Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Damjan Tratnik

Gradnja tridimenzionalnih modelov predmetov z mobilno platformo in barvno-globinsko kamero

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Danijel Skočaj

Ljubljana 2013

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: 01855/2012
Datum: 03.09.2012

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

Kandidat: **DAMJAN TRATNIK**

Naslov: **GRADNJA TRIDIMENZIONALNIH MODELOV PREDMETOV Z MOBILNO PLATFORMO IN BARVNO-GLOBINSKO KAMERO**

BUILDING THREE-DIMENSIONAL OBJECT MODELS USING A MOBILE PLATFORM AND AN RGBD CAMERA

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Svet, ki nas obdaja, je tridimenzionalen. Če hočemo zgraditi realistične 3D modele predmetov iz realnega sveta, je najbolje, da z ustreznimi globinskimi senzorji kar se da natančno posnamemo njihovo obliko in jo pretvorimo v primeren digitalen format. Seveda pa si želimo ta proces čim bolj avtomatizirati. V diplomski nalogi proučite področje avtomatske gradnje 3D modelov in zgradite sistem, ki bo z uporabo mobilne platforme TurtleBot in barvno-globinske kamere Kinect samostojno zgradil 3D model predmeta. Zgrajen sistem tudi primerno ovrednotite.

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 Damjan Tratnik, z vpisno številko **63070142**, sem avtor diplomskega dela z naslovom:

Gradnja tridimenzionalnih modelov predmetov 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, dne 15. marca 2013 Podpis avtorja:

Zahvaljujem se najprej mentorju doc. dr. Danijelu Skočaju za vso pomoč in nasvete pri izdelavi te diplomske naloge. Prav tako se zahvaljujem vsem članom Laboratorija za umetne vizualne spoznavne sisteme, ki so mi omogočali dostop do robotov in pomagali pri delu. Zahvala gre tudi vsem družinskim članom in prijateljem, ki so me tekom študija spodbujali in podpirali.

Moji družini.

Kazalo

Seznam uporabljenih kratic in simbolov

3D-PFH - 3D Point Feature Histogram; 3D geometrijska značilnica, ki za opisovanje okolice točke uporablja histograme
BSD - Berkley Software Distribution; Berkleyeva programska distribucija
ICP - Iterative Closest Point; iterativni algoritem za registracijo oblakov točk
MLS - Moving Least Squares; gibajoči najmanjši kvadrati
NDT - Normal-Distributions Transform; transformacija normalnih distribucij (algoritem za registracijo)
PCL - Point Cloud Library; knjižnjica za obdelavo oblakov točk
pIC - Probabilistic Iterative Correspondence; verjetnostna iterativna korespondenca (algoritem za registracijo)
RANSAC - RANdom SAmple Consensus; soglasje naključnih vzorcev
RGB - Red, Green, Blue; rdeča, modra, zelena (barvni model)
ROS - Robot Operating System; robotski operacijski sistem
SVD - Singular Value Decomposition; dekompozicija na singularne vrednosti
URDF - Unified Robot Description Format; poenoten format za opis robota
XML - Extensible Markup Language; razširljivi označevalni jezik

Povzetek

3D modele danes srečujemo v računalniških igrah, računalniško obogatenih filmih, obogateni resničnosti, animacijah, pa tudi pri načrtovanju novih izdelkov, simulacijah, arhitekturi, medicini itd. Zaradi prihoda novih nizkocenovnih globinskih kamer (npr. Microsoft Kinect in Asus Xtion Pro), je postala gradnja 3D modelov predmetov veliko bolj dostopna in razširjena. V diplomski nalogi smo zato najprej pripravili kratek pregled različnih tehnik zajemanja oblike predmetov in gradnje njihovih modelov, nato pa smo nekatere izmed teh tehnik tudi implementirali v sistemu, ki smo ga zgradili. Ta temelji na mobilni robotski platformi TurtleBot, ki jo sestavljata barvno-globinska kamera Kinect ter robotski sesalnik iRobot Roomba. Za programsko ogrodje smo uporabili Robotski operacijski sistem (ROS) in programsko knjižnjico za obdelavo oblakov točk - Point Cloud Library (PCL). Sistem, ki smo ga razvili, zna avtomatsko zajeti 3D informacijo ciljnega predmeta ter s pomočjo algoritma ICP (*ang. iterative closest point*) zgraditi njegov 3D model, ki ga lahko za nadaljno obdelavo shranimo v obliki barvnega oblaka točk ali poligonske mreže.

Ključne besede:
3D model, algoritem ICP, registracija, mobilna platforma, Kinect, oblak točk

Abstract

Nowadays we can observe 3D models in computer games, computer enriched movies, augmented reality and animations, as well as in the designing of new products, simulations, architecture, medicine etc. Due to the arrival of new low-cost depth cameras (e.g. Microsoft Kinect and Asus Xtion Pro), constructing 3D models of objects has become much more accessible and widespread. In this diploma thesis, we have prepared a short review of several distinct techniques of capturing shapes of objects and constructing their 3D models. Additionally, we implemented some of these techniques in a system that we designed. The system is based on mobile robot platform TurtleBot, which is composed of color-depth camera Microsoft Kinect and robot vacuum cleaner iRobot Roomba. Our software framework is based on Robot operating system (ROS) and Point Cloud Library (PCL), a software library for processing point clouds. Our system is able to automatically capture the 3D information of a target object and build its 3D model by using iterative closest point (ICP) algorithm. The 3D model obtained by the algorithm can be saved as a colored point cloud or as a polygon mesh.

Key words:
3D model, ICP algorithm, registration, mobile platform, Kinect, point cloud

Poglavje 1

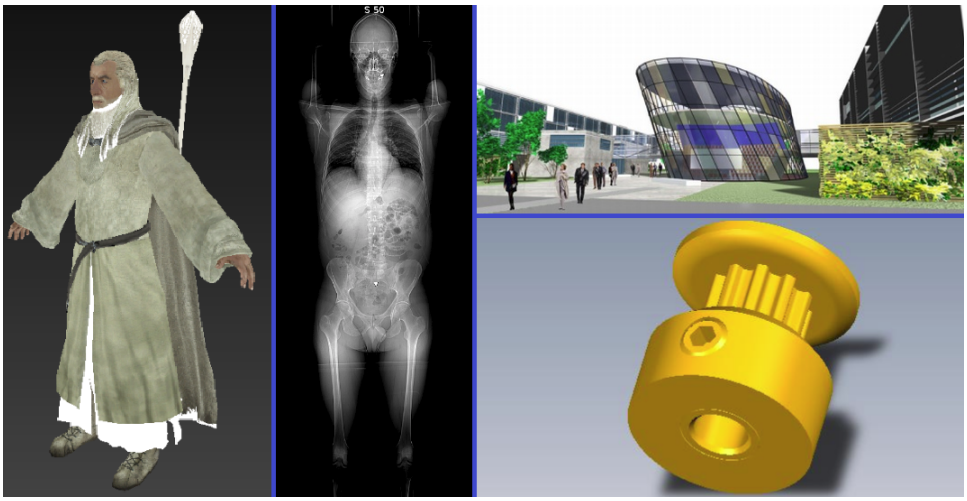
Uvod

Človek zaznava svet preko svojih čutil, predvsem preko vida. Z očmi namreč pridobimo preko 80 % vseh informacij iz okolja. Če nam torej slika predmeta o njem pove več kot tisoč besed, lahko s 3D modelom[1] tega predmeta povemo še več. Omogoča nam namreč opazovanje predmeta z vseh zornih kotov, z nekaterimi tipi modelov pa lahko izvajamo tudi simulacije in opazujemo obnašanje predmeta v različnih okoljih in razmerah. S 3D tiskalniki lahko gremo še korak dlje in 3D model natisnemo v fizični svet.

Zaradi svoje uporabnosti in količine informacije, ki jo nosijo, 3D modele danes najdemo na številnih področjih (slika 1.1). V zabavni industriji so široko prisotni v računalniških igrah in animiranih filmih, v zadnjem času pa tudi v aplikacijah z obogateno resničnostjo. V medicini so nepogrešljivi pri diagnosticiranju poškodb in bolezni ter pri načrtovanju operacij. V arhitekturi in okoljskem načrtovanju omogočajo ogled in oceno nove stavbe ali soseske preden se začne večji poseg v okolje. Prav tako so prisotni pri razvijanju prototipov vseh vrst, kjer zelo zmanjšajo stroške, saj se fizično izdelajo le prototipi, ki so bili prej že testirani v navideznem okolju.

Do 3D modela lahko pridemo na dva načina: lahko ga zgradimo z računalniško aplikacijo ali pa njegovo obliko zajamemo iz realnega sveta. Gradnja modela z računalniškimi programi kot sta npr. AutoCAD in Blender je lahko zelo

[1] Tukaj in skozi celoten tekst z besedama *'3D model'* mislimo na računalniški 3D model.



Slika 1.1: Različne vrste 3D modelov. Od leve proti desni: 3D model junaka računalniške igre, 3D model pacienta uporabljen pri načrtovanju operacije, zgoraj: 3D model načrtovane nove zgradbe, spodaj: 3D model jermenskega zobnika. (Povzeto po [2], [3], [4], [5].)

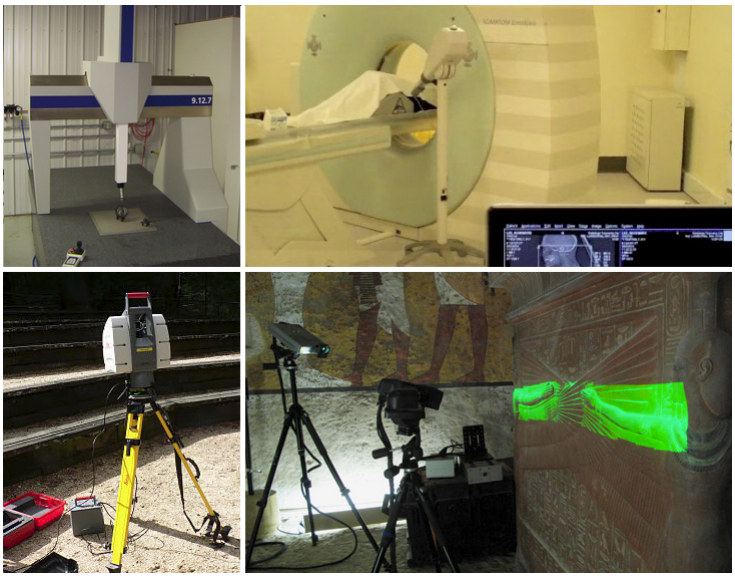
zahtevna in časovno potratna. To je še posebej res, če hočemo imeti natančen model zapletenega predmeta iz realnega sveta. Z zajemanjem 3D oblike takega predmeta lahko ta postopek pohitrimo in poenostavimo. Seveda pa moramo imeti na voljo ustrezne senzorje, ki lahko iz prostora zajamejo 3D informacijo.

Tehnologija pridobivanja 3D informacije se pri različnih vrstah senzorjev precej razlikuje. Kontaktni senzorji se s tipalom neposredno dotikajo predmeta in tako pridobijo informacijo o njegovi obliki. So sicer zelo natančni [1], a tudi najpočasnejši, poleg tega pa lahko tudi deformirajo skenirani predmet. V medicini se uporabljata postopka računalniške tomografije in slikanje z magnetno resonanco. V prvem primeru gre za zajem serije 2D rentgenskih posnetkov iz različnih zornih kotov, nato pa se te posnetke računalniško združi v 3D model. Enako velja tudi za magnetno resonanco, le da se 2D posnetke pridobi z manipuliranjem in merjenjem magnetnih momentov atomskih jeder v telesu, ki ga skeniramo. Za zajemanje oblike velikih in oddaljenih objektov

so primerni senzorji, ki objekt osvetlijo z laserskim žarkom, zaznajo odboj, ter iz časa potovanja žarka določijo oddaljenost oziroma obliko objekta. Drugi način uporabe laserja za zajemanje 3D informacije je v kombinaciji s kamero. Tako zgrajen senzor opazuje lasersko piko na predmetu ter s principom triangulacije izračuna razdaljo do predmeta. Vir svetlobe in kamera se prav tako uporabljata pri pridobivanju 3D informacije s strukturirano svetlobo, kjer se na predmet najprej projicira znana tekstura, nato pa se na podlagi deformacije te teksture, ki jo zaznamo s kamero, izračuna razdalja do predmeta in njegova oblika. Ta pristop se v zadnjem času široko uporablja, glavni razlog za to pa je nizkocenovni senzor Kinect. Kot dodatek za igralno konzolo Xbox je sicer primarno namenjen prepoznavanju človeškega skeleta in njegovih kretenj za namen upravljanja konzole in kot tak ne omogoča natančnega zajemanja 3D podatkov. Kljub temu so raziskovalci že pokazali, da je z ustrezno obdelavo teh podatkov mogoče zgraditi dober 3D model [7]. Za lažjo predstavo, si lahko nekaj od zgoraj naštetih senzorjev ogledamo tudi na sliki 1.2.

Zajemanje celotnega 3D modela predmeta vključuje zajem podatkov iz več zornih kotov, pri čemer so lege, iz katerih so bili zajeti podatki, poznane. Nekateri senzorji (npr. kontaktni senzor, rentgen pri računalniški tomografiji) so že zgrajeni tako, da se sami lahko premaknejo v drugo lego in tako zajamejo podatke iz več zornih kotov, pri ostalih pa moramo za premik poskrbeti sami. Ker želimo zajem 3D modela čim bolj avtomatizirati, si pomagamo z roboti, ki nam poleg premikanja nudijo tudi informacijo o legi v kateri se nahajajo. Z namestitvijo senzorja za zajem 3D informacije na robota, ki se lahko premika po prostoru, dobimo platformo, ki omogoča avtomatsko zajemanje 3D modelov predmetov.

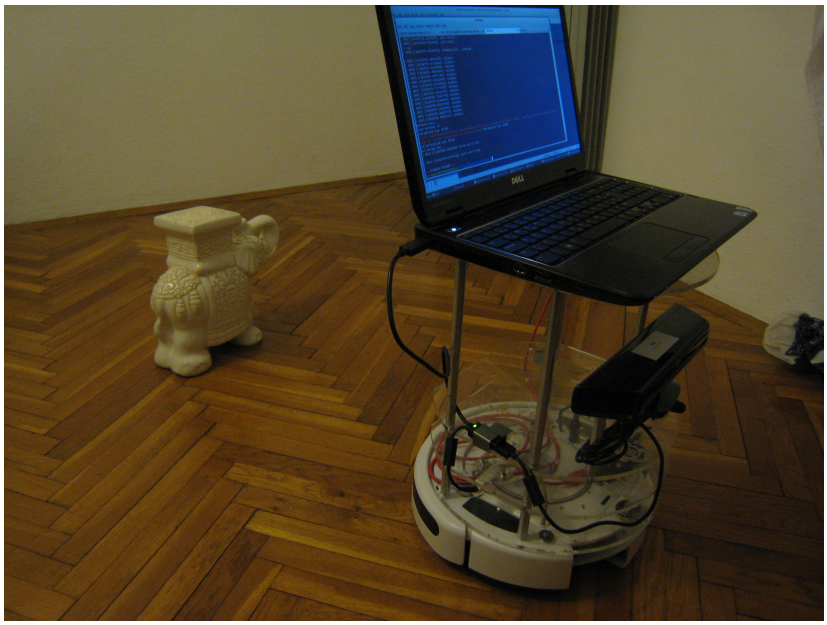
To je bil tudi cilj naše diplomske naloge: zgraditi sistem, ki zna avtomatsko zajeti 3D informacijo predmeta, ki ga postavimo predenj ter zgraditi njegov 3D model. Sistem smo zgradili na robotski platformi TurtleBot. Ta združuje robotski sesalnik Roombo, ki omogoča premikanje po prostoru in senzor Kinect, ki zajema globinsko sliko prostora. Naš sistem mora pra-



Slika 1.2: Različne vrste senzorjev za zajemanje 3D informacije. Zgoraj: levo kontaktni senzor, desno rentgen. Spodaj: levo laserski senzor, primeren za večje objekte, desno senzor s strukturirano svetlobo. (Povzeto po [1],[2],[6].)

vilno navigirati Roombo okrog predmeta in s Kinectom zajeti oblake točk iz različnih zornih kotov, nato pa mora te oblake še pravilno registrirati (sestaviti v 3D model). Navigacija je precej preprosta, saj ne gradimo mape prostora, ampak predpostavimo, da se robot lahko prosto giblje po krožnici okrog predmeta. Zahtevnejša je registracija pridobljenih oblakov točk, saj zaradi zgradbe TurtleBota skenirani predmet ni neprestano v objektivu Kinecta. Na voljo imamo le oblake točk iz nekaj različnih zornih kotov okrog predmeta. Pri gradnji 3D modela se tako zanašamo na odometrijo Roombe, natančnejša registracija pa sloni na algoritmu ICP (*ang. Iterative Closest Point*) [8], s katerim iterativno iščemo najboljšo transformacijo med dvema oblakoma točk. Na sliki 1.3 je prikazan sistem med delovanjem.

V uvodu smo predstavili razširjenost 3D modelov, različne vrste globinskih senzorjev ter nakazali naš pristop k 3D skeniranju. V drugem poglavju pregledamo teoretične osnove gradnje 3D modelov s pomočjo globinskih senzorjev. Osredotočimo se na načine registracije oblakov točk in omenimo



Slika 1.3: Sistem za gradnjo 3D modelov med delovanjem.

pristope, ki uspešno zmanjšajo napako in šum senzorja Kinect pri zajemu oblakov točk. V tretjem poglavju podrobneje opišemo uporabljeno robotsko platformo TurtleBot in predstavimo programsko ogrodje na katerem smo gradili: robotski operacijski sistem ROS in programsko knjižnjico PCL. Predstavitev in opis našega pristopa k skeniranju 3D predmetov sledi v četrtem poglavju. Peto poglavje je namenjeno ovrednotenju natančnosti Roombe in Kinecta ter predstavitvi in ovrednotenju rezultatov. V zadnjem, šestem poglavju pa izpostavimo prednosti in slabosti razvite rešitve ter predlagamo nekaj izboljšav za nadaljni razvoj.

Poglavje 2

Teoretične osnove

Že v uvodu smo našteli najpogostejše načine pridobivanja 3D modelov z različnimi globinskimi senzorji. V našem pristopu uporabljamo globinsko kamero, s katero zajamemo oblake točk iz različnih zornih kotov. Te oblake točk je potrebno postaviti v isti koordinatni sistem in jih združiti, kar imenujemo registracija. Sledi še pretvorba oblaka točk v poligonsko mrežo. Ker je ključen korak tega postopka registracija oblakov točk, se bomo v tem poglavju osredotočili na različne algoritme, ki se za to uporabljajo. V drugem delu tega poglavja pa bomo pregledali še nekaj pristopov, ki se uporabljajo za obvladovanje šuma in napak v zajetih oblakih točk.

2.1 Metode registracije oblakov točk

Problem registracije je preprost le v primeru, ko imamo natančno podane lege (pozicija in orientacija senzorja), s katerih so bili posamezni oblaki točk zajeti. V tem primeru lahko iz samih leg izračunamo transformacije lokalnih koordinatnih sistemov posameznih oblakov točk v globalni koordinatni sistem. Oblake nato transformiramo v ta globalni koordinatni sistem in jih združimo. Navadno imamo na voljo le približne lege, ki nam dajo začetno poravnavo oblakov, zato potrebujemo algoritme, ki lahko to začetno poravnavo izboljšajo.

2.1.1 Registracija na podlagi geometrijskih značilnic

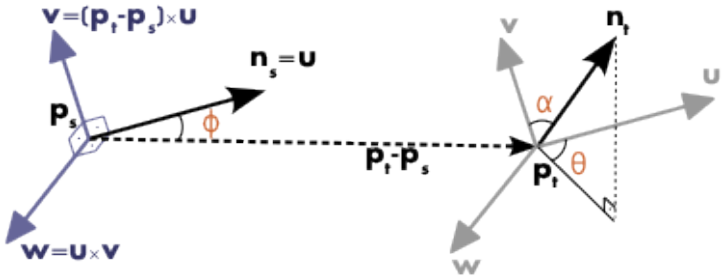
Za izračun transformacije med dvema oblakoma točk po tej metodi moramo identificirati minimalno tri korespondenčne pare točk (*ang. correspondences*); to pomeni, da je prva točka iz prvega oblaka, druga iz drugega, obe pa predstavljata isto točko na objektu. Problem registracije se torej v tem primeru spremeni v problem iskanja korespondenčnih parov točk, pri tem pa si pomagamo z značilnicami.

Geometrijske značilnice (*ang. geometric features*) opisujejo geometrijske značilnosti objektov. Za opisovanje oblakov točk uporabljamo 3D geometrijske značilnice, ki na različne načine zakodirajo informacijo o obliki površine oziroma o razmerju med točkami. Ker lahko neko površino opišemo na različne načine (npr. zgolj s 3D informacijo, z normalami na površino, s kombinacijo obojega ...), niso vse značilnice enakovredne. Posebej za namene registracije so enostavne značilnice (npr. zgolj ukrivljenost) neprimerne, saj je na oblaku navadno veliko točk z enako ukrivljenostjo in tako ne dobimo želenih korespondenc. Različni avtorji zato predlagajo zapletenejše značilnice, ki bolje opisujejo geometrijske značilnosti neke točke in njene okolice. Tu kratko opišemo dve taki značilnici:

- A. E. Johnson v [10] predstavi vrtečo sliko (*ang. spin-image*), ki jo izračunamo tako, da rastersko sliko postavimo z robom ob normalo sredinske točke, kjer računamo značilnico, ter zavrtimo sliko okrog normale. Slikovnemu elementu slike, na katere med vrtenjem pade katera od sosednjih točk, se vrednost poveča in tako ob celem obratu dobimo sliko, ki dobro opisuje celotno okolico sredinske točke.

- R. B. Rusu v [11] predstavi značilnico 3D-PFH (3D Point Feature Histogram), ki je invariantna na toge transformacije, gostoto točk v oblaku in blag šum v podatkih. Omenjena značilnica deluje tako, da za okolico točke, kjer računamo 3D-PFH, za vsak par točk zakodiramo njuno razmerje v 4 vrednosti; kote ϕ, α in θ , ter razdaljo med točkama kot prikazuje slika 2.1. Te vrednosti nato razporedimo v histogram, ki nam

opisuje opazovan skupek točk.



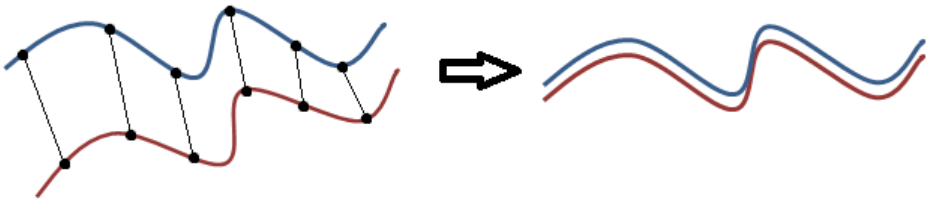
Slika 2.1: Prikaz štirih vrednosti, ki predstavljajo razmerje med točkama pri računanju 3D-PFH: koti ϕ , α in θ ter razdalja med točkama $(P_t - P_s)$.

Računanje kompleksnejših značilnic je časovno zahtevno, zato se te navadno izračunajo samo na nekaj zanimivih točkah (*ang. interest points*). Take so npr. točke na vzboklinah in izboklinah, robovih, kotih ipd. Za njihovo identificiranje uporabljamo detektorje zanimivih točk, kjer med najbolj znanimi zasledimo Harrisov detektor.

Postopek registracije na podlagi geometrijskih značilnic bi torej lahko povzeli takole:

1. iskanje zanimivih točk v oblakih,
2. računanje značilnic v teh zanimivih točkah,
3. iskanje korespondenc s primerjanjem značilnic,
4. računanje transformacije med oblaki na podlagi korespondenc,
5. transformacija oblakov v isti koordinatni sistem.

Ker korespondence velikokrat niso točne, tudi končna poravnava oblakov ni optimalna, zato se ta metoda ponavadi uporablja le kot prvi korak registracije. Z njo pridobimo začetno poravnavo, ki jo z drugimi metodami izboljšamo.



Slika 2.2: Prikaz delovanja algoritma ICP. Levo označeni korespondenčni pari, desno deloma poravnani krivulji.

2.1.2 Algoritem ICP

Najbolj razširjen pristop k registraciji oblakov točk je še vedno algoritem ICP [8]. Algoritem iterativno popravlja oceno transformacije med dvema oblakoma in tako postopoma najde optimalno poravnavo glede na podano metriko ocenjevanja kakovosti transformacije. Delovanje algoritma v 2D je prikazano na sliki 2.2. Poudariti moramo, da ICP ne zagotavlja konvergence k globalnemu optimumu, ampak se lahko ustavi v lokalnem optimumu, zato je za pravilno delovanje potrebna približna začetna poravnava oblakov. To lahko dobimo z zgoraj opisano metodo geometrijskih značilnic ali pa iz približnih pozicij, s katerih so bili oblaki posneti.

Bistvo algoritma za poravnavo dveh oblakov (vhodnega k ciljnemu) lahko podamo takole:

1. za vsako izmed izbranih točk v vhodnem oblaku najdi najbližjo točko v ciljnem oblaku,

2. na podlagi dobljenih parov (korespondenc) izračunaj transformacijo z minimiziranjem izbrane metrike napake,

3. transformiraj vhodni oblak,

4. ponavljaj korake od 1 do 3, dokler ne dosežeš dovolj dobre poravnave glede na podan pogoj ali število iteracij preseže podano vrednost.

Tako podan algoritem pušča veliko odprtih možnosti za konkretno implementacijo. V nadaljevanju bomo s pomočjo [12] analizirali različne variante ICP algoritma glede na štiri kriterije (izbira točk, iskanje korespondenčnih parov točk, uteževanje in zavračanje korespondenčnih parov točk, izbira metrike napake in minimizacija napake), nato pa obrazložili še matematično ozadje najzahtevnejšega koraka algoritma.

Izbira točk

Prvi korak algoritma je izbira točk, s pomočjo katerih bomo računali poravnavo. Najpreprostejša varianta je izbira vseh točk, ki so na voljo [8]. Druga možnost je uniformno vzorčenje točk [13] ali pa naključno vzorčenje točk [14]. Poleg tega lahko te pristope uporabimo samo na enem oblaku točk, ali pa na obeh [15].

Uporaba vseh točk zagotavlja največ informacije, vendar pa pri večjih oblakih točk zelo poveča čas izvajanja algoritma. Uporaba naključno izbrane podmnožice točk zagotavlja večjo hitrost izvajanja, vendar lahko zaradi naključnosti izgubimo nekatere značilnosti oblike oblaka točk in tako poslabšamo natančnost. Izbira reprezentativne podmnožice točk (z uniformnim vzorčenjem) je torej najboljša izbira, saj omogoča večjo hitrost izvajanja algoritma, hkrati pa je informacija oblike oblaka dobro ohranjena. Hitrost izvajanja se še dodatno poveča, če uniformno vzorčenje opravimo na obeh oblakih točk, ki ju registriramo.

Iskanje korespondenčnih parov točk

Ta korak je ključnega pomena, saj vpliva tako na hitrost algoritma, predvsem pa na njegovo natančnost. Pogosti pristopi so iskanje najbližje točke [8], iskanje najbližje točke v smeri normale [16] ali projekcija točke na površino drugega oblaka in iskanje najbližje točke [17].

V prisotnosti šuma bolje delujeto pristopa iskanja najbližje točke s projekcijo ali v smeri normale, najslabše pa samo iskanje najbližje točke, kot lahko vidimo tudi na sliki 2.3. Obratno pa je pri registraciji oblakov s težjo geo-

Slika 2.3: Izbira korespondenčnih parov z metodo projekcije (desno) je v šumnih pogojih boljša od izbire z metodo najbližje točke.

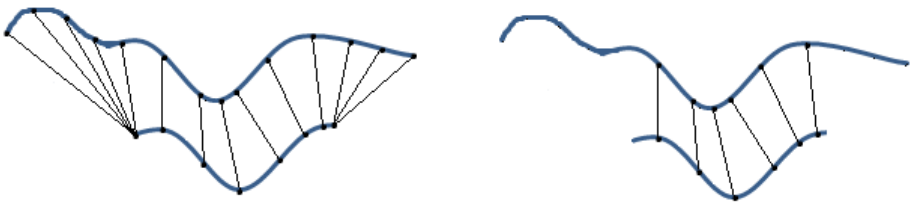
metrijo (npr. ravnine), kjer se najbolje odreže prav pristop iskanja najbližje točke.

Uteževanje in izločanje korespondenčnih parov točk

Za izboljšanje poravnave lahko korespondenčnim parom točk določimo tudi uteži, ki predstavljajo vpliv posameznega korespondenčnega para točk na izračun transformacije. Najpogostejši načini uteževanja so: določitev enakih uteži vsem korespondenčnim parom, določitev manjše uteži korespondenčnim parom z veliko razdaljo in uteževanje na podlagi ujemanja normal v korespondenčnem paru [18]. Izkaže se, da so razlike med različnimi pristopi majhne in se razlikujejo od primera do primera.

Zavračanje korespondenčnih parov točk lahko prav tako izvajamo na različne načine. Tako lahko korespondenčni par točk zavrnemo če:

- ima razdaljo med točkama večjo od razdalje podane s strani uporabnika,
- spada v določen odstotek korespondenčnih parov z najslabšo razdaljo [19],
- ima razdaljo med točkama večjo od 2,5-kratnika standardnega odklona razdalj [14],
- je ena od točk v korespondenčnem paru na robu oblaka točk [13].



Slika 2.4: Prikaz prednosti izločanja korespondenčnih parov, ki vsebujejo točke na robovih.

Zadnjo metodo je vedno dobro uporabiti, saj vedno izboljša kakovost poravnave, kar je prikazano tudi na sliki 2.4. K boljši poravnavi prispevajo tudi ostale, od katerih nam prva omogoča največ nadzora, saj lahko parameter nastavimo za dobro delovanje vsakega primera posebej.

Izbira metrike napake in minimizacija napake

Najpogostejši metriki napake, ki se uporabljata pri algoritmu ICP, sta vsota kvadratov razdalj (*ang. sum of squared distances*) med točkami v korespondenčnih parih in vsota kvadratov razdalj med prvo točko v korespondenčnem paru ter ravnino, na kateri leži druga točka. V prvem primeru se za minimizacijo napake in izračun transformacije največkrat uporablja pristop z uporabo dekompozicije na singularne vrednosti ali SVD (*ang. Singular Value Decomposition*), v drugem primeru pa metoda Levenberg-Marquardt.

Ker v našem sistemu za gradnjo 3D modelov uporabljamo različico algoritma ICP, ki uporablja prvi pristop, v nadaljevanju opišemo minimizacijo vsote kvadratov razdalj ter iskanje ustrezne transformacije s pristopom SVD, kot je bil najprej predstavljen v [20].

Računanje transformacije po metodi SVD

Na voljo imamo dva oblaka točk $P = \{p_i\}$, $P' = \{p'_i\}$; $i = 1, 2, ..., N$, kjer p_i in p'_i predstavljata točke v obliki 3×1 matrik. Relacijo med tema dvema oblakoma lahko zapišemo z enačbo 2.1, kjer je R rotacijska matrika velikosti

3×3, T translacijski vektor velikosti 3×1, N_i pa je vektor šuma. Iščemo torej R in T, ki bosta minimizirala vsoto kvadratov razdalj (Σ^2 v enačbi 2.2).

$$p'_i = Rp_i + T + N_i \tag{2.1}$$

$$\Sigma^2 = \sum_{i=1}^{N} ||p'_i - (Rp_i + T)||^2 \tag{2.2}$$

Če vzamemo, da sta R_s in T_s rešitvi enačbe 2.2, ugotovimo, da imata oblaka točk $P' = \{p'_i\}$ in $P'' = \{p''_i = R_s p_i + T_i\}$ enak centroid. Vzemimo torej, da so p, p' in p'' centroidi oblakov točk P, P' in P'' vsi izračunani po enačbi:

$$c = \frac{1}{N} \sum_{i=1}^{N} c_i. \tag{2.3}$$

Ugotovili smo že, da velja enakost $p' = p''$, centroida oblakov točk P in P' pa nista enaka. Vendar, če obema oblakoma odštejemo njune centroide, imata nova oblaka točk $Q = \{q_i; q_i = p_i - p\}$ in $Q' = \{q'_i; q'_i = p'_i - p'\}$ centroida v koordinatnem izhodišču. Zdaj lahko zapišemo novo enačbo za minimizacijo vsote kvadratov razdalj med točkami:

$$\Sigma^2 = \sum_{i=1}^{N} ||q'_i - Rq_i||^2. \tag{2.4}$$

Ta problem pa lahko rešimo s postopkom SVD. Na podlagi enačbe 2.4 izračunamo matriko H in napravimo SVD te matrike, kot je opisano v enačbah 2.5 in 2.6. Nato po enačbi 2.7 izračunamo še matriko X, ki nam pod pogojem, da je njena determinanta enaka 1, že predstavlja matriko R_s, kar smo ponazorili z enačbo 2.8. V primeru da je determinanta matrike X enaka -1, algoritem ni uspešen, saj nam X v tem primeru predstavlja zrcaljenje, a to se v praksi skoraj nikoli ne zgodi.

$$H = \sum_{i=1}^{N} q_i q_i'^t \tag{2.5}$$

$$H = U\Delta V^t \tag{2.6}$$

$$X = VU^t \tag{2.7}$$

$$det(X) = 1 \Leftrightarrow X = R_s \tag{2.8}$$

Z izračunano matriko rotacije R_s smo uspešno izračunali polovico transformacije, ki jo iščemo. Drugi del nam predstavlja translacija, ki jo izračunamo s pomočjo centroidov prvotnih oblakov točk ter pravkar izračunane rotacije. Natančneje to opisuje enačba 2.9.

$$T_s = p' - R_s p \tag{2.9}$$

Postopek izračuna transformacije med dvema oblakoma točk z uporabo metode SVD je s tem zaključen.

2.1.3 Druge metode

Poleg registracije na podlagi geometrijskih značilnic, ki se velikokrat uporablja le za začetno poravnavo oblakov, ter algoritma ICP, ki je za namene registracije najbolj široko uporabljan, poznamo tudi druge metode registracije oblakov točk. Te večinoma niso široko razširjene in ker jih v našem sistemu za gradnjo 3D modelov ne uporabljamo, jih tukaj ne bomo podrobneje opisovali. V nadaljevanju le naštejemo nekatere metode, ki so obravnavane tudi v [9] in ob vsaki skušamo podati njeno glavno značilnost.

- **Algoritem NDT** *(ang. Normal-Distributions Transform)* [21]: Namesto iskanja točk za ujemanje, je vsaka pozicija na referenčnem oblaku predstavljena z linearno kombinacijo normalnih porazdelitev, ki povedo verjetnost, da se tam nahaja točka.

- **Algoritem pIC** *(ang. Probabilistic Iterative Correspondence)* [22]: Ta metoda skuša upoštevati negotovost šuma v podatkih ter negotovost ocene začetne poravnave tako, da obravnava vhodni oblak točk in začetno oceno poravnave kot naključno spremenljivko z Gaussovim šumom.

- **Točkovna verjetnostna registracija** *(ang. point-based probabilistic registration)* [23]: Algoritem referenčnega oblaka točk ne obravnava kot množice diskretnih točk ampak kot množico verjetnostnih funkcij.

- **Gaussova polja** *(ang. Gaussian fields)* [24]: Podobno kot NDT - tako razdalja med točkama kot tudi podobnost njune okolice je modelirana z Gaussovim modelom.

- **Kvadratne zaplate** *(ang. Quadratic patches)* [25]: Referenčni oblak je tu predstavljen s kvadratnimi približki kvadratne funkcije razdalje do površine oblaka točk.

2.2 Gradnja 3D modela z uporabo nenatančnega senzorja

Senzorji, ki zagotavljajo natančen zajem globinske informacije, so ponavadi zelo dragi. Tako se moramo velikokrat zadovoljiti s slabšimi senzorji, ki ne zagotavljajo velike natančnosti podatkov. V nadaljevanju si bomo zato pogledali nekaj pristopov, ki ponazarjajo kako lahko programsko zmanjšamo vpliv šuma in napak v vhodnih podatkih na končni rezultat - 3D model.

2.2.1 Filtriranje in glajenje

Če je v vhodnih podatkih prisoten naključnen šum, ga lahko z uporabo primernih filtrov lahko uspešno zmanjšamo. Ker imamo ponavadi opravka z urejenimi oblaki točk ali globinskimi slikami, jih lahko filtriramo s premikajočim oknom. Čez sliko torej premikamo okno in v sredino okna vpišemo novo, filtrirano vrednost, ki jo izračunamo na podlagi vseh vrednosti v oknu. Za odstranjevanje naključnega šuma lahko uporabimo medianin filter, (uteženo) povprečje, Gaussov filter, ali kak drug podoben filter. S tem pristopom lahko hkrati tudi zapolnimo luknje v globinski sliki.

Šum lahko zmanjšujemo tudi z glajenjem površine oblaka točk. Za ta pristop najprej izračunamo normale točk, nato pa poravnamo točke z normalami tako, da na površini ni ostrih sprememb.

Opozoriti moramo, da lahko posebej s povprečenjem in glajenjem sicer odstranimo nakljućen šum, vendar pa lahko hkrati tudi uvedemo nov šum v oblak točk, posebej na robovih ali v primeru zelo razgibanih predmetov, ki jih preveč zgladimo.

Filtriranje neurejenih oblakov točk lahko namesto s premikajočim oknom izvajamo na k-najbližjih sosedih ali na sosedih v podanem radiju. Ta način je primeren predvsem za odstranjevanje osamelcev (*ang. outliers*), ali manjših gruč točk - iz oblaka filtriramo točke, ki imajo v podanem radiju premalo sosedov (manj kot podan minimum).

Tukaj lahko omenimo še zmanjševanje števila točk z mrežo vokslov (*ang. voxel grid*) - mrežo 3D kock, saj je to nek način povprečenja. Nova točka v vsakem vokslu se namreč izračuna kot povprečje vseh točk v tem vokslu. Tako sicer zmanjšamo šum, vendar zmanjšamo tudi ločljivost oblaka in s tem izgubimo podrobnosti skeniranega predmeta.

2.2.2 Super-ločljivost

Nasproten pristop od zmanjševanja števila točk v oblaku je super-ločljivost (*ang. super-resolution*). Nizkocenovni senzorji za zaznavanje globine ponavadi ponujajo le nizke ločljivosti oblakov točk. S povečanjem ločljivosti oblakov točk bi hkrati zajeli več podrobnosti in tudi zmanjšali količino šuma, ki bi ostal v oblaku po filtriranju. Za dosego tega cilja lahko uporabimo zaporedje oblakov točk, posnetih z istega mesta. Oblake združimo skupaj in nato interpoliramo nov oblak točk z večjo ločljivostjo.

2.2.3 KinectFusion

Na koncu tega poglavja predstavimo še projekt KinectFusion [7], kot uspešen način gradnje 3D modelov z nizkocenovno barvno-globinsko kamero Kinect.

Slika 2.5: Prikaz uspešnosti sistema KinectFusion: levo eden od oblakov točk, uporabljenih pri konstrukciji modela, na sredini prikaz normal, desno prikaz modela s Phongovim senčenjem. (Povzeto po [27].)

Avtorjem je uspelo zgraditi natančne 3D modele predmetov in tudi celotnih prostorov samo z uporabo globinskih slik oziroma oblakov točk pridobljenih z omenjenim senzorjem. Skeniranje se izvaja z ročnim premikanjem senzorja okrog predmeta oziroma po prostoru. Celoten sistem je optimiziran za paralelno izvajanje na grafičnih procesorjih in tako zagotavlja gradnjo 3D modelov v realnem času.

Glavni koraki algoritma so naslednji:

1. pretvorba oblaka točk v globinsko sliko in računanje normal,
2. sledenje pozicije kamere z algoritmom ICP,
3. posodobitev mreže vokslov,
4. izgradnja 3D modela površine z uporabo metode *ray casting* [26] za izris uporabniku in uporabo pri sledenju pozicije kamere z algoritmom ICP.

Rezultat zajemanja oblike obraza in gradnje 3D modela s sistemom KinectFusion lahko vidimo na sliki 2.5.

Poglavje 3

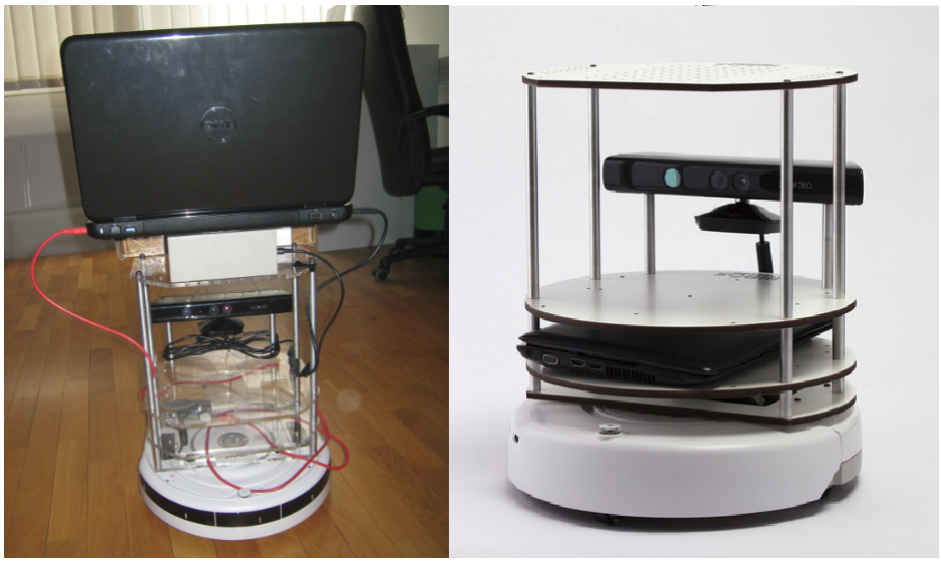
Opis uporabljene platforme

V tem poglavju bomo opisali tako strojno kot tudi programsko platformo, na kateri smo zgradili naš sistem za gradnjo 3D modelov. V prvem podpoglavju se bomo osredotočili na robotsko platformo TurtleBot. V drugem in tretjem podpoglavju bomo podrobneje opisali glavna dela te robotske platforme: robotski sesalnik IRobot Roombo in barvno-globinsko kamero Kinect. V četrtem podpoglavju bomo opisali vezni člen med robotsko platformo TurtleBot in našimi programi - robotski operacijski sistem (ROS). Zadnje podpoglavje pa bomo namenili opisu programske knjižnice Point Cloud Library (PCL), s katero smo si pomagali pri gradnji 3D modelov.

3.1 Mobilna robotska platforma TurtleBot

TurtleBot je nizkocenovna robotska platforma, ki so jo razvili, oziroma bolje rečeno sestavili, v podjetju Willow Garage. TurtleBot je namreč sestavljen iz komponent, ki se sicer prodajajo kot samostojni izdelki. Glavni sestavni deli mobilne robotske platforme Turtlebot so:

- mobilna platforma iRobot Create ali robotski sesalnik iRobot Roomba,
- barvno-globinska kamera Microsoft Kinect,
- skelet Turtlebota (palice in plošče).



Slika 3.1: Prikaz evropske različice platforme TurtleBot, ki smo jo uporabljali za naš sistem (levo) ter ameriške različice (desno). (Povzeto po [28].)

Ker je TurtleBot izdan pod licenco FreeBSD, si ga lahko tudi sami sestavimo in prilagodimo glede na svoje želje, oziroma dele, ki so nam na voljo. Poznamo pa tudi dve uradni različici TurtleBota: ameriško in evropsko. Ameriška različica za mobilno platformo uporablja Create, poleg tega pa ima dodan tudi enoosni giroskop za boljšo odometrijo. Evropska različica, ki smo jo uporabljali tudi mi, za mobilno platformo uporablja robotski sesalnik Roombo in nima giroskopa. Za naš sistem smo uporabljali evropsko različico TurtleBota, ki za mobilno platformo uporablja robotski sesalnik Roombo. Navadno se skupaj s TurtleBotom uporablja tudi Asus netbook, ki je navadno položen na ploščo tik nad robotom Create ali Roombo in na katerem je nameščen ROS. Računalnik, ki smo ga uporabljali za naš sistem, pa je bil večji, zato smo ga postavili na vrhnjo ploščo Turtlebota. Primerjavo med različicami platforme TurtleBot lahko vidimo na sliki 3.1.

3.2 Robotski sesalnik IRobot Roomba

IRobot Roomba je serija avtonomnih robotskih sesalnikov, ki jih proizvaja in prodaja podjetje IRobot [29]. Prva Roomba je bila izdelana leta 2002, danes (2013) pa je na voljo že četrta generacija pod oznako *700 series*. Nizka cena robota in možnost priključitve na računalnik in upravljanja preko posebnega vmesnika sta Roombo popularizirala tudi v akademskih krogih. Podjetje iRobot je zato prav v ta namen izdelalo modificiran model Roombe imenovan Create, ki je tudi standardno del robotske platforme TurtleBot. Platforma, ki smo jo uporabljali mi, pa kot že rečeno namesto robota Create uporablja Roombo 530.

3.2.1 Opis

Kot lahko vidimo na sliki 3.2, je Roomba okrogle oblike, s premerom 34 cm in višino 9 cm. Na sprednji strani ima velik odbijač za zaznavanje trkov. Na odbijaču zgoraj ima infrardeči senzor za zaznavanje virtualnih zidov, na spodnji strani pa ima štiri senzorje za previse. Za premikanje služita dve kolesi, ki se ne premikata levo-desno ampak le naprej-nazaj. Kolesi sta krmiljeni neodvisno eno od drugega, kar omogoča obračanje in zavijanje. Roomba, ki smo jo uporabljali mi, ima na spodnji strani odstranjene krtače za sesanje in na zadnjem delu dodan kroglični ležaj za boljšo stabilnost (slika 3.3).

3.3 Senzor Kinect

Podjetje Microsoft je senzor Kinect javnosti predstavilo leta 2010 kot periferno napravo za igralno konzolo Xbox 360. Njen prvotni namen je omogočanje interakcije s konzolo Xbox zgolj s kretnjami in glasovnimi ukazi [30]. Samo nekaj dni po začetku prodaje so se v javnosti že pojavili odprtokodni gonilniki, ki so omogočali dostop do Kinectovih podatkov. To dejstvo, skupaj z nizko ceno, je Kinect popularizirala tudi v akademski sferi.

Slika 3.2: Robotski sesalnik iRobot Roomba 530.

Slika 3.3: Prikaz spodnjega dela predelane Roombe 530, ki smo jo uporabljali v našem sistemu.

Slika 3.4: Senzor Kinect z označenimi glavnimi deli: 1 - IR projektor, 2 - RGB kamera, 3 - IR kamera, 4 - polje mikrofonov, 5 - motorizirana noga.

3.3.1 Opis

Osnovni deli Kinecta, ki jih lahko vidimo tudi na sliki 3.4, so:

1. projektor laserske IR svetlobe, ki projicira pikast vzorec,
2. 8-bitna RGB kamera z ločljivostjo VGA (640x480 pikslov) s frekvenco do 30 Hz,
3. IR kamera prav tako z ločljivostjo VGA (640x480 pikslov) in frekvenco do 30 Hz,
4. polje štirih mikrofonov ob straneh naprave, od katerih vsak zajame 16-biten zvok ob vzorčni frekvenci 16 kHz,
5. motorizirana noga, ki omogoča 27 stopinjski vertikalni naklon navzgor ali navzdol.

Kinect je globinski senzor, ki deluje po principu strukturirane svetlobe - s projektorjem projicira teksturo v prostor in nato na podlagi deformacije te teksture izračuna informacijo o globini. Glede na [31] se ta informacija izračuna z uporabo dveh principov: globina iz fokusa (bolj zamegljeni predmeti so bolj

oddaljeni) in globina iz sterea (bližnji predmeti so bolj zamaknjeni; namesto dveh kamer Kinect uporablja projektor in kamero).

Globinska slika, ki jo Kinect nudi, ima enako resolucijo kot IR kamera (640x480 slikovnih elementov), globina pa je zakodirana z 11 biti, kar omogoča 2048 stopenj občutljivosti. Horizontalni vidni kot je 57 stopinj, vertikalni 43 stopinj, vidno polje pa se razteza od približno 0,7 - 6 m.

3.4 Robotski operacijski sistem - ROS

ROS je odprtokodno programsko ogrodje (licenca BSD), ki omogoča razvoj programske opreme za širok nabor robotov [32]. Razvoj se je začel leta 2007 na Standfordski univerzi, vendar se je že leto kasneje v veliki meri preselil v podjetje Willow Garage, ki razvoj nadaljuje še danes. Prva večja izdaja ROS-a je bila januarja leta 2010, do danes pa se je zvrstilo že sedem večjih izdaj; trenutno najnovejša (datum izdaje 31. 12. 2012) ima oznako Groovy Galapagos. Naš sistem temelji na šesti večji izdaji ROS-a z imenom Fuerte.

V naslednjih podpoglavjih bomo podali oris sistema ROS, kot je podan v uradni dokumentaciji [33].

3.4.1 Glavne značilnosti

Kot meta-operacijski sistem, ROS ponuja standardne funkcionalnosti operacijskega sistema: abstrakcijo strojne opreme, nizko-nivojsko krmiljenje naprav, implementacijo pogosto uporabljanih funkcionalnosti, pošiljanje sporočil med procesi in upravljanje s paketi. Poleg tega ponuja tudi orodja in knjižnice za pisanje in izvajanje programske kode na mreži računalnikov.

Uradno je podprt samo za operacijski sistem Ubuntu Linux in Mac OS X, deloma za ostale platforme Linux, verzija za operacijski sistem Windows pa je še v eksperimentalni fazi. Trenutno podprti programski jeziki so Python, C++ in Lisp, od katerih je C++ najbolje podprt.

3.4.2 Arhitektura in osnovni koncepti

ROS v zasnovi teži k modularnosti, kar omogoča lažje skaliranje. Tako so posamezne vsebinsko povezane funkcionalnosti združene v sklade (*ang. stacks*), ki jih lahko na sistem nameščamo posamezno. Z namenom hitrejšega razvoja in preglednejše kode se že obstoječe knjižnice ne vgrajujejo neposredno v ROS, ampak se zanje ustvarijo le ROS vmesniki.

Izvajanje robotskega operacijskega sistema temelji na računskem grafu (*ang. computation graph*). Ta je dejansko omrežje procesov ROS, ki so med seboj povezani po načelu točka v točko (*ang. peer to peer*) in skupaj procesirajo podatke. Osnovni koncepti računskega grafa, ki so prikazani tudi na sliki 3.5, so naslednji:

- **Vozlišče** *(ang. node)*: proces, ki izvaja računanje. Zaradi visoko modularne zasnove ROS-a, posamezno vozlišče ponavadi opravlja samo eno sorazmerno preprosto nalogo (npr. upravljanje motorja kolesa, branje podatkov iz senzorja, računanje odometrije, ...).

- **Glavno vozlišče** *(ang. ROS master)*: skrbi za registracijo imen vozlišč, ki se pridružijo računskemu grafu, in imensko poizvedbo po celotnem grafu. To omogoča, da se vozlišča najdejo med seboj, izmenjujejo sporočila ali kličejo storitve. Glavno vozlišče moramo vedno zagnati pred vsemi ostalimi.

- **Parametrični strežnik** *(ang. parameter server)*: omogoča globalno hrambo podatkov v obliki slovarja in je trenutno implementiran kot del glavnega vozlišča.

- **Sporočila** *(ang. messages)*: preprosta podatkovna struktura, ki je sestavljena iz polj. Vsako polje vsebuje v naprej določen tip podatkov. Obliko nestandardnih sporočil določimo sami s posebno datoteko.

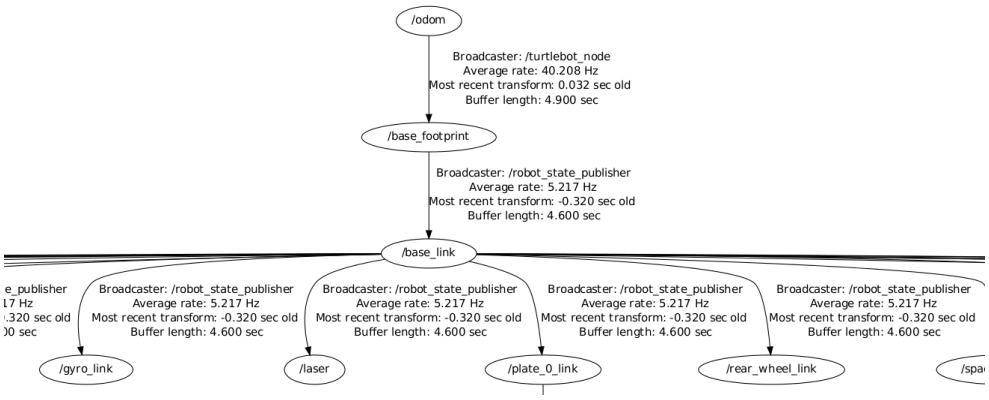
- **Teme** *(ang. topics)*: omogočajo izmenjavo podatkov; so imena, ki označujejo vsebino podatkov v sporočilu. Vozlišča si izmenjujenjo podatke z objavljanjem in naročanjem na teme. Na isto temo lahko hkrati

Slika 3.5: Prikaz glavnih konceptov robotskega operacijskega sistema. Vozlišče1 kliče storitev, ki jo ponuja Vozlišče2. To hkrati obljavlja podatke na temo Tema, na katero je naročeno Vozlišče3.

objavlja več vozlišč in prav tako je lahko na isto temo naročenih več vozlišč. Vozlišča, ki objavljajo in so naročena na določeno temo, na začetku ne vedo drug za drugega. Poiščejo se s pomočjo glavnega vozlišča, nato pa vzpostavijo medsebojne povezave.

- **Storitve** *(ang. services)*: omogočajo izvajanje strežniškega modela *zahteva / odgovor (ang. request / reply)*. Definirana so z zgradbo para sporočil (zahteve in odgovora) ter imenom. Vozlišče, ki potrebuje neko storitev, pošlje zahtevo na ime storitve in počaka na odgovor. Vozlišče, ki storitev ponuja, pa zahtevo sprejme, obdela ter vrne odgovor. Za programerja celoten proces izgleda kot klic oddaljene procedure (*ang. remote procedure call*).

- **Vreče** *(ang. bags)*: format za shranjevanje in ponovno predvajanje ROS sporočil. Uporabne so predvsem pri razvoju in testiranju algoritmov, ki uporabljajo podatke iz raznih senzorjev, saj omogočajo razvoj brez stalnega dostopa do senzorja in večkratno ovrednotenje algoritmov na istih podatkih.

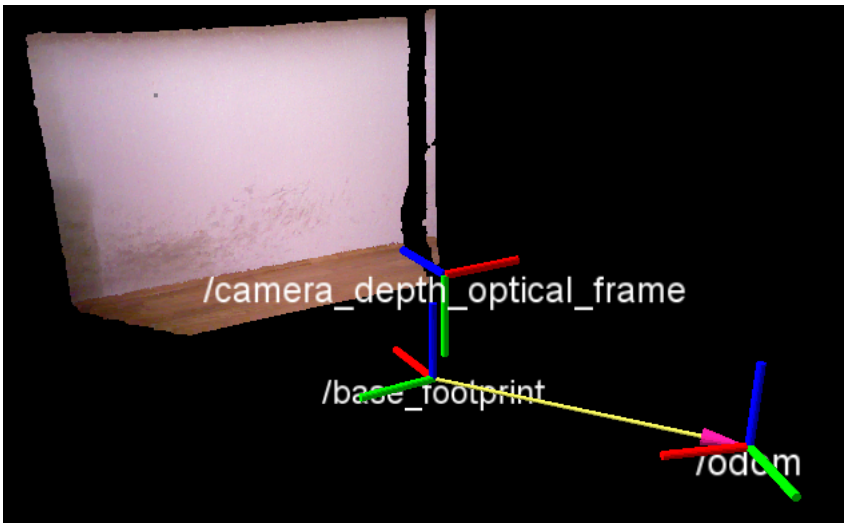


Slika 3.6: Izsek iz drevesa transformacij mobilne platforme TurtleBot. Na vrhu hierarhije je transformacija *odom*, ki predstavlja globalni koordinatni sistem.

3.4.3 Višje-nivojski koncepti

Poleg osnovnih konceptov ROS nudi tudi dodatne funkcionalnosti skozi višjenivojske koncepte. Tukaj jih naštejemo in podrobneje opišemo tiste, ki smo jih uporabili v našem sistemu:

- **Koordinatni sistemi/transformacije** *(ang. coordinate frames/transforms)*: paket *tf* omogoča uporabniku hranjenje množice koordinatnih sistemov skozi čas. Med seboj odvisni koordinatni sistemi so organizirani v drevesno strukturo. Paket omogoča tudi transformacije točk in drugih geometrijskih primitivov med koordinatnimi sistemi, še več; omogoča transformacije v nek koordinatni sistem v točno določenem (preteklem) času. Osnovni dve interakciji, ki ju imajo vozlišča s paketom *tf*, sta oddajanje/objavljanje transformacij in poslušanje/prejemanje transformacij. Proces pridobivanja koordinatnih sistemov torej ni centraliziran, ampak je porazdeljen med vozlišči, ki oddajajo transformacije. Primer drevesne strukture transformacij vidimo na sliki 3.6, na sliki 3.7 pa so prikazani glavni koordinatni sistemi, ki jih uporabljamo v našem sistemu.



Slika 3.7: Prikaz treh glavnih transformacij, ki jih uporabljamo v našem sistemu: globalni koordinatni sistem *odom*, lokalni koordinatni sistem TurtleBota *base_footprint* ter lokalni koordinatni sistem globinske kamere Kinect *camera_depth_optical_frame*.

- **Akcije/naloge** *(ang. actions/tasks)*: paket *actionlib* omogoča programiranje prekinljivih (*ang. preemptable*) nalog in interakcijo z njimi.

- **Ontologija sporočil** *(ang. message ontology)*: sklad *common_msgs* združuje sporočila, ki so široko uporabljana v paketih ROS. To omogoča lažje upravljanje odvisnosti med skladi in odpravlja krožno odvisnost. Glavne skupine sporočil so sporočila za akcije (*actionlib_msgs*), diagnostična sporočila (*diagnostic_msgs*), geometrijski primitivi (*geometry_msgs*), navigacija robotov (*nav_msgs*) in senzorska sporočila (*sensor_msgs*).

- **Vtičniki** *(ang. plugins)*: paket *pluginlib* omogoča programiranje in dinamično nalaganje vtičnikov.

- **Filtri** *(ang. filters)*: paket *filters* omogoča obdelavo podatkov s filtri.

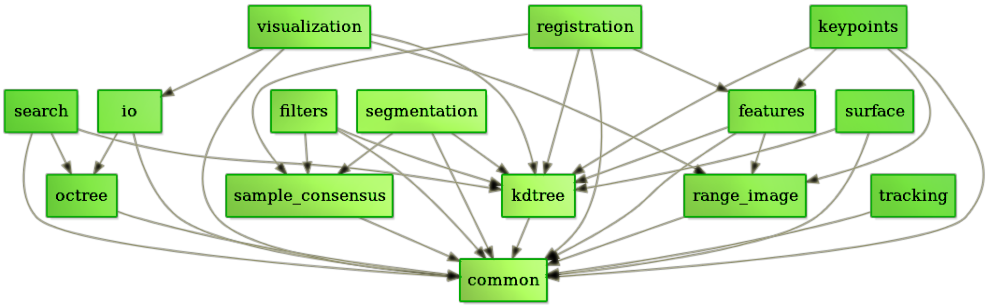
- **Model robota** *(ang. robot model)*: paket *urdf* omogoča interpretiranje XML datotek, ki vsebujejo opis modela robota v formatu URDF (*Unified Robot Description Format*). Omenjeni format nam omogoča opis robota s pomočjo povezav (*ang. links*) in sklepov (*ang. joints*). Povezavam in sklepom lahko določimo lastnosti; sklepom lahko določimo pozicijo, maso, vztrajnost..., sklepom pa npr. os, okrog katere se vrtijo. Za lažje opisovanje robotov v formatu URDF ima ROS na voljo tudi makro jezik *xacro*.

  Spodaj lahko vidimo primer opisovanja robota v formatu URDF. Gre za del opisa našega TurtleBota iz datoteke *turtlebot.urdf.xacro*, ki opisuje desni senzor stopnic (njegovo pozicijo, maso, vztrajnost) ter sklep, s katerim je povezan z ostalim robotom:

```
...
<link name="rightfront_cliff_sensor_link">
  <inertial>
    <mass value="0.01" />
    <origin xyz="0 0 0"/>
    <inertia ixx="0.001" ixy="0.0" ixz="0.0" iyy="0.001" iyz
        ="0.0" izz="0.001" />
  </inertial>
</link>
...
<joint name="base_rightfront_cliff_sensor_joint" type="fixed
    ">
  <origin xyz="0.15 -0.04 0.01" rpy="0 1.57079 0" />
  <parent link="base_link"/>
  <child link="rightfront_cliff_sensor_link" />
</joint>
...
```

3.5 Point Cloud Library - PCL

PCL je odprtokodna knjižnica (BSD licenca) za procesiranje oblakov točk in 2D ter 3D geometrije [34]. Prvotno je bila zasnovana v podjetju Willow



Slika 3.8: Prikaz grafa modulov knjižnice PCL. (Slika vzeta iz [34].)

Garage, danes pa jo podpira in razvija veliko število podjetij, raziskovalnih ustanov ter posameznih navdušencev. Dostopna je za vse najbolj razširjene operacijske sisteme: Linux, MacOS, Windows in celo Android ter iOS. Izvajanje na platformah z manjšo računsko močjo ji omogoča modularna zgradba, ki je predstavljena na sliki 3.8. Na njej vidimo posamezne module, ki vsebujejo vsebinsko povezane algoritme in razrede, ter povezave med njimi, ki predstavljajo medsebojno odvisnost modulov.

Kot knjižnica za obdelavo slik in oblakov točk PCL vsebuje številne sodobne algoritme za filtriranje, ocenjevanje značilnic, rekonstrukcijo površine, registracijo, segmentacijo in prileganje modelov.

V našem sistemu smo uporabili algoritme filtriranja, segmentacije, registracije ter rekonstrukcije površine. Čeprav je PCL 1.5 že vključena v distribuciji ROS-a, smo v našem sistemu uporabili novejšo, trenutno še nestabilno verzijo 1.7, saj ta vključuje nekaj dodatnih algoritmov in funkcij ki olajšajo predvsem izris rezultatov.

Poglavje 4

Sistem za zajemanje oblike predmetov in gradnjo 3D modelov

V tem poglavju opišemo konkretno implementacijo našega sistema za zajemanje 3D oblike predmetov in gradnjo njihovih modelov. V prvem podpoglavju podrobneje opišemo postopek navigacije okrog predmeta ter izbiranja leg za zajemanje globinskih podatkov. V drugem podpoglavju predstavimo način zajemanja oblakov točk ter njihovo procesiranje pred registracijo. V tretjem podpoglavju podrobneje razčlenimo uporabljen algoritem za registracijo oblakov točk, v zadnjem podpoglavju pa predstavimo še algoritme za obdelavo registriranega modela in gradnjo poligonske mreže.

4.1 Navigacija okrog predmeta

Pomembna lastnost našega sistema je avtomatiziranost celotnega postopka gradnje 3D modela. Velik del tega predstavlja samodejna navigacija platforme TurtleBot okrog predmeta, ki ga zajemamo. Kot smo že omenili v poglavju 1, predvidevamo, da se lahko robot prosto giblje po krožnici okrog predmeta, kar problem navigacije poenostavi, saj se nam ni potrebno izogi-

bati oviram. Osredotočimo se torej na postopke navigacije, ki pripomorejo k gradnji dobrega 3D modela: na minimizacijo napake odometrije, ki se uporablja pri registraciji, ter izbiro primernih leg za zajemanje 3D infromacije glede na konkreten predmet.

Za premikanje platforme TurtleBot skrbi robotski sesalnik Roomba, ki smo ga opisali v poglavju 3.2. Omejili smo se na počasno premikanje robota ter tako minimizirali napako odometrije, kar bomo še natančneje opisali v naslednjem poglavju.

ROS nam omogoča neposredno krmiljenje Roombe preko objavljanja želene linearne in kotne hitrosti na temo *cmd_vel*. Vozlišče *turtlebot_node*, ki je naročeno na to temo, pa to hitrost nato s frekvenco 30 Hz pošilja robotu, ki ustrezno nastavi hitrost koles. Za ustavitev robota preprosto objavimo hitrost 0. Z namenom zmanjšanja možnosti zdrsa koles in posledično napak odometrije, pri speljevanju robota postopoma povečujemo hitrost, pri ustavljanju pa jo postopoma znižujemo.

Načrtovanje leg, s katerih zajemamo globinske podatke, in premikanje do njih izvajamo po naslednjem postopku:

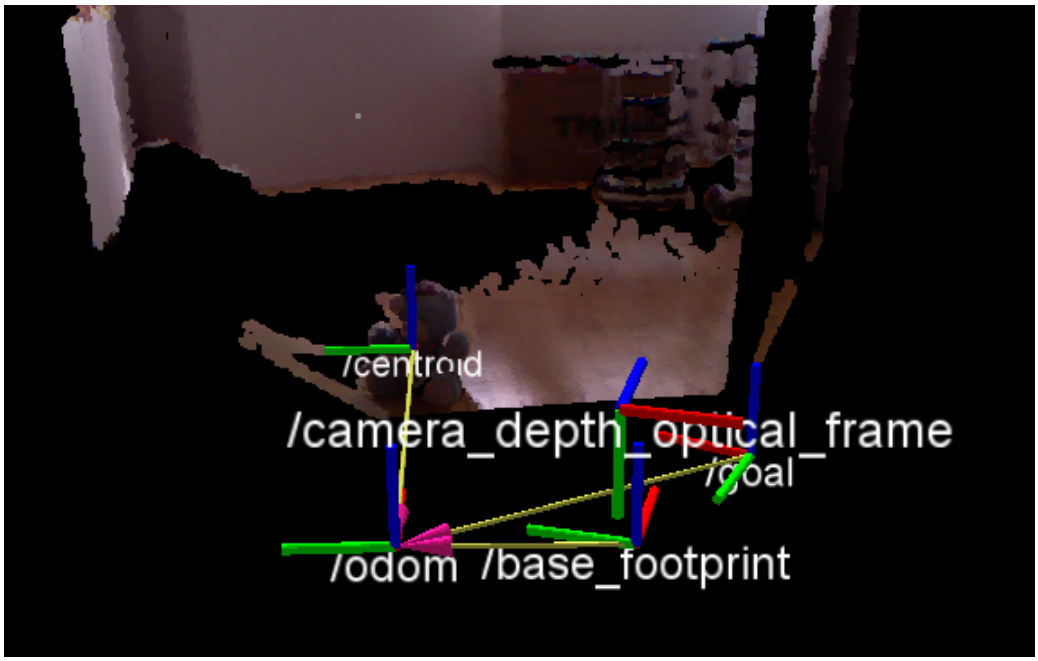
1. izračunaj centroid trenutnega modela predmeta ter prostor, ki ga zavzema na tleh,

2. na podlagi ugotovljene velikosti predmeta prilagodi velikost kota med legami skeniranja (glede na krog s središčem v centroidu predmeta) ter njihovo razdaljo od centroida predmeta,

3. izračunaj novo lego skeniranja glede na parametre izračunane v prejšnjem koraku,

4. izvedi premik robota v treh korakih: obrat proti novi poziciji, premik na novo pozicijo, obrat proti centroidu predmeta.

Navigacija platforme Turtlebot okrog predmeta, ki ga zajemamo, je prikazana na sliki 4.1.



Slika 4.1: Prikaz navigacije robota okrog predmeta. Na sliki poleg lokalnih koordinatnih sistemov Roombe (*base_footprint*) in Kinecta (*camera_depth_optical_frame*) vidimo globalno koordinatno izhodišče (*odom*), centroid predmeta (*centroid*) ter naslednjo lego (*goal*).

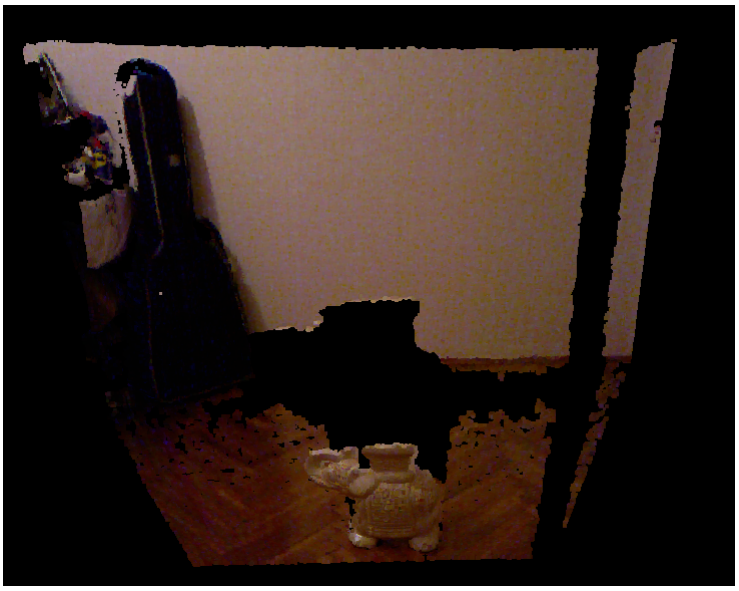
S prilagajanjem števila leg zajemanja globinskih podatkov skušamo pohitriti celoten postopek, kjer je to mogoče. Registracija večjih predmetov je navadno lažja od registracije manjših predmetov, saj prilegamo večje površine, ki so pogosto tudi razgibane, kar nam daje veliko verjetnost, da bo algoritem ICP našel pravo poravnavo. Zato pri zajemanju podatkov večjih predmetov zmanjšamo število leg, s katerih zajemamo podatke in tako pohitrimo celoten postopek gradnje 3D modela predmeta. V naslednjih dveh podpoglavjih bomo opisali in s slikami predstavili gradnjo 3D modela predmeta, ki je predstavljen na sliki 4.2.

Slika 4.2: Kip slona, ki nam bo služil za predstavitev postopka gradnje 3D modela z našim sistemom.

4.2 Zajemanje oblakov točk

Algoritem za registracijo oblakov, ki ga uporabljamo v našem sistemu, deluje z upoštevanjem zgolj 3D informacije in ne uporablja informacije o barvi. Ker pa nam Kinect nudi preprost dostop do barvnega oblaka točk, ga uporabljamo, saj nam nudi boljšo vizualizacijo predmeta na vmesnih stopnjah gradnje 3D modela. Relacije med RGB in IR kamero nismo posebej kalibrirali, saj nam je za naše potrebe zadostovala tovarniška kalibracija.

Z vsake lege zajemanja globinskih podatkov zajamemo serijo desetih oblakov točk, saj nam to omogoča zmanjšanje šuma v podatkih in zapolnjenje nekaterih lukenj v posameznih oblakih. Primer zajetega oblaka točk lahko vidimo na sliki 4.3.

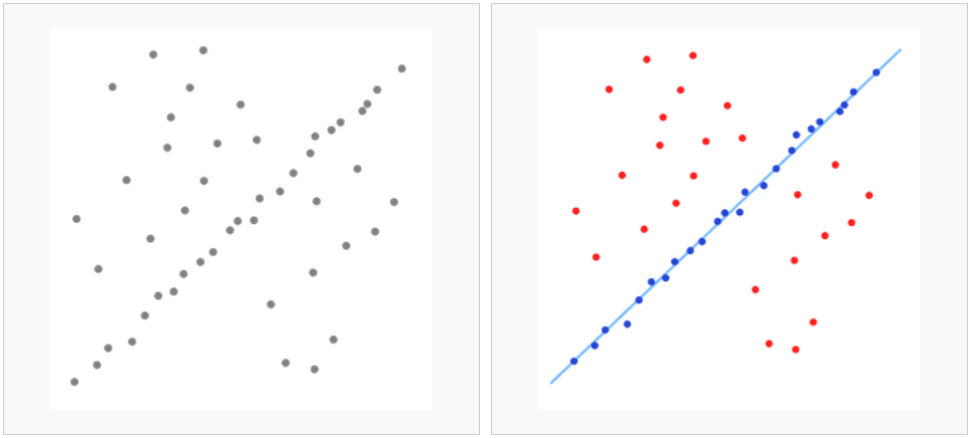


Slika 4.3: Primer zajetega oblaka točk.

4.2.1 Odstranjevanje tal

Ker nas zanima le model predmeta in ne celotnega prostora, ob zajemu oblakov točk iz njih s filtrom najprej odstranimo ozadje. To storimo tako, da okrog centroida postavimo kvader velikosti $1,5m \times 1,5m \times 1,5m$ in izločimo točke, ki so izven tega kvadra. Tako dobimo oblak točk brez ozadja, a ta še vedno vsebuje del tal. Da ob odstranjevanju tal ne bi predmeta preveč odrezali, uporabimo algoritem RANSAC (*RANdom SAmple Consensus*), ki sta ga prva predlagala Fischler in Bolles v [38]. Gre za iterativno metodo ocenjevanja parametrov matematičnega modela v množici točk, ki vsebuje tako elemente tega modela (*ang. inliers*), kot tudi točke, ki niso elementi iskanega modela (*ang. outliers*). Sam algoritem deluje po naslednjem postopku:

1. naključno izberi podmnožico vseh točk - hipotetični elementi modela,
2. prilegaj model tej podmnožici točk in oceni parametre modela,
3. prilegaj vse ostale točke trenutnemu modelu in točke, ki se dobro prilegajo, dodaj hipotetičnim elementom modela,



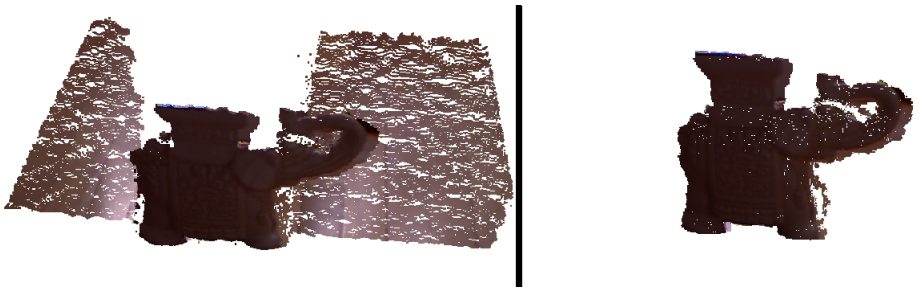
Slika 4.4: Prikaz delovanja algoritma RANSAC. Levo začetna množica točk, desno prilegan model premice ter označeni elementi modela z modro ter zavrnjene točke z rdečo. (Povzeto po [37].)

4. če ima model dovolj hipotetičnih elementov oceni njegovo napako glede na te elemente, sicer ga zavrzi,

5. postopek ponavljaj k iteracij in sproti hrani model z najmanjšo napako.

Kot je iz postopka razvidno, gre za nedeterministični algoritem, ki zagotavlja pravilen rezultat le z neko verjetnostjo, ki je odvisna od števila iteracij. Na sliki 4.4 lahko vidimo primer prileganja premice, v našem primeru pa smo prilegali ravnino in dobili model v obliki enačbe $ax + by + cz = 0$. Točke, ki so bile del modela, smo nato odstranili in tako dobili oblak točk, ki je predstavljal samo 3D informacijo predmeta, katerega 3D model smo želeli zgraditi (slika 4.5).

Ker algoritem RANSAC ne zagotavlja rešitve v vsakem primeru, smo za tak slučaj implementirali tudi odstranjevanje tal s filtriranjem oblaka točk po višini, ki pa poleg tal odstrani tudi nekaj točk samega predmeta.



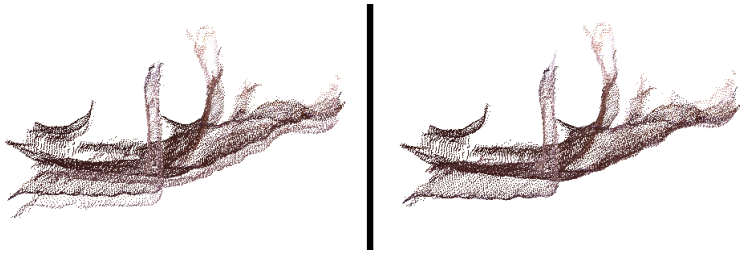
Slika 4.5: Prikaz oblaka točk z odstranjenim ozadjem (levo) ter oblaka točk z odstranjenim ozadjem in tlemi (desno).

4.2.2 Zmanjševanje šuma

Za zmanjševanje šuma izkoriščamo strukturo urejenih oblakov točk, ki jih lahko obravnavamo kot globinske slike. Vsak oblak filtriramo z medianinim filtrom, kjer je velikost filtrirnega okna 5x5 slikovnih elementov. Medianin filter smo izbrali, ker dobro filtrira impulziven šum, hkrati pa ob izbrani velikosti okna ne pokvari robov. Tako filtrirano celotno serijo oblakov točk nato s povprečenjem po slikovnih elementih združimo en oblak točk. Preden začnemo z registracijo oblaka, mu skušamo odstraniti še šum na zunanjih robovih oblaka. To storimo tako, da najprej ugotovimo skrajne robove minimalnega kvadra, ki zajema vse točke oblaka, nato pa ta kvader v vsaki dimenziji zmanjšamo za 6 odstotkov velikosti kvadra v tej dimenziji. Točke, ki ostanejo zunaj novega kvadra, odstranimo.

4.3 Registracija oblakov točk

Registracijo oblakov točk, ki jih dobimo iz serij oblakov točk, opravimo postopoma - na vsaki poziciji registriramo nov oblak točk. Za registracijo uporabljamo Algoritem ICP, ki smo ga podrobneje opisali že v poglavju 2.1.2 in je tudi že realiziran v knjižnici PCL. Da se nam napaka odometrije ne bi akumulirala za vsako novo registracijo, vedno registriramo samo sosednje oblake točk, nato pa dobljeno transformacijo pomnožimo s transformacijo



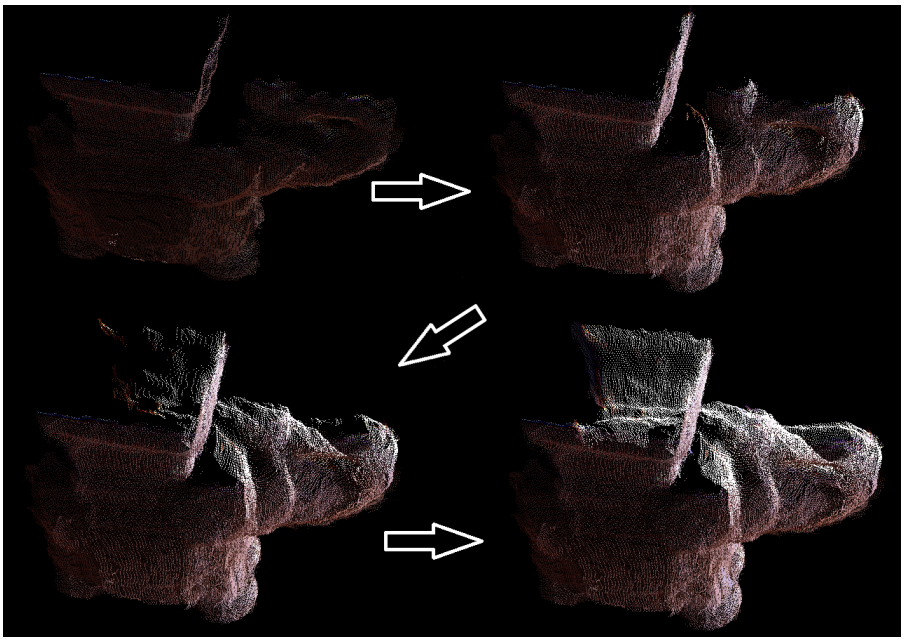
Slika 4.6: Registracija dveh oblakov: levo pred izvajanjem algoritma ICP, desno po izvajanju.

prejšnjega oblaka. To nazadnje uporabimo za poravnavo novega oblaka točk z ostalimi. Registriranje oblakov točk si lahko ogledamo tudi na sliki 4.6 in 4.7.

Sledi opis uporabljane različice algoritma ICP:

- oblak točk, ki ga prilegamo, decimiramo z mrežo vokslov resolucije 0,25 cm^3, da zajamemo celotno obliko oblaka točk, hkrati pa precej pohitrimo izvajanje algoritma,
- vse točke decimiranega oblaka točk prilegamo vsem točkam nedecimiranega oblaka točk zajetega s prejšnje lege,
- korespondenčne pare točk med oblakoma določimo po principu najbližje točke (upoštevanje zgolj 3D informacije),
- vsi korespondenčni pari točk so enako uteženi, zavrnemo pa korespondenčne pare, katerih razdalja presega podan prag; ta prag po desetih iteracijah zmanjšamo za boljšo natančno poravnavo,
- metrika napake, ki jo minimiziramo, je vsota kvadratov razdalj,
- računanje transformacije se izvede s pomočjo metode SVD.

Algoritem ICP se ustavi, ko doseže enega od treh pogojev: število iteracij doseže maksimalno dovoljeno vrednost ali evklidska razdalja med oblakoma



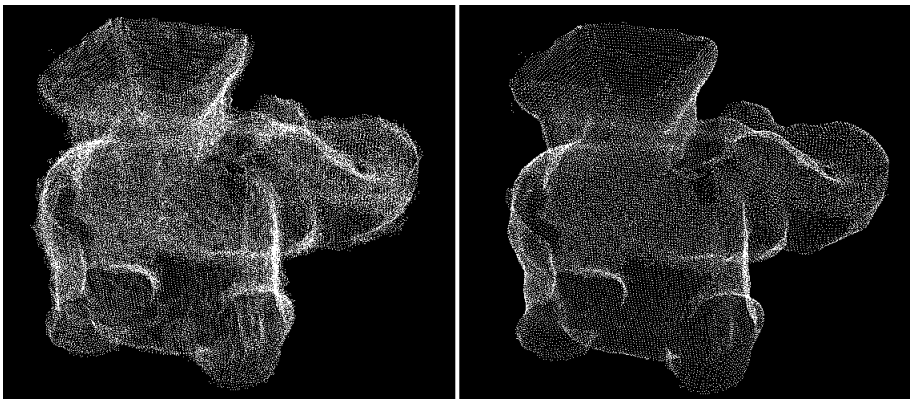
Slika 4.7: Prikaz sestavljanja 3D modela.

je manjša od podane vrednosti ali razlika med izračunanimi transformacijami v zadnjih dveh iteracijah je manjša od podane vrednosti.

Ko je registracija vseh oblakov končana, še enkrat izvedemo decimacijo oblaka z mrežo vokslov resolucije 0,25 cm^3, kar nam malo poslabša ločljivost oblaka, vendar pa hkrati združi točke, ki se morda slabše prekrivajo in tako za majhno ceno izniči efekt dvojne površine. Rezultat postopka lahko vidimo na sliki 4.8.

4.4 Gradnja poligonske mreže

Ko s TurtleBotom obkrožimo predmet, imamo že na voljo 3D model predmeta v obliki oblaka točk. Na tej točki izvedemo še glajenje z algoritmom gibajočih najmanjših kvadratov (*moving least squares - MLS*) [35]. Omenjeni algoritem postopoma poravna normale celotnega oblaka točk na podlagi bližnjih okolic točk. Če želimo, lahko tako dobljen 3D model shranimo v obliki barvnega



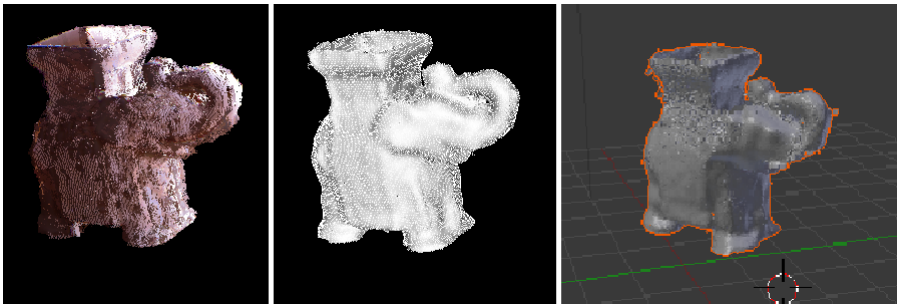
Slika 4.8: Prikaz decimacije vseh registriranih oblakov skupaj. Levo množica registriranih oblakov, desno decimiran oblak.

oblaka točk.

Ker veliko aplikacij uporablja 3D modele v obliki poligonskih mrež, tudi naš sistem ponuja hitro pretvorbo oblaka točk v poligonsko mrežo. Za to uporabljamo algoritem požrešna projekcijska triangulacija (*ang. greedy projection triangulation*) [36], s katerim lahko trianguliramo neorganizirane oblake točk, sestavljene iz več (registriranih) oblakov točk. Najbolje deluje v primeru, ko je površje oblaka zglajeno in ni hitrih prehodov med področji z veliko gostoto točk in majhno gostoto točk. Vse to velja tudi za naš 3D model v obliki oblaka točk.

Metoda deluje tako, da v seznamu hrani točke, na katere se poligonska mreža lahko razširi (robne točke ali *ang. fringe points*) ter ta seznam razširja dokler vse možne točke niso povezane. Triangulacija se izvaja lokalno s projiciranjem sosednjih točk na normalo sredinske točke in povezovanjem še nepovezanih točk.

Omenjeni algoritem je implementiran tudi v knjižnici PCL. V njej prav tako najdemo algoritem za shranjevanje polinomskih mrež v obliki datotek .obj, ki ga prav tako uporabimo. S tem je gradnja 3D modela končana. Na sliki 4.9 si lahko ogledamo končni rezultat tako v obliki oblaka točk kot poligonske mreže.



Slika 4.9: Prikaz končnega rezultata gradnje 3D modela. Levo v obliki barvnega oblaka točk, na sredimi v obliki poligonske mreže, desno datoteka .obj vizualizirana v programu Blender.

Poglavje 5

Vrednotenje platforme in rezultati

V tem poglavju bomo predstavili in analizirali rezultate v prejšnjem poglavju opisanega sistema za gradnjo 3D modelov. Analizo bomo opravili na štirih predmetih različnih oblik in velikosti ter ene sestavljene scene. Kakovost pridobljenih modelov bomo ocenjevali vizualno, hkrati pa bomo opisali težave, s katerimi se sistem sooča v posameznih primerih in komentirali kako uspešno jih lahko rešimo z dodatnim nastavljanjem parametrov sistema. Pred tem pa bomo najprej ovrednotili platformo, na kateri smo gradili, oziroma njene dele. Ovrednotili bomo torej odometrijo robotskega sesalnika Roomba ter natančnost barvno-globinske kamere Kinect.

5.1 Ovrednotenje platforme

Ovrednotenje kakovosti podatkov, ki jih imamo na voljo preko robotske platforme TurtleBot, je ključnega pomena za ocenjevanje celotnega sistema. Poleg tega lahko sistem izboljšamo le, če poznamo njegove šibkosti in jih znamo popraviti ali se jim znamo izogniti.

5.1.1 Natančnost odometrije robotskega sesalnika Roomba

Roomba meri svojo lego v prostoru le na podlagi vrtenja pogonskih koles. Ta ne ponujajo zelo natančnih meritev, poleg tega pa se napaka skozi čas samo povečuje. Delno lahko izboljšamo odometrijo s pomočjo kalibracijske skripte, ki nam jo ponuja ROS. Kalibracijo opravimo tako, da platformo TurtleBot postavimo približno 30 cm pred ravno steno, nato pa poženemo kalibracijsko skripto. TurtleBot se nato štirikrat zavrti okrog svoje osi z različnimi hitrostmi, s senzorjem Kinect pa zazna kdaj je naredil obrat. Na podlagi tega in odometrije koles kalibracijska skripta izračuna faktor popravka, ki ga vpišemo v zagonsko skripto Roombe.

Natančnost odometrije je odvisna tudi od načina vožnje Roombe in od podlage; sunkovito premikanje, smeti ali spolzka tla povzročajo zdrse koles in s tem napačno odometrijo. Robot Create ima zato dodan enoosni giroskop, ki blaži napake odometrije ob zdrsih koles. V našem primeru tega nismo imeli na voljo, zato smo skušali z dobro kalibracijo in previdno vožnjo čim bolj zmanjšati napako odometrije. To smo storili tako, da smo se pri premikanju TurtleBota omejili na nizke hitrosti in za tak način premikanja tudi primerno kalibrirali Roombo. Za potrebe našega sistema torej nismo upoštevali predlaganega popravka, ampak smo s poizkušanjem optimizirali točnost odometrije pri počasnem premikanju Roombe. Kalibracijska skripta namreč izračuna napako odometrije pri vsaki hitrosti posebej, na koncu pa predlaga kompromisni popravek za vse štiri hitrosti. Za naš sistem smo optimizirali natančnost odometrije le za prvi dve hitrosti.

5.1.2 Natančnost globinske kamere

Da bi ocenili velikost napake in šuma v globinskih podatkih senzorja Kinect in določili najprimernejšo razdaljo za zajemanje globinskih podatkov, smo izvedli analizo natančnosti globinske kamere. Predvsem nas je zanimala velikost in porazdelitev šuma, saj lahko sistemsko napako (zamik celotnega

Slika 5.1: Predalnik, s pomočjo katerega smo opravili analizo senzorja Kinect.

oblaka) odpravimo z izvajanjem algoritma ICP.

Postopek

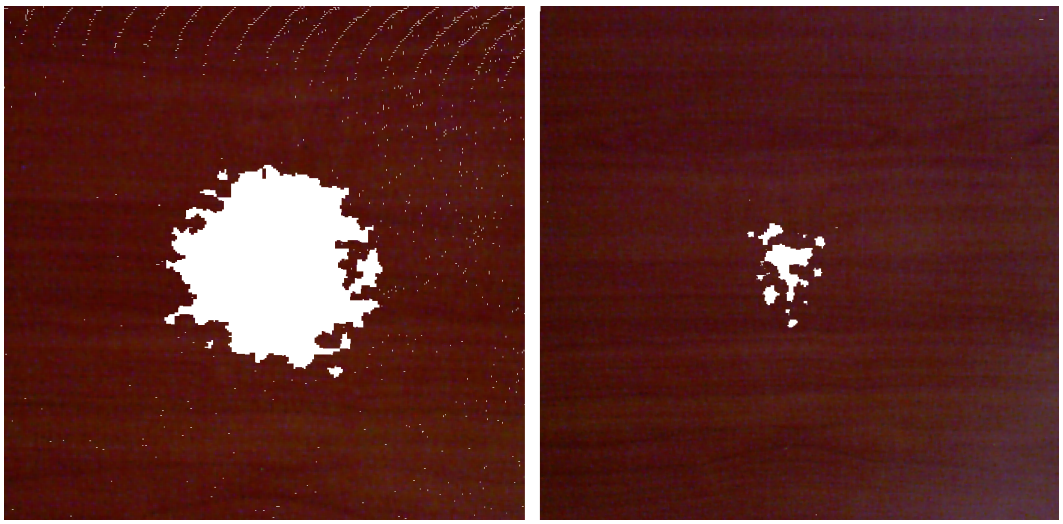
Analizo smo izvedli na ravni površini predalnika velikosti 40x40 cm (slika 5.1), ki smo jo zajemali na razdaljah od 0,5 m do 1,5 m z razmakom 10 cm med posameznimi posnetki. V posnetkih smo najprej s filtrom odstranili vse točke razen ravnine predalnika, nato pa smo na preostale točke z že opisanim algoritmom RANSAC prilegali ravnino. Ker je bila ravnina predalnika vedno pravokotno obrnjena proti Kinectu, smo razdaljo do predalnika izračunali z razdaljo do prilegane ravnine. S primerjanjem izračunane in izmerjene razdalje do predalnika smo ocenili napako, ki jo senzor Kinect napravi na posamezni razdalji. Z računanjem standardnega odklona odmika točk od prilegane ravnine pa smo ocenili šum.

$d_{izmerjena}[cm]$	$\overline{d}_{Kinect}$ $[cm]$	$\overline{\sigma}_{tock}$ $[cm]$
50	50,07	0,14
60	60,12	0,19
70	69,08	0,23
80	79,92	0,25
90	89,90	0,25
100	99,85	0,31
110	109,90	0,35
120	119,75	0,25
130	129,31	0,35
140	139,51	0,34
150	149,66	0,41

Tabela 5.1: Rezultati analize senzorja Kinect na podlagi prileganja ravnine. Prvi stolpec: izmerjena razdalja do predalnika, drugi stolpec: povprečna razdalja do prilegane ravnine, tretji stolpec: povprečni standardni odklon odmika točk od prilegane ravnine.

Rezultati

Z vsake razdalje smo izvedli več meritev. Povprečni rezultati so prikazani v tabeli 5.1, iz katere lahko pričakovano razberemo, da je bolje zajemati podatke z manjših razdalj. Napaka senzorja je bila povsod majhna, še posebej če upoštevamo tudi to, da naša meritev razdalje ni bila do milimetra natančna. Šum, ki smo ga predstavili s povprečnim standardnim odklonom razdalje do prilegane ravnine, je z razdaljo naraščal. Vendar moramo opozoriti, da razdalje pod 65 cm niso primerne za zajemanje globinski podatkov s senzorjem Kinect, saj na tako majhni razdalji senzor ne more določiti globine, kar lahko tudi lepo vidimo na sliki 5.2. Za najbolj primerno razdaljo smo torej izbrali razdaljo 70 cm od površine predmeta.



Slika 5.2: Prikaz stranice predalnika na razdaljah 50 (levo) in 60 (desno) cm. V sredini oblaka točk opazimo luknjo, kjer senzor Kinect zaradi premajhne razdalje ni zmogel oceniti globine.

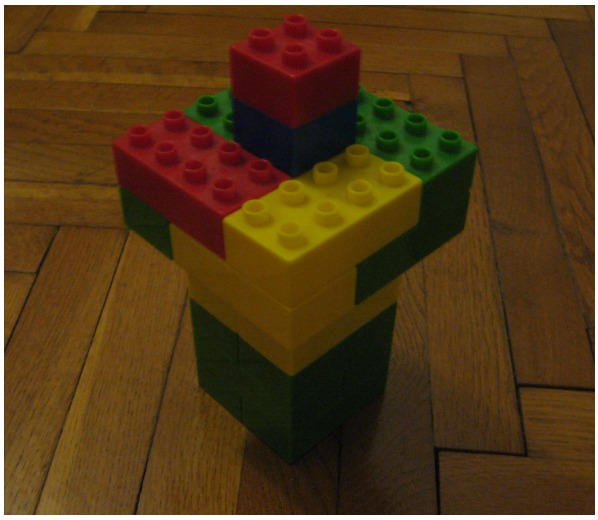
5.2 Rezultati

Sledi predstavitev dobljenih rezultatov zgrajenega sistema za gradnjo 3D modelov in njihovo vrednotenje.

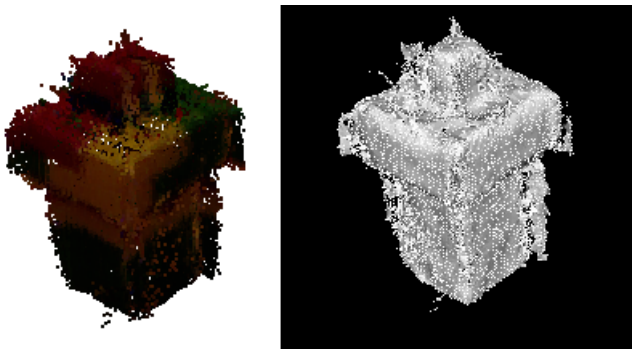
5.2.1 Manjši predmet

Za primer manjšega predmeta smo si izbrali stolpič iz lego kock (slika 5.3). Kot smo že v prejšnjem poglavju opisali, je sistem prepoznal, da gre za manjši predmet in predmet zajel iz 12-ih zornih kotov (namesto iz 8-ih, kot večje predmete). Sistem je uspešno zgradil model, ki ujame glavno obliko predmeta, a hkrati vsebuje precej šuma, kot lahko vidimo tudi na slikah 5.4 in 5.5.

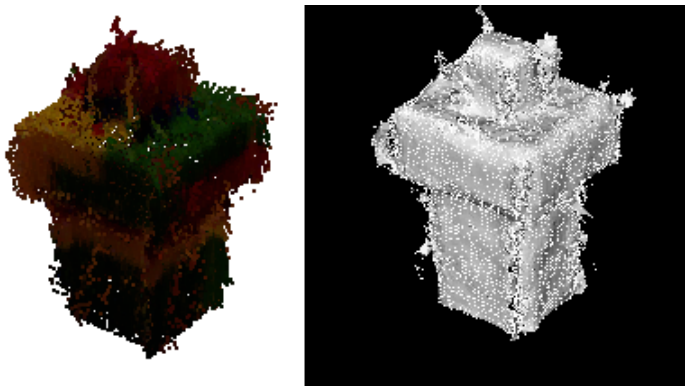
Pri majhnih predmetih imamo za registracijo na voljo malo točk. Vendar samo število točk ni problem, saj smo enak rezultat dobili v poskusih z nedecimiranim oblakom. Težava je v razporejenosti točk po prostoru. Ker so vse



Slika 5.3: Stolpič lego kock kot primer majhnega predmeta.



Slika 5.4: Končni model stolpiča lego kock v obliki poligonske mreže in barvnega oblaka točk.



Slika 5.5: Končni model stolpiča lego kock v obliki poligonske mreže in barvnega oblaka točk iz drugega zornaga kota.

točke skupaj na majhnem prostoru, algoritem ICP v kombinaciji s slabšo odometrijo napačno registrira posamezne oblake točk. Da ublažimo ta problem, moramo, poleg zajemanja več oblakov točk, tudi dodatno zmanjšati razdaljo za sprejemanje korespondenc pri algoritmu ICP. Z omenjenimi ukrepi nam je uspelo precej izboljšati rezultate, saj registracija ni več čisto napačna, ampak se pojavlja le rahlo križanje med poravnanimi oblaki.

Na omenjenih slikah so lepo vidne tudi posamezne točke, ki 'štrlijo' iz modela ter zelo vidno pokvarijo triangulacijo oblaka točk. To se zgodi zaradi nepopolno odstranjenega šuma na robovih oblakov, ki je pri zajemanju majhnih predmetov še bolj opazen kot pri zajemanju večjih predmetov. Ta problem nas ne skrbi tako kot problem napačne registracije, saj bi te točke oziroma poligone lahko odstranili naknadno s postprocesiranjem, a to ni bila tema naše diplomske naloge in se zato s tem nismo ukvarjali.

5.2.2 Večji predmeti

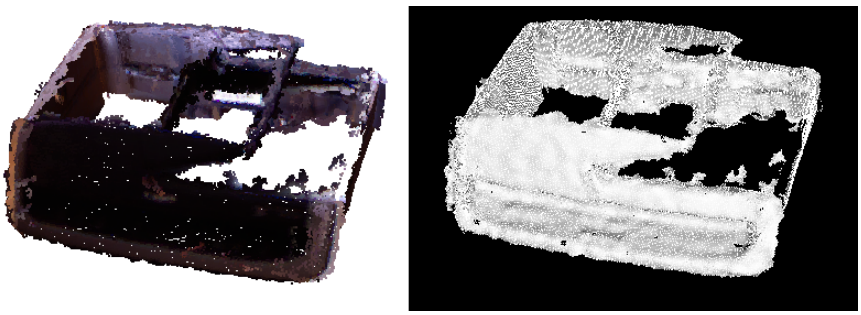
Pri večjih predmetih je sistem bolje deloval, zato nas je v tej točki zanimalo predvsem delovanje sistema glede na obliko predmeta. Tako smo analizirali delovanje sistema na razgibanem kvadrastem predmetu, tankem predmetu ter cilindričnem predmetu. V nadaljevanju predstavimo vsak primer posebej.

Slika 5.6: Tiskalnik kot primer večjega kvadrastega predmeta.

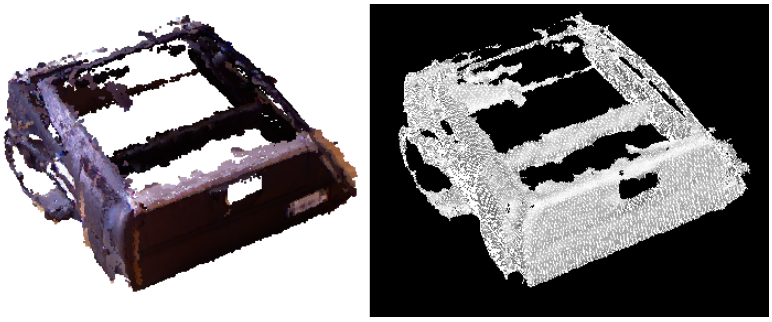
Razgiban kvadrast predmet

Kot primer kvadrastega predmeta smo vzeli tiskalnik, kot je viden na sliki 5.6. Ker gre za večji predmet, ga je sistem skeniral samo z 8 pozicij, kar pa je kljub temu zadostovalo za dobro registracijo (sliki 5.7 in 5.8), saj so bile točke v nasprotju s stolpičem lego kock precej bolj razporejene po prostoru. Naša verzija algoritma ICP take predmete dobro registrira - sosednji oblaki se lepo prekrivajo in ne pride do križanja. Vidimo lahko tudi, da je bil v tem primeru šum na robovih bolje odstranjen, saj smo v zglajenem modelu dobili gladke robove, kakršni so tudi v resnici. Rezultat nam malo pokvarijo le posamezne luknje v modelu, kjer je površina predmeta odsevna. Znano je namreč, da ima senzor Kinect težave z zaznavanjem takih površin.

Pri tem predmetu lahko vidimo tudi omejitev same platforme, ki smo jo uporabljali. Zaradi nizke namestitve senzorja Kinect namreč ni bilo mogoče zajeti vrhnje ploskve predmeta, zato ima dobljeni model le stranske ploskve.



Slika 5.7: Končni model tiskalnika v obliki poligonske mreže in barvnega oblaka točk.



Slika 5.8: Končni model tiskalnika v obliki poligonske mreže in barvnega oblaka točk iz drugega zornega kota.

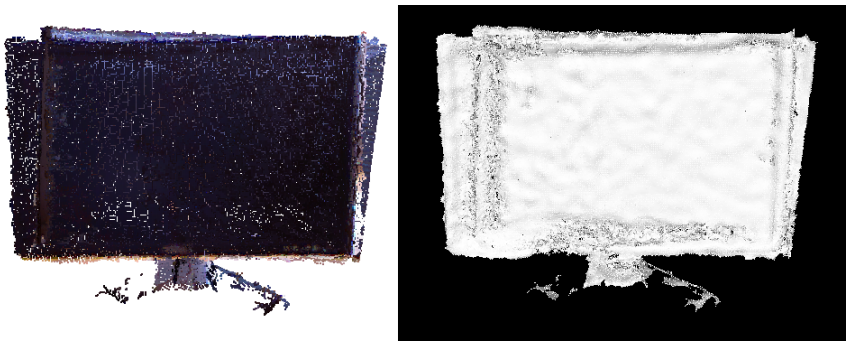
Slika 5.9: Zaslon kot primer večjega tankega predmeta.

Tanek predmet

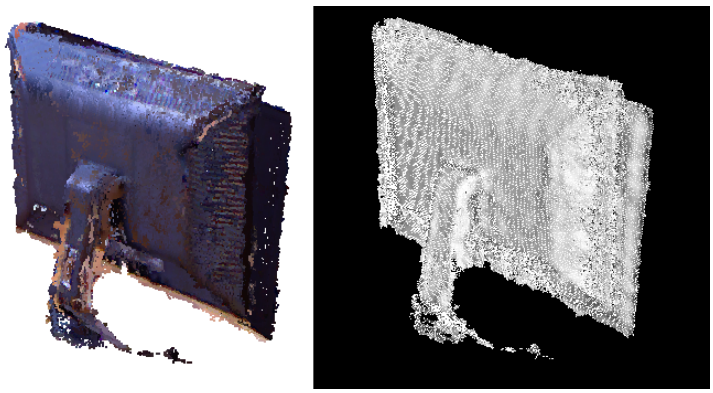
Za primer tankega predmeta smo vzeli zaslon osebnega računalnika (slika 5.9). Čeprav gre tu za večji predmet, pri katerem so točke porazdeljene po prostoru, lahko rečemo, da registracija predmeta ni uspela. Kot lahko vidimo tudi na slikah 5.10 in 5.11, je namreč sprednja stran zaslona zamaknjena glede na zadnjo. Razlog za to je v sami obliki predmeta - ker sta stranski stranici tako ozki, smo imeli pri registraciji oblakov točk, ki ju vsebujejo, probleme. Tu namreč točke niso porazdeljene po veliki površini in zato naletimo na enake probleme kot pri majhnem predmetu. Še dodatno pa tu naletimo na problem povečanja robnega šuma, saj dolgo stranico zaslona z nekaterih leg zajemamo skoraj vzporedno, pri čemer ima senzor Kinect večje težave s šumom (slabši podatki celotne stranice ter podaljšan rob).

Cilindričen predmet

Nekateri sistemi za registracijo oblakov točk imajo probleme s cilindričnimi predmeti, zato smo se tudi mi odločili testirati na takem primeru. Za testni predmet smo izbrali okrogel koš za dežnike. Izkazalo se je, da naš sistem s takimi predmeti nima težav, kar pokažeta tudi sliki 5.13 in 5.14. Vsi oblaki so pravilno registrirani, rjave črte na barvnem oblaku točk pa so le posledica zamika med barvno in globinsko kamero, ki je še vedno prisoten, kljub uporabi



Slika 5.10: Končni model zaslona v obliki poligonske mreže in barvnega oblaka točk.



Slika 5.11: Končni model zaslona v obliki poligonske mreže in barvnega oblaka točk iz drugega zornega kota.

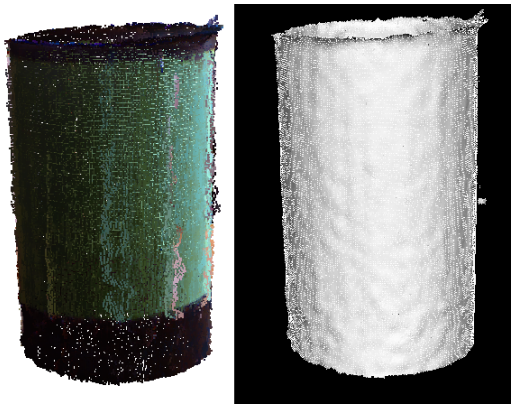
Slika 5.12: Koš za dežnike kot primer večjega cilindričnega predmeta.

tovarniške kalibracije obeh kamer. Čeprav smo zajemali mrežast predmet (slika 5.12), ki je bil le delno prekrit, seveda nismo pričakovali mrežastega modela, saj gre za premajhen detajl. Smo pa ob tem ugotovili, da s senzorjem Kinect lahko zajamemo tudi mrežaste površine.

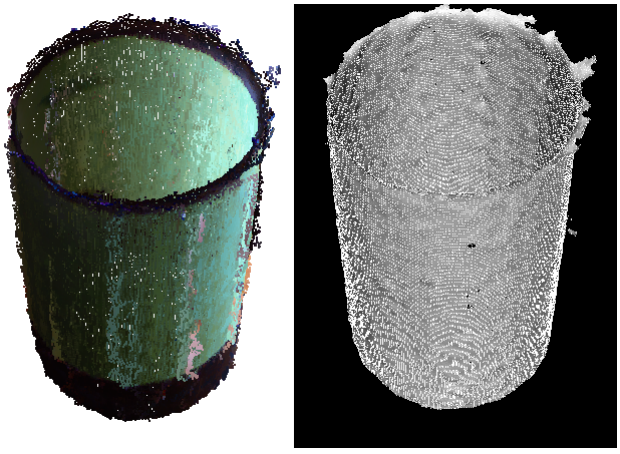
5.2.3 Razgibana scena

Sistem smo nazadnje testirali še na malo večji razgibani sceni sestavljeni iz različnih predmetov: škatel, pločevink, igrač. Zaradi večje površine, ki jo je scena zajemala na tleh, je sistem navigiral v večjem krogu, kot pri ostalih predmetih. Kljub temu smo dobili sorazmerno dobre rezultate, ki jih lahko vidimo tudi na slikah 5.15 in 5.16.

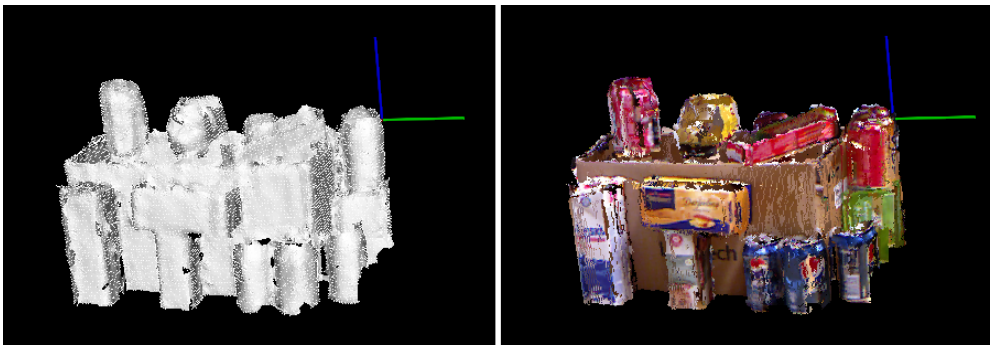
Zaradi velike razgibanosti in širine v vsakem zajetem oblaku točk je sistem dobro registriral vse oblake točk. Dobro lahko razločimo vsako posamezno škatlico in pločevinko. Nepopolnost modela, ki se je pojavila v tem primeru, je posledica zgradbe scene in načina zajemanja oblakov. Nekaterih delov predmetov namreč sistem ni zajel, ker so vidni le iz ozkega zornega kota, ki ga sistem pri izbiranju leg ni pokril. Delno bi lahko ta problem rešili z



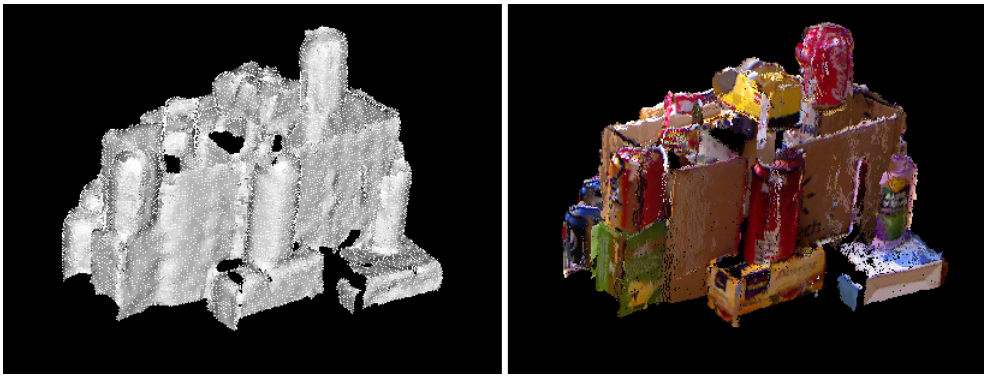
Slika 5.13: Končni model koša za dežnike v obliki poligonske mreže in barvnega oblaka točk.



Slika 5.14: Končni model koša za dežnike v obliki poligonske mreže in barvnega oblaka točk iz drugega zornega kota.



Slika 5.15: Končni model večje razgibane scene v obliki poligonske mreže in barvnega oblaka točk.

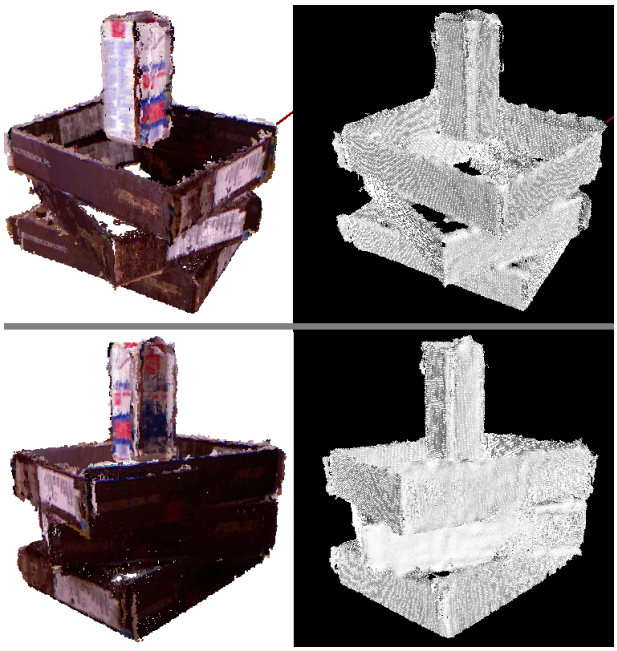


Slika 5.16: Končni model večje razgibane scene v obliki poligonske mreže in barvnega oblaka točk iz drugega zornega kota.

zajemanjem oblakov točk iz večih leg, kar pa bi posledično pomenilo daljši celoten proces gradnje 3D modela.

5.2.4 Splošna ocena

Ugotovili smo, da sistem dobro deluje na večini večjih predmetov, probleme pa ima z manjšimi in tankimi predmeti ter takimi, ki jih senzor Kinect ne more zajeti. Sistem največ časa porabi za navigacijo okrog predmeta zaradi počasne vožnje, ki zmanjša verjetnost napak v odometriji. Registracija z



Slika 5.17: Model scene sestavljene iz škatel v obliki oblaka točk in poligonske mreže. Prikaz iz dveh zornih kotov.

našo različico algoritma ICP je dokaj hitra, saj se povprečno število iteracij giblje okrog trideset - deset iteracij v fazi grobe poravnave in deset do trideset iteracij v fazi fine poravnave. Na slikah 5.17, 5.18 in 5.19 lahko vidimo še nekaj primerov uspešne gradnje 3D modelov.

Slika 5.18: Model klobuka v obliki barvnega oblaka točk in poligonske mreže prikazan iz dveh zornih kotov.

Slika 5.19: Model človeka v obliki barvnega oblaka točk in poligonske mreže prikazan iz več zornih kotov.

Poglavje 6

Zaključek

Za namen gradnje tridimenzionalnih modelov smo uspešno uporabili mobilno platformo Turtlebot in barvno-globinsko kamero Kinect. Zgradili smo sistem, ki za gradnjo tridimenzionalnega modela uporablja odometrijo robota ter oblake točk posnete iz nekaj zornih kotov okrog predmeta, glavni del sistema pa je prilagojeni algoritem za registracijo oblakov točk ICP. Sistem se pri zajemanju podrobnosti ne more kosati z nekaterimi najnovejšimi pristopi ali natančnejšimi senzorji, vendar pa kljub temu uspešno zgradi modele predvsem večjih predmetov, kjer ujame spremembe v obliki, ki so večje od 5 mm.

6.1 Prednosti in omejitve sistema

Bistvena prednost sistema izhaja iz uporabljene mobilne platforme Turtlebot, saj nam ta omogoča avtomatizacijo celotnega postopka zajemanja oblike predmeta in gradnje 3D modela. Ni se nam potrebno ubadati z nastavitvami senzorja ali obračanjem predmeta, ampak preprosto postavimo platformo Turtlebot pred predmet in čez nekaj minut že imamo na voljo 3D model v obliki oblaka točk in poligonske mreže.

Glavna omejitev prav tako izhaja iz uporabljene mobilne platforme Turtlebot in sicer iz njegove zgradbe. Linearno premikanje platforme je namreč

omejeno na smeri naprej in nazaj, senzor Kinect pa je nameščen tako, da zajema le prostor pred robotsko platformo. To pomeni, da pri gibanju okrog predmeta le-tega ni mogoče imeti stalno v objektivu senzorja Kinect, kar nam onemogoča uporabo nekaterih pristopov za gradnjo 3D modelov ter otežuje registracijo. Poleg tega nizka namestitev senzorja Kinect omogoča zajemanje globinskih podatkov le od strani, kar pomeni da imajo pridobljeni večji modeli na vrhu luknjo.

Prav tako ima sistem težave pri gradnji 3D modelov manjših predmetov. Uporabljeni pristop namreč slabše deluje v primerih, ko zajeti oblaki točk zajemajo majhno površino.

6.2 Nadaljnje delo

V trenutnem sistemu za registracijo oblakov uporabljamo samo 3D informacijo, končni modeli pa niso teksturirani. Ker imamo preko senzorja Kinect na voljo tudi RGB informacijo, je prvi korak za nadaljnje delo uporaba te informacije za pomoč pri registraciji težjih primerov ter za teksturiranje končnega modela. Pred tem je seveda potrebna še natančna kalibracija ujemanja barvne in globinske slike senzorja.

Kot naslednji korak k izboljšanju sistema predlagamo drugačno namestitev senzorja Kinect na mobilno platformo Turtlebot, ki bo omogočala, da bo predmet, ki nas zanima, neprestano v objektivu senzorja. Hkrati predlagamo tudi višjo namestitev senzorja. S tem se odpira možnost natančnejše registracije oblakov točk, hitrejše navigacije (obračanje ni več potrebno) ter gradnje zaprtih 3D modelov (tudi vrhnja površina modela je zajeta).

Kot smo omenili v prejšnjem poglavju, je mogoče 'štrleče' točke modelov odstraniti s postprocesiranjem, ki pa ni bilo predmet te naloge. V nadaljnem razvoju sistema pa bi bilo seveda smiselno v sistem integrirati tudi to funkcionalnost, saj bi privedla do vizualno boljših rezultatov.

Uporaba mobilne platforme pri zajemanju globinskih podatkov in gradnji tridimenzionalnih modelov se je izkazala za zelo uporabno. Zato kot

zadnji predlog podajamo možnost prilagoditve sistema za gradnjo modelov celotnih prostorov. Pri tem lahko sistem pridobljeni model uporablja tudi za natančnen zemljevid pri premikanju ali izvajanju drugih nalog.

Literatura

[1] 3D Scanner - wikipedia. (15.3.2013). Dostopno na: http://en.wikipedia.org/wiki/3D_scanner.

[2] X-ray computed tomography - wikipedia. (15.3.2013). Dostopno na: http://en.wikipedia.org/wiki/X-ray_computed_tomography.

[3] The Free 3D Models, Gandalf 3d model. (15.3.2013). Dostopno na: http://thefree3dmodels.com/stuff/characters/gandalf/14-1-0-5102

[4] Fakulteta za kemijo in kemijsko tehnologijo. Zaključen natečaj za arhitekturno rešitev novih stavb FKKT in FRI Univerze v Ljubljani. (15.3.2013). Dostopno na: http://www.fkkt.uni-lj.si/si/?1067.

[5] 3D CAD Drawings - Timing Belts, Synchronous Drives and Sprockets. (15.3.2013). Dostopno na: http://www.bbman.com/3d-cad-drawings.html.

[6] Factum Foundation, A report on the recording of the tomb of Tutankhamun. (15.3.2013). Dostopno na: http://www.factumfoundation.org/tut_recording.php.

[7] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, A. Fitzgibbon, "KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera", UIST '11 Proceedings of the 24th annual ACM symposium on User interface software and technology: 559-568, New York, USA, 2011.

[8] P. J. Besl, N. D. McKay, "A Method for Registration of 3-D Shapes". IEEE Trans. on Pattern Analysis and Machine Intelligence, 14 (2): 239–256, Los Alamitos, CA, USA, 1992.

[9] M. Magnusson, "The Three-Dimensional Normal Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loop Detection", Ph.D. Dissertation, Örebro University, 2009.

[10] A. E. Johnson, "Spin Images: A Representation for 3-D Surface Matching", Ph.D. Dissertation, Carnegie Mellon University, 1997.

[11] R. B. Rusu, "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments", Ph.D. Dissertation, Technische Universität München, 2009.

[12] S. Rusinkiewicz, M. Levoy, "Efficient Variants of the ICP Algorithm", Third International Conference on 3D Digital Imaging and Modeling, 2001.

[13] G. Turk, M. Levoy, "Zippered Polygon Meshes from Range Images", Proc. SIGGRAPH, 1994

[14] T. Masuda, K. Sakaue, N. Yokoya, "Registration and Integration of Multiple Range Images for 3-D Model Construction", Proc.CVPR, 1996.

[15] G. Godin, M. Rioux, R. Baribeau, "Three-dimensional Registration Using Range and Intensity Information", Proc. SPIE: Videometrics III, Vol. 2350, 1994.

[16] Chen, Y. and Medioni, G. "Object Modeling by Registration of Multiple Range Images", Proc. IEEE Conf. on Robotics and Automation, 1991.

[17] R. Benjemaa, F. Schmitt, "Fast Global Registration of 3D Sampled Surfaces Using a Multi-Z-Buffer Technique", Proc. 3DIM, 1997.

[18] G. Godin, M. Rioux, R. Baribeau, "Three-dimensional Registration Using Range and Intensity Information", Proc. SPIE: Videometrics III, Vol. 2350, 1994.

[19] K. Pulli, "Surface Reconstruction and Display from Range and Color Data", Ph.D. Dissertation, University of Washington, 1997.

[20] K. Arun, T. Huang, S. Blostein, "Least-Squares Fitting of Two 3-D Point Sets", Trans. PAMI, Vol. 9, No. 5, 1987.

[21] P. Biber, W. Straßer, "The normal distributions transform: A new approach to laser scan matching", In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS), pages 2743–2748, Las Vegas, USA, October 2003.

[22] L. Montesano, J. Minguez, L. Montano, "Probabilistic scan matching for motion estimation in unstructured environments", In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS), 2005.

[23] D. Hähnel, W. Burgard, "Probabilistic matching for 3D scan registration", In Proceedings of the VDI-Conference Robotik 2002 (Robotik), 2002.

[24] F. Boughorbel, A. Koschan, B. Abidi, M. Abidi, "Gaussian fields: a new criterion for 3D rigid registration", Pattern Recognition, 37(7):1567–1571, 2004.

[25] N. J. Mitra, N. Gelfand, H. Pottmann, L. Guibas, "Registration of point cloud data from a geometric optimization perspective", In Proceedings of the Symposium on Geometry Processing, pages 22–31, 2004.

[26] S. D. Roth, "Ray Casting for Modeling Solids", Computer Graphics and Image Processing 18 (2): 109–144, February, 1982.

[27] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, A. Fitzgibbon, "KinectFusion: KinectFusion: Real-time dense surface mapping and tracking", ISMAR '11 Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality: 127-136, IEEE Computer Society Washington, DC, USA, 2011.

[28] Turtlebot. (15.3.2013). Dostopno na: http://turtlebot.com.

[29] Wikipedia. Roomba - wikipedia. (15.3.2013). Dostopno na: http://en.wikipedia.org/wiki/Roomba.

[30] Wikipedia. Kinect - wikipedia. (15.3.2013). Dostopno na: http://en.wikipedia.org/wiki/Kinect.

[31] John MacCormick. How does the Kinect work? (15.3.2013). Dostopno na: http://users.dickinson.edu/ jmac/selected-talks/kinect.pdf.

[32] Wikipedia. Robot Operating System - wikipedia. (15.3.2013). Dostopno na: http://en.wikipedia.org/wiki/ROS_(Robot_Operating_System).

[33] ROS.org. ROS/Introduction. (15.3.2013). Dostopno na: http://www.ros.org/wiki/ROS/Introduction.

[34] PCL. About - Point Cloud Library. (15.3.2013). Dostopno na: http://pointclouds.org/about/.

[35] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D., C. T. Silva, "Computing and Rendering Point Set Surfaces", IEEE TVCG 9(1), Jan 2003

[36] Z. C. Marton, R. B. Rusu, M. Beetz, "On Fast Surface Reconstruction Methods for Large and Noisy Datasets", Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), May 12-17, 2009, Kobe, Japan.

[37] Wikipedia. RANSAC - wikipedia. (15.3.2013). Dostopno na: http://en.wikipedia.org/wiki/RANSAC.

[38] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography", Comm. Of the ACM 24: 381–395, June 1981.