Segmentacija dubinske slike na ravne i valjkaste površine metodom nasumičnog uzorkovanja

Findžanović, Duško

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek

Sveučilišni preddiplomski studij računarstva

SEGMENTACIJA DUBINSKE SLIKE NA RAVNE I VALJKASTE POVRŠINE METODOM NASUMIČNOG UZORKOVANJA

Završni rad

Duško Findžanović

Osijek, 2017.

SADRŽAJ

1. UVOD

U današnje vrijeme od suvremenih računalnih sustava očekuje se mogućnost korištenja i manipuliranja informacijama iz stvarnog svijeta, no ti podaci nisu pogodni za računala, te ih ona teško mogu obraditi i još teže koristiti u radu.
Računala ne mogu dobiti sve potrebne informacije iz obične slike, pa je neophodno sliku obraditi postupcima kao što je segmentacija slike kako bi se informacija približila računalnom sustavu i omogućilo istom da procesira informacije i dobivene podatke iz okoline.
Pomoću određenih programskih alata, funkcija i metoda se može dobiti precizna segmentacija slike koja dobro prenosi potrebne informacije iz okoline prilagođene računalnom sustavu.

Ovisno o primjeni, slika se može segmentirati na objekt od interesa i pozadinu, na objekte prisutne na sceni te na geometrijske strukture, koje se koriste na višim razinama analize slike. U ovom radu se obrađuje zadnja od navedenih vrsta segmentacije, tj. segmentacija na ravne i valjkaste površine. Za ostvarivanje programa korištena je metoda RANSAC. Promjenom parametara navedene metode može se segmentirati svaka slika, te sama kvaliteta rezultata uvelike ovisi o njima. Međutim, to nije praktično, zbog toga što nije jednostavno pronaći parametre pri kojima se dobiva dobra segmentacija za sve scene, te se zbog toga teži univerzalnom skupu parametara koji možda nisu optimalni, ali daju dovoljnu aproksimaciju informacije iz okoline.

Korištenjem drugih parametara za svaku sliku se može dobiti optimalan prikaz modela iz stvarnog svijeta, no to nije primjenljivo u praksi, jer se ne može očekivati da će korisnik za svaku sliku podešavati parametre, čime bi se izgubio smisao automatizacije koja se želi postići primjenom računalnog vida. Stoga je potrebno pronaći univerzalne parametre za odgovarajuću primjenu, koji će računalnom sustavu omogućiti prepoznavanje i obradu velikog broja ulaznih slika, uz što je moguće manji gubitak korisne informacije
Cilj i zadatak ovog završnog rada je napraviti program koji radi segmentaciju dubinske slike na ravne i valjkaste površine, testirati ga na 20 primjera koristeći iste parametre i detaljno objasniti dobivene rezultate.
U drugom poglavlju opisan je projekt Point Cloud Library i pojam oblaka točaka.
U trećem poglavlju objašnjen je proces segmentacije slike i opisana je metoda RANSAC, način njezina rada, te prednosti i nedostaci u odnosu na ostale slične metode.

U četvrtom poglavlju naveden je način rada programa izrađenog u okviru ovog završnog rada, te je dan detaljan opis relevantnih naredbi i funkcija.
U petom poglavlju su prikazani i komentirani rezultati testiranja koda na dubinskim slikama.

1.1. Zadatak završnog rada

Korištenjem programske biblioteke Point Cloud Library (PCL) izraditi računalni program za segmentaciju dubinske slike snimljene 3D kamerom na ravne i valjkaste površine. Ispitati učinkovitost razmatrane metode na ispitnom skupu dubinskih slika usporedbom s ručno generiranim referentnim segmentiranim slikama.

2. POINT CLOUD LIBRARY

Point Cloud Library (PCL) je biblioteka algoritama otvorenog koda za procesiranje oblaka točaka i obradu geometrije, primjenjivih za ostvarivanje trodimenzionalnog računalnog vida. PCL okvir sadržava brojne moderne algoritme koji omogućavaju filtriranje, rekonstrukciju površine, procjenu značajki, registraciju i segmentaciju. Zbog lakšeg razvoja, PCL je podijeljen u manje biblioteke koda koje se mogu prevoditi zasebno. Također, takav način rada omogućava instalaciju samo pojedinih komponenti, trošenje manje resursa i mogućnost instalacije PCL-a na velik broj različitih platformi, kao što su Android, Linux, Windows, MacOS i iOS.

Point Cloud Library je pisan u programskom jeziku C++ i objavljen pod uvjetima troklauzne BSD licence koja postavlja minimalne restrikcije na korištenje i distribuciju programskog paketa, te je PCL besplatan za korištenje u edukacijske, ali i u komercijalne svrhe.

Razvoj PCL-a je započeo u ožujku 2010. u Willow Garage, tehnološkom inkubatoru i laboratoriju za istraživanje robotike. Godinu dana kasnije PCL projekt je preseljen na novu web stranicu te je, u svibnju 2011. objavljena prva službena verzija PCL-a, 1.0. Trenutna verzija je 1.8.0.

Iako je PCL projekt nezavisan i sam razvoj nadgledan od strane neprofitne organizacije, velik broj sveučilišta, investitora i tvrtki, kao što su Toyota, Nvidia, Google i Intel, financijski podržava i pomaže razvoju PCL projekta.

2.1.Oblak točaka (Point Cloud)

Oblak točaka je skup točaka u nekom koordinatnom sustavu. Točke su određene svojim koordinatama i koriste se za prikaz površine ili objekta.

Oblaci točaka se mogu koristiti za velik broj funkcija kao što su kreiranje 3D CAD modela objekata, u industrijskoj metrologiji, te za vizualizaciju i animaciju. Iako se oblaci mogu koristiti direktno, oni se najčešće pretvaraju u modele mreže poligona ili trokuta, NURBS modele površine i CAD modele.

3. SEGMENTACIJA SLIKE

Segmentacija slike je proces podjele digitalne slike u segmente, odnosno skupove piksela ili superpiksele. Cilj segmentacije je pojednostavljivanje i pretvaranje slike u nešto što je lakše za razumjeti i analizirati.

Segmentacija podrazumijeva proces lociranja objekata i granica na slikama, te dodjeljivanje oznaka pikselima prema njihovim svojstvima, tako da pikseli sa zajedničkim svojstvima imaju iste oznake. Kao rezultat segmentacije slike se dobiva skup segmenata ili kontura, ovisno o načinu segmentacije, koji pokrivaju cijelu sliku. Svaki od piksela u nekom segmentu dijeli određenu karakteristiku s ostalima, npr. boju ili teksturu. Susjedni segmenti se značajno razlikuju po uzoru na iste karakteristike ili obilježja.

Segmentacija slike primijenjena na oblak točaka se svodi na podjelu oblaka na više različitih grupa. Takav način segmentacije daje najbolje rezultate ukoliko je oblak točaka podijeljen u prostorno izolirane regije. U tom slučaju, grupe se mogu podijeliti na sastavne dijelove, koji se onda mogu zasebno segmentirati.

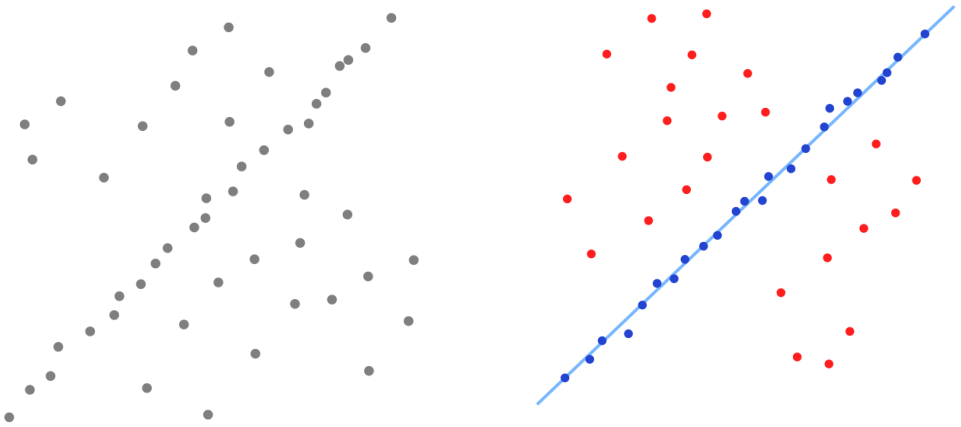
3.1. RANSAC

RANSAC (RANdom SAmple Consensus) je iterativna metoda koja se koristi za procjenu parametara matematičkog modela iz skupa podataka koji sadrže greške (outliers), a te greške ne smiju imati utjecaja na procijenjene vrijednosti. Zbog toga se može smatrati i metodom detekcije grešaka. Algoritam je ne-deterministički, te daje razumne rezultate s određenom točnošću, no ponavljanjem algoritma se dobiva bolja točnost.

RANSAC je popularna metoda zbog svoje robusnosti i jednostavnost, te se najčešće koristi za ostvarivanje računalnog vida.

Algoritam je napisan od strane Martina Fischlera i Roberta Bollesa 1981. godine. [2]

RANSAC algoritam pretpostavlja da se ulazni podatci sastoje od podataka koji pripadaju objektu i podataka koji ne pripadaju objektu ili grešaka. Podatci koji pripadaju objektu su oni podatci koji se mogu objasniti pomoću parametara modela, dok se oni podatci koji se ne mogu objasniti i predstavljaju ekstremne vrijednosti smatraju greškama ili točkama koje ne pripadaju objektu. Greške mogu nastati na razne načine, najčešće zbog šuma ili pogrešnih mjera i interpretacije podataka. Točke koje ne pripadaju objektu nisu uvijek greške, već mogu pripadati nekom drugom objektu.



Slika 3.1.1. Primjer rada RANSAC algoritma (Slika preuzeta iz [1])

Slika 3.1.1. predstavlja primjer primjene RANSAC metode za detekciju ravnih linija u skupu točaka. Algoritam je odabrao samo točke koje odgovaraju parametrima modela te kreirao pravac, dok su ostale točke van modela odbačene i nisu imale utjecaja na odabir parametara pravca.

RANSAC radi na principu uzorkovanja unesenog skupa podataka, te analize tih podataka u svrhu kreiranja modela. Postoje dvije bitne pretpostavke koje su važne za rad ove metode:

- Greške nisu konstantne kao ispravni podaci, te neće imati prevelik utjecaj na odabir modela
- Točnih podataka ima dovoljno kako bi se mogao odabrati odgovarajući model.

RANSAC algoritam se sastoji od dva koraka koji se iterativno ponavljaju.

- Od minimalne količine podataka napravi se podskup uzoraka. Podatci su nasumično odabrani iz unesenog skupa podataka. Model se kreira samo od elemenata tog podskupa.
- Algoritam određuje podskup elemenata ulaznog skupa koji odgovaraju modelu napravljenom u prethodnom koraku. Element se smatra greškom ukoliko ne odgovara modelu, uz određenu toleranciju.

Skup podataka koji se koriste u odabiru modela se naziva skup konsenzusa. RANSAC algoritam će se iterativno ponavljati sve dok se ne sakupi dovoljan broj podataka.

Jedna od najvećih prednosti RANSAC algoritma je robusnost procjene. Algoritam može procijeniti parametre s velikom točnošću, čak i kada ulazni skup podataka sadržava veliki broj grešaka i podataka koji se nalaze izvan traženog modela. RANSAC je također jednostavan, te se koristi kao baza za ostale algoritme procjene parametara.

S druge strane, nedostatci ove metode su to što nije dovoljno točna ukoliko se koristi broj iteracija koji je manji od optimalnog, te se mogu dogoditi slučajevi u kojima odabrani model zapravo ne opisuje podatke ulaznog skupa. Isto tako, RANSAC daje loše rezultate ukoliko broj elemenata koji se nalaze unutar modela nije veći od 50% ali taj problem se može riješiti promjenom parametara procesa, no to nije optimalno za slučajeve u kojima promjena parametara nije moguća.

RANSAC ima problema u slučajevima kada skup ulaznih podataka sadržava dva ili više modela, budući da je algoritam namijenjen za skupove sa samo jednim modelom. Tada se može dogoditi da se RANSAC zbuni i ne uspije pronaći niti jedan model. U slučajevima sa više od jednog modela je nabolje koristiti Houghovu transformaciju koja radi na sličan način. S druge strane, RANSAC je primjenljiv na modele s većim brojem parametara, dok Houghova transformacija postaje nepraktična za modele s više od tri parametra.

Također, RANSAC nema gornji vremenski limit rada, što znači da se algoritam, u nekim slučajevima, može veoma dugo izvršavati. Međutim, postoje varijante koje imaju mogućnost zadavanja fiksnog broja iteracija, odnosno postavlja se vremenski limit.

RANSAC se može primijeniti za detekciju ravnina tako da se u svakoj iteraciji iz ulaznog skupa točaka nasumično odaberu tri točke, odredi se ravnina koja prolazi kroz te tri točke te se odredi podskup točaka ulaznog skupa koje leže na toj ravnini unutar neke zadane tolerancije. Taj se podskup točaka naziva skup konsenzusa. Ravnina koja odgovara najvećem skupu konsenzusa se uzima kao dominantna ravnina, a njezin skup konsenzusa predstavlja jednu ravnu površinu. Taj se skup zatim oduzima od ulaznog skupa i postupak se ponavlja na ulaznom skupu. Na sličan način se mogu detektirati i valjkaste površine.

4. PROGRAMSKO RJEŠENJE ZA SEGMENTACIJU DUBINSKE SLIKE

Programsko rješenje je izrađeno u razvojnom okruženju Microsoft Visual Studio 2013, a za programiranje je korišten programski jezik C++.

Cilj ovog završnog rada je izraditi program za segmentaciju. Iz tog razloga potrebno je dobro odrediti parametre, tako da se iz ulazne datoteke može izraditi model.

Za ostvarivanje segmentacije i vizualizacije rješenja su korištene naredbe i funkcije, koje su definirane u bibliotekama PCL-a, te ih je potrebno uključiti u program.

Kao ulazne podatke, program prima PCD (Point Cloud Data) datoteku, a za izlaz prikazuje rezultat segmentacije na 3D Viewer-u. PCL-ov 3D Viewer je definiran u posebnoj biblioteci koju je potrebno uključiti ako se tako želi prikazati rezultat. Alternativa tomu je spremanje rezultata u datoteku, te korištenje nekog drugog alata za 3D prikaz.

Na početku programa se vrši učitavanje biblioteka potrebnih za rad programa. Osim biblioteke <iostream>, koja sadrži definicije standardnog ulaza i izlaza u programskom jeziku C++, učitavaju se biblioteke PCL-a. One omogućuju korištenje PCL funkcija, npr. biblioteka <pcl/visualization/cloud_viewer.h> omogućuje korištenje PCL-ovog 3D Viewer-a za pregled rezultata programa.

Nakon pokretanja program učitava ulazni oblak točaka iz datoteke i obavlja filtriranje. Nakon toga se određuju parametri segmentatora, te se provjerava koji objekti su dominantniji, plohe ili cilindri. U slučaju da program prepoznaje više ploha, prvo segmentira plohe, a u suprotnom, cilindre. Program nastavlja s radom sve dok se ne izvrši 95% filtriranog oblaka točaka, a ostatak se ignorira. Svaki objekt koji program prepozna se dodaje u zajednički oblak točaka, koji se na kraju prikazuje na zaslonu, te se boja nasumično odabranom bojom.

U nastavku su objašnjeni glavni dijelovi koda, a cijeli kod ovog programa se nalazi u prilozima.

```
reader.read("C:/Users/Duško/Desktop/PCD Datoteke/Blocks_and_stones_Exp00015/image-00003.pcd",
*cloud);
```

Ova naredba služi za učitavanje Point Cloud Data (PCD), odnosno datoteka koje sadržavaju oblake točaka. Za učitavanje se koristi PCL naredba **reader.read** kojoj se predaje apsolutna putanja do datoteke i naziv oblaka točaka u koji se podatci dobiveni iz te datoteke spremaju, u ovom slučaju **cloud.**

```
pass.setInputCloud(cloud);
pass.setFilterFieldName("z");
pass.setFilterLimits(0, 2000.0);
pass.filter(*cloud_filtered);
std::cerr << "PointCloud after filtering has: " << cloud_filtered->points.size() << " data
points." << std::endl;

ne.setSearchMethod(tree);
ne.setInputCloud(cloud_filtered);
ne.setKSearch(50);
ne.compute(*cloud_normals);
```

U ovom dijelu programa se radi filtriranje koje služi za uklanjanje NaN (Not a Number) vrijednosti. NaN-ovi su numeričke vrijednosti koje nisu definirane ili ih nije moguće predstaviti. Potom se obavlja procjena normala.

```
pass.setFilterLimits(0, 2000.0);

segPlane.setMaxIterations(10000);
segPlane.setDistanceThreshold(10.0);
segPlane.setNormalDistanceWeight(0.2);

segCylinder.setNormalDistanceWeight(0.07);
segCylinder.setMaxIterations(15000);
segCylinder.setDistanceThreshold(10.0);
segCylinder.setRadiusLimits(10.0, 200.0);
```

Ove naredbe služe za postavljanje parametara segmentacije. **`pass.setFilterLimits`** označava opseg filtriranja, a **`seg.setMaxIterations`** maksimalni broj iteracija RANSAC algoritma. Parametar **`seg.setDistanceThreshold`** određuje kolika je maksimalna dopuštena udaljenost točke da se ona može smatrati kao dio modela. **`seg.setNormalDistanceWeight`** postavlja koliko važnosti se daje normalama, a **`seg.setRadiusLimits`** granice polumjera valjka.

Postavljanjem boljih parametara se dobiva kvalitetnija segmentacija, no time se povećava i vrijeme rada programa. U zagradama su postavljene vrijednosti parametara koje daju najbolju segmentaciju slika odabranih za pokus.

```
uint8_t r = (BYTE)(rand() % 255);
uint8_t g = (BYTE)(rand() % 255);
uint8_t b = (BYTE)(rand() % 255);

	for (i = 0; i < cloud_plane->points.size(); i++)
	{
		cloud_plane->points[i].r = r;
		cloud_plane->points[i].g = g;
		cloud_plane->points[i].b = b;
	}
```

Ove naredbe služe za pridruživanje boje točkama oblaka. Svaka točka objekta dobiva RGB vrijednost, koja je slučajan broj od 0 do 255. Na taj način svaki objekt sa slike će imati svoju boju.

```
pcl::visualization::CloudViewer viewer("Cloud Viewer");
viewer.showCloud(cloud_all);
while (!viewer.wasStopped())
	{
		user_data++;
	}
```

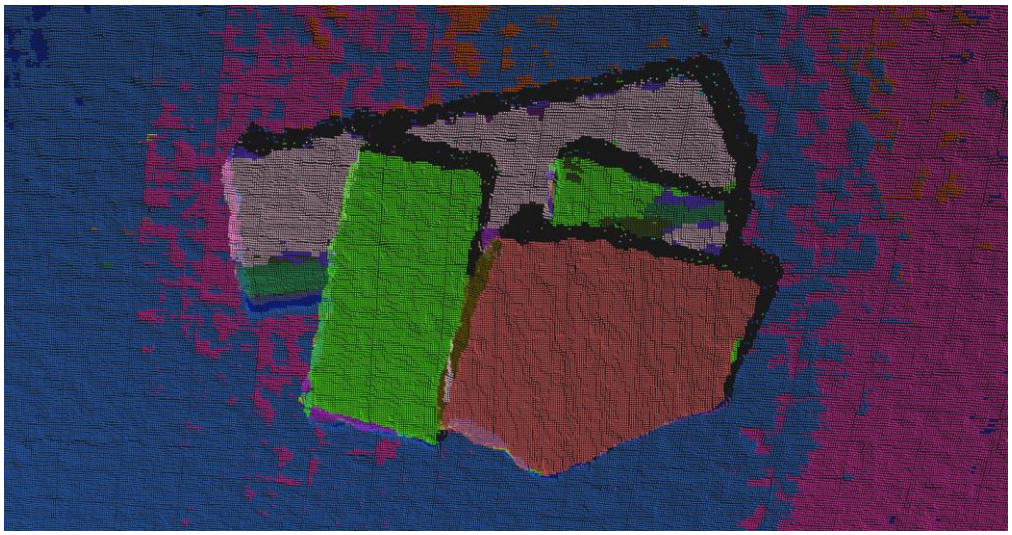
Ove naredbe služe za prikaz dobivenog oblaka točaka na zaslon, korištenjem ugrađenog Point Cloud-ovog 3D Viewer-a.

5. REZULTATI POKUSA

U ovom dijelu rada su prikazani rezultati pokusa i prateći komentari. Prilikom provođenja pokusa koristilo se ukupno 20 slika, rezolucije 640x480 piksela. Svaki pokus je opisan ulaznom slikom (a), slikom sa označenim bridovima objekata ulaza (b), slikom segmentacije (c) i komentarom.

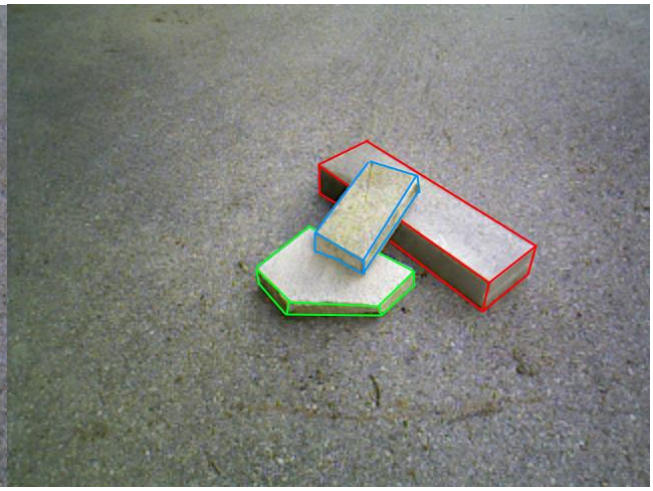
Slika 8.1.a Ulazna slika

Slika 8.1.b Označeni bridovi

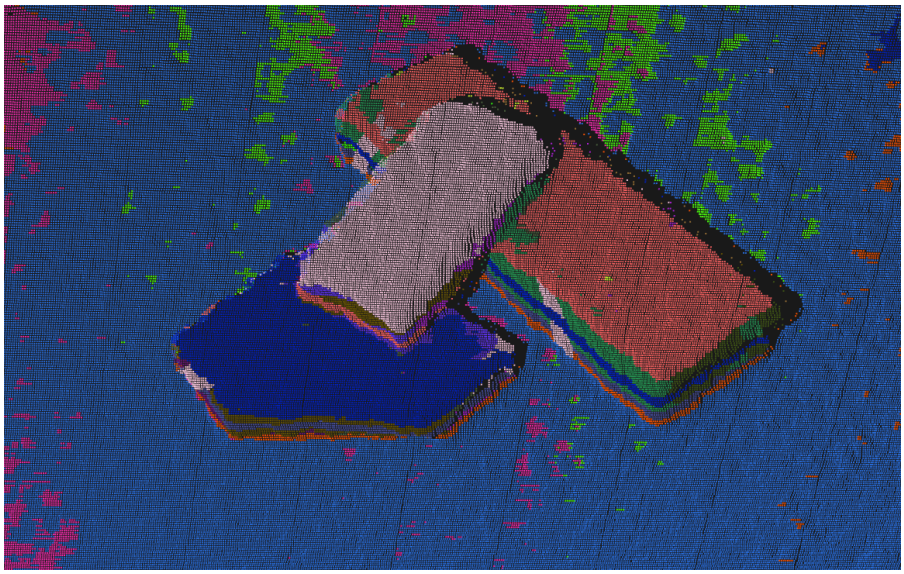


Slika 8.1.c Segmentirana slika

Slika 8.2.a Ulazna slika

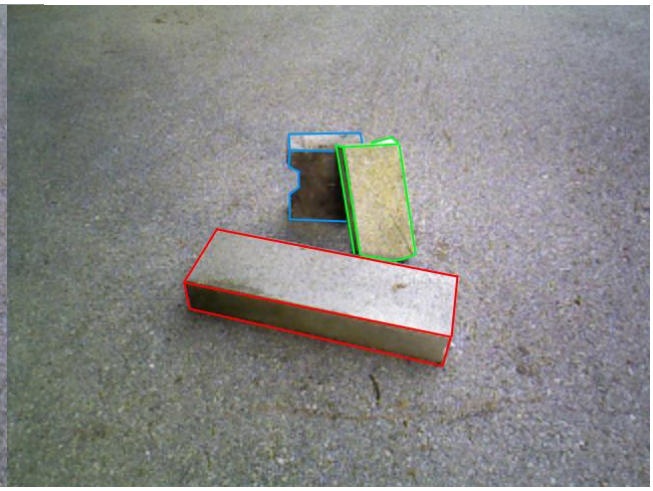


Slika 8.2.b Označeni bridovi

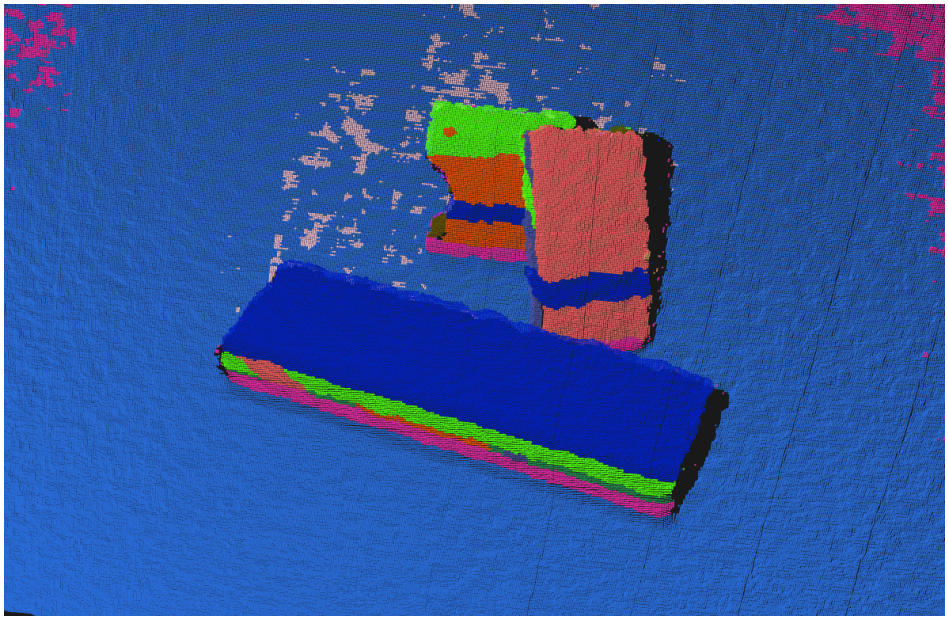


Slika 8.2.c Segmentirana slika

Slika 8.3.a Ulazna slika



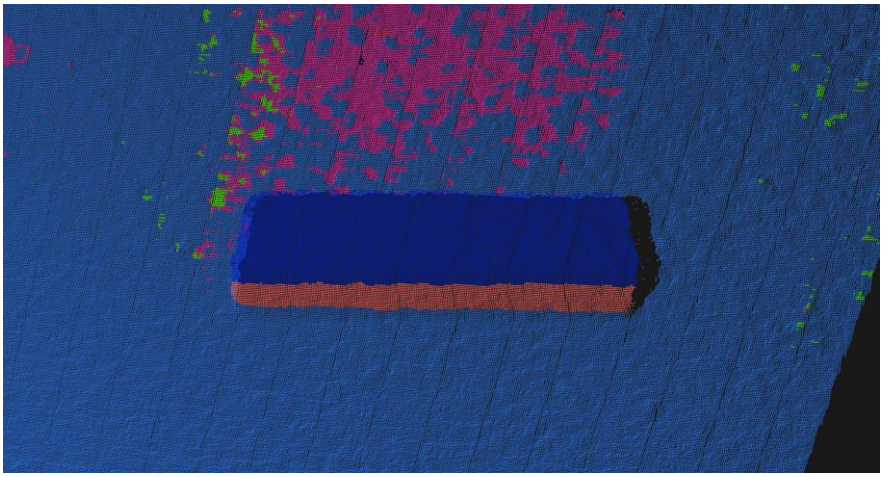
Slika 8.3.b Označeni bridovi



Slika 8.3.c Segmentirana slika

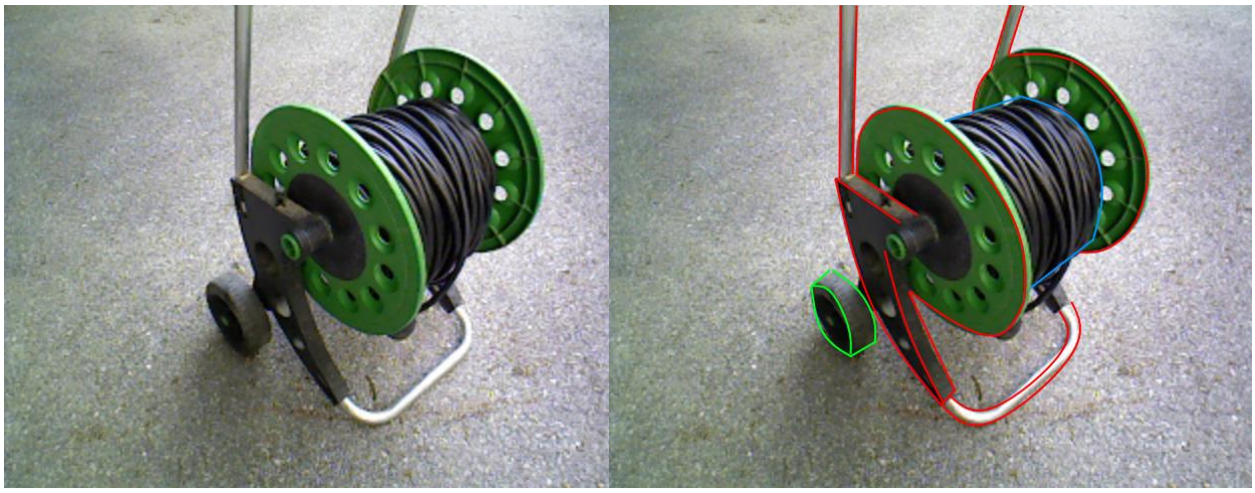
Slika 8.4.a Ulazna slika

Slika 8.4.b Označeni bridovi



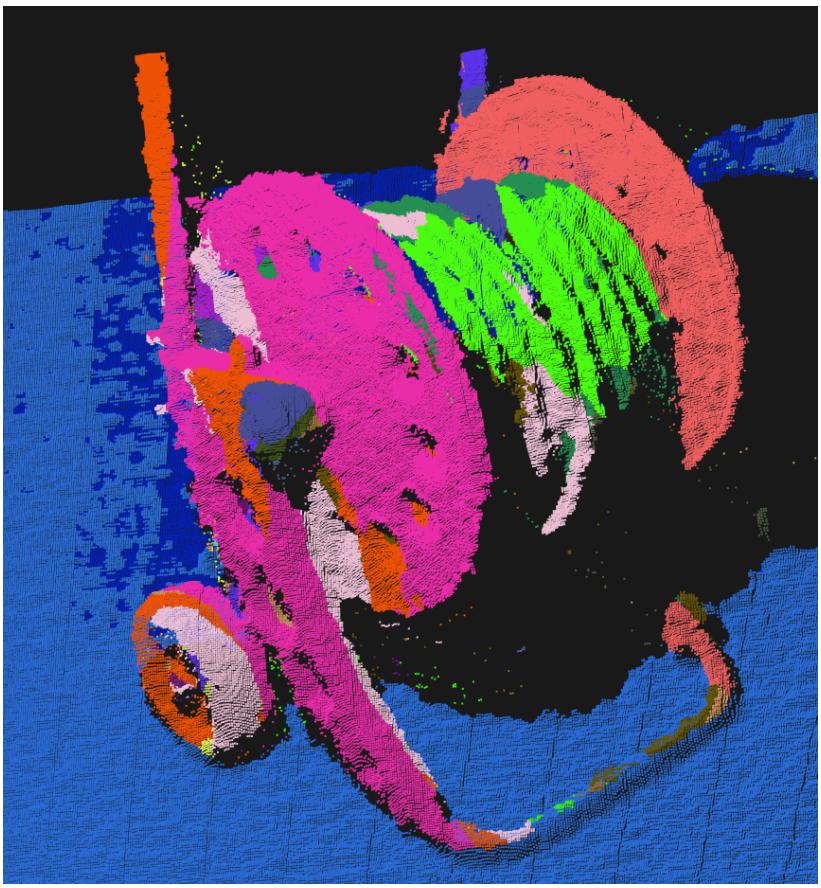
Slika 8.4.c Segmentirana slika

Kao što se može vidjeti na slikama 8.1., 8.2., 8.3. i 8.4. oblik modela uspješno je segmentiran, no iz obojanih dijelova vidi se da program neke od ploha nije prepoznao kao jedinstvene. To se može objasniti time što program zaista smatra dijelove određenih ploha kao zasebne objekte sa ulazne slike zbog npr. sjene, osvjetljenja, nepravilnosti plohe itd.



Slika 8.5.a Ulazna slika

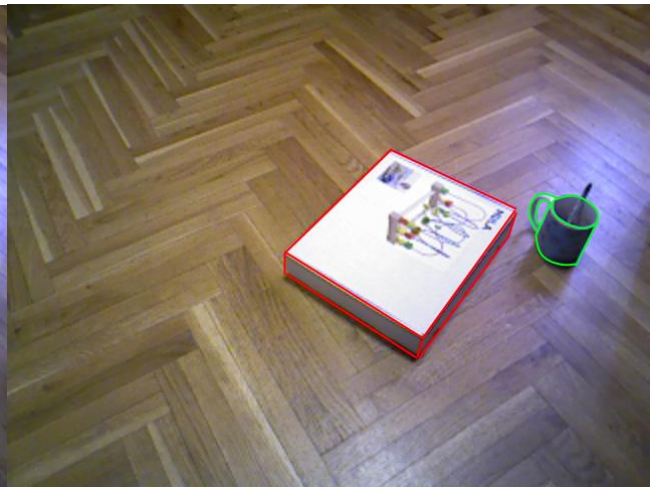
Slika 8.5.b Označeni bridovi



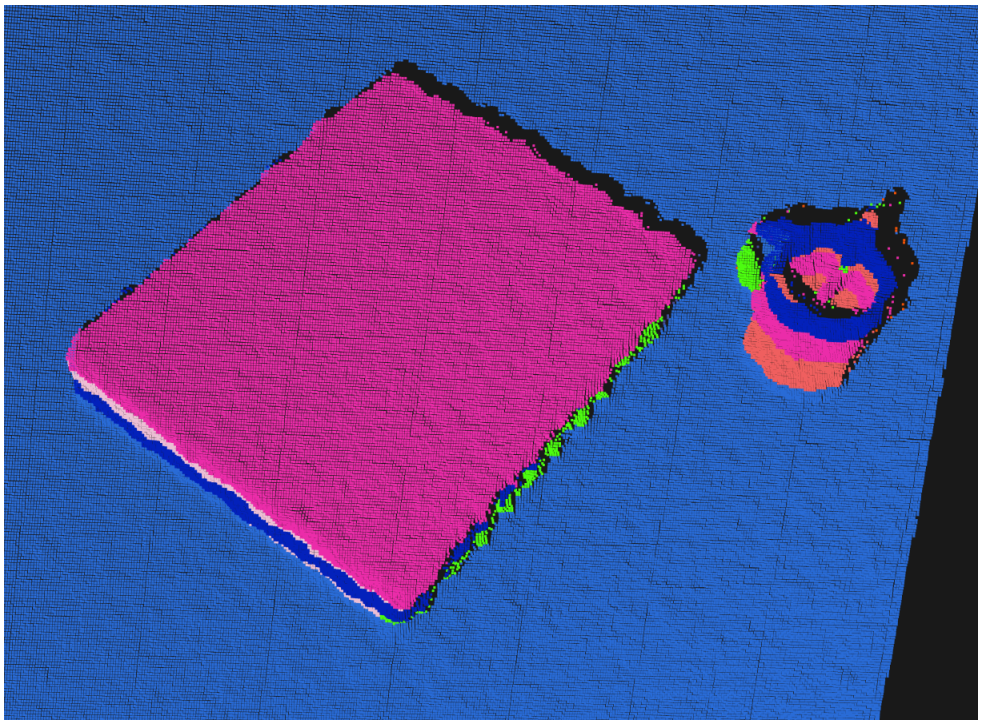
Slika 8.5.c Segmentirana slika

Sa slike 8.5. se može vidjeti da je oblik modela uspješno segmentiran, ali vidljive su greške kod crijeva, jer program ne prepoznaje namotaje i spaja ih u jedan objekt. Zbog razlike u boji vidljivo je da program nije prepoznao prednji dio kao jedinstven objekt, te ga je segmentirao u više dijelova, što nije slučaj sa stražnjim dijelom koji je ostao cjelovit.

Slika 8.6.a Ulazna slika



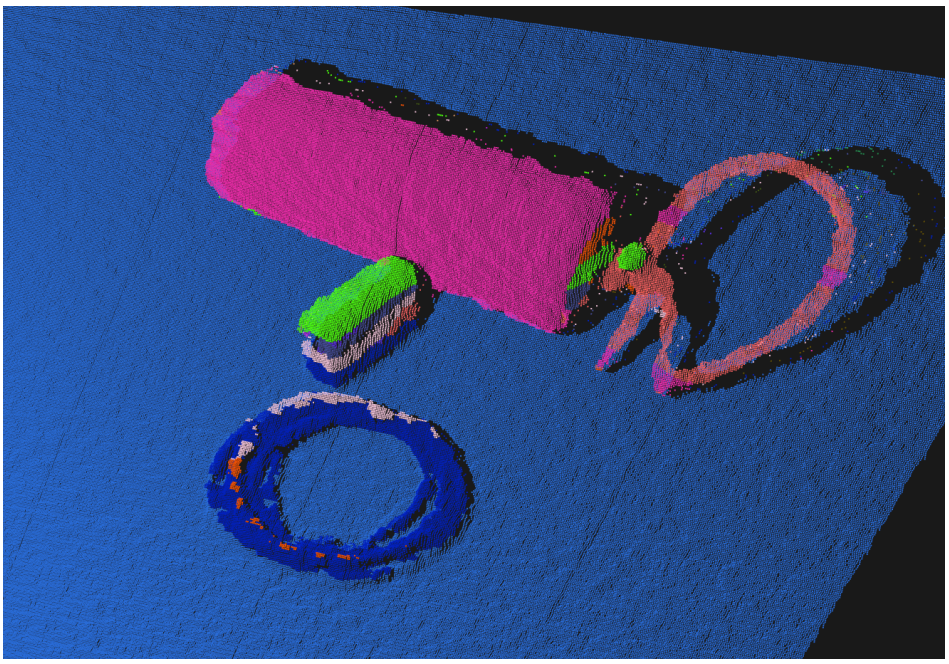
Slika 8.6.b Označeni bridovi



Slika 8.6.c Segmentirana slika

Slika 8.7.a Ulazna slika

Slika 8.7.b Označeni bridovi



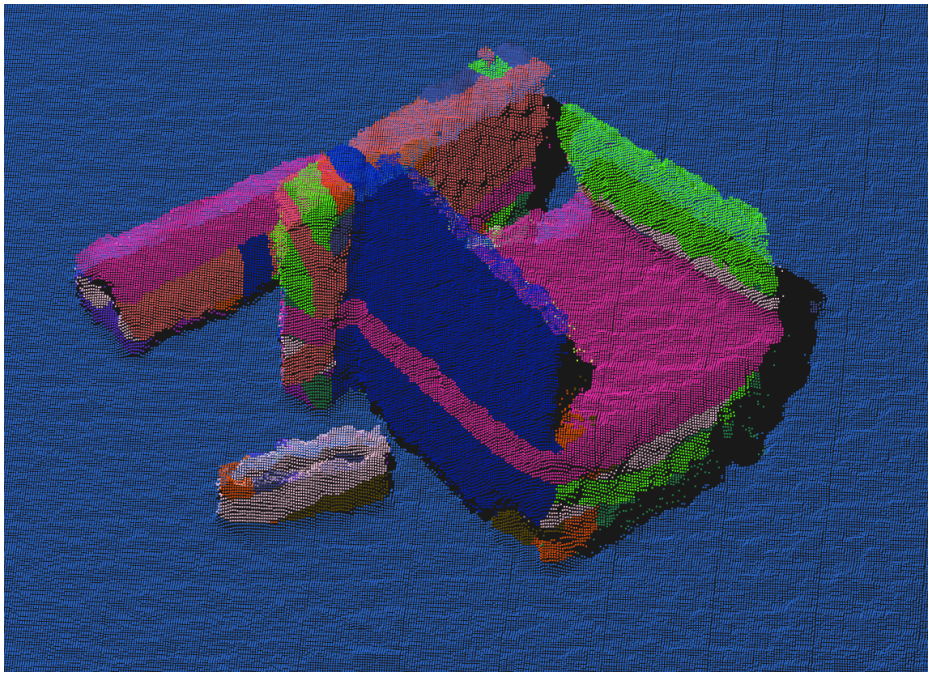
Slika 8.7.c Segmentirana slika

Na slici 8.6. može se vidjeti dobra segmentacija ravne plohe kutije, dok je valjkasta ploha loše odrađena. Naime, valjak je podijeljen u više objekata i samim tim vidljivo je više boja.

U slučaju 8.7. situacije je obrnuta, jer je valjkasti objekt uspješno segmentiran i program ga je prepoznao kao cjelovit objekt, dok je kutija podijeljena na više objekata u različitim bojama. Razlog tome može biti pogreška pri učitavanju ulazne datoteke budući da su neki slični objekti dobro segmentirani. Program je također prepoznao oba crijeva kao jedinstvene objekte i sukladno tome ih obojao u istu boju.

Slika 8.8.a Ulazna slika

Slika 8.8.b Označeni bridovi

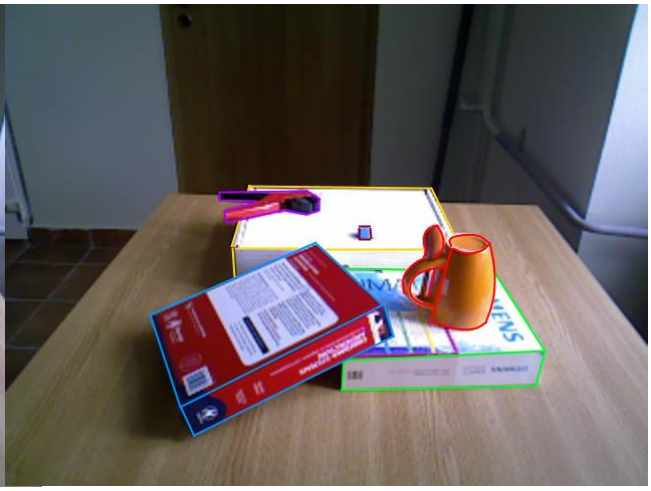


Slika 8.8.c Segmentirana slika

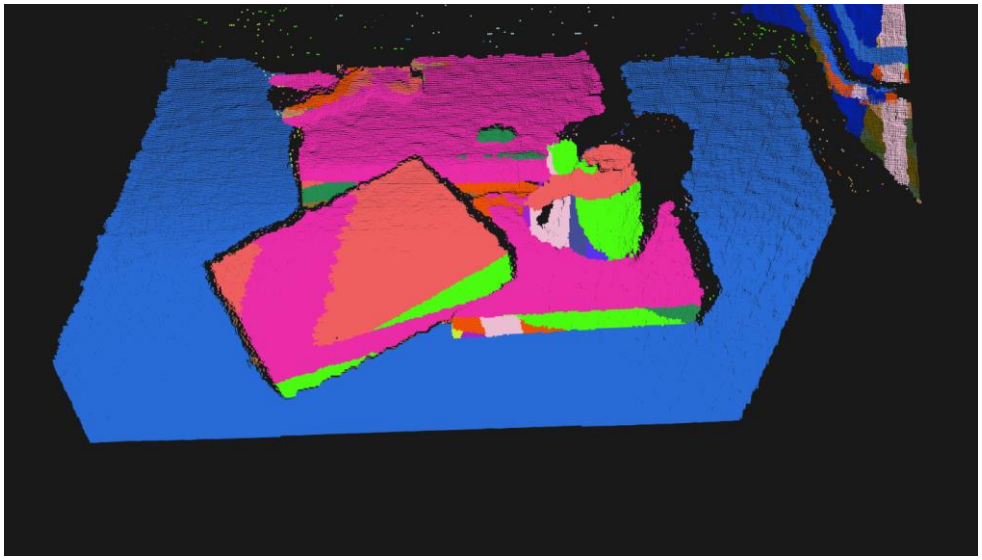
Iako objekti sa segmentirane slike 8.8. oblikom odgovaraju ulaznoj datoteci, prema bojama mogu se uočiti greške. Program je podijelio svaki objekt na više dijelova i svaki dio drugačije obojao.

Također je vidljivo da je jedan valjak dobro segmentiran i prepoznat kao jedinstven objekt, dok drugi nije i program ga je podijelio na više dijelova.

Slika 8.9.a Ulazna slika



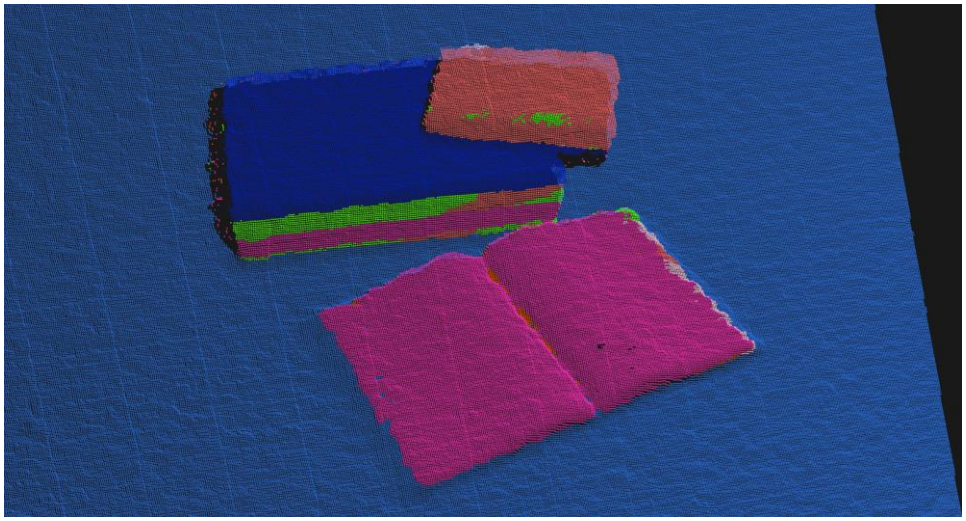
Slika 8.9.b Označeni bridovi



Slika 8.9.c Segmentirana slika

Slika 8.10.a Ulazna slika

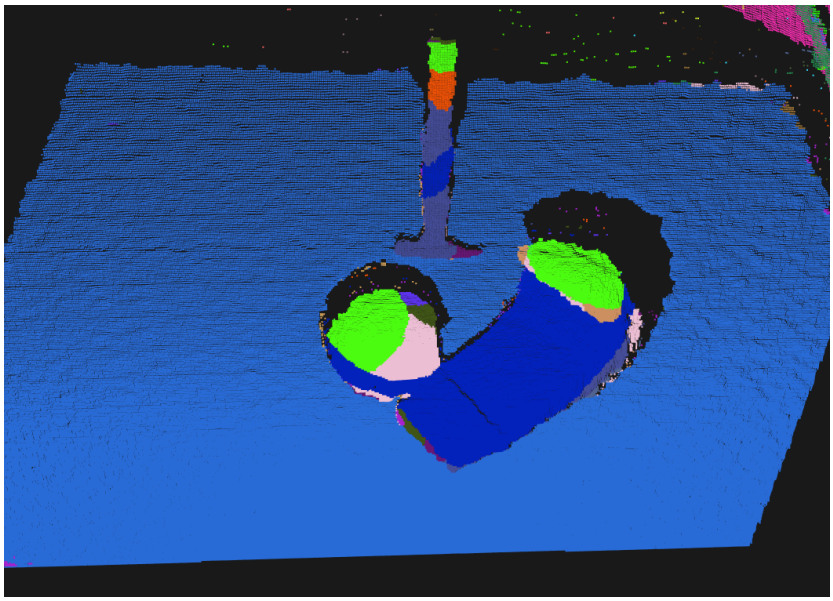
Slika 8.10.b Označeni bridovi



Slika 8.10.c Segmentirana slika

Slika 8.11.a Ulazna slika

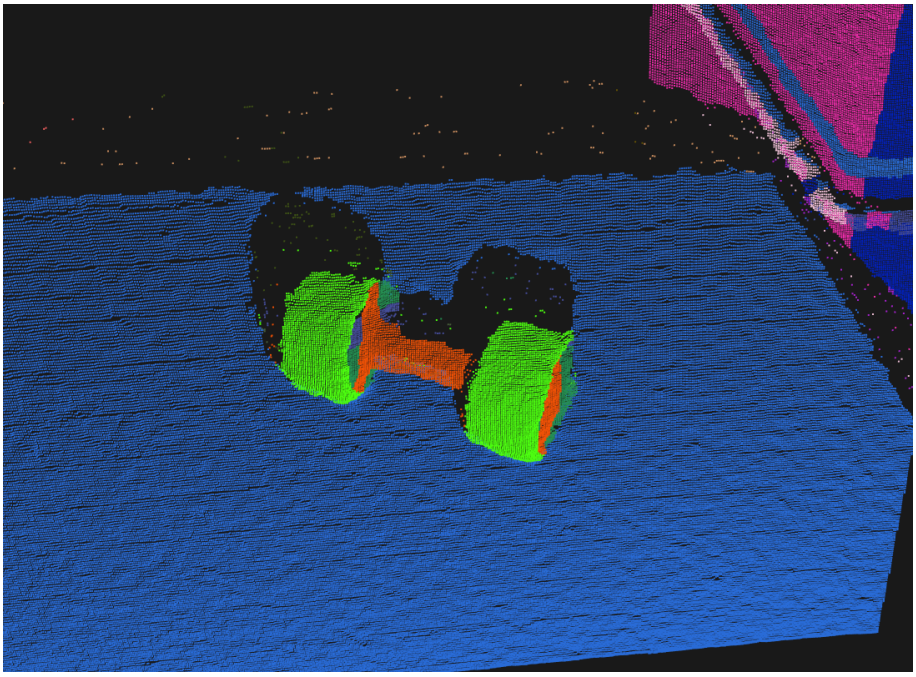
Slika 8.11.b Označeni bridovi



Slika 8.11.c Segmentirana slika

Slika 8.12.a Ulazna slika

Slika 8.12.b Označeni bridovi



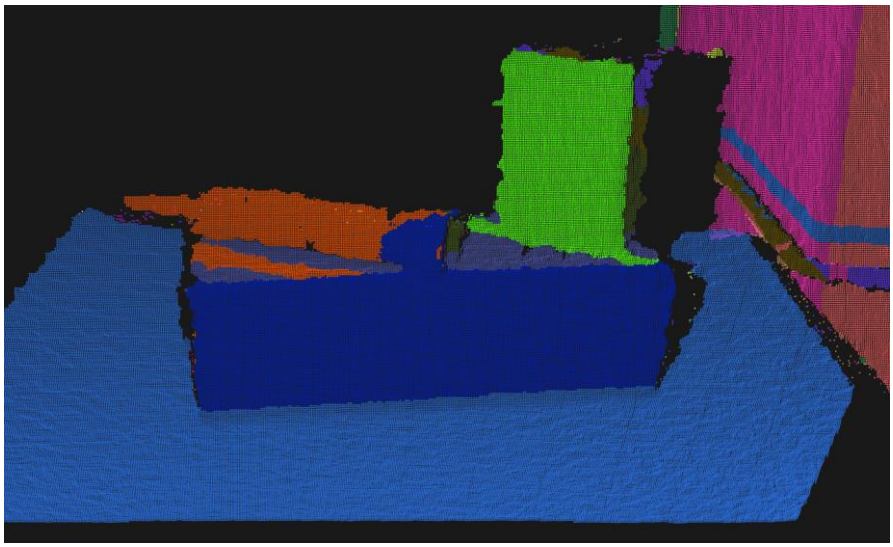
Slika 8.12.c Segmentirana slika

Slike 8.9. i 8.11. su loše segmentirane. Objekti su podijeljeni i raznobojno obojani, te program nije prepoznao niti jednu plohu u cijelosti. Greške se mogu objasniti neodgovarajućim parametrima i lošim osvjetljenjem zbog kojeg program nije u mogućnosti točno raspoznati podatke iz ulazne datoteke.

S druge strane, primjer dobre segmentacije su slike 8.10. i 8.12. na kojima se objekti mogu prepoznati, a greške su zanemarive i mogu se objasniti kuta kamere i sjene na pojedine površine.

Slika 8.13.a Ulazna slika

Slika 8.14.b Označeni bridovi

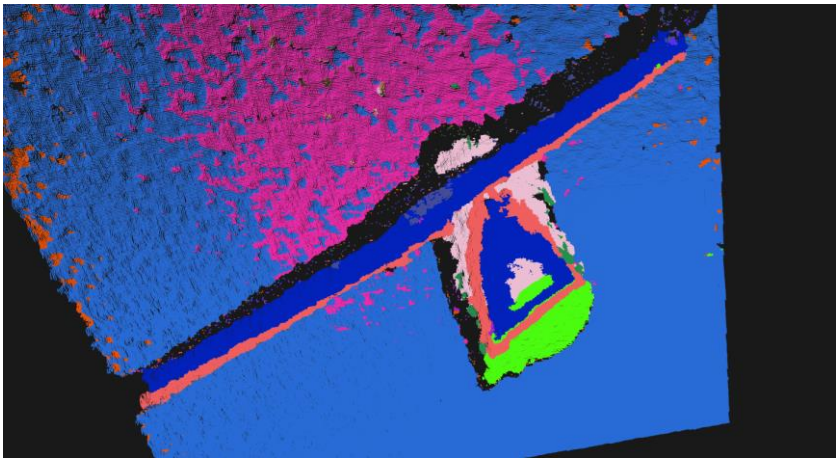


Slika 8.13.c Segmentirana slika

Slika 8.13. je relativno dobro segmentirana i uočavaju se male greške na mjestima gdje se spajaju dva objekta.

Slika 8.14.a Ulazna slika

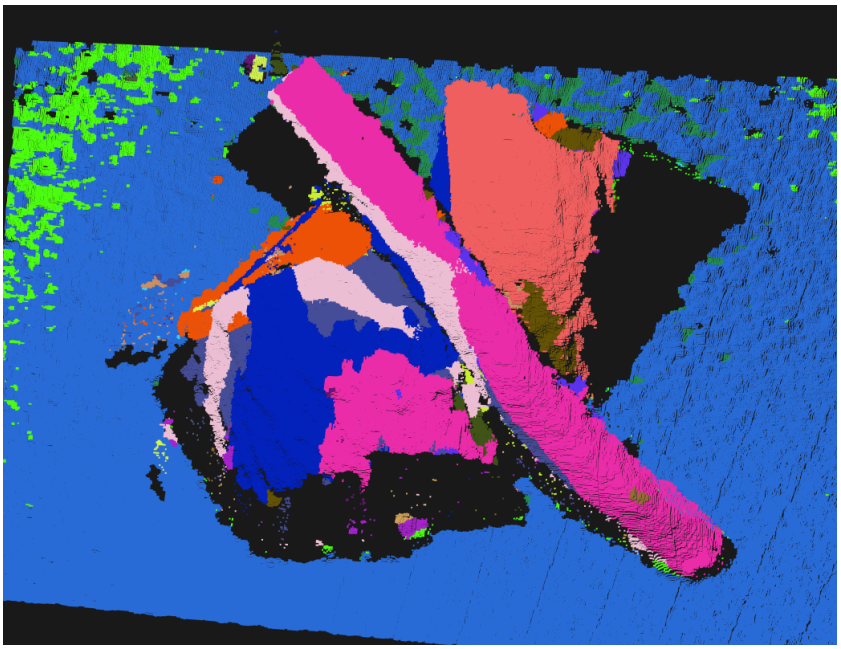
Slika 8.14.b Označeni bridovi



Slika 8.14.c Segmentirana slika

Slika 8.15.a Ulazna slika

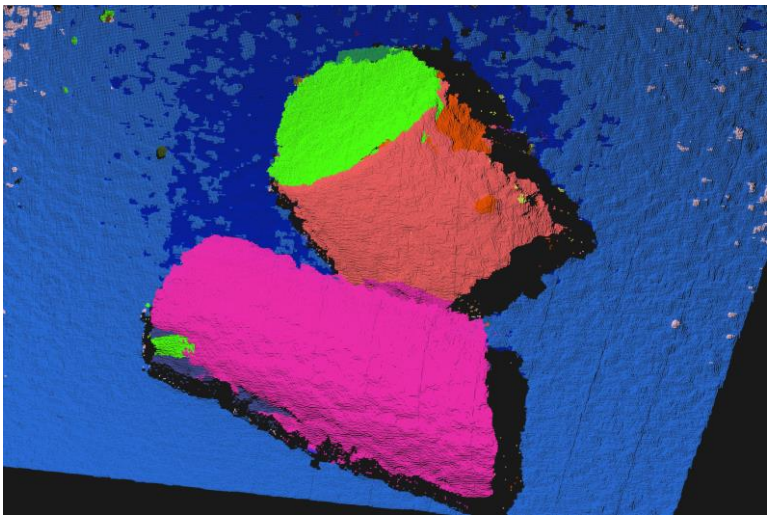
Slika 8.15.b Označeni bridovi



Slika 8.15.c Segmentirana slika

Slika 8.16.a Ulazna slika

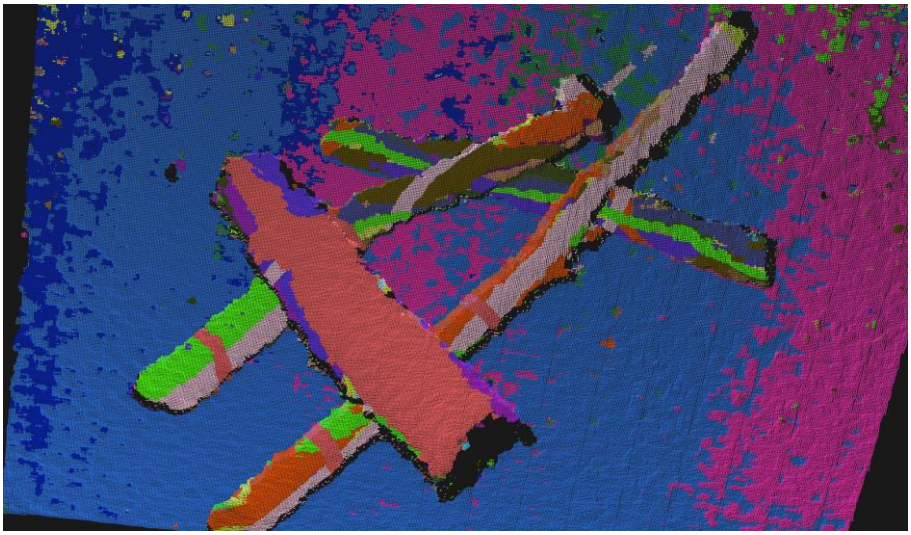
Slika 8.16.b Označeni bridovi



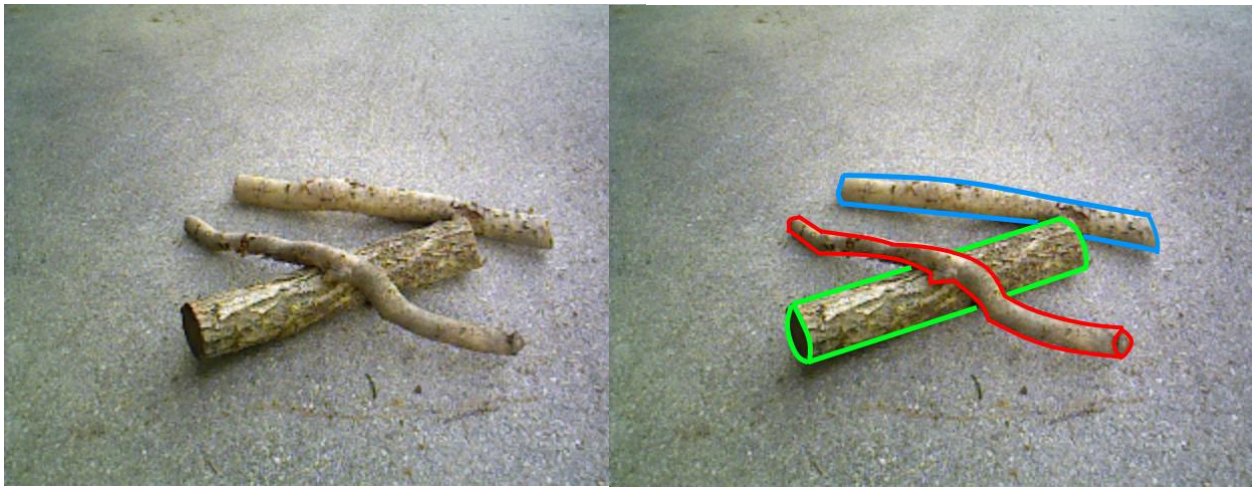
Slika 8.16.c Segmentirana slika

Slika 8.17.a Ulazna slika

Slika 8.17.b Označeni bridovi

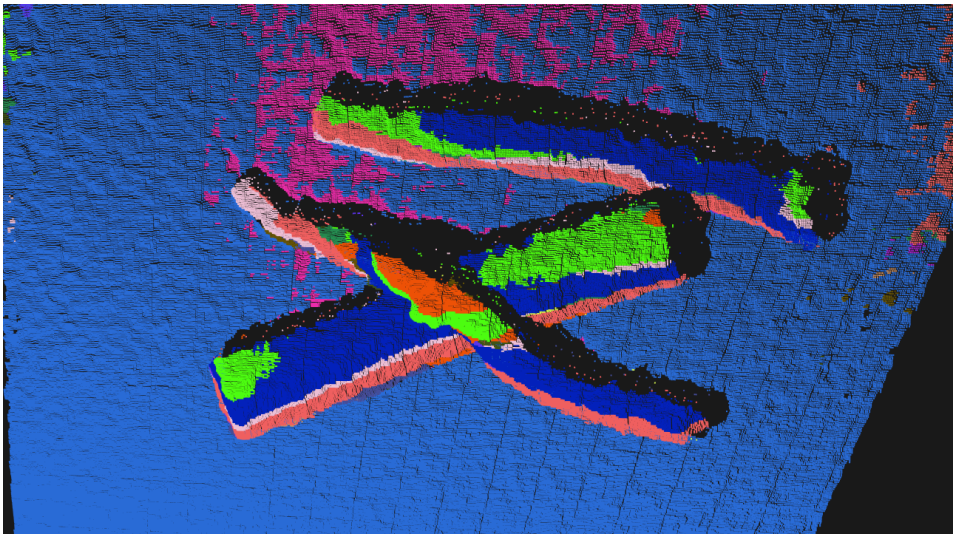


Slika 8.17.c Segmentirana slika



Slika 8.18.a Ulazna slika

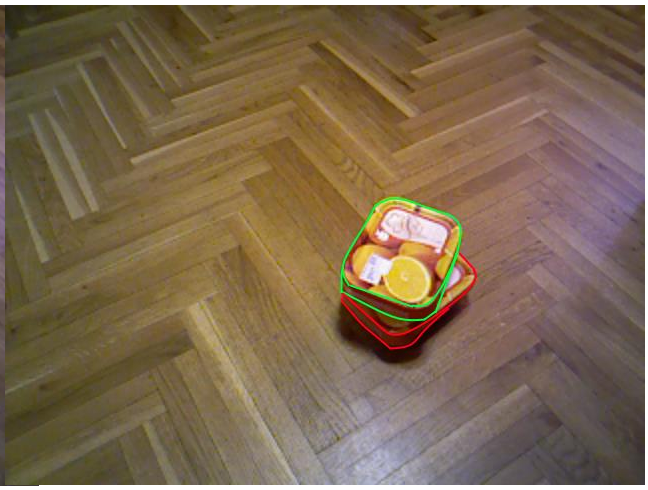
Slika 8.18.b Označeni bridovi



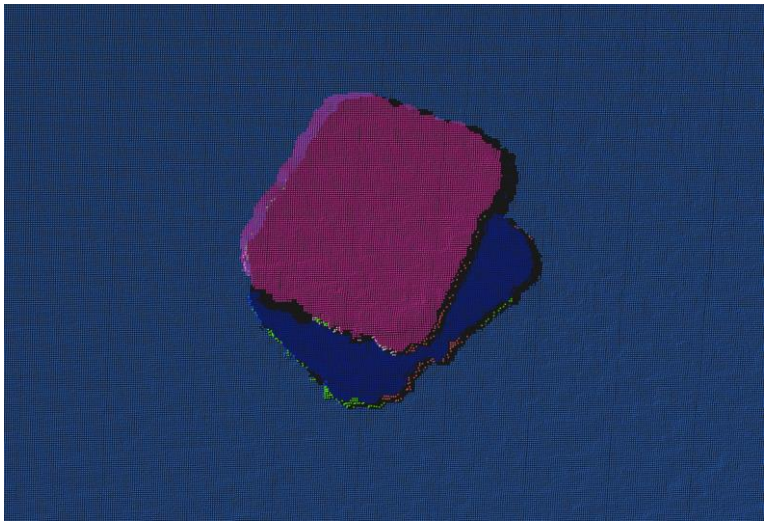
Slika 8.18.c Segmentirana slika

Na slikama 8.14., 8.15., 8.17. i 8.18. se vidi loša segmentacija, svaki model je podijeljen na više objekta i obojan drugom bojom. Može se pretpostaviti da je do toga došlo zbog nepravilnih površina objekata. Objekti su dobro prikazani, ali se zbog boja može zaključiti da nisu prepoznati. Program se nije pokazao učinkovitim za segmentaciju ovakve vrste slika, te je jedino uspješno segmentirana slika 8.16.

Slika 8.19.a Ulazna slika



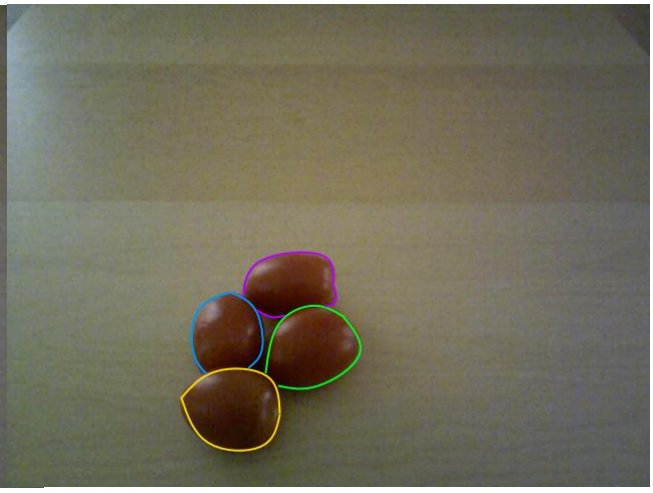
Slika 8.19.b Označeni bridovi



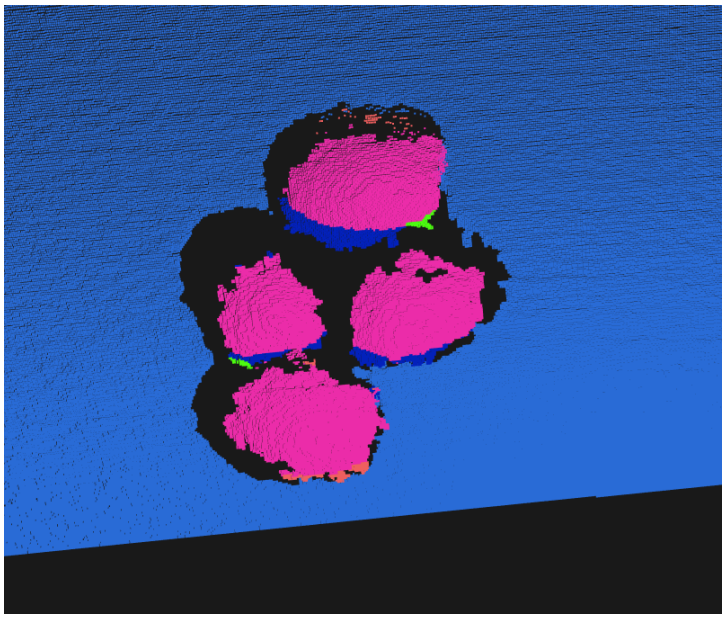
Slika 8.19.c Segmentirana slika

Slika 8.19. uspješno je segmentirana.

Slika 8.20.a Ulazna slika



Slika 8.20.b Označeni bridovi



Slika 8.20.c Segmentirana slika

Na slici 8.20. se može vidjeti da program nije prepoznao objekte, niti ih je dobro segmentirao što možemo objasniti oblikom istih. Naime, program nije predviđen za segmentaciju objekata tih oblika već za ravne i valjkaste površine.

6. ZAKLJUČAK

Nakon obavljanja pokusa, te detaljnog proučavanja dobivenih rezultata, može se zaključiti da napisani program i odabrani parametri daju zadovoljavajuću segmentaciju za neke slike, a za neke ne. Kao primjer dobre segmentacije se mogu navesti valjkasta površina vatrogasnog aparata, te ravne površine cigli i kutija. S druge strane, kao primjer loše segmentacije se mogu uzeti grane drveća, kod kojih program nije prepoznavao oblike površina, te su zbog toga bile raznobojno obojane. Također, ulazne slike sa sfernim objektima nisu dale dobru segmentaciju i prepoznavanje ploha, te bi se to moglo popraviti korištenjem druge metode RANSAC algoritma, no to nije bio cilj ovog završnog rada.

Iz toga se može doći do zaključka da su neki objekti sa ulaznih slika lakši za segmentaciju, a neki teži, što samim tim dovodi do saznanja da bi optimalno bilo koristiti iste parametre za određenu skupinu slika, a za neku drugu skupinu slika potrebno je prilagoditi parametre radi bolje kvalitete segmentacije.

Zbog naprijed navedenog, zaključak je da se segmentacija dubinske slike može odraditi sa istim parametrima na velikom broju primjera, uz zadovoljavajuće rezultate. No, s druge strane određeni primjeri daju nedovoljno jasne rezultate, što znači da ovaj program nije uvijek optimalan za korištenje. Stoga je u tim slučajevima potrebno napraviti promjenu na programu u cilju dobivanja bolje segmentacije.

7. LITERATURA

[1]http://pointclouds.org/documentation/tutorials/random_sample_consensus.php#random-sample-consensus

[2] M. A. Fischler & R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography", Comm. of the ACM. br. 24, Lipanj 1981, dostupno na: http://www.dtic.mil/dtic/tr/fulltext/u2/a460585.pdf [21.9.2017.]

http://pointclouds.org/documentation/tutorials/planar_segmentation.php

http://www.bmva.org/bmvc/2002/papers/50/full_50.pdf

http://pointclouds.org/documentation/tutorials/cylinder_segmentation.php#cylinder-segmentation

http://docs.pointclouds.org/trunk/classpcl_1_1visualization_1_1_p_c_l_visualizer.html

http://pointclouds.org/documentation/tutorials/cloud_viewer.php#cloud-viewer

http://pointclouds.org/media/

https://github.com/PointCloudLibrary/pcl/releases

8. SAŽETAK

Izrađen je program za segmentaciju slike na ravne i valjkaste površine slike pomoću RANSAC metode. Kao ulaz se uzima PCD datoteka koja sadrži parametre slike, a kao rezultat program daje 3D prikaz određene segmentacije slike u obliku oblaka točaka, gdje je svaki segment prikazan drugom bojom. Program je pisan u programskom jeziku C++, te se koristio razvojni alat Microsoft Visual Studio 2013. Također koristile su se funkcije i naredbe iz programske biblioteke PCL (Point Cloud Library). Funkcionalnost programa ispitana je na 20 RGB-D slika.

Ključne riječi: Point Cloud Library, segmentacija, RANSAC, C++

Abstract

Segmentation of the depth image into planar and cylindrical surfaces using the random sampling consensus method

A program was created for image segmentation into planar and cylindrical surfaces using the RANSAC method is developed. The input to the program is a PCD file and as a result, a 3D visualization of the obtained segmentation is provided, where every segment is depicted with a different color. The program is written in C++ programing language and Microsoft Visual Studio 2013 is used as the main development tool. Furthermore, functions and commands from program library PCL (Point Cloud Library) have also been used. The functionality of the program was tested on 20 RGB-D pictures.

Key words: Point Cloud Library, segmentation, RANSAC, C++

9. ŽIVOTOPIS

Duško Findžanović je rođen 7.11.1995. godine u Somboru. Pohađao je Osnovnu školu "dr. Franjo Tuđman" u Belom Manastiru, a srednjoškolsko obrazovanje je stekao u Prvoj srednjoj školi Beli Manastir, smjer Tehničar za računalstvo.

Fakultetsko obrazovanje je započeo 2014. godine i to na Fakultetu Elektrotehnike, Računarstva i Informacijskih Tehnologija u Osijeku, smjer preddiplomski studij računarstva. Trenutno je redovni student treće godine.

10. PRILOZI

Kod programa:

```
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/io/io.h>
#include <pcl/point_types.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/ModelCoefficients.h>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/filters/extract_indices.h>
#include <pcl/filters/passthrough.h>
#include <pcl/features/normal_3d.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>

int user_data, i;
typedef pcl::PointXYZRGB PointT;

int
main(int argc, char** argv)
{
	// Objekti
	pcl::PCDReader reader;
	pcl::PassThrough<PointT> pass;
	pcl::NormalEstimation<PointT, pcl::Normal> ne;
	pcl::SACSegmentationFromNormals<PointT, pcl::Normal> segPlane;
	pcl::SACSegmentationFromNormals<PointT, pcl::Normal> segCylinder;
	pcl::PCDWriter writer;
	pcl::ExtractIndices<PointT> extract;
	pcl::ExtractIndices<pcl::Normal> extract_normals;
	pcl::search::KdTree<PointT>::Ptr tree(new pcl::search::KdTree<PointT>());

	// Skupovi podataka
	pcl::PointCloud<PointT>::Ptr cloud(new pcl::PointCloud<PointT>);
	pcl::PointCloud<PointT>::Ptr cloud_filtered(new pcl::PointCloud<PointT>);
	pcl::PointCloud<pcl::Normal>::Ptr cloud_normals(new pcl::PointCloud<pcl::Normal>);
	pcl::ModelCoefficients::Ptr coefficients_plane(new pcl::ModelCoefficients),
coefficients_cylinder(new pcl::ModelCoefficients);
	pcl::PointIndices::Ptr inliers_plane(new pcl::PointIndices), inliers_cylinder(new
pcl::PointIndices);
	pcl::PointCloud<PointT>::Ptr cloud_remaining(new pcl::PointCloud<PointT>);
	pcl::PointCloud<pcl::Normal>::Ptr cloud_normals_remaining(new
pcl::PointCloud<pcl::Normal>);
	pcl::PointCloud<PointT>::Ptr cloud_plane(new pcl::PointCloud<PointT>());
	pcl::PointCloud<PointT>::Ptr cloud_cylinder(new pcl::PointCloud<PointT>());
	pcl::PointCloud<PointT>::Ptr cloud_all(new pcl::PointCloud<PointT>());


	// Ucitavanje datoteke
	reader.read("C:/Users/Duško/Desktop/PCD Datoteke/Blocks_and_stones_Exp00014/image-
00004.pcd", *cloud);

	//Odredivanje velicine ulaznog clouda
	std::cerr << "PointCloud has: " << cloud->points.size() << " data points." << std::endl;

	// Filtriranje i unklanjenje NaN-ova
	pass.setInputCloud(cloud);
	pass.setFilterFieldName("z");
	pass.setFilterLimits(0, 2000.0);
	pass.filter(*cloud_filtered);
	std::cerr << "PointCloud after filtering has: " << cloud_filtered->points.size() << " data
points." << std::endl;

	float x = cloud_filtered->points.size() / 20; //5% ulazog clouda nakon filtriranja
```

```
        std::cerr << "5% of the filtered PointCloud: " << x << std::endl;

        // Odredjivanje normala
        ne.setSearchMethod(tree);
        ne.setInputCloud(cloud_filtered);
        ne.setKSearch(50);
        ne.compute(*cloud_normals);

        *cloud_remaining = *cloud_filtered;
        *cloud_normals_remaining = *cloud_normals;

        // Parametri segmentacije
        segPlane.setOptimizeCoefficients(true);
        segPlane.setModelType(pcl::SACMODEL_NORMAL_PLANE);
        segPlane.setNormalDistanceWeight(0.2);
        segPlane.setMethodType(pcl::SAC_RANSAC);
        segPlane.setMaxIterations(10000);
        segPlane.setDistanceThreshold(10.0);

        segCylinder.setOptimizeCoefficients(true);
        segCylinder.setModelType(pcl::SACMODEL_CYLINDER);
        segCylinder.setMethodType(pcl::SAC_RANSAC);
        segCylinder.setNormalDistanceWeight(0.07);
        segCylinder.setMaxIterations(15000);
        segCylinder.setDistanceThreshold(10.0);
        segCylinder.setRadiusLimits(10.0, 200.0);

        while (cloud_remaining->points.size() > x)
        {

                segPlane.setInputCloud(cloud_remaining);
                segPlane.setInputNormals(cloud_normals_remaining);
                segPlane.segment(*inliers_plane, *coefficients_plane);

                segCylinder.setInputCloud(cloud_remaining);
                segCylinder.setInputNormals(cloud_normals_remaining);
                segCylinder.segment(*inliers_cylinder, *coefficients_cylinder);


                if (inliers_plane->indices.size() > inliers_cylinder->indices.size())
                {
                        std::cerr << "Extracting the Planar component, " << cloud_remaining-
>points.size() << " data points remaining..." << std::endl;

                        extract.setInputCloud(cloud_remaining);
                        extract.setIndices(inliers_plane);
                        extract.setNegative(false);
                        extract.filter(*cloud_plane);

                        uint8_t r = (BYTE)(rand() % 255);
                        uint8_t g = (BYTE)(rand() % 255);
                        uint8_t b = (BYTE)(rand() % 255);

                        for (i = 0; i < cloud_plane->points.size(); i++)
                        {
                                cloud_plane->points[i].r = r;
                                cloud_plane->points[i].g = g;
                                cloud_plane->points[i].b = b;
                        }
                        *cloud_all = *cloud_all + *cloud_plane;

                        extract.setInputCloud(cloud_remaining);
                        extract.setIndices(inliers_plane);
                        extract.setNegative(true);
                        extract.filter(*cloud_remaining);

                        extract_normals.setInputCloud(cloud_normals_remaining);
                        extract_normals.setIndices(inliers_plane);
                        extract_normals.setNegative(true);
                        extract_normals.filter(*cloud_normals_remaining);
```

```
		}
		else
		{
			std::cerr << "Extracting the Cylinder component, " << cloud_remaining->points.size() << " data points remaining..." << std::endl;
			extract.setInputCloud(cloud_remaining);
			extract.setIndices(inliers_cylinder);
			extract.setNegative(false);
			extract.filter(*cloud_cylinder);

			uint8_t r = (BYTE)(rand() % 255);
			uint8_t g = (BYTE)(rand() % 255);
			uint8_t b = (BYTE)(rand() % 255);

			for (i = 0; i < cloud_cylinder->points.size(); i++)
			{
				cloud_cylinder->points[i].r = r;
				cloud_cylinder->points[i].g = g;
				cloud_cylinder->points[i].b = b;
			}

			*cloud_all = *cloud_all + *cloud_cylinder;

			extract.setInputCloud(cloud_remaining);
			extract.setIndices(inliers_cylinder);
			extract.setNegative(true);
			extract.filter(*cloud_remaining);

			extract_normals.setInputCloud(cloud_normals_remaining);
			extract_normals.setIndices(inliers_cylinder);
			extract_normals.setNegative(true);
			extract_normals.filter(*cloud_normals_remaining);
		}
		if (inliers_plane->indices.size() == 0 && inliers_cylinder->indices.size() == 0)
		{
			break;
		}
	}
	//Vizualizacija
	pcl::visualization::CloudViewer viewer("Cloud Viewer");
	viewer.showCloud(cloud_all);
	while (!viewer.wasStopped())
	{
		user_data++;
	}
	return (0);
}
```