SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

# IZGRADNJA 3D MODELA SCENE POMOĆU 3D KAMERE

Diplomski rad

Marijan Svalina

Osijek, 2013.

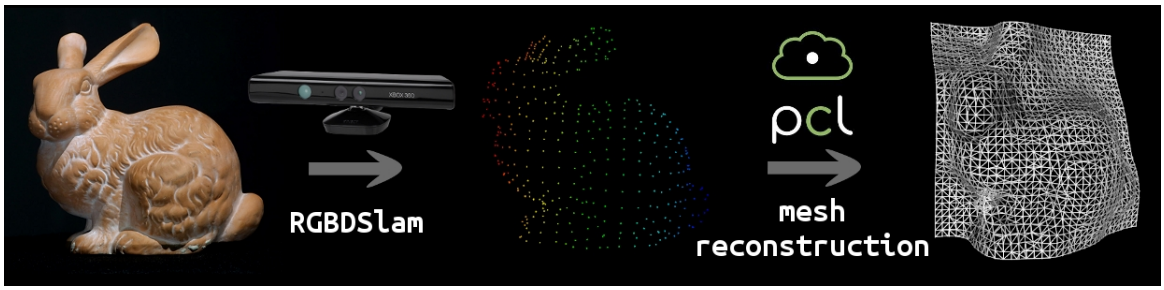
# Sadržaj

# 1. UVOD

## 1.1. Zadatak diplomskog rada

**PRIVREMEN C/P opisa diplomskog**

Program RGBDSLAM raspoloživ u okviru programske biblioteke OpenSLAM omogućava izgradnju 3D modela objekata i scena pomoću 3D kamere. Razviti program za izgradnju 3D modela u obliku mreže trokuta koristeći biblioteku PointCloud. 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.




Slika 1.1.: Grafički prikaz projekta upotrebom Standfordovog zeca[1]

[1]Standford Bunny 3D model su originalno konstruirali 1994 Greg Turk i Marc Levoy i od tada je postao najčešće upotrebljevani 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. Microsof Kineckt 3D kamera

## 2.2. ROS biblioteka i alati

## 2.3. Biblioteka Pointcloud

## 2.4. Istovremena lokalizacija i mapiranje

## 2.5. Poisson algoritam za rekonstrukciju površine

# 3. IZGRADNJA 3D MODELA SCENE

## 3.1. Snimanje scene 3D kamerom i RGBDSlam programom

Program RGBDSlam omogućava brzo prikupljanje 3D modela objekata i scena u unutrašnjosti prostorija (oblak točaka u boji) rukom upravljanom kamerom tipa Kineckt. Razvijen je suradnjom sveučilišta Albert-Ludwigs-Unversität[1] u Freiburgu i Technische Universität München[2]. Slobodan je program objavljen pod GPLv3[3] licencom. Izvorni kod je dostupan na Google code[4] stranicama. U prilogu diplomskog rada nalaze se upute za prevođenje i instaliranje programa.

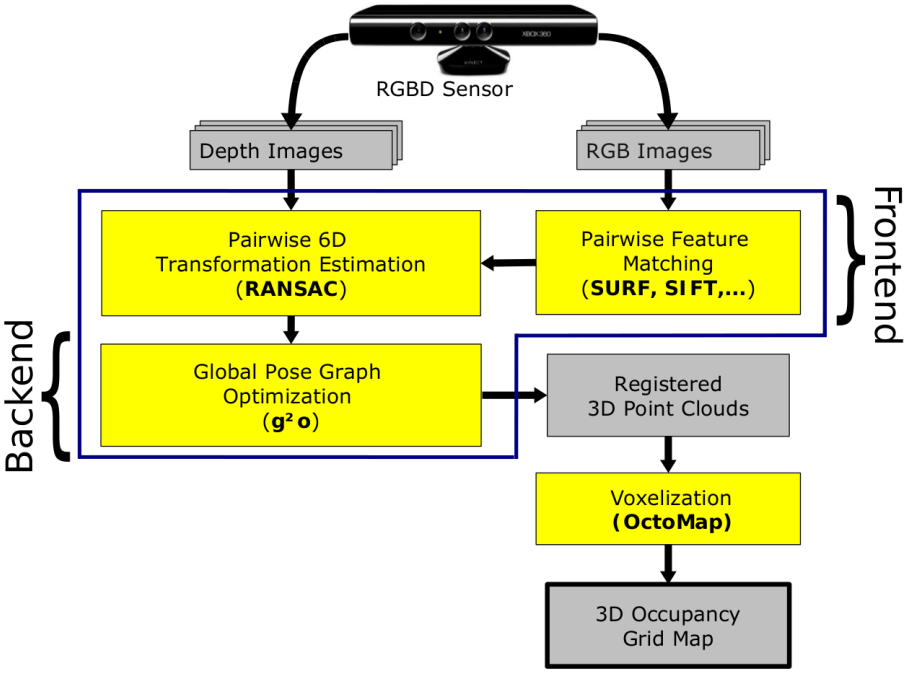
[1] Felix Endres i Juergen Hess sa odijela Autonomous Intelligent Systems koji vodi Prof. Dr. Wolfram Burgard.

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

[3] GNU 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

[4] RGBDSlam 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 pregled[1] RGBDSlam programa

Kao što je vidljivo iz grafikona 3.1. program je podijeljen u četiri osnovna dijela. Prvi dio računa značajke[2] iz ulaznih slika u boji. Zatim se te značajke sparuju sa značajkama iz prethodnih slika. Drugi dio ispituje dubinu slika na lokacijama izračunatih značajki. Time je dobiveno znanje o 3D korespondencijama točaka između bilo koje dvije sličice. Zatim je tim korespondencijama estimirana relativna transformacija između sličica upotrebom RANSACa[3]. Kako parovi estimiranih poza između sličica nisu globalno konzistentni treći dio programa optimizira graf poza upotrebom g$^2$o[4]. 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. Zadnji četvrti dio upotrijebljava Octomap[5] biblioteku kako bi generirio volumetrijski prikaz okoline. Taj dio se uključuje posebnom datotekom prilikom pokretanja programa i nije korišten u ovom radu.

[1] Shema je preuzeta iz znanstvenog rada "An Evaluation of the RGB-D SLAM System" autora Endres, Hess, Burgard, Engelhard, Cremers, Sturm [3].

[2] RGBDSlam se oslanja na OpenCV [2] biblioteku u kojoj su implementirani SURF [1], SIFT [7] i ORB [10] algoritmi za pronalazak značajki.

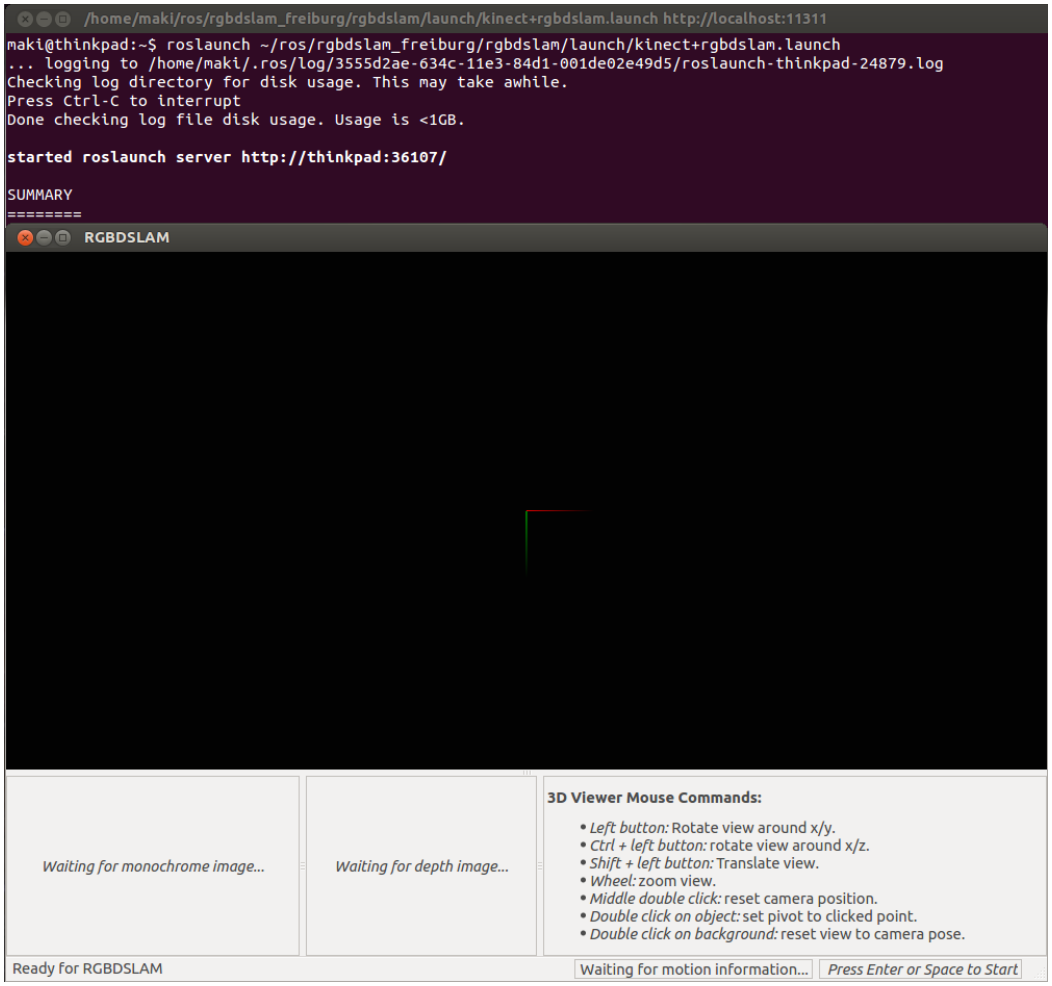
[3] RANSAC Random Sample Consensus [4] je iterativna metoda estimiranja parametara matematičkog modela iz promatranih mjerenja koji sadržavaju šum.

[4] g$^2$o: A General Framework for Graph Optimization [6] je okvir otvorenog koda za optimiziranje graph-based? nelinearnih error? funkcija.

[5] OctoMap [5] je efikasni probabilistički okvir za 3D mapiranje baziran na octree strukturi.

### 3.1.2. Pokretanje RGBDSlam programa

Instaliran program pokreće se preko komandne linije koristeći `roslaunch` koji je dio ROS [9] 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 `kineckt+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

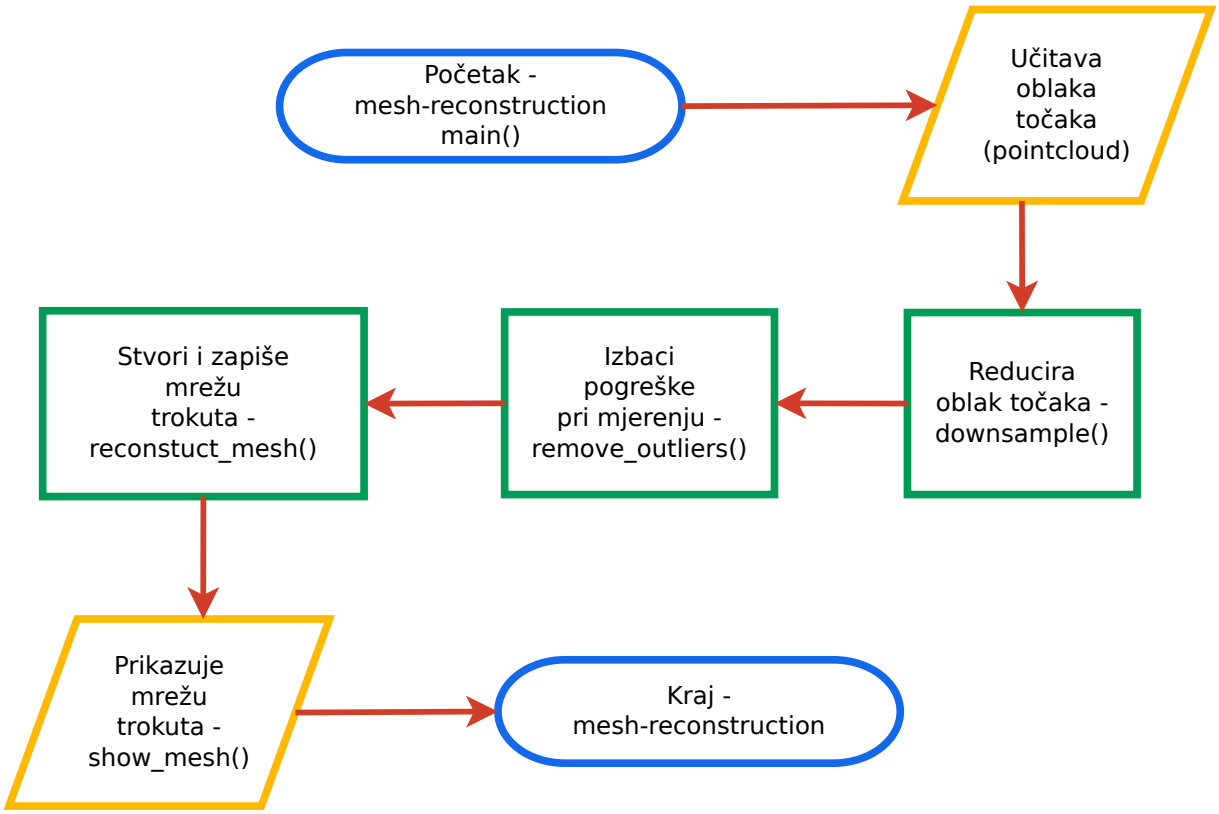
Za snimanje scena upotrijebljen je ručni način rada RGBDSlam programa. Prednost ručnog načina rada je što snimatelj kontrolira broj slika uzetih s kamere. Nedostatak je što je nezgodno jednoj osobi baratati s kamerom, gledat u računalo i pritiskati Enter za slikanje. Zato je pri snimanju scena sudjelovalo više osoba ili je korištena skripta[1] koja umjesto korisnika šalje signal Enter programu nakon proizvoljnog broja sekudni. 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 osam scena odnosno prostorija koje su obrađene u poglavlju 4. Snimljene scene su spremane u `.pcd` formatu i kao takve korištene za daljnju obradu u diplomskom radu.

[1] Skripta za slikanje je dostupna u prilogu.

## 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:

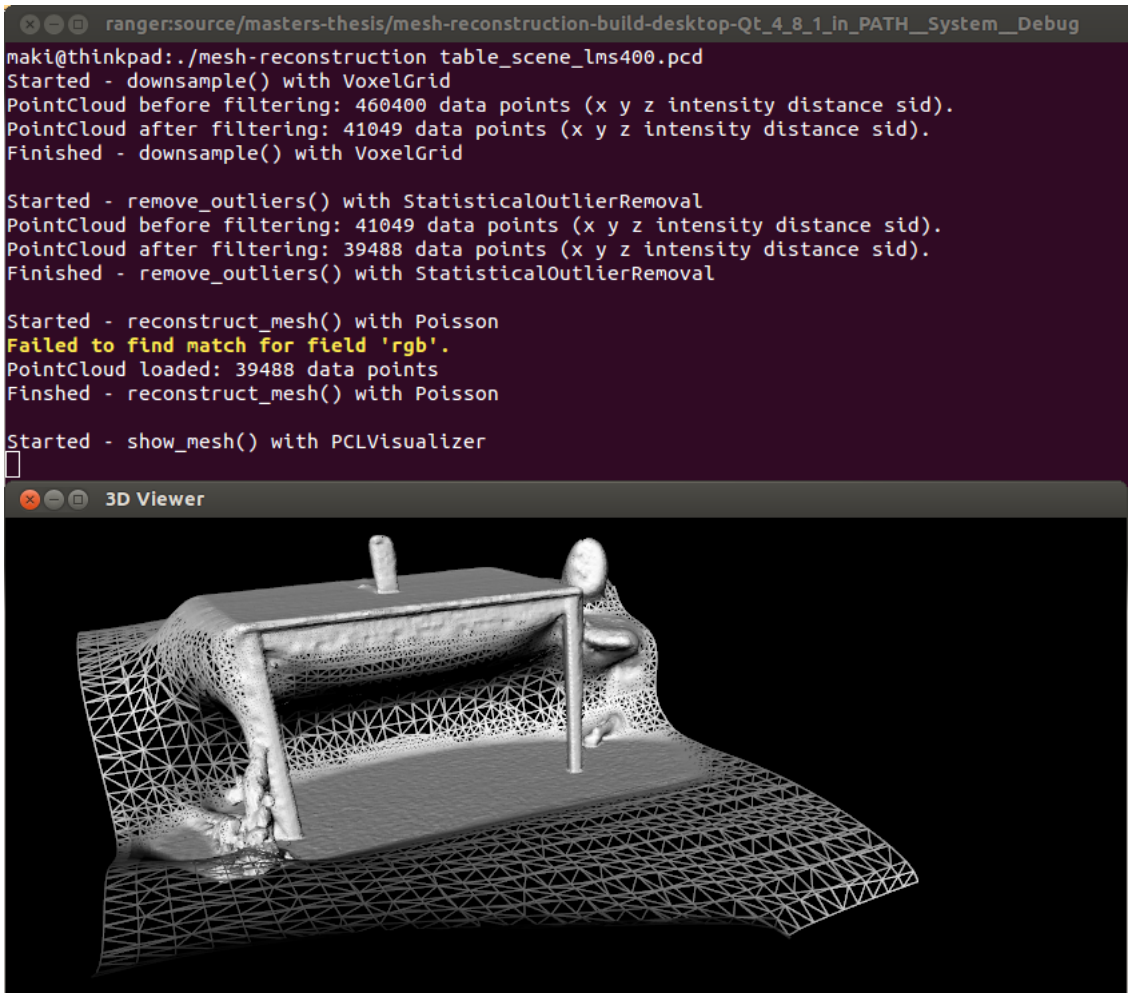
- Učitavanje oblaka točaka snimljenih RGBDSlam programom.
- Reduciranje oblaka točaka.
- Uklanjanje pogrešaka pri mjerenju.
- Stvaranje i zapisivanje mreže trokuta.
- Prikaz mreže trokuta.




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

U sljedećim podpoglavljima dan je pregled funkcija i PCL [11] 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.

[1] Program `mesh-reconstruction` je slobodan program dostupan pod uvijetima MIT licence. Izvorni kod se nalazi u prilogu te na web stranici `github.com/msvalina/`




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

### 3.2.1. Pregled `main()` funkcije

Ispis koda 3.1.: Izvorni kod `main()` funkcije

```
int main (int argc, char *argv[])
{
    downsample (argc, argv);
    remove_outliers (argc, argv);
    pcl::PolygonMesh mesh_of_triangles;
    reconstruct_mesh (argc, argv, mesh_of_triangles);
    show_mesh (mesh_of_triangles);
    return 0;
}
```

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 učitava podatke na početku svake funkcije, te ih zapisuje na izlazu iz funkcije kako bi prije i poslije svake operacije bio dostupan oblak točaka. Za to koristi `PCDReader` i `PCDWriter` klase. Predložak takvog koda se nalazi u ispisu koda 3.2. Nakon učitavanja oblaka točaka `reader` objektom, nad njim se vrše operacije npr. reduciranje oblaka točaka. Nakon toga kreiranjem i korištenjem `writer` objekta promjenji oblak se zapisuje u datoteku.

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

```
    // Init cloud variables
    pcl::PCLPointCloud2::Ptr cloud (new pcl::PCLPointCloud2());
    pcl::PCLPointCloud2::Ptr cloud_filtered (new pcl::PCLPointCloud2());

    // Fill in the cloud data
    pcl::PCDReader reader;
    reader.read ("pointcloud.pcd", *cloud);
    /*
     * Do something with cloud e.g. downsample pointcloud
     */
    // Write cloud to a file
    pcl::PCDWriter writer;
    writer.write ("pointcloud-downsampled.pcd",
            *cloud_filtered, Eigen::Vector4f::Zero(),
            Eigen::Quaternionf::Identity(), false);
```

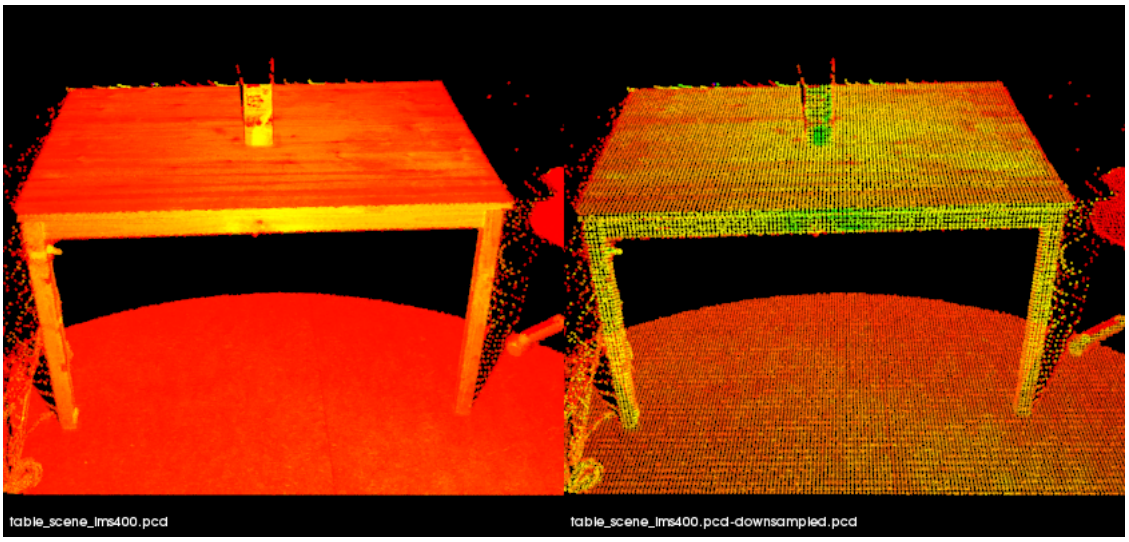
### 3.2.3. Reduciranje oblaka točaka

Reduciranje obalaka ne unosi bitne gubitake informacija, a izvodi se zbog lakše daljnje 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()`

```
// Create the filtering object
pcl::VoxelGrid<pcl::PCLPointCloud2> vg;
vg.setInputCloud (cloud);
// voxel size to be 1cm^3
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

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

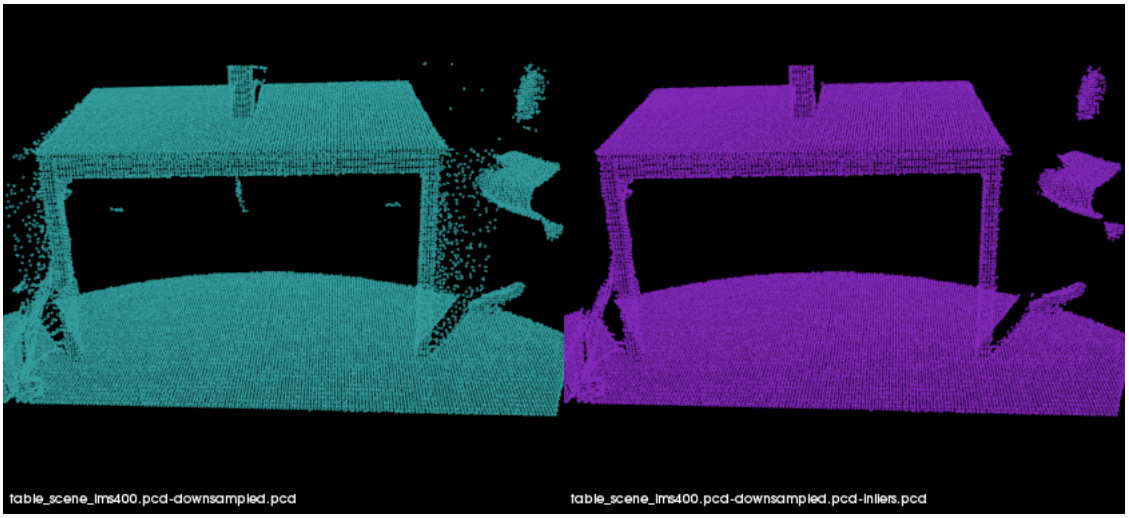
### 3.2.4. Uklanjanje pogrešaka pri mjerenju

Šum pri mjerenju je sastavni dio svakog mjernog uređaja pa tako i Kineckt kamere. PointCloud biblioteka ima ugrađenu `StatisticalOutlierRemoval` klasu koja uklanja šum a implementirana je u funkciji `remove_outlieres()`. Iz ispisa koda 3.4. se vidi kako se klasa koristi.

Ispis koda 3.4.: Dio izvornog koda o uklanjanju šuma 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. Sve točke izvan normalne razdiobe određene jednadžbom $\mu \pm \sigma \cdot$ `StddevMulThresh` bit će označene kao (*outlier*) i odbačene. Odnosno točke u okolini ispitane točke čije su udaljenosti veće od jedne standardne devijacije od očekivane udaljenosti biti označne kao šum i odbačene. 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 o estimaciji normala iz funkcije `reconstruct_mesh()`

```
1     // Normal estimation
      pcl::NormalEstimation<PointType, Normal> normEst;
3     pcl::PointCloud<Normal>::Ptr normals (new pcl::PointCloud<Normal>);

5     // Create kdtree representation of cloud,
      // and pass it to the normal estimation object.
7     pcl::search::KdTree<PointType>::Ptr tree (new
              pcl::search::KdTree<PointType>);
9     tree->setInputCloud (cloud);
      normEst.setInputCloud (cloud);
11    normEst.setSearchMethod (tree);
      // Use 20 neighbor points for estimating normal
13    normEst.setKSearch (20);
      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 normala[2].

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.

[1] K dimenzionalno stablo [8] je detaljno objašnjeno i na stranici http://pointclouds.org/documentation/tutorials/kdtree_search.php

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

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

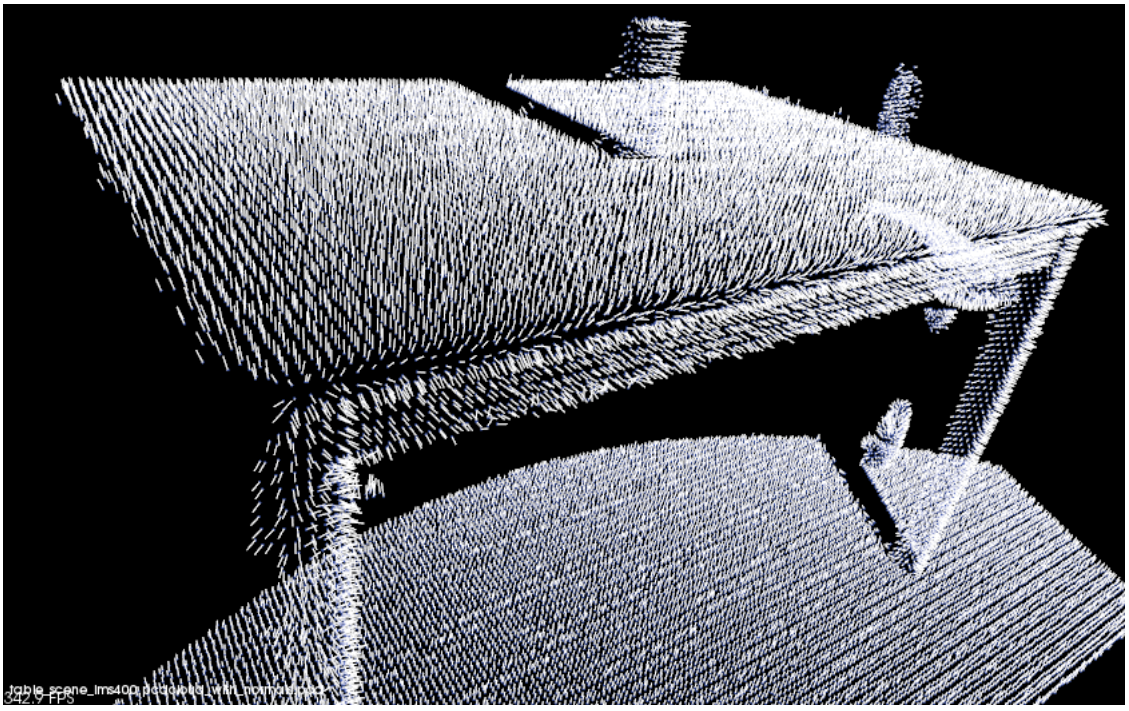
```
    // Concatenate the XYZ and normal fields
    pcl::PointCloud<PointTypeN>::Ptr cloud_with_normals (new
            pcl::PointCloud<PointTypeN>);
    pcl::concatenateFields (*cloud, *normals, *cloud_with_normals);
    // cloud_with_normals = cloud + normals

    // Create search tree
    pcl::search::KdTree<PointTypeN>::Ptr tree2 (new
            pcl::search::KdTree<PointTypeN>);
    tree2->setInputCloud (cloud_with_normals);

    // Initialize objects
    // psn - for surface reconstruction algorithm
    // triangles - for storage of reconstructed triangles
    pcl::Poisson<PointTypeN> psn;
    pcl::PolygonMesh triangles;

    psn.setInputCloud(cloud_with_normals);
    psn.setSearchMethod(tree2);
    psn.reconstruct (triangles);
    psn.setOutputPolygons(false);
```

Nad stvorenim oblakom s normalama stvara se stablo za pretraživanje. Zatim se inicijaliziraju objekti `psn` i `triangles`. `psn` predstavlja `Poisson`[1] 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.



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

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

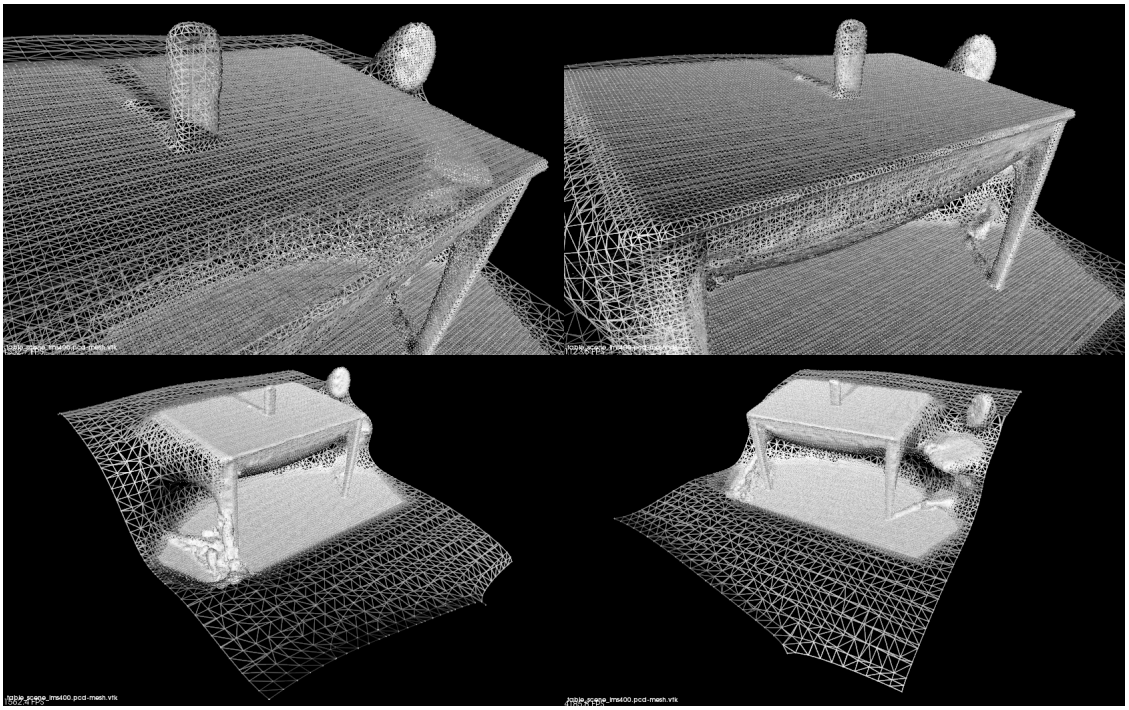
Ispis koda 3.7. prikazuje korištenje klase `saveVTKFile` za spremanje objekta `triangles` u datoteku s vtk ekstenzijom.

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

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

### 3.2.6. Prikazivanje mreže trokuta

Prikazivanje mreže trokuta omogućava `PCLVisualizer` klasa. Ista klasa se koristi u komandno linijskom programu za prikaza 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()` odrađuje 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()`

```
void show_mesh (const pcl::PolygonMesh& mesh_of_triangles)
{
    std::cout << "Started - show_mesh() with PCLVisualizer\n";
    // Create viewer object and show mesh
    boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
          pcl::visualization::PCLVisualizer ("3D Viewer"));
    viewer->setBackgroundColor (0, 0, 0);
    viewer->addPolygonMesh (mesh_of_triangles, "sample mesh");
    viewer->initCameraParameters ();
    while (!viewer->wasStopped ())
    {
        viewer->spinOnce (100);
        boost::this_thread::sleep
            (boost::posix_time::microseconds (100000));
    }
    std::cout << "Finshed - show_mesh() with PCLVisualizer\n";
}
```

# 4. REZULTATI

# 5. ZAKLJUČAK

# 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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[8] Andrew Moore. A tutorial on kd-trees. Technical report, 1991.

[9] 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.

[10] 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.

[11] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *International Conference on Robotics and Automation*, Shanghai, China, 2011 2011.

# SAŽETAK

# ŽIVOTOPIS

# PRILOZI

## Prevođenje i instalacija RGBDSlam programa