

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

IZGRADNJA 3D MODELA SCENE POMOĆU 3D
KAMERE

Diplomski rad

Marijan Svalina

Osijek, 2014.

Sadržaj

1. Uvod	1
1.1. Zadatak diplomskog rada	2
2. Pregled korištenih tehnologija i algoritama	3
2.1. Microsoft Kinect 3D kamera	3
2.2. ROS biblioteka i alati	5
2.3. Biblioteka Pointcloud	6
2.4. Metoda istovremene lokalizacije i mapiranja	7
2.5. Poisson algoritam za rekonstrukciju površine	10
3. Izgradnja 3D modela scene	14
3.1. Snimanje scene 3D kamerom i RGBDSlam programom	14
3.2. Izgradnja 3D modela scene pomoću mreže trokuta	18
4. Rezultati pokusa	29
4.1. Prikaz izgrađenih modela scena i objekata	30
4.2. Korišteni parametri i podaci o izrađenim modelima	33
5. Zaključak	34
Literatura	35
Sažetak	37
Životopis	38
Prilozi	39

1. UVOD

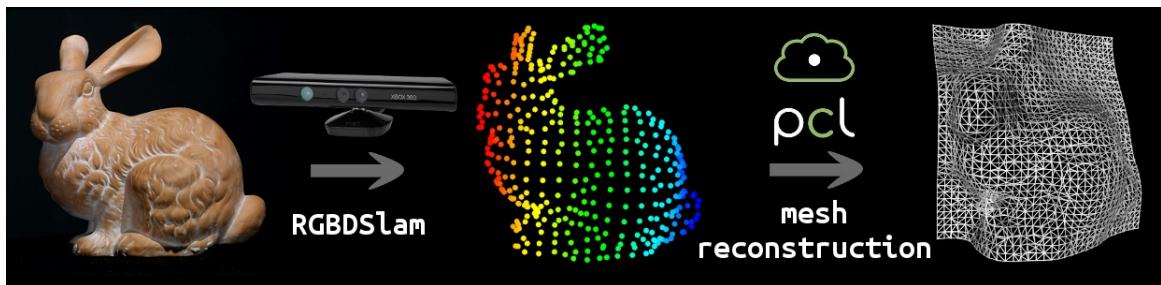
Pojava i dostupnost jeftinog i kvalitetnog 3D senzora Microsoft Kinect kamere 2010. godine za igračku konzolu Xbox 360 uvelike je doprinjela razvoju računalnog vida. Senzor u VGA rezoluciji frekvencijom od 30Hz daje sliku u boji te informaciju o dubini točaka slike, tj. njihovoj udaljenosti od kamere. Time omogućava da se na prirodniji način pristupi rješavanju osnovnih problema računalnog vida. Upravo to se i dogodilo te su znanstvenici, programeri i hakeri razvili upravljačke programe, alate i algoritme za korištenje Kinect senzora i sličnih uređaja za snimanje 3D oblaka točaka. Većina izrađenog softvera je objavljena pod slobodnim licencama koje omogućavaju slobodu upotrebe programa u bilo koje svrhe, slobodu proučavanja i primjenjivanja stečenog znanja, slobodu distribuiranja kopija u cijelosti ili u dijelovima te slobodu mijenjanja, poboljšavanja i distribuiranja derivacijskih programa. Upravo ti programi su postavili temlje za ovaj diplomski rad.

Osim Kinect kamere u radu se koristi još niz suvremenih tehnologija i algoritama: ROS (engl. *Robot Operating System*) biblioteka i alati, biblioteka Pointcloud, metoda istovremene lokalizacije i mapiranja te Poisson algoritam za rekonstrukciju površine.

Rad se sastoji iz tri dijela. Prvi dio odnosi se na pregled korištenih tehnologija i algoritama. Slijedi praktični dio koji govori o izgradnji 3D modela scene. Prvo je objašnjen postupak snimanja scene 3D kamerom i RGBDSlam programom, a zatim je dan pregled izgradnje 3D modela scene u obliku mreže trokuta. Treći dio rada prikazuje rezultate pokusa snimanja i izrade 3D modela te ispituje funkcionalnost i kvalitetu postupka.

1.1. Zadatak diplomskog rada

Program RGBDSLAM raspoloživ u okviru incijative dijeljenja algoritama OpenSLAM omogućava izgradnju 3D modela objekata i scena pomoću 3D kamere. Zadatak ovog rada je razviti program za izgradnju 3D modela u obliku mreže trokuta koristeći programsku biblioteku PCL. Kombinacijom ova dva programa mogu se izgraditi 3D modeli objekata i scena snimljenih iz više pogleda. Zadatak je ispitati funkcionalnost navedenog postupka kao i kvalitetu dobivenog rezultata izgradnjom nekoliko 3D modela objekata i scena. Na slici 1.1. grafički je prikazan zadatak diplomskog rada.



Slika 1.1.: Grafički prikaz projekta upotrebom Standfordovog zeca¹

¹Standford Bunny 3D model su originalno konstruirali 1994 Greg Turk i Marc Levoy i od tada je postao najčešće upotrebljевani model za testiranje tehnika u računalnoj grafici. <http://www.gvu.gatech.edu/people/faculty/greg.turk/bunny/bunny.html>

2. PREGLED KORIŠTENIH TEHNOLOGIJA I ALGORITAMA

2.1. Microsoft Kinect 3D kamera

Kinect je RGB-D senzor koji daje sliku u boji sinkroniziranu s dubinskom slikom. Inicijalno je korišten kao ulazni uređaj za Microsoft Xbox igračku konzolu. Algoritmima za prepoznavanje ljudskih pokreta omogućava interakciju između igre i igrača bez potrebe za korištenjem upravljačkog uređaja. Nakon pojave senzora zajednica znanstvenika koji proučavaju računalni vid otkrila je da tehnologija korištena za dohvaćanje dubine može biti upotrebljena za puno više od igranja za puno nižu cijenu od tradicionalne 3D kamere kao što su stereo i Time-of-flight kamere.

Dodatno, komplementarna priroda informacije o dubini na slici koju pruža Kinect, olakšava nova potencijalna rješenja klasičnih problema računalnog vida. U samo dvije godine nakon puštanja u prodaju Kinecta, velik broj znanstvenih članaka i demonstracija pojavio se na raznim konferencijama.



Slika 2.1.: Shematski prikaz Kinect kamere, izvor: [9]

2.1.1. Tehničke specifikacije

Slika 2.1. prikazuje osnovne komponente Kinect senzora. Kinect se sastoji od infrarevnog (IR) projektorja, IR kamere i RGB kamere. Dubinski senzor se sastoji od IR projektorja i IR kamere. Kinect radi na principu strukturirane svjetlosti. IR projektor projicira IR uzorak točkastih mrlja (engl. *speckle*) na 3D scenu dok IR kamera hvata reflektirane IR mrlje. Geometrijska veza između IR projektorja i IR kamere se ostvaruje offline kalibracijskom procedurom. IR svjetlost je nevidljiva RGB kameri ali je vidljiva IR kameri. Kako je svaki lokalni uzorak projiciranih točaka jedinstven, spajanje promatranih lokalnih uzoraka točaka sa slike s kalibriranim projektorovim uzorkom točaka je ostvarivo. Više o principu strukturirane svjetlosti u [8].

Tehničke informacije o kamери:

- **RGB kamera** daje slikу rezolucije 640×480 piksela pri osvježavanju slike od 30Hz. Postoji i opcija davanja slike rezolucije 1280×1024 piksela ali se dobije samo 10 sličica po sekundi.
- **3D Dubinski senzor** se sastoji od IR laserskog projektorа i IR kamere. Zajedno, projektor i kamera kreiraju dubinsku mapu, koja pruža informaciju o udaljenosti između objekta i kamere. Senzor ima praktično ograničenje u dometu 0.8m - 3.5m. Kamera daje video od 30 sličica po sekundi s rezolucijom 640×480 piksela. Vidno polje senzora je 57° horizontalno i 43° vertikalno.
- **Motor** upravlja nagibom Kinecta. Kinect može biti nagnut 27° prema gore ili prema dolje.

2.2. ROS biblioteka i alati

ROS [20] (engl. *Robot Operating System*) je skup programskih biblioteka i alata koji pomažu programerima kreirati aplikacije za robote. Pruža hardversku apstrakciju, upravljačke programe, biblioteke, vizualizaciju, komunikaciju, upravljanje paketima i dr. ROS je licenciran BSD licencom, koja spada pod licence otvorenog koda.



Slika 2.2.: Shematski prikaz ROSa, izvor: [7]

U ovom radu ROS se dotiče kroz program RGBDSlam koji je opisan u potpoglavlju 3.1. te se oslanja na njegove biblioteke, upravljačke programe i alate. Tako RGBDSlam koristi OpenNI upravljački program za komunikaciju s Kinect kamerom u obliku ros programskega paketa (`ros-fuerte-openni-launch`). Također, RGBDSlam koristi i niz drugih paketa iz ROS biblioteke [4].

2.3. Biblioteka Pointcloud

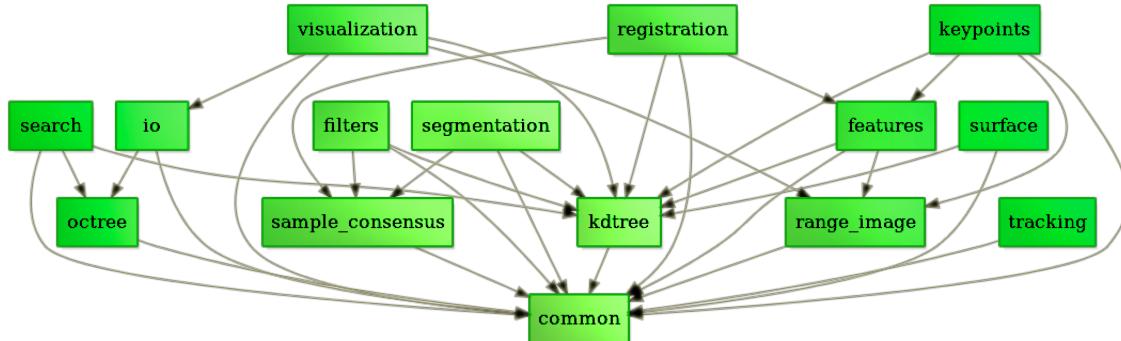
PCL [23] (engl. *Point Cloud Library* slika 2.3.) je otvoren projekt za obradu 2D/3D slika i oblaka točaka. PCL biblioteka sadržava niz najsuvremenijih algoritama za filtriranje, estimaciju značajki, rekonstrukciju površina, registraciju, uklapanje modela i segmentaciju. Ti algoritmi se mogu koristiti za: izbacivanje odudarajućih vrijednosti iz šumovith podataka, spajanje 3D oblaka točaka, segementiranje relevantnih dijelova scene, izvlačenje ključnih točaka i računanje deskriptora za prepoznavanje objekata na temelju njihove geometrije, kreiranje i prikazivanje površina iz oblaka točaka itd.



Slika 2.3.: PCL logo, izvor: [19]

PCL objavljen je pod BSD licencom i softver je otvorenog koda. Što znači da je slobodan za upotrebu u komercijalne i akademiske svrhe.

PCL uspešno se prevodi na GNU/Linux, MacOS, Windows i Android/iOS platformama. Zbog olakšanog razvijanja podijeljen je na više manjh biblioteka koje se mogu prevoditi zasebno. PCL se može prikazati kao graf biblioteka kao što je prikazano na slici 2.1.



Grafikon 2.1.: Graf PCL biblioteka, izvor: [19]

2.4. Metoda istovremene lokalizacije i mapiranja

Jedno od najznačajnijih područja istraživanja na polju mobilne robotike su metode istovremene lokalizacije i mapiranja (engl. *Simultaneous Localization and Mapping* - SLAM). SLAM su originalno razvili Hugh Durrant-Whyte i John J. Leonard [15] koji su rad bazirali na prethodnom radu Smitha, Selfa and Cheesemana [24]. SLAM se bavi rješavanjem problema izgradnje karte nepoznate okoline mobilnog robota te istovremene navigacije robota okolinom korištenjem te karte.

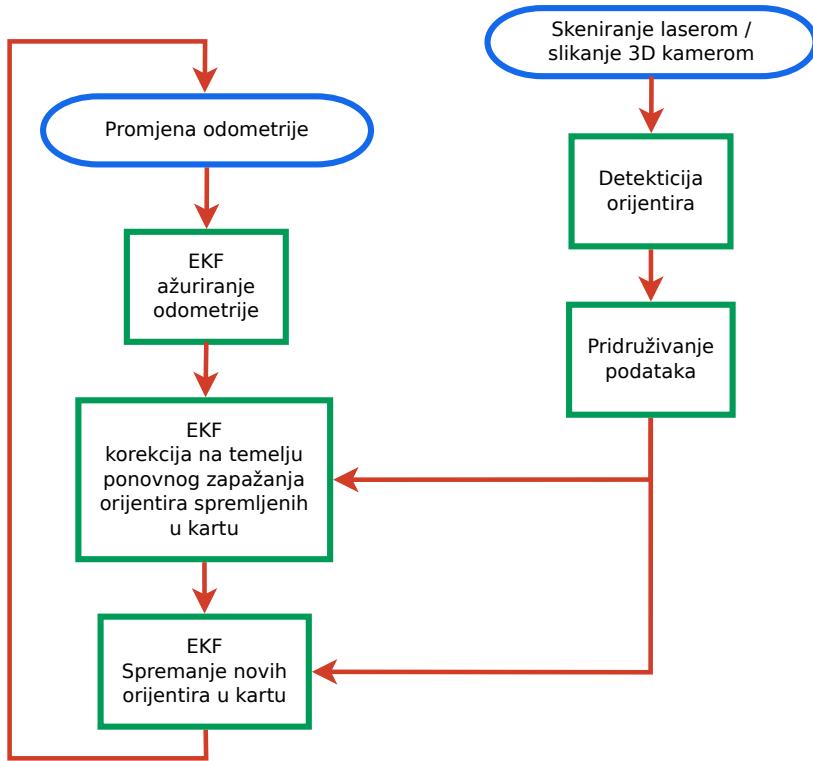
SLAM se sastoji iz nekoliko koraka: detekcije orijentira, pridruživanja podataka, estimiranja stanja, ažuriranja stanja i ažuriranja orijentira. Postoji više načina kako realizirati svaki od tih koraka. SLAM se može primjeniti na kreiranje 2D i 3D karti.

2.4.1. Osnovna ideja i kratak pregled koraka

Proces istovremene lokalizacije i mapiranja se sastoji iz nekoliko koraka. Cilj procesa je koristiti precepciju okoline za ažuriranje pozicije robota. Zbog nesavršenosti odometrije robota nije dobro osloniti se samo na nju kako bi se pronašla pozicija robota već se može koristiti lasersko skeniranje okoline ili, kao u slučaju RGBDSlam programa, Kinect 3D kamera kako bi pronašli pravu poziciju robota/kamere. To se postiže detekcijom značajki u okolini i promatranjem tih značajki kada se robot pomakne. EKF (engl. *Extended Kalman Filter*) odnosno prošireni Kalmanov filter je jezgra SLAM procesa. EKF je odgovoran za ažuriranje pozicije na kojoj robot "misli" da se nalazi koristeći detektirane značajke. Takve značajke se još nazivaju i orijentiri. EKF prati estimaciju nesigurnosti pozicije robota i nesigurnosti orijentira iz okoline. Pregled SLAM procesa¹ se nalazi na grafikonu 2.2.

Kad se odometrija promijeni zato što se robot pomakao, nesigurnost koja se tiče robotove nove pozicije se ažurira u EKF koristeći ažuriranje odometrije. Orijentiri se tada detektiraju u okolini iz robotove nove pozicije. Robot tada pokušava asocirati detektirane orijentire s prije zapaženim orijentirima spremlijenim u kartu. Ponovno zapaženi orijentiri se tada koriste za ažuriranje pozicije robota u EKFu. Orijentiri koji prije nisu zapaženi se dodaju u kartu kako bih se mogli koristiti kasnije. U svakom od opisanih koraka EKF računa estimaciju robotove trenutne pozicije.

¹Grafikon je inspiriran sličnim grafikonom iz rada SLAM for Dummies [21]

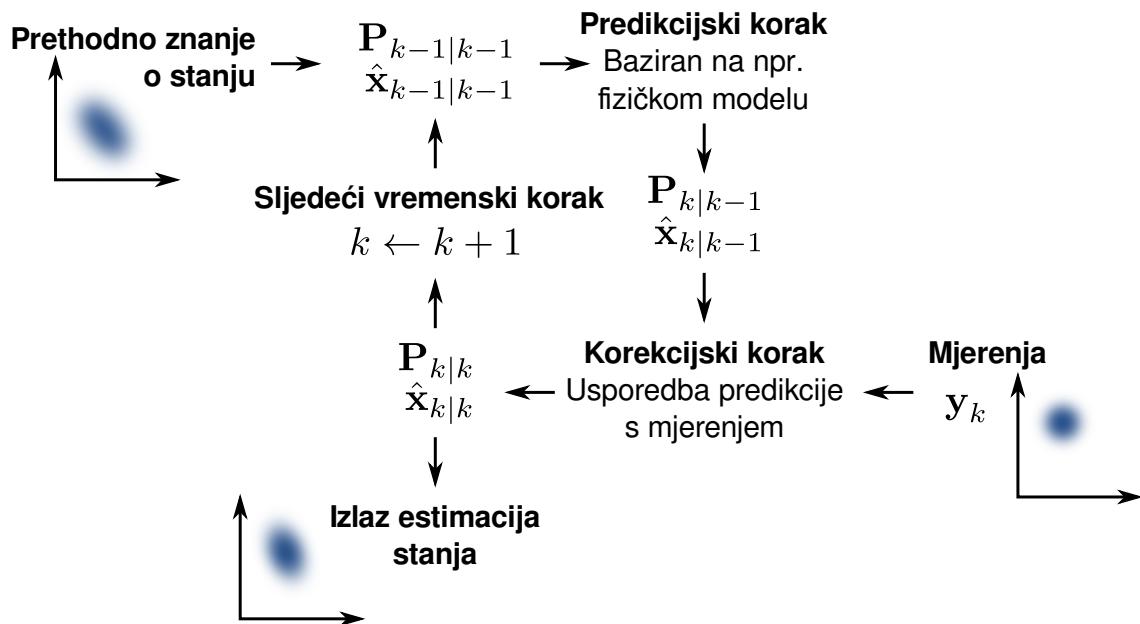


Grafikon 2.2.: Pregled SLAM procesa

2.4.2. Proširen Kalmanov filter - EKF

Kalmanov filter je algoritam koji koristi niz mjerena pribavljenih tijekom vremena, koja sadržavaju šum i ostale netočnosti, te estimira nepoznate varijabli koje su, u slučaju zadovoljenja pretpostavki algoritma, preciznije od računa temeljenog samo na jednom mjerenuju. Ukratko, Kalmanov filter se izvršava rekursivno na nizu šumovitih ulaznih podataka i računa statistički optimalnu estimaciju stanja sustava. Filter je nazvan po Rudolfu E. Kalmanu [11] koji je jedan od prvih razvijatelja te teorije.

Algoritam je podijeljen u dva koraka kao što se vidi na grafikonu 2.3. U prvom predikcijskom koraku Kalmanov filter računa estimaciju trenutnih varijabla stanja s njihovim nesigurnostima na temelju modela kretanja robota. U drugom korekcijskom koraku, kada je promotren rezultat sljedećeg mjerena (koje isto ima greške pri mjerenu i šum mjerena), izračunata estimacija se korigira upotrebom težinske sredine, gdje se više težine daje estimaciji s većom vjerojatnošću.



Grafikon 2.3.: Osnova Kalmanovog filtra¹

Prošireni Kalmanov filter je nelinearna verzija Kalmanovog filtra gdje promjena stanja i promatrani model ne moraju biti opisani linearnim funkcijama.

¹Grafikon osnova Kalmanovog filtra prevedena je verzija grafikona s http://en.wikipedia.org/wiki/Kalman_filter

2.5. Poisson algoritam za rekonstrukciju površine

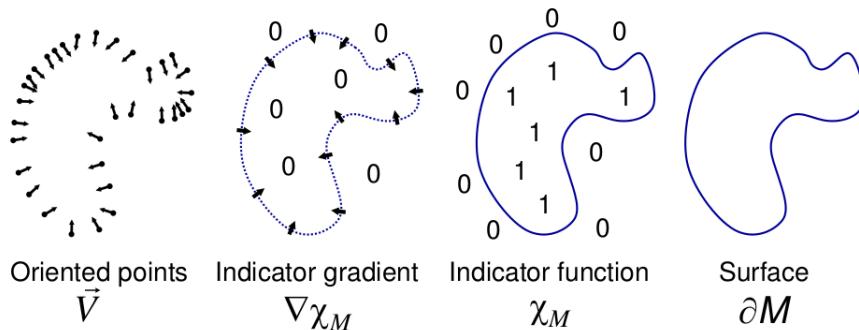
Poisson algoritam za rekonstrukciju površine [12] razvijen je suradnjom Michaela Kazhdana i Matthewa Bolitha s Johns Hopkins sveučilišta u Baltimoru i Huguesa Hoppea iz Microsoft Researcha u Redmondu. Također Kazhdan i Bolitho su implementirali¹ Poisson algoritam i objavili kod pod BSD licencom. Na osnovu tog rada algoritam je dodan i u PCL biblioteku.

U ovom potpoglavlju nalazi se osnovna ideja i kratak matematički pregled algoritma. Opisano je ograničenje algoritma te parametri kojima se može upravljati rekonstrukcijom površine.

2.5.1. Osnovna ideja i kratak matematički pregled

Poisson algoritam pristupa problemu rekonstrukcije površine rješavanjem Poissonove jednadžbe. To čini upotrebom metode implicitne funkcije. Točnije računanjem 3D indikacijske funkcije χ definirane s 1 u točkama unutar modela, odnosno s 0 u točkama izvan i dohvaćanjem rekonstruktuirane površine izvlačenjem odgovarajuće izopovršine.

Algoritam se oslanja na ideju da postoji cjelovita veza između orijentiranih normala uzetih s površine modela i indikacijske funkcije modela. Točnije, gradijent indikacijske funkcije je polje vektora koje je uglavnom popunjeno nulama (jer je indikacijska funkcija uglavnom konstantna), osim kod točaka blizu površine gdje je jednako unutrašnjim normalama površine. Stoga, uzorci orijentiranih normala mogu biti promatrani kao gradijent modela indikacijske funkcije kao što je prikazano na slici 2.4.



Slika 2.4.: Prikaz Poisson rekonstrukcije u 2D, izvor: [12]

Problem računanja indikacijske funkcije se svodi na invertiranje operatorka gradijenta, odnosno pronalazak funkcije skalara χ čiji gradijent najbolje aproksimira polje vektora \vec{V} definirano uzorcima, odnosno

$$\min_{\chi} \|\nabla \chi - \vec{V}\|.$$

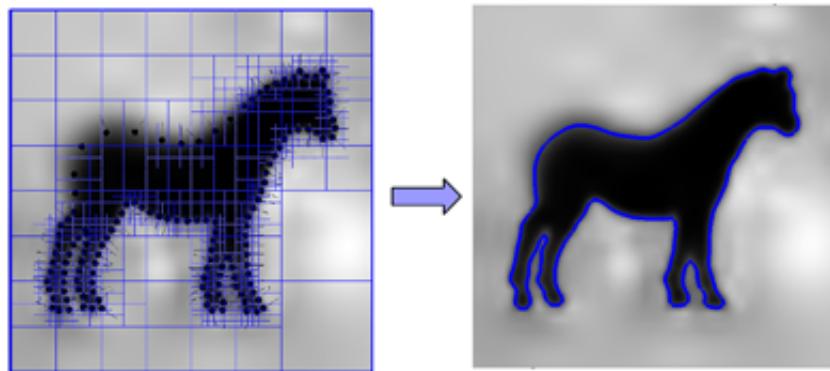
¹Originalna implementacija Poisson algoritma se nalazi na <http://www.cs.jhu.edu/~misha/Code/PoissonRecon/Version5.5/>

Ako se primjeni operator divergencije, tada se taj problem pretvara u standardni Poissonov problem: računanje funkcije skalara χ čiji laplasijan (divergencija gradijenta) je jednak divergenciji polja vektora \vec{V} ,

$$\Delta\chi \equiv \nabla \cdot \nabla\chi = \nabla \cdot \vec{V}.$$

Predstavljanje rekonstrukciju površine kao Poissonov problem pruža nekoliko prednosti. Mnoge implicitne metode uklapanja površina segmentiraju podatke u regije za lokalno uklapanje i onda te lokalne aproksimacije spajaju upotrebom funkcija stapanja. Za razliku od njih, Poisson rekonstrukcija je globalno rješenje koje razmatra sve podatke odjednom, bez upotrebe heurističkih podijela i stapanja. Zbog toga Poisson rekonstrukcija kreira izrazito glatku površinu koja robusno aproksimira šumovite podatke.

Za izvlačenje izopovršine Poisson algoritam koristi Marching Cubes algoritam [16] koji kreira octree strukturu podataka za prikaz površine. Kao što se vidi na slici 2.5. Marching Cubes algoritam dijeli oblak točaka u mrežu voxela marširajući kroz oblak i analizira koje točke čine izopovršinu objekta. Detektiranjem koji rubovi voxela presjecaju izopovršinu modela algoritam kreira mrežu trokuta. Više informacija o izvlačenju površine se mogu pronaći u radu “Unconstrained Isosurface Extraction on Arbitrary Octrees” Michaela Kahzdana [13]



Slika 2.5.: Prikaz Marching cubes algoritma, izvor: [13]

2.5.2. Ograničenje Poisson algoritma

Ograničenje implementacije Poisson algoritma je u tome što ne uzima u obzir informacije vezane uz način snimanja oblaka točaka. Slika 2.6. pokazuje kip Bude i vidi se primjer takvog ograničenja. Budući da nema točaka između Budinih nogu, Poisson algoritam spaja te dvije regije. Algoritam se može unaprijediti ugradnjom dodatne informacije poput vidokruga i na taj način izbjegići to ograničenje.



Slika 2.6.: Rekonstrukcija modela “Happy Buddha” VRIP² algoritam (lijevo) i Poisson algoritma (desno), izvor: [12]

2.5.3. Parametri Poisson algoritma

Postoji nekoliko parametara koji utječu na rezultat rekonstrukcije.

- **Depth:** dubina octree stabla koje se koristi za rekonstrukciju. Prepostavljena vrijednost 8.
- **SolverDivide:** postavlja dubinu kod kojeg bloka Gauss-Seidel metoda riješava Laplaceovu jednadžbu. Prepostavljena vrijednost 8.
- **IsoDivide:** postavlja dubinu kod kojeg bloka ekstraktor izopovršine izvlači izopovršinu. Prepostavljena vrijednost 8.
- **SamplesPerNode:** postavlja minimalni broj točaka koje se trebaju nalaziti unutar octree čvora kako se octree konstrukcija prilagođava gustoći uzorkovanja. Za podatke bez šuma 1 - 5, sa šumom 15 - 20. Prepostavljena vrijednost 1.
- **Scale:** omjer između promjera kocke korištne za rekonstrukciju i promjera kocke koja omeđuje uzorke. Prepostavljena vrijednost 1.25.
- **Confidence:** postavljanje zastavice govori rekonstrukciji da koristi veličinu normala kao informaciju o pouzdanosti. Ako nije postavljena sve normale se normaliziraju prije rekonstrukcije.

²VRIP - Volumetric Range Image Processing [3]

Od nabrojanih parametara najvažniji utjecaj na generiranu mrežu imaju `SamplesPerNode` i `Depth`. Veća dubina octree stabla rezultira većom preciznosti mreže voxela jer Marching Cubes algoritam ulazi dublje u stablo. Manja dubina (između 5 i 7) daje glađi model ali s manje detalja. `SamplesPerNode` parametar definira koliko će točaka Marchin Cubes algoritma staviti u jedan čvor rezultantnog octree stabla. Ako algoritam radi s podacima punim šuma velik uzorak točaka (15 - 20) po čvoru pruža glađenje ali se gube detalji. Dok rad s malim vrijednostima (1 - 5) održava razinu detalja visokom. Velike vrijednosti reduciraju krajnji broj vrhova poligona, dok male održavaju broj vrhova visokim.

3. IZGRADNJA 3D MODELA SCENE

3.1. Snimanje scene 3D kamerom i RGBDSlam programom

Program RGBDSlam omogućava izgradnju 3D modela objekata i scena u unutrašnjosti prostorija (oblak točaka u boji) rukom upravljanom kamerom tipa Kinect. Razvijen je suradnjom sveučilišta Albert-Ludwigs-Universität¹ u Freiburgu i Technische Universität München². Slobodan je program objavljen pod GPLv3³ licencom. Izvorni kod je dostupan na Google code⁴ stranicama. U prilogu diplomskog rada nalaze se upute za prevođenje i instaliranje programa.

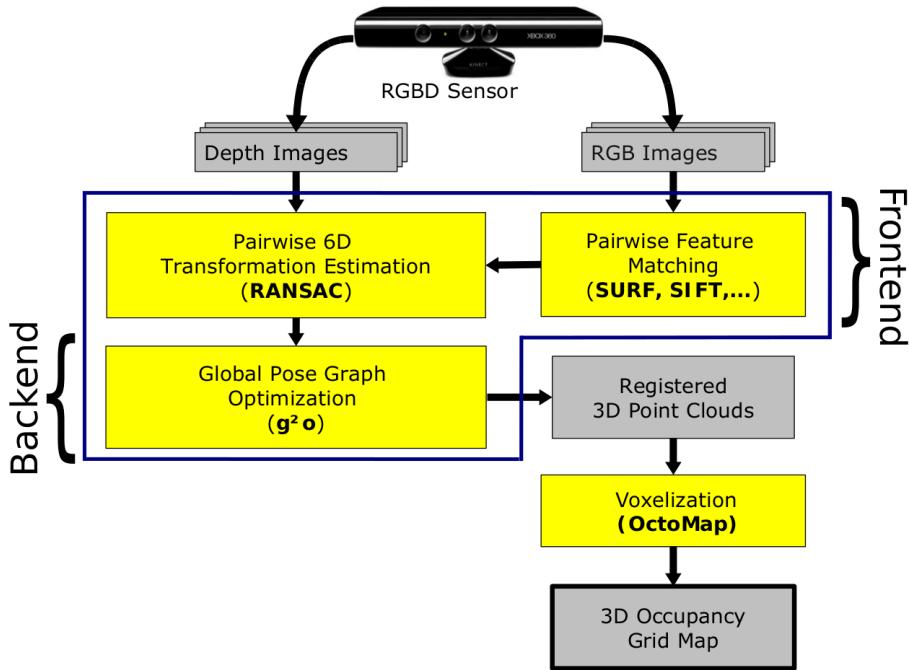
¹Felix Endres i Juergen Hess sa odijela Autonomous Intelligent Systems koji vodi Prof. Dr. Wolfram Burgard.

²Nikolas Engelhard sa odijela Computer Vision Group koji vodi Dr. Juergen Sturm.

³GNU General Public License version 3 slobodna je licenca koja osigurava osnovna prava slobodnih programa. Pravo na korištenje, proučavanje, kopiranje i poboljšavanje. Izvor: <http://www.gnu.org/licenses/gpl-3.0.html>

⁴RGBDSlam program moguće je preuzeti sa svn programom sa stranice http://alufr-ros-pkg.googlecode.com/svn/trunk/rgbdslam_freiburg/

3.1.1. Sažet opis rada RGBDSlam programa



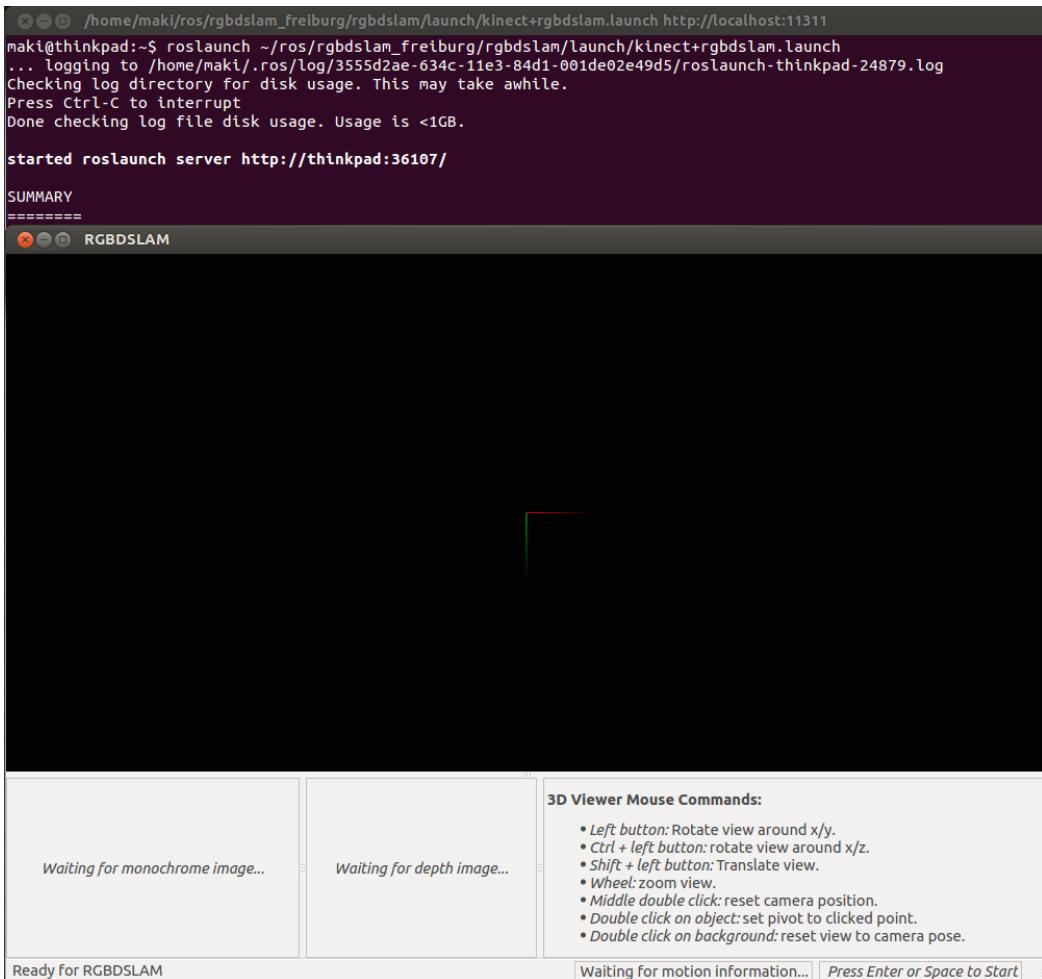
Grafikon 3.1.: Shematski pregled¹ RGBDSlam programa

Kao što je vidljivo iz grafikona 3.1. program je podijeljen u četiri osnovna dijela. Prvi dio računa značajke iz ulaznih slika u boji. RGBDSlam se oslanja na OpenCV [2] biblioteku u kojoj su implementirani SURF [1], SIFT [17] i ORB [22] algoritmi za pronađak značajki. Zatim se te značajke sparaju sa značajkama iz prethodnih slika. Drugi dio ispituje dubinu slika na lokacijama izračunatih značajki. Usporedbom lokalnih deskriptora dobiva se informacija o mogućim 3D korespondencijama točaka između bilo koje dvije sličice. Zatim je na temelju tih korespondencija estimirana relativna transformacija između sličica upotrebom RANSACa. RANSAC (engl. *Random Sample Consensus*) [6] je iterativna metoda estimiranja parametara matematičkog modela iz promatranih mjerena koja sadržavaju šum i odudarajuće vrijednosti. RANSAC postupkom se uklanjaju netočne korespondencije ostvarne na temelju lokalnih deskriptora. Kako parovi estimiranih položaja između sličica nisu globalno konzistentni treći dio programa optimizira graf položaja upotrebom g^2o algoritma. g^2o (engl. *General Framework for Graph Optimization*) [14] je okvir otvorenog koda za optimiziranje nelinearnih funkcija pogreške zasnovanih na grafu. Algoritam u ovoj fazi daje globalno konzistentni 3D model promatrane okoline predstavljen oblakom točaka u boji. Takav oblak točaka je upotrebljen u diplomskom radu za stvaranje mreže trokuta. RGBDSlam je dizajniran da osim oblaka točaka generira i volumetrijski prikaz okoline upotrebljavajući OctoMap [10] biblioteku ali takav prikaz nije korišten za potrebe ovog rada.

¹Shema je preuzeta iz znanstvenog rada “An Evaluation of the RGB-D SLAM System” autora Endres, Hess, Burgard, Engelhard, Cremers, Sturm [5].

3.1.2. Pokretanje RGBDSlam programa

Instaliran program pokreće se preko komandne linije koristeći `roslaunch` koji je dio ROS [20] biblioteke i alata koji su detaljnije objašnjeni u poglavlju 2.2. Kao što je prikazano na slici 3.1. `roslaunch` za parametar prima XML datoteku s ekstenzijom `.launch` u kojoj su definirani parametri s kojima se pokreće program.



Slika 3.1.: Prikaz pokretanja RGBDSlam programa iz komandne linije

Prilikom upotrebe RGBDSlama nije bilo potrebe za mijenjanjem zadanih postavki te je korištena zadana `kinect+rgbdslam.launch` datoteka za pokretanje. Program podržava dva načina rada, automatski i ručni. Kod automatskog načina program neprestano uzima slike s kamere i procesira ih, što u kratkom vremenu rezultira velikom količinom podataka. Ručni način korisniku omogućava uzimanje slike na pritisak tipke Enter.

3.1.3. Snimanje scena RGBDSlam programom

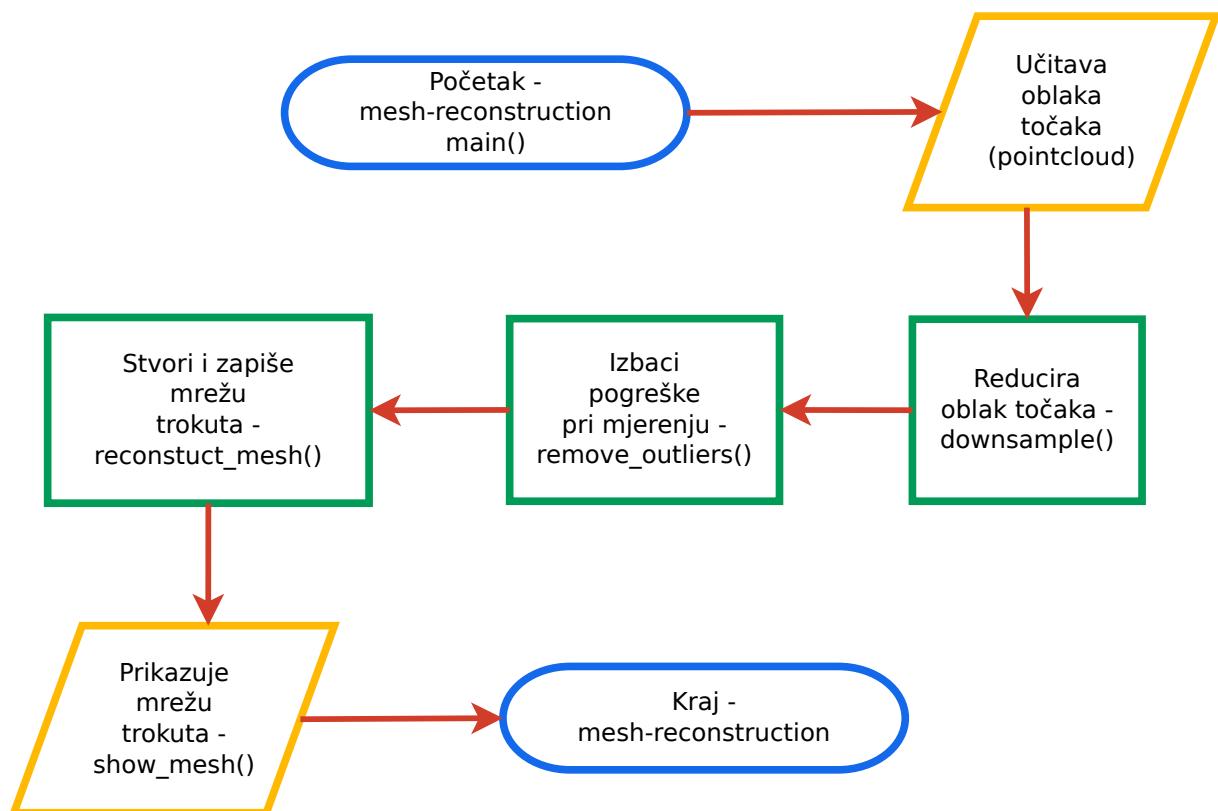
Za snimanje scena upotrijebljen je ručni način rada RGBDSlam programa. Prednost ručnog načina rada je što snimatelj kontrolira broj slika uzetih s kamere. Nedostatak je što je nezgodno jednoj osobi baratati s kamerom, gledati u računalo i pritiskati Enter za slikanje. Zato je pri snimanju scena sudjelovalo više osoba ili je korištena skripta¹ koja umjesto korisnika šalje signal Enter programu nakon proizvoljnog broja sekundi. Tijekom snimanja trebalo je obratiti pozornost na značajke scene koja se snima kako bih program mogao spariti značajke s prethodnom scenom. Tijekom izrade diplomskog rada snimljeno je šest scena odnosno prostorija koje su obrađene u poglavlju 4. Snimljene scene spremane su u formatu PCL biblioteke .pcd i kao takve korištene za daljnju obradu u diplomskom radu.

¹Skripta za slikanje dostupna je na DVDu i na web stranici <http://github.com/msvalina/pcl-surface-mesh-reconstruction>

3.2. Izgradnja 3D modela scene pomoću mreže trokuta

Izgradnja 3D modela scene pomoću mreže trokuta je implementirana u programu nazvanom `mesh-reconstruction`.¹ Program se intenzivno oslanja na biblioteku PointCloud koja je opisana u podpoglavlju 2.3. Kao što je vidljivo iz grafikona 3.2. program je podijeljen u pet osnovnih funkcija:

- Učitavanje oblaka točaka snimljenih RGBDSlam programom.
- Reduciranje oblaka točaka.
- Uklanjanje pogrešaka pri mjerenu.
- Stvaranje i zapisivanje mreže trokuta.
- Prikaz mreže trokuta.



Grafikon 3.2.: Dijagram toka programa `mesh-reconstruction`

U sljedećim podpoglavlјima dan je pregled funkcija i PCL [23] klasa na kojima se baziraju. Također na slici 3.2. se vidi kako izgleda pokretanje programa, što sve ispisuje na standardni izlaz te kako prikazuje mrežu trokuta.

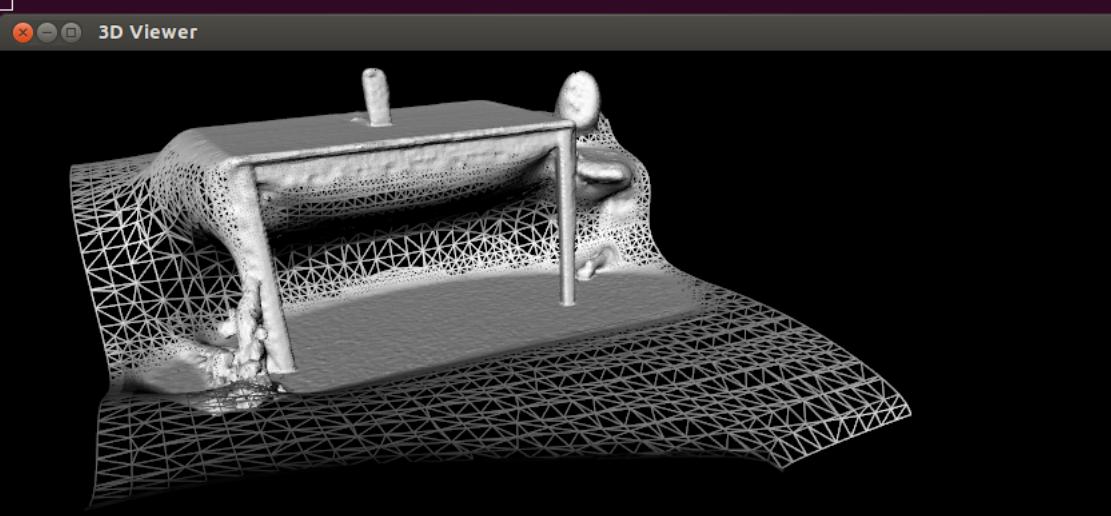
¹Program `mesh-reconstruction` slobodan je program dostupan pod uvjetima MIT licence. Izvorni kod se nalazi na DVD-u i na web stranici github.com/msvalina/

```
ranger:source/masters-thesis/mesh-reconstruction-build-desktop-Qt_4_8_1_in_PATH_System_Debug
maki@thinkpad:./mesh-reconstruction table_scene_lms400.pcd
Started - downsample() with VoxelGrid
PointCloud before filtering: 460400 data points (x y z intensity distance sid).
PointCloud after filtering: 41049 data points (x y z intensity distance sid).
Finished - downsample() with VoxelGrid

Started - remove_outliers() with StatisticalOutlierRemoval
PointCloud before filtering: 41049 data points (x y z intensity distance sid).
PointCloud after filtering: 39488 data points (x y z intensity distance sid).
Finished - remove_outliers() with StatisticalOutlierRemoval

Started - reconstruct_mesh() with Poisson
Failed to find match for field 'rgb'.
PointCloud loaded: 39488 data points
Finshed - reconstruct_mesh() with Poisson

Started - show_mesh() with PCLVisualizer
```



Slika 3.2.: Prikaz pokretanja programa `mesh-reconstruction` iz terminala

3.2.1. Pregled main() funkcije

Ispis koda 3.1.: Izvorni kod main() funkcije

```
1 int main (int argc, char *argv[])
{
3     downsample (argc, argv);
4     remove_outliers (argc, argv);
5     pcl::PolygonMesh mesh_of_triangles;
6     reconstruct_mesh (argc, argv, mesh_of_triangles);
7     show_mesh (mesh_of_triangles);
8     return 0;
9 }
```

Kao što se vidi iz ispisa koda 3.1. ideja je da funkcija bude što manja te da se iz nje samo pozivaju druge funkcije.

3.2.2. Učitavanje oblaka točaka

Program se sastoji od funkcija `downsample`, `remove_outliers`, `reconstruct_mesh` i `show_mesh`. Na početku svake funkcije program učitava oblik točaka, obrađuje ga i zapisuje na izlazu iz funkcije kako bi prije i poslije svake operacije bio dostupan. Za to koristi `PCDReader` i `PCDWriter` klase. Predložak takvog koda se nalazi u ispisu koda 3.2. Nakon učitavanja oblika točaka `reader` objektom, nad njim se vrše operacije npr. reduciranje oblika točaka. Nakon toga kreiranjem i korištenjem `writer` objekta promjeni oblik se zapisuje u datoteku.

Ispis koda 3.2.: Predložak izvornog koda za učitavanje oblika točaka

```
1 // Init cloud variables
2     pcl::PCLPointCloud2::Ptr cloud (new pcl::PCLPointCloud2());
3     pcl::PCLPointCloud2::Ptr cloud_filtered (new pcl::PCLPointCloud2());
4
5     // Fill in the cloud data
6     pcl::PCDReader reader;
7     reader.read ("pointcloud.pcd", *cloud);
8     /*
9      * Do something with cloud e.g. downsample pointcloud
10     */
11    // Write cloud to a file
12    pcl::PCDWriter writer;
13    writer.write ("pointcloud-downsampled.pcd",
14                  *cloud_filtered, Eigen::Vector4f::Zero(),
15                  Eigen::Quaternionf::Identity(), false);
```

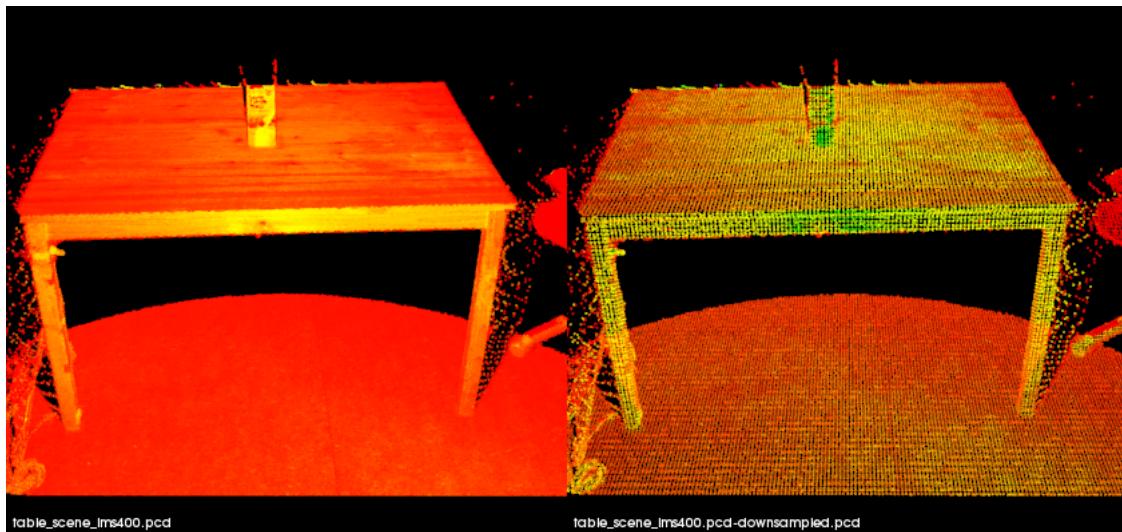
3.2.3. Reduciranje oblaka točaka

Reduciranje oblaka točaka ne unosi bitne gubitke informacije, a izvodi se zbog lakše daljne obrade oblaka. Izvodi se pomoću **VoxelGrid** klase i implementirano je u `downsample()` funkciji. Dijelovi funkcije prikazani su u ispisu koda 3.3. **VoxelGrid** dolazi od riječi *volume pixel grid* i predstavlja niz malih kocaka u prostoru.

Ispis koda 3.3.: Dio izvornog koda za reduciranje točaka iz funkcije `downsample()`

```
1 // Create the filtering object  
2 pcl::VoxelGrid<pcl::PCLPointCloud2> vg;  
3 vg.setInputCloud (cloud);  
4 // voxel size to be 1cm^3  
5 vg.setLeafSize (0.01f, 0.01f, 0.01f);  
vg.filter (*cloud_filtered);
```

Kao što se vidi iz ispisa koda 3.3. nakon kreiranja objekta `vg` predaje mu se oblak točaka nad kojim se vrši reduciranje. Postavlja se veličina kocke (*voxel*) u našem slučaju to je 1cm^3 . Nad tim oblakom prilikom filtriranja će se kreirati mreža kocaka te će se sve točke unutar jedne kocke zamjeniti centralnom točkom. Tim postupkom značajno se smanjuje broj točaka u oblaku kao što je vidljivo iz slike 3.3.



Slika 3.3.: Oblak točaka `table_scene`¹ lijevo prije `downsample()` 460400 točaka i poslije desno 41049 točaka

¹Oblak točaka `table_scene_lms400.pcd` je objavljen pod uvjetima BSD licence izvor: github.com/PointCloudLibrary/.../table_scene_lms400.pcd

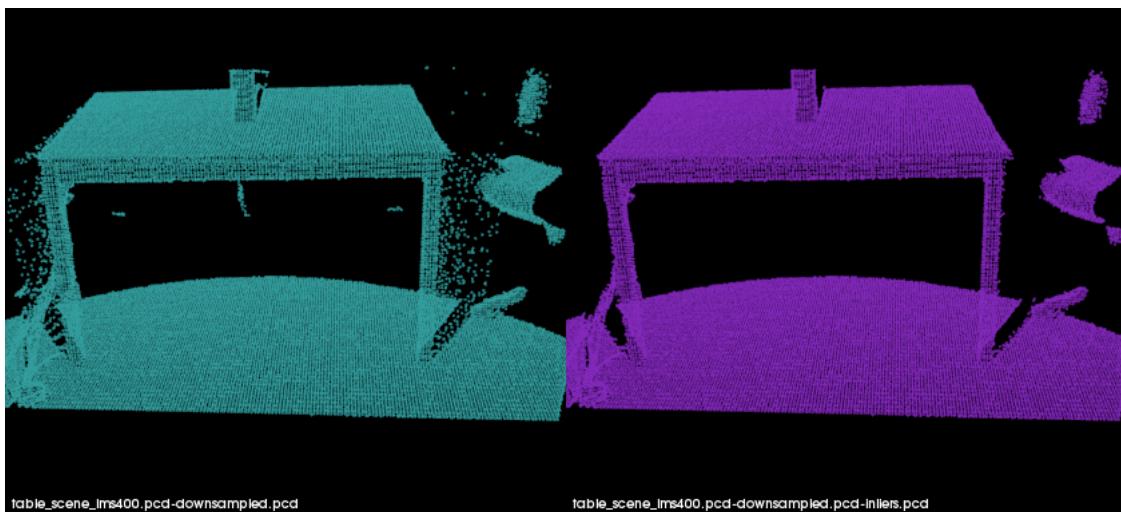
3.2.4. Uklanjanje pogrešaka pri mjerenu

Pogreške pri mjerenu su sastavni dio svakog mjernog uređaja. U slučaju Kinect senzora često se javljaju odudarajuće vrijednosti (engl. *outliers*). PointCloud biblioteka ima ugrađenu `StatisticalOutlierRemoval` klasu koja uklanja odudarajuće vrijednosti, a implementirana je u funkciji `remove_outliers()`. Iz ispisa koda 3.4. se vidi kako se klasa koristi.

Ispis koda 3.4.: Dio izvornog koda za uklanjanje odudarajućih vrijednosti iz funkcije `remove_outliers()`

```
// Create the filtering object  
2  pcl::StatisticalOutlierRemoval<pcl::PCLPointCloud2> sor;  
    sor.setInputCloud (cloud);  
4  // Set number of neighbors to analyze  
    sor.setMeanK (50);  
6  sor.setStddevMulThresh (1.0);  
    sor.filter (*cloud_filtered);
```

Nakon kreiranja objekta `sor` i predavanja oblaka postavljena su još dva parametra. Prvi `setMeanK` je broj susjednih točaka koje će filter analizirati. Drugi `setStddevMulThresh` postavlja multiplikator praga standardne devijacije. Algoritam dva puta prolazi kroz oblak točaka. U prvoj iteraciji računa srednju vrijednost udaljenosti svake točke do njenih K susjeda. Zatim računa očekivanje μ i standardnu devijaciju σ svih udaljenosti kako bi mogao odrediti prag udaljenosti. Prag udaljenosti je određen jednadžbom $\mu + \text{StddevMulThresh} \cdot \sigma$. U sljedećoj iteraciji točke će biti označene kao outlier ukoliko su njihove srednje vrijednosti udaljenosti od susjeda veće od praga. Rezultati rada funkcije se vide na slici 3.4.



Slika 3.4.: Oblak točaka lijevo poslije `downsample()` 41049 točka i desno poslije `remove_outliers()` 39488 točka

3.2.5. Stvaranje i zapisivanje mreže trokuta

Nakon pripreme oblaka točaka funkcijama `downsample()` i `remove_outliers()` slijedi stvaranje mreže trokuta unutar funkcije `mesh_reconstruction()`. Stvaranje mreže trokuta se može podijeliti u tri koraka. Prvi je estimiranje normala nad oblakom točaka. Drugi je spajanje estimiranih normala i oblaka točaka u zajedniči oblak točaka s normalama. Treći korak je pozivanje algoritma za stvaranje mreže nad novo stvorenim oblakom.

Ispis koda 3.5.: Dio izvornog koda za estimaciju normala iz funkcije `reconstruct_mesh()`

```
1 // Normal estimation
2 pcl::NormalEstimation<PointType, Normal> normEst;
3 pcl::PointCloud<Normal>::Ptr normals (new pcl::PointCloud<Normal>);
4
5 // Create kd-tree representation of cloud,
6 // and pass it to the normal estimation object.
7 pcl::search::KdTree<PointType>::Ptr tree (new
8     pcl::search::KdTree<PointType>);
9 tree->setInputCloud (cloud);
10 normEst.setInputCloud (cloud);
11 normEst.setSearchMethod (tree);
12 // Use 20 neighbor points for estimating normal
13 normEst.setKSearch (20);
14 normEst.compute (*normals);
```

Iz ispisa koda 3.5. se vidi da je prije estimiranja normala nad oblakom točaka potrebno inicijalizirati objekt za spremanje normala i za estimaciju. Nakon toga definira se stablo za pretraživanje oblaka tipa `KdTree`.¹ Stablu se tada predaje oblak za pretraživanje. Objektu za estimaciju `normEst` tada se predaje oblak i stablo te broj susjednih točaka nad kojima se vrši estimacija normala².

Nakon estimacije normala slijedi spajanje estimiranih normala i oblaka u novi oblak točaka s normalama. Kao što je prikazano u ispisu koda 3.6. Taj oblak točaka je prikazan na slici 3.5.

¹K dimenzionalno stablo [18] je detaljno objašnjeno i na stranici http://pointclouds.org/documentation/tutorials/kdtree_search.php

²Estimacija normala detaljno je objašnjena na stranici http://pointclouds.org/documentation/tutorials/normal_estimation.php

Ispis koda 3.6.: Dio izvornog koda za stvaranju mreže iz funkcije `reconstruct_mesh()`

```
1 // Concatenate the XYZ and normal fields
2 pcl::PointCloud<PointTypeN>::Ptr cloud_with_normals (new
3     pcl::PointCloud<PointTypeN>);
4 pcl::concatenateFields (*cloud, *normals, *cloud_with_normals);
5 // cloud_with_normals = cloud + normals
6
7 // Create search tree
8 pcl::search::KdTree<PointTypeN>::Ptr tree2 (new
9     pcl::search::KdTree<PointTypeN>);
10 tree2->setInputCloud (cloud_with_normals);
11
12 // Initialize objects
13 // psn - for surface reconstruction algorithm
14 // triangles - for storage of reconstructed triangles
15 pcl::Poisson<PointTypeN> psn;
16 pcl::PolygonMesh triangles;
17
18 psn.setInputCloud(cloud_with_normals);
19 psn.setSearchMethod(tree2);
20 psn.reconstruct (triangles);
21 psn.setOutputPolygons(false);
```

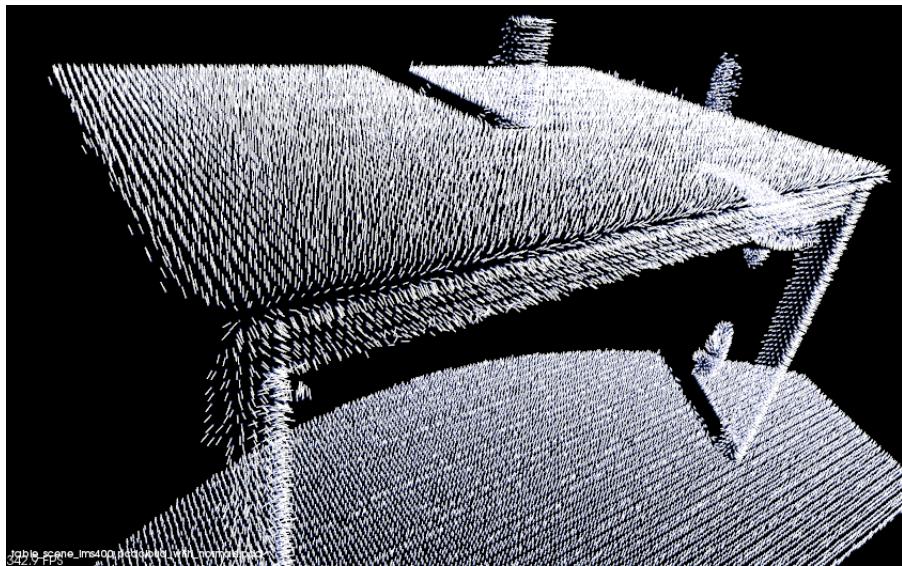
Nad stvorenim oblakom s normalama stvara se stablo za pretraživanje. Zatim se inicijaliziraju objekti `psn` i `triangles`. `psn` predstavlja Poisson¹ algoritam za stvaranje mreže trokuta. `triangles` je objekt tipa `PolygonMesh` za spremanje izračunatih koordinata trokuta. Algoritmu se sada predaje ulazni oblak, stablo pretraživanja i poziva se rekonstrukcija.

Ispis koda 3.7. prikazuje korištenje klase `saveVTKFile` za spremanje objekta `triangles` u datoteku s vtk ekstenzijom.

Ispis koda 3.7.: Dio izvornog koda za zapisivanju mreže iz funkcije `reconstruct_mesh()`

```
1 // Write reconstructed mesh
2 if (argc < 2){
3     pcl::io::saveVTKFile
4         ("pointcloud-downsampled-outliers-mesh.vtk",
5          triangles);
6 }
7 else {
8     std::string str;
9     str.append(argv[1]).append("-mesh.vtk");
10    pcl::io::saveVTKFile (str, triangles);
11 }
```

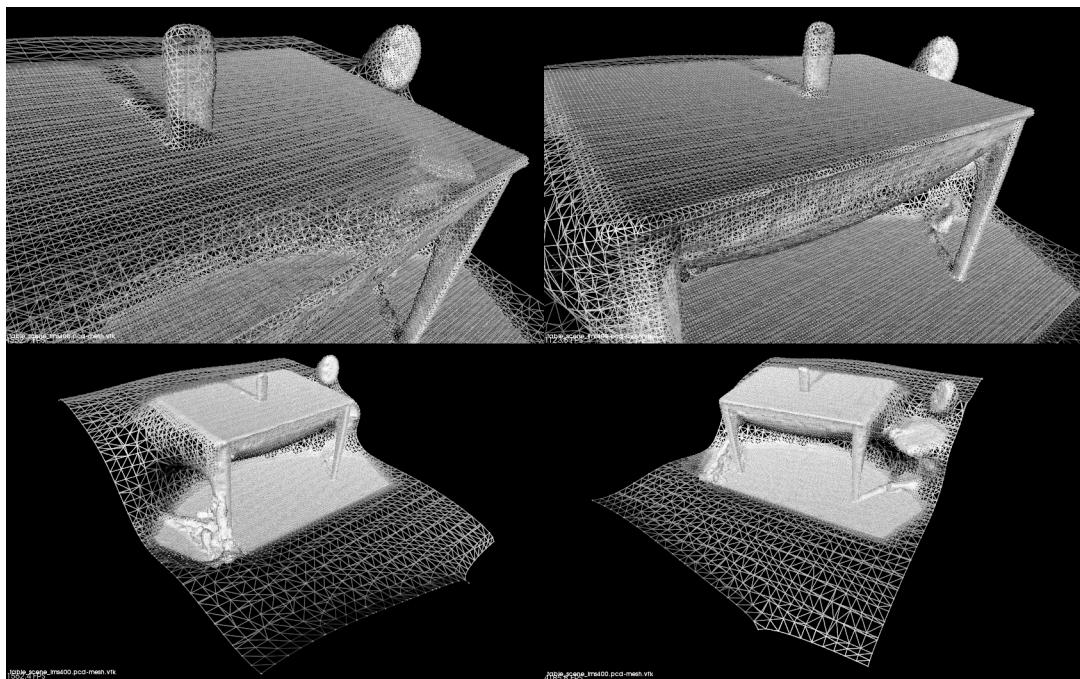
¹Poisson algoritam su razvili Michael Kazhdan i Matthew Bolitho, objavljen je pod BSD licencom. Izvor: <http://www.cs.jhu.edu/~misha/Code/PoissonRecon/Version5.5/>



Slika 3.5.: Prikaz oblaka točaka `tablescene` s estimiranim normalama

3.2.6. Prikazivanje mreže trokuta

Prikazivanje mreže trokuta omogućava `PCLVisualizer` klasu. Ista klasa se koristi u komandno linijskom programu za prikaz oblaka točaka `pcl_vieweru`. U ispisu koda 3.8. se vidi jednostavnost upotrebe klase. Nakon kreiranja objekta `viewer` i postavljanja parametara poziva se beskonačna petlja. Metoda `spinOnce()` izvodi crtanje mreže, prikazivanje na ekranu i daje `vieweru` vremena za procesiranje i time omogućava interaktivnost s mrežom. Klikom na tipku `q` izlazi se iz petlje i program završava. Slika 3.6. prikazuje izgled mreže iz četiri pogleda.



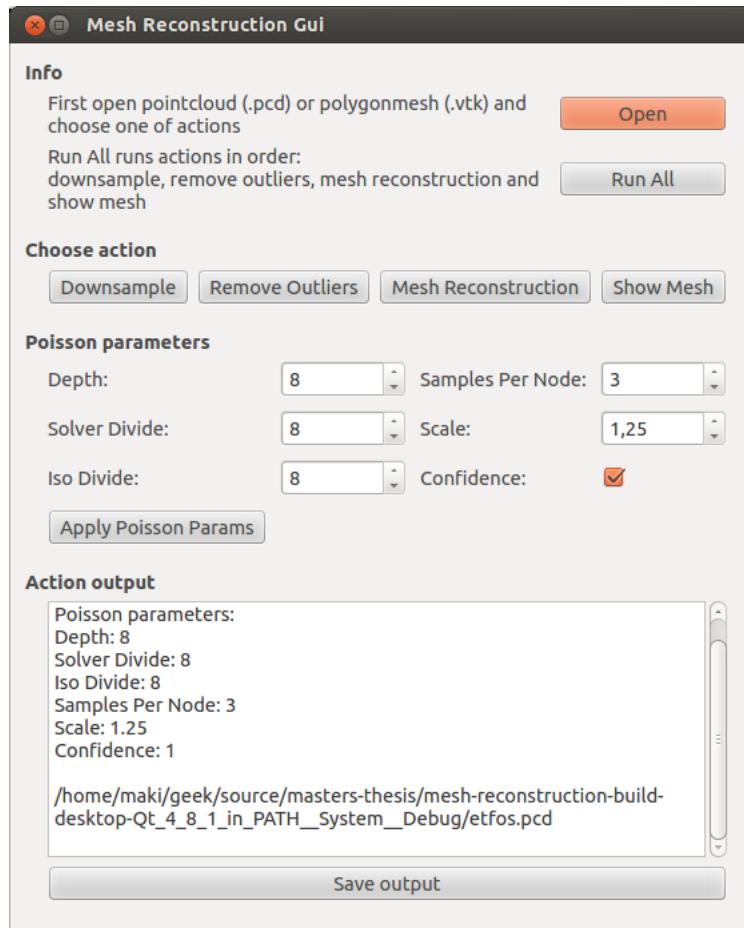
Slika 3.6.: Prikaz mreže trokuta funkcijom `show_mesh()`

Ispis koda 3.8.: Izvorni kod funkcije `show_mesh()`

```
1 void show_mesh (const pcl::PolygonMesh& mesh_of_triangles)
{
3     std::cout << "Started - show_mesh() with PCLVisualizer\n";
4     // Create viewer object and show mesh
5     boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
6         pcl::visualization::PCLVisualizer ("3D Viewer"));
7     viewer->setBackgroundColor (0, 0, 0);
8     viewer->addPolygonMesh (mesh_of_triangles, "sample mesh");
9     viewer->initCameraParameters ();
10    while (!viewer->wasStopped ())
11    {
12        viewer->spinOnce (100);
13        boost::this_thread::sleep
14            (boost::posix_time::microseconds (100000));
15    }
16    std::cout << "Finshed - show_mesh() with PCLVisualizer\n";
17 }
```

3.2.7. Grafičko sučelje programa

Osim komandno linijske verzije programa izrađena je i verzija s grafičkim sučeljem. Grafičko sučelje je napravljeno upotrebom Qt okvira. Qt je C++ višeplatformski okvir za razvijanje aplikacija i korisničkih sučelja. Na slici 3.7. se vidi izgled programa.



Slika 3.7.: Prikaz glavnog prozora programa `mesh-reconstruction`

Sučelje programa je jednostavno i jasno te se sastoji od četiri sekcije i dva prozora. Glavni prozor `Mesh Reconstruction Gui` prikazan je na slici 3.7., a prozor za prikazivanje mreže trokuta `Visualisation` prikazan je na slici 3.8..

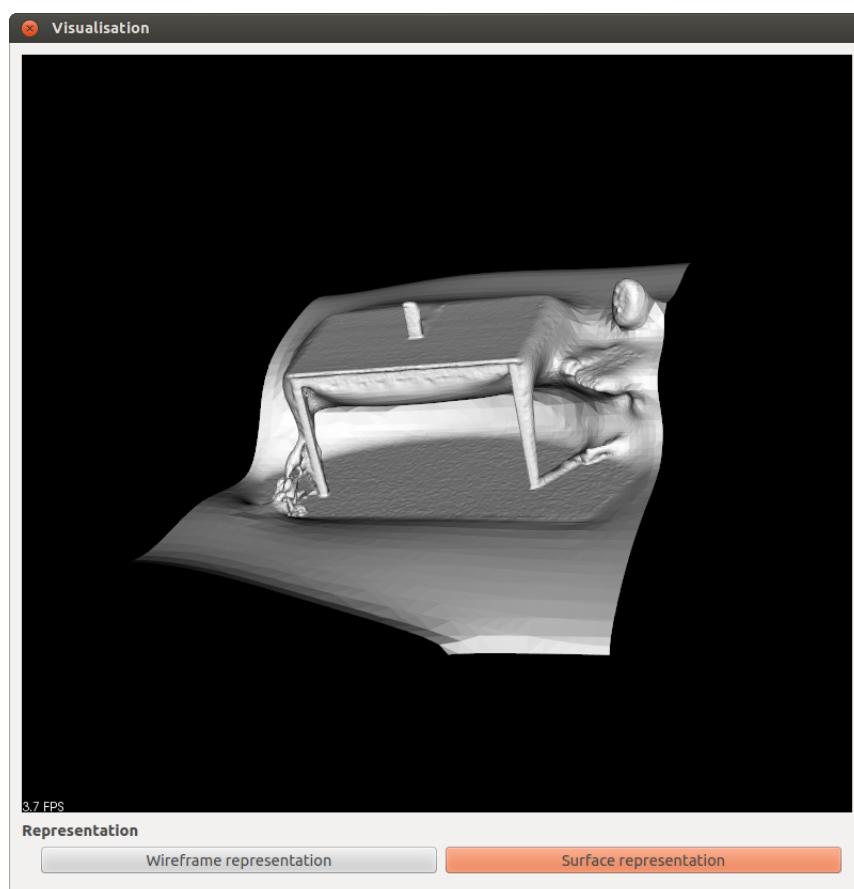
Prozor Mesh Reconstruction Gui

- **Info** sekcija sadrži kratak opis i dugmad:
 - `Open` za odabir datoteke s oblakom točaka (`.pcd`) odnosno datoteke s mrežom trokuta (`.vtk`).
 - `Run All` za pokretanje svih akcija nad odabranim oblakom točaka.
- **Choose action** sekcija sadrži dugmad:

- **Downsample** za reduciranje oblaka točaka.
 - **Remove Outliers** za izbacivanje odudarajućih vrijednosti.
 - **Mesh Reconstruction** za izgradnju mreže trokuta.
 - **Show Mesh** za prikaz izrađene mreže trokuta.
- **Poisson parameters** sekcija prikazuje parametre Poisson algoritma za rekonstrukciju površine. Parametri i njihove vrijednosti su opisani u podpoglavlju 2.5.3.
 - **Action output** sekcija prikazuje ispis informacija pojedinih akcija te ima dugme **Save Output** za spremanje tektsa.

Prozor Visualisation

- Vizualizira kreiranu mrežu trokuta. Sadrži dva dugma:
 - **Surfce representation** prikazuje mrežu trokuta pomoću površine.
 - **Wireframe representation** prikazuje mrežu trokuta upotrebom trokutića.



Slika 3.8.: Prikaz prozora za vizualizaciju mreže trokuta

4. REZULTATI POKUSA

U ovom poglavlju prikazani su rezultati izgradnje, ispitana je funkcionalnost izgradnje 3D modela scena pomoću 3D kamere te su dani podaci o snimljenim scenama i korištenim parametrima. Tijekom izrade diplomskog rada snimljeno je šest scena RGBDSlam programom i iz tih snimaka (slika 4.1.) izgrađeni su 3D modeli objekata i scena upotrebom `mesh-reconstruction` programa. Sve snimke i izrađeni modeli nalaze se na priloženom DVDu. Za ispitivanje funkcionalnosti metode odabrane su tri scene: `etfos-ured-2013-07-30`, `etfos-hol-2013-11-20`, `etfos-hodnik-11-27`. Prije prikaza rezultata potrebno je istaknuti nekoliko problema odnosno ograničenja ove metode.

Proces snimanja scena RGBDSlam programom odnosno akvizicije oblaka točaka je obilježen s par problema i ograničenja. Program se nekoliko puta srušio tijekom snimanja te su snimljeni podaci bili izgubljeni. Ustanovljeno je da se RBGSLam sruši ako je u ručnom načinu rada te se pritisne tipka `Backspace` koja bi trebala obrisati zadnju uzetu sliku. Također prilikom snimanja pažljivo je odabran "put" snimanja svake scene kako bi program uspješno mogao spariti značajke s prethodnim slikama te na kraju zatvoriti petlju i ispraviti akumulirane greške. Naime, slike moraju biti snimane tako da se scene prikazane na uzastopno snimljenim slikama djelom preklapaju, kako bi program mogao detektirati značajke koje su zajedničke tim slikama te na temelju njih odrediti pomak kamere. Unatoč tome kao što se vidi među rezultatima na nekim mjestima program nije uspješno spasio značajke niti je zatvorio petlju.

Proces izgradnje 3D modela scene `mesh-reconstruction` programom obilježen je ograničenjem Poisson algoritma. Algoritam spaja odnosno popunjava regije/rupe bez točaka u modelu. Isto tako implementacija algoritma u PCL biblioteci ne uzima u obzir informaciju o boji prilikom izgradnje modela.



Slika 4.1.: Prikaz svih snimljenih scena

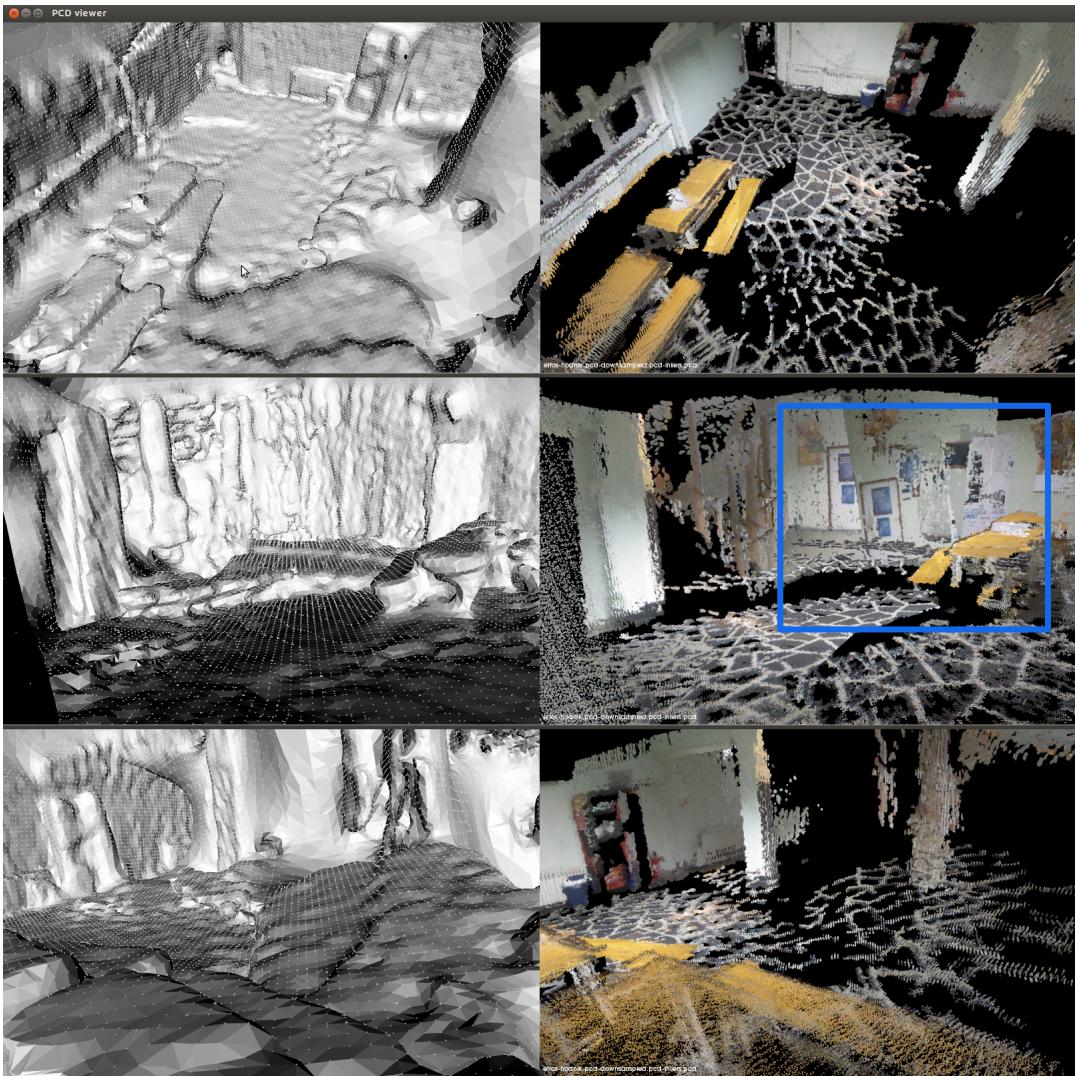
4.1. Prikaz izgrađenih modela scena i objekata

Za prikazivanje izrađenih modela scena i oblaka točaka upotrebljen je komandnolinjski program `pcl_viewer` koji ima opciju usporednog prikazivanja više izgrađenih modela i oblaka točaka iz istog kuta gledanja. Također program ima mogućnost predočavanja izgrađenih modela površinom (svi modeli na slici 4.2.) kao i mrežom trokuta.



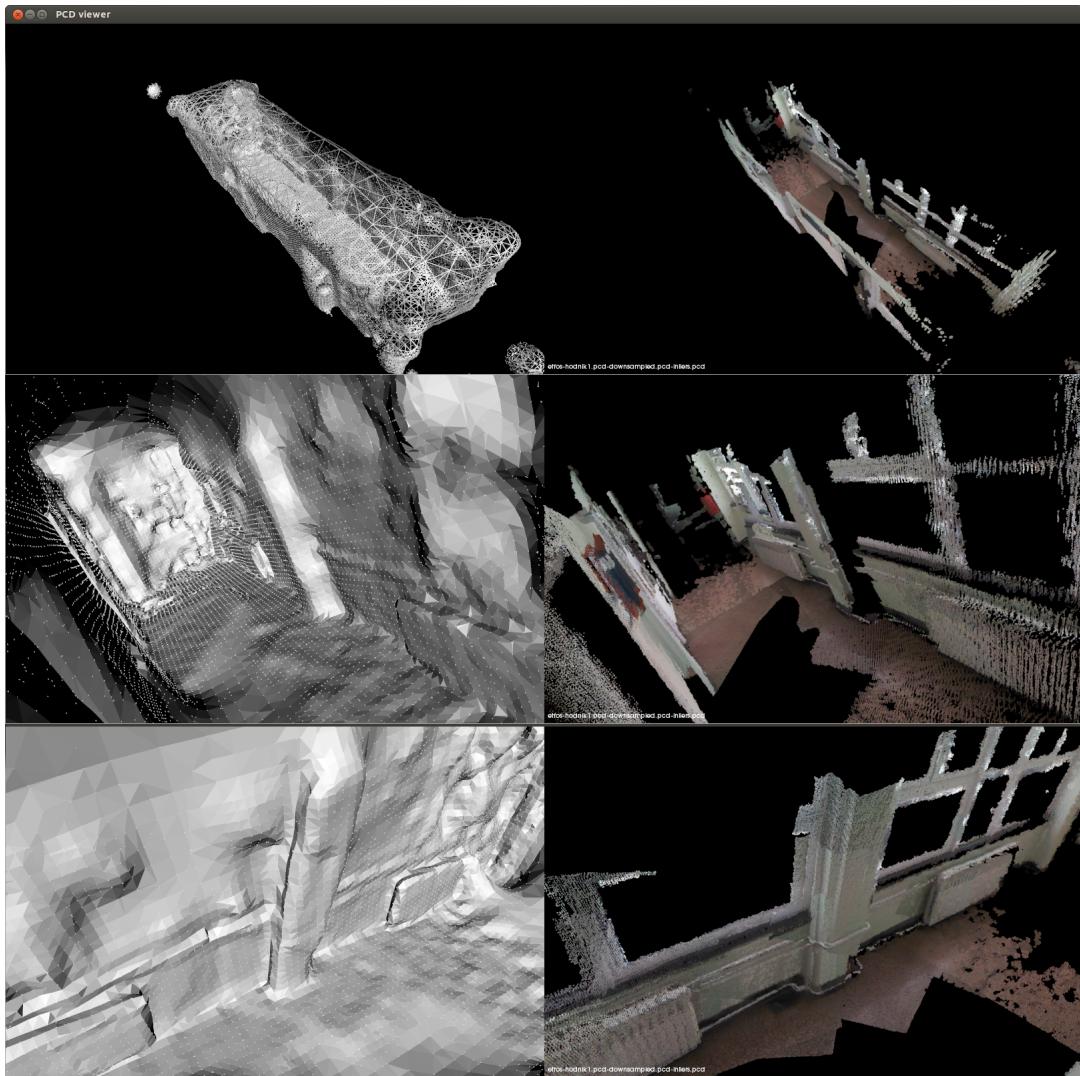
Slika 4.2.: Prikaz izrađene mreže trokuta (lijevo) i snimljenog oblaka točaka (desno) `etfos-ured-2013-07-30`

Slika 4.2. prikazuje izrađenu mrežu trokuta i snimljeni oblak točaka iz tri različita kuta. Oblak točaka u okolini stola ima veliku količinu rupa odnosno regija bez točaka. Ta su mjesta u mreži trokuta popunjena ali zbog nedostatka informacije ne prikazuju stvarno stanje. Predmeti poput zida, ormara, stola i stolice u prostoriji koji su adekvatno predočeni u oblaku točaka su isto tako prepoznatljivi i u izrađenoj mreži trokuta.



Slika 4.3.: Prikaz izrađene mreže trokuta (lijevo) i snimljenog obalaka točaka (desno)
etfos-hol-2013-11-20

Snimanje oblaka točaka na sceni sa slike 4.3. obilježeno je neuspješnim zatvaranjem petlje koje je rezultiralo krivim preklapanjem istih snimki (označeno plavim pravokutnikom). Razlog tome neuspješno je povezivanje zapaženih orijentira jer je scena "monotona", s malom količinom detalja i s jednoličnim stepeništem na visinskoj razlici. Takav oblak točaka iskoristen je za izgradnju modela što se jasno vidi po odstupanju nivoa poda. Preostali dio scene uspješno je snimljen i izgrađen.



Slika 4.4.: Prikaz izrađene mreže trokuta (lijevo) i snimljenog oblaka točaka (desno) **etfos-hodnik-2013-11-27**

Snimanje scene **etfos-hodnik-2013-11-27** uspješno je izvršeno i program je izradio oblak točaka. Kao što je i očekivano staklene površine nisu prikazane jer ne reflektiraju IR zrake. Izrađeni model sa slike 4.4. dobro prikazuje ograničenje Poisson algoritma. Primjer ograničenja metode je izradjivanje mreže trokuta po stropu hodnika. Iako očekivan takav rezultat je nepoželjan. Moguće poboljšanje algoritma bi bilo da ne izrađuje mrežu troukuta na velikim rupama. Ovo svojstvo Poisson algoritma, koje u navedenom primjeru daje neželjeni učinak, u nekim drugim slučajevima daje poželjan rezultat. Primjer je izradjivanje mreže trokuta po prozorima i drugim manjim rupama bez točka. Takvo zatvaranje površina je poželjno i na ovoj sceni uspješno odrađeno.

4.2. Korišteni parametri i podaci o izrađenim modelima

Izgradnja mreže trokuta snimljenih scena izvedena je upotrebom pretpostavljenih vrijednosti Poisson algoritma kao što se vidi iz ispisa 4.9. Pretpostavljene vrijednosti daju zadovoljavajuće rezultate u pogledu detalja, glatkoće i brzine izrade mreže trokuta.

Ispis koda 4.9.: Ispis korištenih parametara iz aplikacije `mesh-reconstruction`

```

1 Writing reconstructed mesh and cloud with normals
2 Finshed - reconstruct_mesh() with Poisson
3 Using these parameters:
4   Depth: 8
5   Solver Divide: 8
6   Iso Divide: 8
7   Samples per Node: 1
8   Scale: 1.1
9   Confidence: 0

```

Tablica 4.1.: Prikaz osnovnih podataka o snimljenim scenama i izgrađenim modelima

etfos-ured-2013-07-30 etfos-hol-2013-11-10 etfos-hodnik-2013-11-27

Veličina datoteke [MByte]	373	211	175
Broj točaka snimljenog oblaka [$*10^3$]	9042	5197	4253
Broj točaka poslije <code>downsample()</code> [$*10^3$]	1709	3336	1027
Broj točaka poslije <code>remove_outliers()</code> [$*10^3$]	1506	2963	938
Veličina datoteke poslije <code>remove_outliers()</code> [MByte]	67	132	42
Broj trokuta u mreži [$*10^3$]	167	79	25
Veličina datoteke mreže trokuta [MByte]	11.9	5.31	1.65

Kao što se vidi iz tablice 4.1. snimke su reda veličine stotinu megabajta, a njihov broj točaka reda je veličine nekoliko milijona iz tog razloga korištene su funkcije `downsample` i `remove_outliers` prije izgradnje mreže trokuta. Te funkcije znatno utječu na veličinu oblaka točaka upotrebljenog za izgradnju mreže trokuta. Iz tablice se isto vidi da koeficijent smanjivanja veličine datoteka prije i poslije funkcija nije konzistentan između snimki. To je očekivano zbog načina na koji funkcije reduciraju oblak točaka i uklanjuju odudarajuće vrijednosti odnosno snimke su imale različitu količinu redundantnih točaka. Količina redundantnih točaka ovisi o načinu snimanja i o broju snimki od kojih je oblak točaka sastavljen. Tablica još prikazuje broj trokuta u mreži i veličinu datoteke mreže trokuta. Veličina datoteke je znatno manja jer izrađena mreža ne sadržava informacije o boji već samo koordinate vrhova trokuta koje čine mrežu trokuta.

5. ZAKLJUČAK

U ovom diplomskom radu razvijen je i predstavljen program za izgradnju mreže trokuta mesh-reconstruction. Program se upotrebljava nad oblakom točaka prikupljenim RGBDSlam programom. Ispitana je funkcionalnost i kvaliteta postupka snimanjem i izgradnjom nekoliko 3D modela scena pomoću 3D kamere. Tijekom izrade rada pojavilo se niz prepreka koje je trebalo riješiti da bi se došlo do upotrebljivog programa. Glavne prepreke su bile prevođenje, instalacija, proučavanje i pokretanja programa RGBDSlam te postavljanje radne okoline za pisanje programa.

Predstavljeni postupak snimanja i izgradnje 3D modela ima svoje prednosti i nedostatke. Nedostatci su vezani uz ograničenja korištenih tehnologija i algoritama. Npr. snimanje Kinectom ograničeno je na prostore u unutrašnjosti, uspješnost snimanja RGBDSlam programom uvelike ovisi o detekciji i sparivanju značajki, Poisson algoritam prilikom rekonstrukcije povšine odbacuje informaciju o boji itd. Prednosti je svakako nabavna cijena kamere koja se može nabaviti za manje od tisuću kuna. Isto tako svi korišteni programi objavljeni su pod slobodnim licencama te su dostupni za proučavanje, poboljšavanje i upotrebu u bilo koje svrhe.

Postoji potencijal opisanog postupka izgradnje odnosno generalna ideja ima značajne mogućnosti buduće primjene. Sve veća dostupnost 3D printerja na tržištu stvara potražnju za jednostavnim i jeftinim rješenjem skeniranja 3D scena i objekata. Osim toga dizajneri FPS (engl. *First Person Shooter*) igara mogli bi koristiti sličan postupak za skeniranje prostorija od kojih bi izgrađivali 3D modele.

Mogućnosti za poboljšanja postoje na svim razinama. Najveći utjecaj na krajnje rezultate bilo bi proširivanje Poisson algoritma logikom za očuvanje informacije o boji. Isto tako algoritmu bi se mogla dodati logika koja uklanja zatvaranje velikih rupa, ideja je da se prođe još jednom kroz izrađenu mrežu te pronađu i odbace veliki trokuti. Kod snimanja scene jednostavno poboljšanje bilo bi korištenje jačeg računala i automatskog snimanja. Najzahtjevniji je, mehanizam zatvaranja petlje o kojemu značajno ovisi kvaliteta rezultata. To je još uvijek predmet intenzivnog istraživanja.

LITERATURA

- [1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.
- [2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [3] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.
- [4] Felix Endres. Rgbdsłam wiki page on ros.org, April 2013. List of dependencies as well as other info on running RGBDslam <http://wiki.ros.org/rgbdslam>.
- [5] Felix Endres, Juergen Hess, Nikolas Engelhard, Juergen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the RGB-D SLAM system. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, Minnesota, 2012.
- [6] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
- [7] Open Source Robotic Foundation. About ros page on ros.org, June 2014. <http://www.ros.org/about-ros/>.
- [8] Jason Geng. Structured-light 3d surface imaging: a tutorial, September 2010. Tutorial is available at <http://www.opticsinfobase.org/aop/viewmedia.cfm?uri=aop-3-2-128&seq=0>.
- [9] Jungong Han, Ling Shao, Dong Xu, and Jamie Shotton. Enhanced computer vision with microsoft kinect sensor: A review. *IEEE T. Cybernetics*, 43(5):1318–1334, 2013.
- [10] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at <http://octomap.github.com>.
- [11] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basic Engineering*, 82(Series D):35–45, 1960.
- [12] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

- [13] Michael Kazhdan, Allison Klein, Ketan Dalal, and Hugues Hoppe. Unconstrained iso-surface extraction on arbitrary octrees. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, pages 125–133, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.
- [14] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.
- [15] J. J. Leonard and Durrant H. Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3), 1991.
- [16] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *COMPUTER GRAPHICS*, 21(4):163–169, 1987.
- [17] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.
- [18] Andrew Moore. A tutorial on kd-trees. Technical report, 1991.
- [19] Pointclouds.org. About pcl page on pointclouds.org, June 2014. <http://pointclouds.org/about/>.
- [20] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- [21] Søren Riisgaard and Morten Rufus Blas. Slam for dummies, Spring 2005. Tutorial is available at http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam blas_repo.pdf.
- [22] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *International Conference on Computer Vision*, Barcelona, 11/2011 2011.
- [23] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *International Conference on Robotics and Automation*, Shanghai, China, 2011 2011.
- [24] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Proceedings of the Second Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-86)*, pages 267–288, Corvallis, Oregon, 1986. AUAI Press.

SAŽETAK

Izgradnja 3D modela scene pomoću 3D kamere je metoda koja upotrebljava nekoliko komplementarnih tehnologija. U radu je razvijen i predstavljen program za izgradnju mreže trokuta iz snimljenog oblaka točaka te je ispitana kvaliteta i funkcionalnost razvijene metode izgradnjom nekoliko 3D modela objekata i scena. Snimanje scena se izvršava upotrebom RGBDSlam programa i Microsoft Kinect kamere. RGBDSlam program kontrolira uzimanje dubinskih slika s kamere i od njih sastavlja oblak točaka koji prikazuje 3D scenu primjenom tehnike istovremene lokalizacije i mapiranja - SLAM. Program je baziran na ROS programskom okviru i OpenCV biblioteci. Dobiveni oblak točaka se koristi za izgradnju 3D mreže trokuta razvijenim programom mesh-reconstruction. mesh-reconstruction se oslanja na PCL biblioteku u kojoj je implementiran Poisson algoritam za izgradnju mreže trokuta. Program ima grafičko sučelje razvijeno pomoću Qt okvira koje omogućava reduciranje i uklanjanje odudarajućih vrijednosti iz snimljenog oblaka, podešavanje Poisson parametara te izgradnju i prikaz mreže trokuta.

Ključne riječi: Kinect, oblak točaka, 3D modeliranje, RGBDSlam, ROS, OpenCV, SLAM, PCL, Poisson, Qt

ABSTRACT

Title: 3D model reconstruction with 3D camera

3D model reconstruction using 3D camera is a process that relies on a few complementary technologies. This thesis presents a developed application for triangle mesh reconstruction from acquired point cloud. The thesis also examines quality and functionality of the developed method by reconstructing few 3D models of objects and scenes. Scene recording is performed using RGBDSlam application with a Microsoft Kinect camera. RGBDSlam controls acquisition of RGB-D images and assembles point cloud representing the observed 3D scene by fusion of the acquired images using SLAM technique. The Program is based on ROS framework and OpenCV library. The obtained point cloud is used for 3D mesh reconstruction using developed program mesh-reconstruction. The program is based on PCL library which contains an implementation of Poisson algorithm for mesh reconstruction. It has a graphical user interface developed in Qt which features functions for downsampling and removing outliers from captured point clouds, it also offers Poisson parameters configuration and has functions for reconstruction and visualisation of constructed triangle mesh.

Keywords: Kinect, point cloud, 3D modelling, RGBDSlam, ROS, OpenCV, SLAM, PCL, Poisson, Qt

ŽIVOTOPIS

Marijan Svalina rođen je 19. kolovoza 1988. godine u Osijeku. Živi u Belišću do 2008. godine i pohađa osnovnu školu Ivana Kukuljevića u Belišću. Godine 2003. upisuje strukovni smjer elektrotehiničar u Srednjoj školi Valpovo, 2007. polaže maturu i iste godine upisuje Prediplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku. Tijekom druge godine studija seli se u Osijek i zapošljava preko studenskog servisa u Hrvatskom Telekomu kao Helpdesk agent. Radi i redovno studira te 2010. završva Prediplomski studij računarstva i upisuje Diplomski studij procesnog računarstva. Početkom 2012. godine prestaje raditi u HTu upisuje Cisco akademiju i uspješno polaže certifikacijski CCNA ispit. U apsolventskoj 2013. godini zapošljava se preko studenstkog servisa kao sistemski i mrežni administrator u Siemens CVCu i radi do kraja akademske godine. Od petog mjeseca 2014. godine radi kao mladi softverski inžinjer u Siemens CVCu u Osijeku.

Potpis:

PRILOZI

Prevođenje i instalacija RGBDSlam programa

Prevođenje i instalacije programa rađena je na Ubuntu 12.04 GNU/Linux distribuciji pomoću skripte prikazane u ispisu 5.10. Upute za ostale distribucije se nalaze na <http://wiki.ros.org/rgbdslam>. Skripta automatizira dodavanje ROS repozitorija i poziva instalaciju svih potrebnih paketa. Zatim postavlja ROS varijable okruženja potrebne za pokretanje RGBDSlam programa. Tada se klonira izvorni kod programa i pozivaju naredbe za instaliranje i prevodenje programa.

Ispis koda 5.10.: Ispis shell skripte za instalaciju rgbdslam programa

```
1 # Better pick a mirror close to you.
2 # See http://ros.org/wiki/ROS/Installation/UbuntuMirrors
3 sudo sh -c '.
4   /etc/lsb-release && echo "deb http://packages.ros.org/ros/
5     ubuntu $DISTRIB_CODENAME main" > /etc/apt/sources.list.d/ros-latest.
6     list'
7
8
9 # This will draw gigabytes from the network:
10 sudo apt-get install ros-fuerte-perception-pcl ros-fuerte-vision-opencv
11   ros-fuerte-octomap-mapping python-rosdep
12 sudo apt-get install ros-fuerte-openni-launch
13
14 echo 'source /opt/ros/fuerte/setup.bash' >> ~/.bashrc
15 echo 'export ROS_PACKAGE_PATH=~/ros:$ROS_PACKAGE_PATH' >> ~/.bashrc
16 . ~/.bashrc
17
18 svn co http://alufr-ros-pkg.googlecode.com/svn/trunk/rgbdslam_freiburg ~/ros/rgbdslam_freiburg
19
20 sudo rosdep init
21 rosdep update
22 rosdep install rgbdslam_freiburg
23   roscd rgbdslam
24
25 # This will take a while:
26 rosmake rgbdslam_freiburg
```