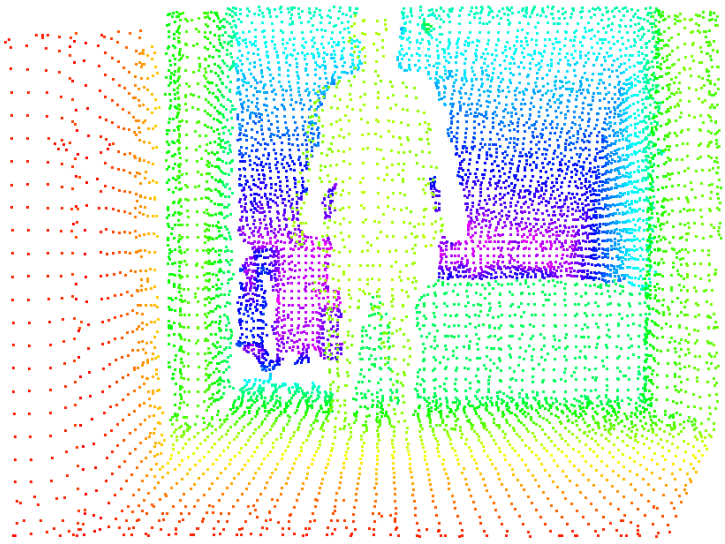


Slika 4.2: Prikaz oblaka točk v polni ločljivosti, kjer temno rdeča barva predstavlja točke, najbližje senzorju, temno vijolična barva pa točke, najdlje od senzorja.



Slika 4.3: Postopek detekcije človeka.

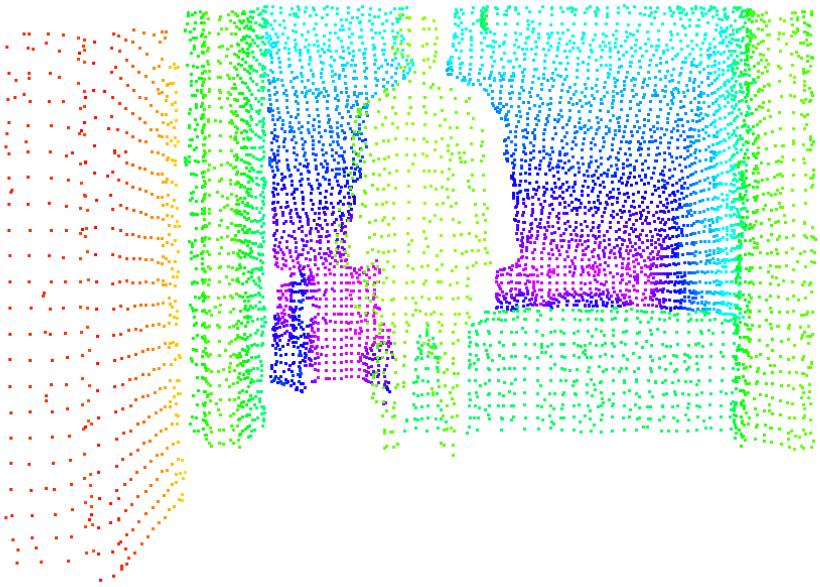
S Kinecta dobimo oblak točk, ki mu zaradi hitrejšega nadaljnjega obdelovanja zmanjšamo ločljivost. Primer oblaka točk z zmanjšano ločljivostjo je prikazan na Sliki 4.4. Sledi odstranjevanje tal s pomočjo algoritma segmenta-



Slika 4.4: Prikaz oblaka točk z zmanjšano ločljivostjo.

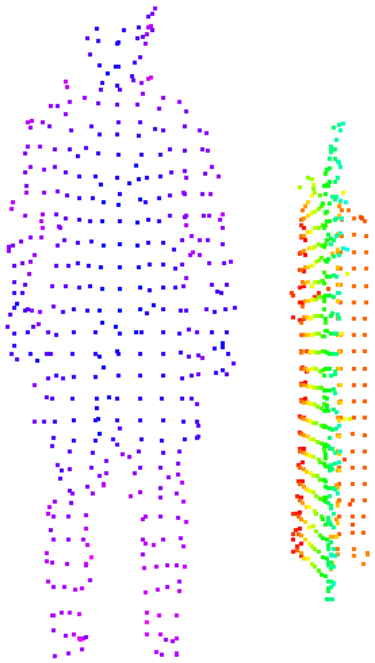
cije in *RANSAC-a*. Zaradi večje natančnosti se oblak najprej razdeli na dva dela. Spodnji del, ki vsebuje tla, in zgornji del, ki vsebuje preostali oblak točk.

Algoritem za odstranitev tal poženemo le na spodnjem delu. Tako zmanjšamo možnosti za napako in postopek pohitrimo. Na koncu postopka oba, zgornjega in spodnjega, oblaka točk ponovno združimo v enega.
Tako iz oblaka točk preprosto odstranimo velik del točk, kar olajša in pohitri nadaljnje procesiranje. Na Sliki 4.5 je prikazan oblak točk z odstranjenimi tlemi.



Slika 4.5: Prikaz oblaka točk po odstranitvi tal.

S pomočjo gručenja točke združujemo glede na njihovo evklidsko razdaljo. Tako že dobimo prve skupke objektov, med katerimi naj bi se nahajal tudi človek. Prikaz gruč po gručenju je prikazan na Sliki 4.6.



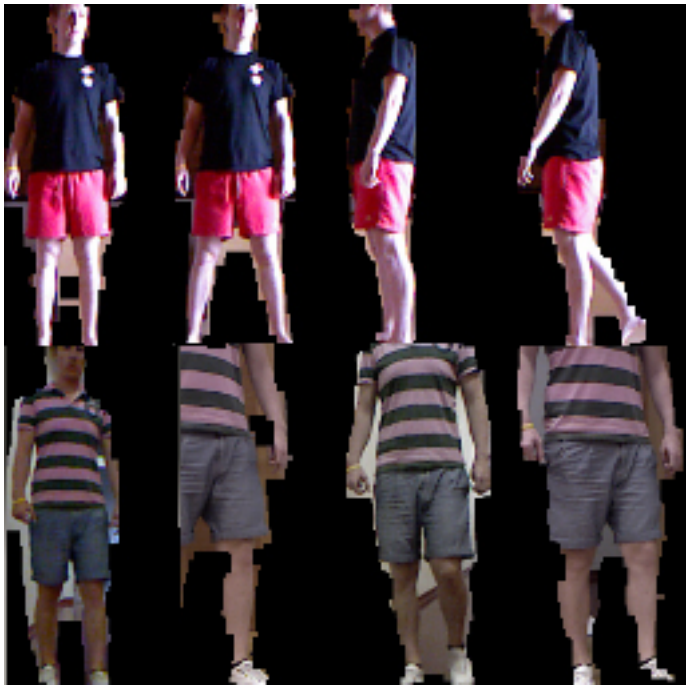
Slika 4.6: Prikaz gruč po postopku gručenja in selekcije. Na sliki sta vidni dve gruči točk.

Dobljene gruče je potrebno preveriti - ali je katera izmed njih človek, saj algoritem vrže gruče, ki so lahko ali pa niso človek. Na vsaki dobljeni gruči algoritem opravi sledeč postopek:

- vsak **slikovni element v gruči preslika v točko** na prazno *RGB* sliko (skupek) in mu priredi belo barvo,
- s pomočjo **dilatacije in erozije** *2D* točke na sliki zgosti,
- glede na Kinectovo barvno sliko, ki mora biti sinhrona z oblakom točk, **določi vsem točkam v 2D skupku ustrezno barvo,**

- tako dobljene skupke gruč nato **klasificira s pomočjo *HOG-a*** in za sledenje človeka uporabi tisto, ki ima največje zaupanje.

RGB slika skupka gruče je velikosti $64\,x\,128$ točk, enako kot učni primeri, na katerih je bil naučen *HOG* klasifikator. Prikaz primerov pozitivno klasificiranih gruč je prikazan na Sliki 4.7, medtem ko so negativno klasificirani primeri prikazani na Sliki 4.8.



Slika 4.7: Barvni skupki pozitivno klasificiranih gruč.

Zaradi sprememb okolja, svetlobe in poze se marsikdaj zgodi, da klasifikator *HOG* pozitivno klasificira kakšno gručo, ki ni človek. Zaradi pogostih sprememb v notranjem okolju smo uvedli še dodaten klasifikator – **inkrementalni Adaboost**. Ob prvi pozitivni detekciji *HOG-a* poženemo še inicializacijo Adaboost-a. Tako klasifikator naučimo, kakšen objekt naj pozitivno klasificira

Slika 4.8: Barvni skupki negativno klasificiranih gruč.

in mu sledi. Tekom sledenja klasifikator posodabljamo tako z negativnimi kot s pozitivnimi primeri in ga na tak način stalno izboljšujemo in delamo vedno bolj robustnega.

Pred ustvarjanjem sporočila za paket sledenja moramo izračunati še **višino človeka, 3D centroid pozitivno klasificiranega skupka** in **pozicijo ter velikost kvadra** (*angl. bounding box*), znotraj katerega je človek. Tako na koncu ustvarimo sporočilo, ki vsebuje sledeče podatke:

- oddaljenost človeka od robota,
- višino človeka,

- zaupanje zaznanega skupka,
- lokacijo in velikost kvadra, ki vsebuje človeka,
- x, y, z lokacijo centroida človeka.

4.2 Sledenje

Za potrebe sledenja smo napisali **paket ROS Tracking**, s katerim se prijavimo na temo *Detections*. Preko te teme dobimo vse potrebne informacije za sledenje zaznanemu človeku.

Ko imamo zaznano lokacijo človeka v času t, se nam s tem nadaljnja zaznava olajša, tako da **predvidevamo, da se človek v času $t + 1$ ne bo premaknil veliko**. Zaradi tega lahko območje naslednje zaznave zožimo. Tako v naslednjem koraku ni potrebe po procesiranju celotnega oblaka točk, temveč le majhne okolice prejšnje zaznave. Celotno **izvajanje se pohitri in postane bolj robustno**.

Za sledenje smo prvotno vzeli kar največjo gručo znotraj zmanjšanega okna. Ta princip deluje zelo robustno v primeru, ko je robot stalno obrnjen v smeri prejšnje detekcije. Vendar smo po testiranjih sledenja robota s pomočjo navigacijskega sklada ROS ugotovili, da robot velikokrat ni obrnjen proti človeku in tako kaj hitro zamenja človeka za kakšno drugo gručo znotraj okna.

Tako smo se odločili za manj hitro, a toliko bolj robustno rešitev. Za sledenje vzamemo tisto gručo, ki jo **klasifikator Adaboost najbolje klasificira** znotraj že prej omenjenega zmanjšanega okna.

Lahko se zgodi, da človeka ne zaznamo v omenjenem oknu zaradi razlogov, kot so:

- v oknu ne zaznamo gruč,
- negativna adaboost klasifikacija vseh najdenih gruč znotraj okna.

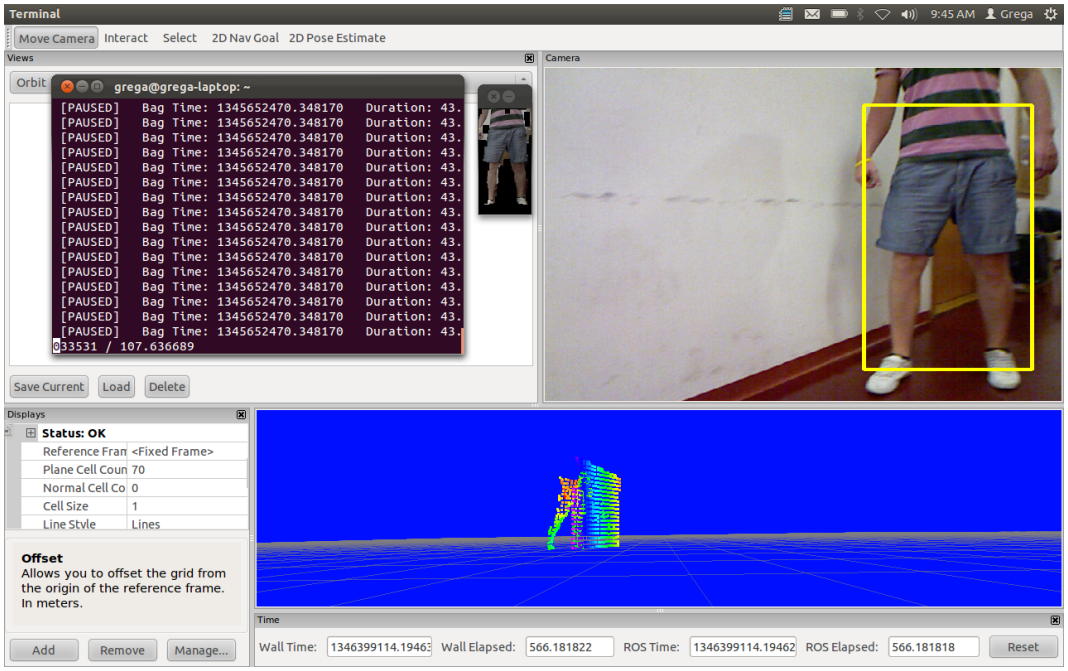
V takih primerih **okno iskanja postopoma povečujemo,** dokler ponovno ne zaznamo človeka oz. smo iskanje ponovno razširili na celoten oblak točk.
V primeru, ko človek izgine in ga večkrat ponovno ne zaznamo, začne robot slediti pozicijam, ki jih napove Kalmanov filter. Tak primer nastopi, ko npr. človek zavije skozi vrata ali okoli ovinka. Zaradi specifičnosti Kinecta mora biti človek od robota oddaljen vsaj $1.8m$. Do teh rezultatov smo prišli pri testiranju sistema. Zaradi tako velike oddaljenosti od robota robot hitro izgubi človeka iz svojega vidnega polja, zato se je primoran zanašati na te napovedi.

4.3 Navigacija

Za navigacijo po prostoru smo uporabili **sklad *ROS* »navigation stack«**, ki že vsebuje vse potrebne mehanizme za navigacijo po prostoru kot tudi izogibanje oviram.
Navigacijski sklad sprejema informacije pozicije in senzorjev robota ter izdaja robotu ukaze za premik. Na poti do cilja se s pomočjo planiranja poskuša izogniti oviram in kar najbolj **optimalno prispeti na cilj**.
Pozicije zaznav iz koordinatnega sistema Kinecta s pomočjo tf knjižnice **pretvorimo v svetovni koordinatni sistem**. Vse pozicije zaznav kot tudi na-

povedi Kalmanovega filtra shranimo v listo pozicij človeka. S pomočjo ločene niti in zanke se konstantno sprehajamo skozi osveženo listo pozicij človeka in izdajamo robotu preko navigacijskega sklada ukaze za premik na določeno x, y koordinato v svetu.

Tako robot sledi sprotnim zaznavam sistema in se poskuša izogniti oviram na poti, kot so stene, podboji ipd. Na Sliki 4.9 je prikazan primer delovanja sistema v primeru, ko je človek s telesom obrnjen proti robotu.



Slika 4.9: Prikaz sledenja in navigacije robota v programskem okolju *rviz*, kjer je objekt, ki mu sledimo, označen z rumenim pravokotnikom, medtem ko so gruče, ki jih sistem zazna, prikazane v spodnjem - modrem delu.

Poglavje 5

Ovrednotenje sistema

V petem poglavju bomo predstavili ovrednotenje celotnega sistema na treh različno zahtevnih poligonih in kvalitativno ocenili uspešnost detekcije in sledenja človeka. Na vsakem poligonu smo detekcijo in sledenje ponovili 12-krat in zapisali, kolikrat je bil sistem uspešen in neuspešen ter zakaj. Na vseh poligonih se je človek premikal počasi s spreminjajočo se hitrostjo.

5.1 Preprost poligon

Za preprost poligon smo v našem primeru izbrali dolg hodnik s postavljenimi ovirami, kot so npr. stoli in mize. Poligon je prikazan na Sliki 5.1.
Človek se je gibal vijugajoče v obliki črke S z neostrimi in s počasnimi zavoji. Osvetlitev je bila konstantna s pomočjo luči na hodniku. Na preprostem poligonu je robotski sistem vsakič pravilno zaznal in sledil človeku čez celoten hodnik. V tem primeru je bila uspešnost našega sistema 100 %, kot je

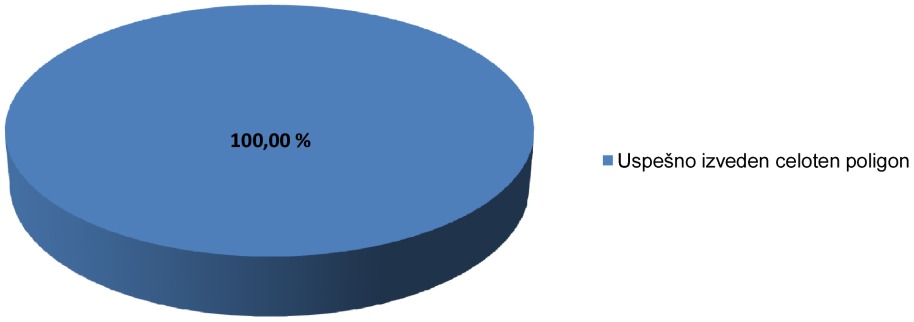
prikazano na Sliki 5.2.

Slika 5.1: Prikaz preprostega poligona - dolgega hodnika.

5.2 Srednje zahteven poligon

Srednje zahteven poligon je vseboval sledeče korake:

- vožnja skozi sobo z dvema srednjima ostrima zavojema,
- sledenje skozi vrata in nato spust po manjši klančini,
- oster zavoj in nato nadaljevanje s preprostejšim sledenjem po hodniku z neostrimi zavoji.



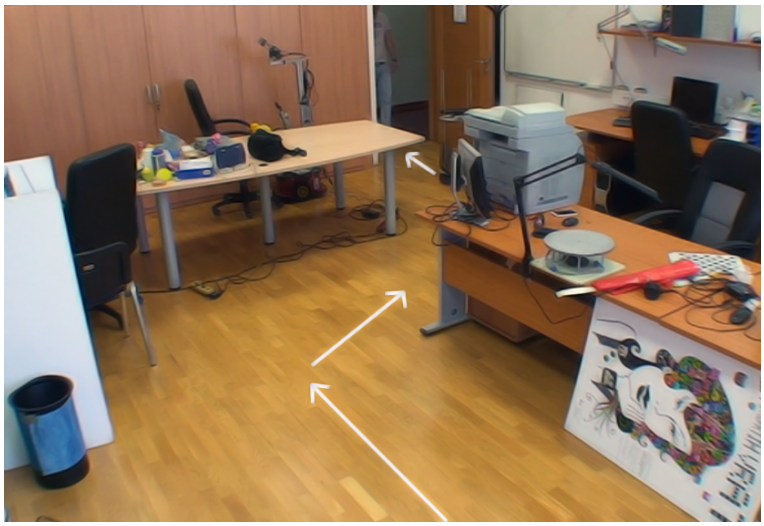
Slika 5.2: Grafični prikaz uspešnosti robotskega sistema na preprostem poligonu.

Osvetlitev je bila spreminjajoča. Iz dobro osvetljene sobe se je robotski sistem premaknil v slabše osvetljen hodnik. Na Slikah 5.3, 5.4 in 5.5 je prikazana pot robota po poligonu.

Sistem je od dvanajstih ponovitev skozi celoten poligon sledil 8-krat, kar je 66,7 % uspešnost sistema. Štiri nepravilna delovanja sistema je povzročilo:

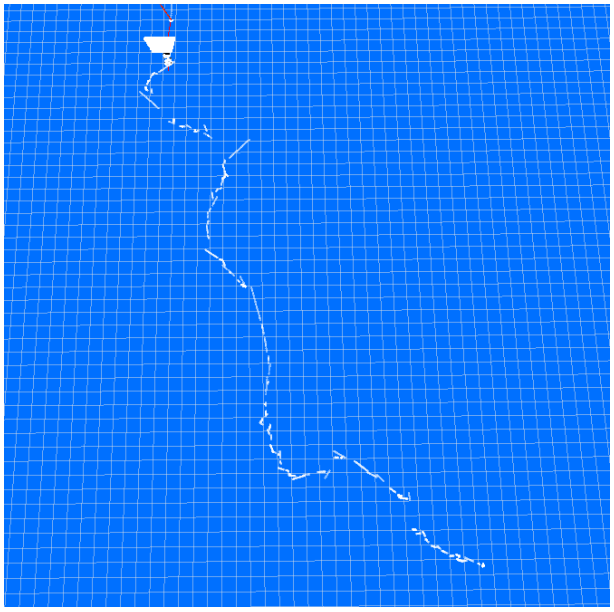
- 2-krat zatik robota pri prehodu iz klančine na ravna tla hodnika, kar je posledica slabše narejene klančine,
- 2-krat napačna klasifikacija inkrementalnega Adaboost algoritma zaradi izgube človeka iz vidnega polja.

Grafični prikaz uspešnosti je predstavljen na Sliki 5.6.

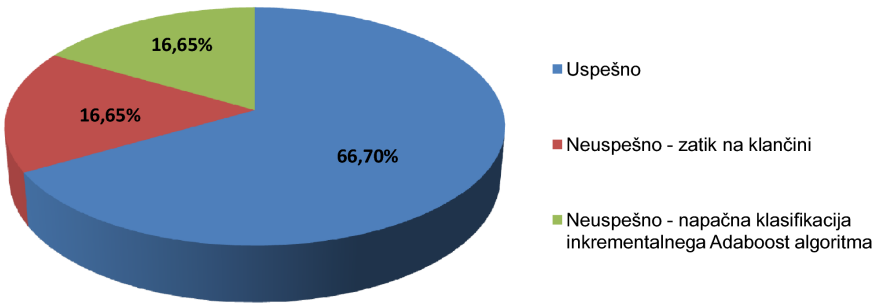


Slika 5.3: Prikaz prvega dela srednje zahtevnega poligona - sobe.

Slika 5.4: Prikaz drugega dela srednje zahtevnega poligona - hodnik.



Slika 5.5: Prikaz poti robota v programu *rviz* čez srednje zahteven poligon. Pot je označena z belo barvo.



Slika 5.6: Grafični prikaz uspešnosti sistema na srednje zahtevnem poligonu.

5.3 Zahteven poligon

Zahteven poligon smo sestavili iz sledečih korakov:

- krajše sledenje po hodniku z neostrimi zavoji,
- oster zavoj skozi vrata v učilnico in krajše sledenje po ravnini,
- oster zavoj v ozko območje med klopi in krajše sledenje po ravnini med klopmi.

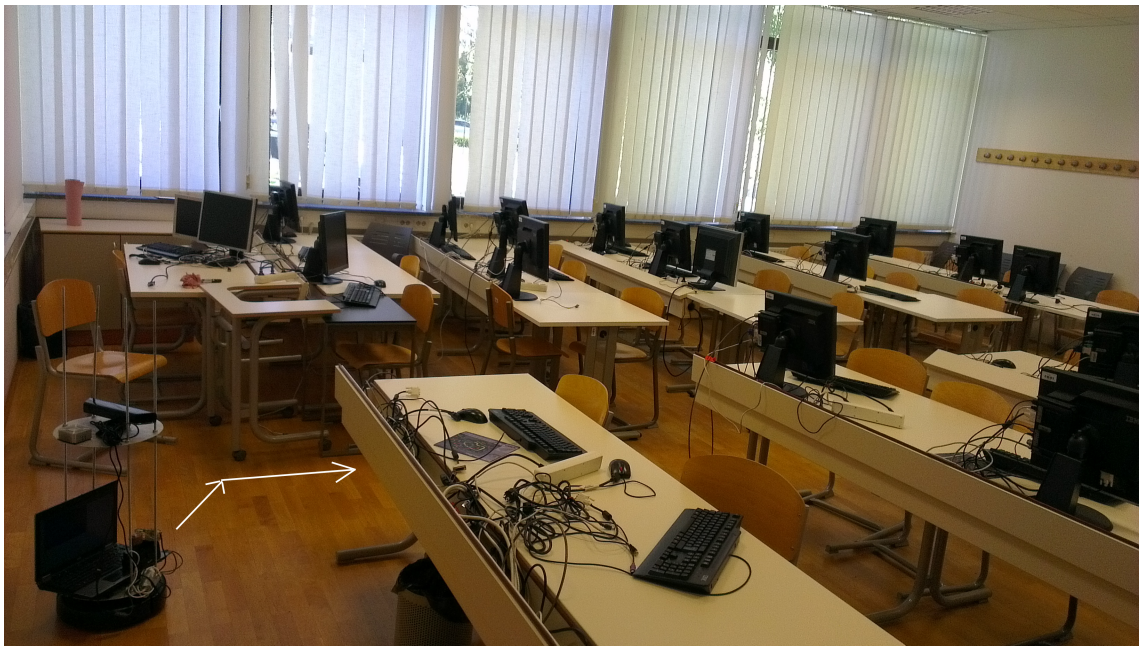
Osvetlitev je bila tudi tu spreminjajoča. Iz slabše osvetljenega hodnika je sistem sledil v boljše osvetljeno učilnico. Na Slikah 5.7 in 5.8 je prikazana pot, ki jo je moral robotski sistem narediti.
Skozi dvanajst ponovitev smo z robotskim sistemom izmerili sledeče rezultate:

- 4-krat je uspešno deloval skozi celotno pot,
- 4-krat je uspešno detektiral in sledil skozi prvi oster zavoj v učilnico in sledil po ravnini. Nato nepravilno izvedel drugi ostri zavoj in izgubil človeka. Vzrok je nepravilna klasifikacija inkrementalnega Adaboost algoritma zaradi izgube človeka iz vidnega polja,
- 4-krat je sledil le po hodniku in nepravilno začel delovati pri prvem ostrem zavoju v učilnico. Vzrok je nepravilna klasifikacija inkrementalnega Adaboost algoritma zaradi izgube človeka iz vidnega polja.

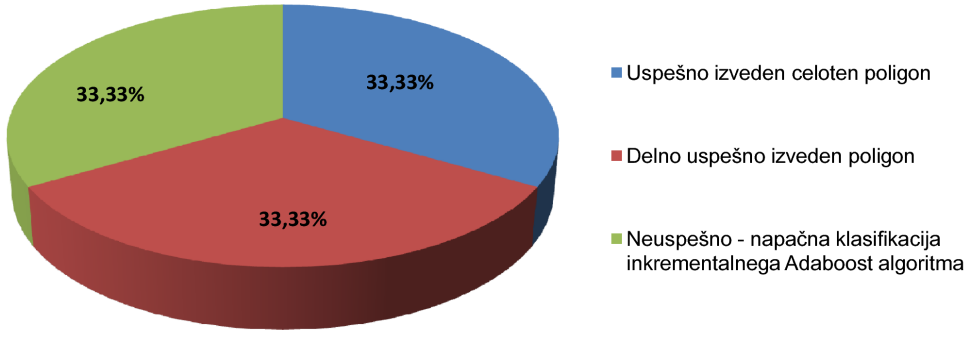
Tako lahko povzamemo, da je pri zahtevnem poligonu naš sistem v 33 % popolnoma deloval skozi celotno pot, v 33 % je sistem deloval do polovice poti

in v 33 % se je ustavil že pri prvem težjem zavoju. Grafični prikaz uspešnosti sistema je prikazan na Sliki 5.2.

Slika 5.7: Prikaz prvega dela zahtevnega poligona - hodnika, ki mu sledi oster zavoj skozi vrata v učilnico.



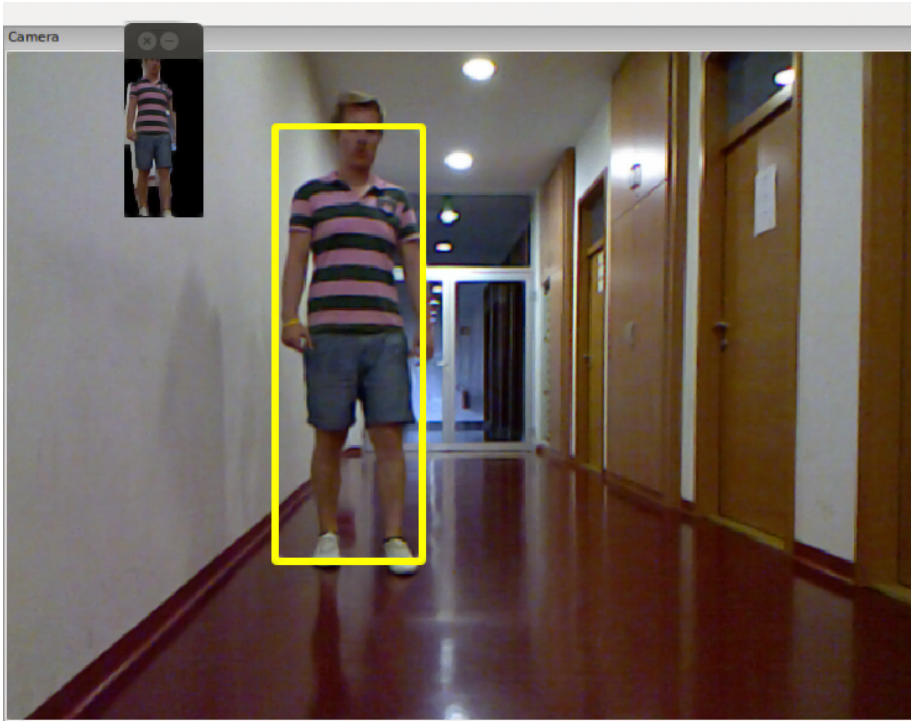
Slika 5.8: Prikaz drugega dela zahtevnega poligona - učilnica, ki mu sledi oster zavoj in sledenje med klopmi.



Slika 5.9: Grafični prikaz uspešnosti sistema na zahtevnem poligonu.

5.4 Diskusija

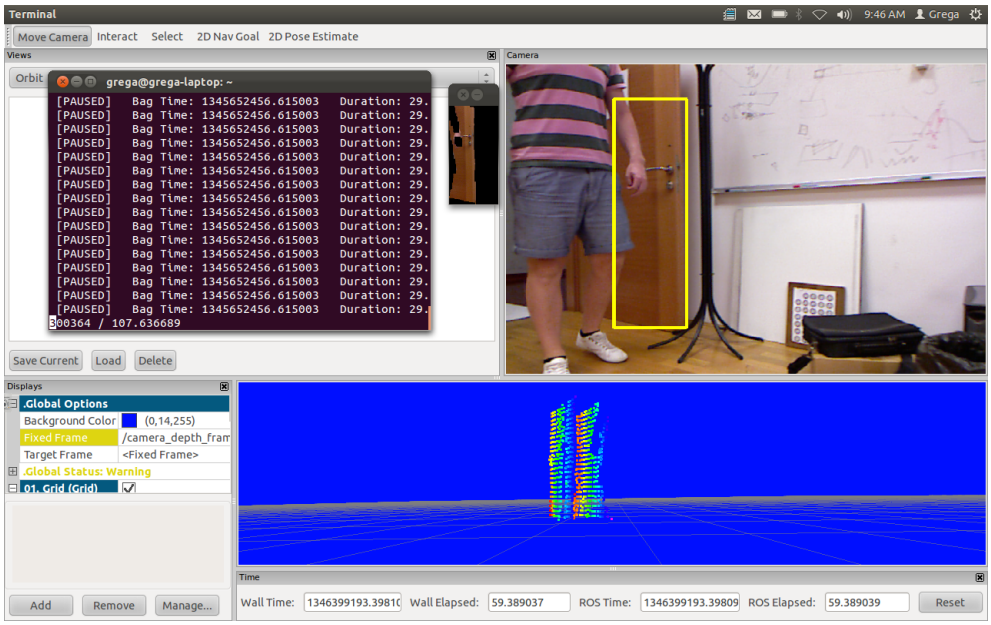
Sistem v dobrih svetlobnih pogojih, kjer človek ne zavija ostro, odlično detektira in sledi človeku. Kljub stalnemu spreminjanju smeri in hitrosti človeka se sistem ne zmoti in mu robustno sledi v realnem času. Tudi v ožjih prostorih sistem deluje brez težav. Na Sliki 5.10 je prikazan primer pravilnega delovanja sistema.



Slika 5.10: Prikazan je primer pravilnega detektiranja in sledenja človeka. Pozitivno klasificiran skupek je prikazan v ločenem oknu in na sliki obrobljen z rumenim okvirjem.

Kljub odličnim rezultatom po preprostem poligonu ima sistem težave pri težjem poligonu. Težave dela predvsem ostro spreminjanje smeri sledečega človeka, saj ga zaradi ozkega vidnega polja barvno-globinske kamere hitro izgubi.

Za pravilno detekcijo je v našem primeru uporaben le $40 - 45$ stopinjski horizontalni vidni kot Kinect-a, saj v nasprotnem primeru ne dobimo dovolj točk za združevanje v gruče. Kljub temu, da robot ob izgubi človeka sledi napovedanim pozicijam Kalmanovega filtra, se lahko zgodi, da inkrementalni Adaboost pozitivno klasificira napačno gručo. Primer nepravilne pozitivne klasifikacije je prikazan na Sliki 5.11. Tako začne robot slediti drugemu objektu, kar povzroči neuspešno nadaljnjo detekcijo.



Slika 5.11: Prikaz napačne pozitivne klasifikacije inkrementalnega Adaboost algoritma - pozitivno klasificiranih vrat.

Cilj celotnega sistema je bilo zaznavanje in sledenje človeka s pomočjo barvno-globinske kamere na mobilni platformi v realnem času. Sistem deluje tako v primeru, ko je človek s telesom obrnjen proti robotu, kot tudi, ko je obrnjen od robota. Pri tem dosegamo nekje do 5 sličic na sekundo, kar je

povsem zadovoljivo za sledenje človeku pri počasni hoji. V primeru, da za ponovno zaznavo ne uporabljamo Adaboost algoritma in za pravilno privzamemo le največjo gručo, se hitrost dvigne tudi do 10 sličic na sekundo. Pri tem naj omenimo, da uporabljamo srednje zmogljiv prenosni računalnik z dvo-jedrnim 2.4 GHz procesorjem, Intel Core 2 Duo P8400, 4 GB DDR3 pomnilnika in diskom *SSD*. Sistem učinkovito sledi človeku tudi v primeru, ko le-ta izgine iz vidnega polja ali ko zavije skozi vrata.

Pohitritev sistema bi bila vsekakor mogoča predvsem z uporabo implementacij procesorsko potratnih funkcij na *GPE* kot tudi z uporabo novejših več-jedrnih *CPE*. Tako je npr. klasifikator *HOG* znotraj knjižnice OpenCV že implementiran na *GPE*, a ga zaradi neprimerne grafične kartice nismo uporabljali. S tako optimizacijo bi se krepko približali hitrosti 20-30 sličic na sekundo za celoten sistem, kar bi pripomoglo k še večji robustnosti sistema.

Poglavje 6

Povzetek

Problem detekcije in sledenja človeka z mobilno platformo in barvno - globinsko kamero, smo razdelili na tri podprobleme - detekcijo, sledenje in navigacijo po prostoru. Tako smo s pomočjo barvno-globinske kamere Kinect zajemali tako *2D* kot *3D* sliko. V *ROS-u* smo napisali algoritme, ki so s pomočjo programskih knjižnic človeka uspešno detektirali in mu sledili. Vse skupaj smo povezali z mobilno platformo IRobot Roombo, ki je služila za sledenje človeka skozi prostor. Robotski sistem uspešno detektira in sledi človeku v realnem času.

Seveda sistem še zdaleč ne deluje idealno. Problem vidimo predvsem v omejitvah barvno-globinskega senzorja Kinect, kjer moramo biti za dobro zaznavo od robota oddaljeni vsaj 1.8 m in ne več kot 4 m. Te razdalje smo določili z empiričnimi poskusi. Tako je prostor gibanja človeka precej omejen. Problem predstavljajo tudi nagli zavoji človeka, saj kaj hitro izgine iz vidnega polja kamere. Kljub temu, da za primere, ko nas kamera ne vidi, uporabljamo

napovedi gibanja Kalmanovega filtra, je mnogokrat napačna klasifikacija inkrementalnega Adaboost algoritma vzrok za napačno sledenje. Tako za učinkovito sledenje in manevriranje potrebujemo kar velik prostor in ne preveč naglo gibanje človeka.

Globinski senzor Kinect deluje odlično, saj tudi v slabši svetlobi in večji oddaljenosti dobimo dobre gruče, a jih sistem velikokrat nepravilno klasificira tako z algoritmom *HOG* kot inkrementalnim Adaboost klasifikatorjem. Tu težave povzročajo slaba osvetljenost, premikanje in tresenje robota. Tako velikokrat dobimo deloma razmazano sliko, ki jo klasifikator ne klasificira pravilno. Z boljšim barvnim senzorjem, ki bi zajemal boljše slike pri slabi svetlobi, bi zaznavo naredili še bolj robustno.

V sistemu uporabljamo že naučeni klasifikator *HOG*. Klasifikator je bil naučen na učni množici ljudi v zunanjem okolju pod boljšimi svetlobnimi pogoji. Tako smo morali mejo zaupanja klasifikacije spustiti nižje, kar je prineslo sicer boljšo klasifikacijo človeka, a s tem tudi več nepravilno pozitivno klasificiranih gruč. Tu bi lahko klasifikacijo znatno izboljšali s tem, da bi klasifikator naučili na lastni učni množici v notranjem okolju.

Nekaj težav se pojavi tudi pri uporabi IRobot Roombe, saj je zaradi teže, ki je na robota naložena, z dnom zelo nizko pri tleh. Originalne Roombine vzmeti niso namenjene vsej tej teži, zato se kaj hitro zatakne v rahlo neravna tla, kar povzroči napake v lokalizaciji robota v prostoru in s tem tudi nepravilno navigacijo.

Kljub mnogim omejitev sistema nam je uspelo doseči zastavljeni cilj - detekcija

in sledenje človeka z barvno-globinsko kamero na mobilni platformi. Tako nam robotski sistem robustno zazna človeka in mu učinkovito sledi po prostoru.

Literatura

[1] T. Sonoura, T. Yoshimi, M. Nishiyama, H. Nakamoto, S. Tokura, in N. Matsuhira, *Person Following Robot with Vision-based and Sensor Fusion Tracking Algorithm*, Japonska, 2008.

[2] J. Satake in J. Miura, "Robust stereo-based person detection and tracking for a person following robot," v *Proceedings of the IEEE ICRA 2009, Workshop on People Detection and Tracking*, Kobe, Japonska, maj 2009.

[3] N. Dalal in B. Triggs, "Histograms of oriented gradients for human detection," v *International Conference on Computer Vision & Pattern Recognition*, zv. 2, USA, jun 2005, str. 886–893.

[4] K. Levi in Y. Weiss, "Learning object detection from a small number of examples: the importance of good features," v *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2004*, zv. 2, Washington, DC, ZDA, 2004, str. 53–60.

[5] D. G. Lowe, "Object recognition from local scale-invariant features," v *Proceedings of the Seventh IEEE International Conference on Computer Vision*, zv. 2, Los Alamitos, CA, ZDA, avg 1999, str. 1150–1157.

[6] L. Spinello in K. O. Arras, "People detection in RGB-D data," v *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, ZDA, 2011, str. 3838–3843.

[7] L. Spinello, M. Luber, in K. O. Arras, "Tracking people in 3D using a bottom-up top-down detector," v *Proc. IEEE International Conference on Robotics and Automation (ICRA'11)*, Shanghai, Kitajska, 2011.

[8] L. Spinello, K. O. Arras, R. Triebel, in R. Siegwart, "A layered approach to people detection in 3[d] range data," v *Proc. 24th AAAI Conference on Artificial Intelligence, PGAI Track (AAAI'10)*, Atlanta, ZDA, 2010.

[9] E. A. Topp in H. I. Christensen, "Tracking for following and passing persons," v *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05)*, 2005, str. 70–76.

[10] F. Basso, "RGB-D people tracking by detection for a mobile robot, magistrsko delo," Universita degli Studi di Padova, 2011.

[11] Wikipedia. Irobot roomba - wikipedia. (10. 8. 2012). Dostopno na: http://en.wikipedia.org/wiki/Roomba

[12] ——. Microsoft kinect. (10. 8. 2012). Dostopno na: http://en.wikipedia.org/wiki/Kinect

[13] J. MacCormick. How does kinect work? (10. 8. 2012). Dostopno na: http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf

[14] ROS.org. Ros concepts. (10. 8. 2012). Dostopno na: http://www.ros.org/wiki/ROS/Concepts

[15] D. Skočaj, "Razvoj inteligentnih sistemov, transformacije med koordinatnimi sistemi," str. 2, 2012.

[16] ROS.org. Ros - urdf/xacro. (10. 8. 2012). Dostopno na: http://www.ros.org/wiki/urdf/Tutorials/Create%20your%20own%20urdf%20file

[17] OpenCV. OpenCV. (10. 8. 2012). Dostopno na: http://opencv.willowgarage.com/wiki

[18] Willow garage. (25. 8. 2012). Dostopno na: http://www.willowgarage.com/

[19] Itseez. (25. 8. 2012). Dostopno na: http://itseez.com/

[20] M. by Vincent Rabaud in T. Foote. Openni kinect. (10. 8. 2012). Dostopno na: http://www.ros.org/wiki/openni_kinect

[21] M. W. Maintained by Michael Ferguson, Tully Foote. Turtlebot - ros.org. (10. 8. 2012). Dostopno na: http://ros.org/wiki/Robots/TurtleBot

[22] PCL. PCL - PointCloud Library. (10. 8. 2012). Dostopno na: http://pointclouds.org

[23] Google. (25. 8. 2012). Dostopno na: http://google.com/

[24] Toyota. (25 .8. 2012). Dostopno na: http://www.toyota.com/

[25] Nvidia. (25. 8. 2012). Dostopno na: http://nvidia.com/

[26] M. A. Fischler in R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Comm. ACM*, zv. 24, št. 6, str. 381–395, junij 1981.

[27] Wikipedia. RANSAC - wikipedia. (10. 8. 2012). Dostopno na: http://en.wikipedia.org/wiki/RANSAC

[28] ——. (10. 8. 2012). Dostopno na: http://en.wikipedia.org/wiki/K-d_tree

[29] K. Khoshelham, "Accuracy analysis of kinect depth data," v *ISPRS workshop laser scanning 2011*, Kanada, 2011.

[30] Y. Freund in R. E. Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting*, ser. EuroCOLT '95. London, UK: Springer-Verlag, 1995.

[31] H. Grabner in H. Bischof, "On-line boosting and vision," v *In Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, zv. 1, ZDA, 2006, str. 260–267.

[32] H. Grabner, M. Grabner, in H. Bischof, "Real-time tracking via on-line boosting," v *Proceedings British Machine Vision Conference*, zv. 1, 2006, str. 47–56.

[33] Computer Vision Lab - ETH-Zurich. On-line boosting for tracking library. (12. 8. 2012). Dostopno na: http://www.vision.ee.ethz.ch/boostingTrackers/onlineBoosting.html

[34] N. Bellotto in H. Hu, "People Tracking with a Mobile Robot: a Comparison of Kalman and Particle Filters," v *Proc. of the 13th IASTED International Conference on Robotics and Applications (RA 2007)*, Würzburg, Germany, avg 2007, str. 388–393.

[35] G. Welch in G. Bishop, "An Introduction to the Kalman Filter," ZDA, 1995.

[36] ROS.org. Kinect accuracy. (11 .8. 2012). Dostopno na: http://www.ros.org/wiki/openni_kinect/kinect_accuracy

[37] M. Stevens. Bayes++: Open source bayesian filtering classes. (12. 8. 2012). Dostopno na: http://bayesclasses.sourceforge.net/Bayes++.html