Vizualno prepoznavanje i lokalizacija objekata

Orsag, Luka

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Luka Orsag

Zagreb, 2014.

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Prof.dr.sc. Bojan Jerbić

Student:

Luka Orsag

Zagreb, 2014.

Izjavljujem, da sam ovaj rad radio samostalno uz pomoć mentora te navedene literature koristeći znanja stečena na Fakultetu strojarstva i brodogradnje

Zahvaljujem mentoru na pomoći koju sam primio tijekom izrade ovoga rada.

Veliku zahvalnost dugujem i doc.dr.sc. Tomislavu Stipančiću

na svoj pomoći i idejama pri izradi zadatka.

Također zahvaljujem ocu Goranu i majci Vesni

na iznimnom strpljenju i potpori tijekom studija

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **LUKA ORSAG** Mat. br.: 0035159343

Naslov rada na hrvatskom jeziku: **Prepoznavanje i lokalizacija objekata primjenom PCL programske biblioteke i 3D CAD modela**

Naslov rada na engleskom jeziku: **Objects recognition and localisation using PCL software library and 3D CAD models**

Opis zadatka:

U radu je potrebno razviti postupak i programsku podršku koja omogućava vizualno prepoznavanje te lokalizaciju 3D objekata u radnoj okolini robota koristeći CAD opis uzora. Programsko rješenje potrebno je izvesti koristeći PCL biblioteku (Point Cloud Library) povezanu s MS Kinect stereovizijskim senzorom. Nakon prepoznavanja i lokalizacije objekta, dobivene informacije potrebno je povezati s robotom koji potom treba izvesti predviđene operacije rukovanja u nestrukturiranoj okolini.

Razvijenu primjenu potrebno je provjeriti koristeći opremu dostupnu u Laboratoriju za projektiranje izradbenih i montažnih sustava.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
25. rujna 2014.	27. studenog 2014.	3., 4. i 5. prosinca 2014.

Zadatak zadao: Predsjednik Povjerenstva:

Prof. dr. sc. Bojan Jerbić Prof. dr. sc. Franjo Cajner

Sadržaj

Popis slika I

Popis tabela III

Popis oznaka IV

POPIS DIJAGRAMA V

Sažetak rada VII

1. Uvod 1

2. Opis zadatka 2

2.1. Oblak točaka 2

2.2. Prikaz točke i oblaka točaka u koordintatnom sustavu 3

2.3. Pcl (Point cloud library) [2] 4

2.4. Ms kinect [3] 6

2.4.1. Tehnologija i izvedba senzora 6

2.5. Idejno rješenje 7

2.5.1. Određivanje VOI (Volume Of Interest) 8

2.5.2. Kalibracija 8

2.5.3. Prepoznavanje i lokalizacija 10

3. Koncept susjedstva [7] 11

3.1. Algoritam najbližeg susjeda 12

3.1.1. Algoritam brzog aproksimiranog najbližeg susjeda (ANN) 13

3.1.2. k-d tree algoritam 13

3.1.2.1. Konstrkcija k-d stabla 14

3.1.2.2. Nearest Neighbor algoritam modificran k-d stablom 15

3.1.2.3. Nasumična k-d stabla 18

4. Koncept susjedstva (implementacija pomoću PCL) 19

4.1. Kratki opis klase KdTreeFLANN 19

4.2. Pretraga u slučaju *k* najbližih susjeda 20

4.3. Pretraga u slučaju odabranog radijusa pretrage 21
5. Estimacija normala i zakrivljenosti površine [7] 23
6. Estimacija normala i zakrivljenosti površine (implementacija pomoću PCL) 26
6.1. Kratki opis klase *NormalEstimation* 26
6.2. Rezultati estimacije normala pomoću PCL 27
7. Segmentacija ravnine [7] 29
8. Segmentacija ravnine (implementacija pomoću PCL) [4] 31
8.1. Kratki opis klase SACSegmentation 32
8.2. Rezultati segmentacije ravnine 35
8.3. Funkcija u C++ programskom jeziku 36
9. Grupiranje objekata (Clustering) [7] 37
10. Grupiranje objekata (implementacija pomoću PCL) [6] 39
10.1. Kratki opis klase *EuclideanClusterExtraction* 39
10.2. Rezultati algoritma grupiranja 40
10.3. Funkcija u C++ programskom jeziku 41
11. Prostorna transformacija (implementacija u C++ programskom jeziku) 43
11.1. Implementacija algoritma za računanje transformacije u C++ programskom jeziku 43
11.2. Implementacija određivanja VOI u C++ programskom jeziku 43
11.2.1. Konačni rezultati nakon provedbe svih navedenih algoritama 44
12. Zaključak 46
13. Literatura 47
14. Prilog 48

POPIS SLIKA

Slika 1 Robot FANUC M3iA/6a
Slika 2 Točka u XYZ koordinatnom sustavu
Slika 3 Oblak točaka P_{xyz}
Slika 4 Oblak točaka $P_{realna\ situacija}$
Slika 5 Point Cloud Library logo
Slika 6 Struktura PCL-a
Slika 7 MS Kinect
Slika 8 MS Kinect opis komponenata
Slika 9 Algoritam najbližeg susjeda (k=3 i k=5)
Slika 10 Izgled k-d stabla za zadani set točaka
Slika 11 Particioniranje prostora (lijevo) i prikaz k-d stabla (desno) za zadani set točaka: P=[(2,3), (5,4), (9,6), (4,7), (8,1)]
Slika 12 Euclid-ov prostor sa zadanim točkama i k-d stablom
Slika 13 Prikaz prostora sa unesenom točkom za klasifikaciju
Slika 14 Algoritam započinje pretragu od korijenskog čvora (desno); Sfera presjeca sva područja, niti jedno se u ovom trenutku ne može izostaviti (lijevo)
Slika 15 Odabrana točka B ima manju udaljenost od A i sprema se kao "trenutno najbolja"
Slika 16 Provjerava se da li su D i E bliži zadanoj točki od B; U ovom slučaju nisu pa se izbacuju iz pretrage
Slika 17 Sfera ne presjeca ravninu (lijevo) i cijeli desni dio stabla može se ukloniti iz pretrage (desno)
Slika 18 Rezultat pretrage sa K = 10
Slika 19 Rezultat pretrage sa K = 10 (povećani prikaz)
Slika 20 Rezultati pretrage sa R = 100
Slika 21 Rezultati pretrage sa R = 100 (povećani prikaz)
Slika 22 Loša orijentacija normala
Slika 23 Kvalitetna orijentacija normala
Slika 24 Eksperimentalni postav
Slika 25 Rezultati estimacije normal pomoću klase *NormalEstimation*
Slika 26 Rezultati estimacije normal (povećani prikaz)

Slika 27 Primjer segmentacije ravnine
Slika 28 Eksperimentalni postav
Slika 29 Pokretna traka u laboratoriju fakulteta
Slika 30 Oblak točaka prikupljenog sa eksperimentalnog postava
Slika 31 Rezultati segmentacije ravnine
Slika 32 Podešavanje ravnine u laboratoriju fakulteta
Slika 33 Površina sa objektima
Slika 34 Primjer grupiranih oblaka točaka
Slika 35 Ulazni oblak (segmentirana površina označena je crvenom bojom)
Slika 36 Rezultat algoritma grupiranja (plavom bojom označena pronađena površina; crvenom bojom označene su ostale grupe koje se odbacuju)
Slika 37 Prikaz rezultata (segmentacija, VOI, grupiranje oblaka točaka u objekte i lokalizacija)
Slika 38 Prikaz rezultata provedenih algoritama (povećani prikaz)

POPIS TABELA

Tabela 1 Member funkcije klase *kdtree*

Tabela 2 Member funkcije klase *NormalEstimation*

Tabela 3 Member funkcije klase *SACSegmentation*

Tabela 4 Member funkcije *EuclideanClusterExtraction* klase

POPIS OZNAKA

P_x	*-oblak točaka*
p_i^k	*-točka u oblaku točaka*
d_i	*-udaljenost od točke*
$\vec{n}$	*-normala*
$\boldsymbol{x}$	*-centroida (težište)*
C	*-matrica kovarijance*
$g(\sigma)$	*-Gaussova funkcija*
D_{xx}	*-aproksimacja Gaussove funkcije*
w	*-Forbenijeva norma*
σ	*-oznaka mjerila*
v	*-vektor deskriptora*
P	*-ulazni vektor sa početnim podacima*
$\dot{X}$	*-ulazi vektor sa podacima za klasifikaciju*
c_j	*-oznaka za klasu*
d	*-Euklidova udaljenost*
D_x^K	*-vektor sa najbližim susjedima*
x	*-trodimenzionalna točka*
P^2	*-oznaka za ravninu*
σ_p	*-zakrivljenost površine u točki p*
λ_j	*-svojstvene vrijednosti*
q	*-4x4 homogena matrica transformacije*
ϵ	*-vjerojatnost da je međuovisnost uzoraka gruba pogreška*

POPIS DIJAGRAMA

Dijagram 1 Idejno rješenje
Dijagram 2 Idejno rješenje (Određivanje VOI)
Dijagram 3 Idejno rješenje (Kalibracija)
Dijagram 4 Idejno rješenje (Prepoznavanje i lokalizacija)

SAŽETAK RADA

Ovaj rad bavi se problematikom prepoznavanja i lokalizacije objekata pomoću algoritama vizijskih sustava. Koristi se MS Kinect 3D kamera, PCL (Point Cloud Library) i fanuc M3iA/6a industrijski robot. U sklopu rada obrađeni su algoritmi za obradu 3D oblaka točaka u koje spadaju segmentacija (ravnina i cilindar), grupiranje (clustering) i određivanje značajki. Proučena je i implementirana TCP/IP komunikacija (Sockets) te Fanuc Karel programski jezik za upravljanje robotom.

Ključne riječi: Vizijski sustavi, vizualno prepoznavanje i lokalizacija, PCL, k-NN algoritam, k-d tree algoritam, Kinect

1. UVOD

U okviru diplomskog rada potrebno je osmisliti sustav prepoznavanja i lokalizacije objekata. Biti će prikazane metode za akviziciju podataka u obliku oblaka točaka te njihova obrada i sakupljanje smislenih podataka. Ispitati će se i implementirati algoritmi za filtriranje podataka, određivanje smislenih značajki kao skup točaka koje najbolje opisuju površinu koja se promatra i njihova interpretacija računalu (numeričkim vrijednostima) te interpretacija korisniku preko vizualizacije. Implementacija algoritama vršiti će se pomoću PCL-a (Point Cloud Library) koji predstavlja skup algoritmama za obradu podataka prikupljenih pomoću vizijskih sustava posebnih akvizicijskih sposobnosti. Prepoznavanje objektata potrebno je provesti ugađanjem CAD modela u prikupljenom stohastičkom okruženju, lokalizacija će se vršiti određivanjem translacije i rotacije objekta u 3D koordinatnom sustavu u odnosu na zajednički koordinatni sustav. Za ispitivanje kvalitete algoritma i lokalizacije aplikacija će se povezati sa industrijskim robotom preko TCP/IP veze. Cilj ovoga rada je implementirati kvalitetan algoritam dovođenja robota u željenu, unaprijed nepoznatu, poziciju te akvizicija i paletizacija objekata na industrijskoj traci.

2. OPIS ZADATKA

U sklopu rada potrebno je osmisliti postupak prepoznavanja objekata u industrijskom okruženju pomoću kamere posebnih akvizicijskih sposobnosti (u radu će se koristiti MS Kinect). Također je potrebno upotrijebiti posebnu knjižnicu algoritama za obradu oblaka točaka koji će za razliku od 2D vizijskih sustava, gdje glavni oblik informacije predstavlja piksel, predstavljati 3D prostorne informacije. Aplikaciju je kasnije potrebno povezati sa industrijskim robotom FANUC M-3ia 6A putem TCP/IP Socket komunikacijskog protokola.

Postupci obrade oblaka točaka biti će detaljno obrađeni što uključuje terojisku i praktičnu podlogu.

Točnost sustava ispitati će se na industrijskom robotu u laboratoriju fakulteta.

Slika 1 Robot FANUC M3iA/6a

2.1. Oblak točaka

Pod oblakom točaka smatra se set točaka u istom koordinatnom sustavu. U trodimenzionalnom sustavu te točke su opisane X, Y i Z koordinatama i obično predstavljaju nekakvu površinu.

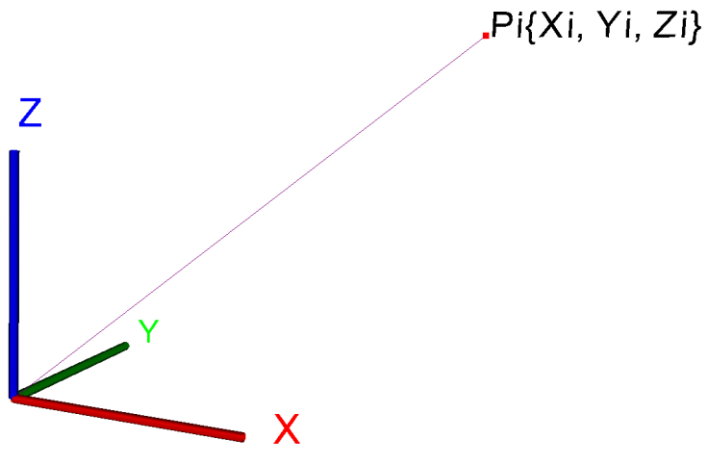
Oblaci točaka se u praksi prikupljaju pomoću 3D skenera ili posebnih senzora koji imaju mogućnost određivanja dubine (u ovom radu koristi se MS Kinect). Oblaci točaka

prikupljeni pomoću takvih uređaja koriste se za 3D prikaz objekata iz pravog svijeta, na računalu. Koriste se također i za stvaranje 3D CAD modela, rekonstrukciju itd.

Isto kao za 2D sliku i za oblake točaka postoje algoritmi koji se koriste za filtriranje i obradu istih te na kraju i analizu scene.

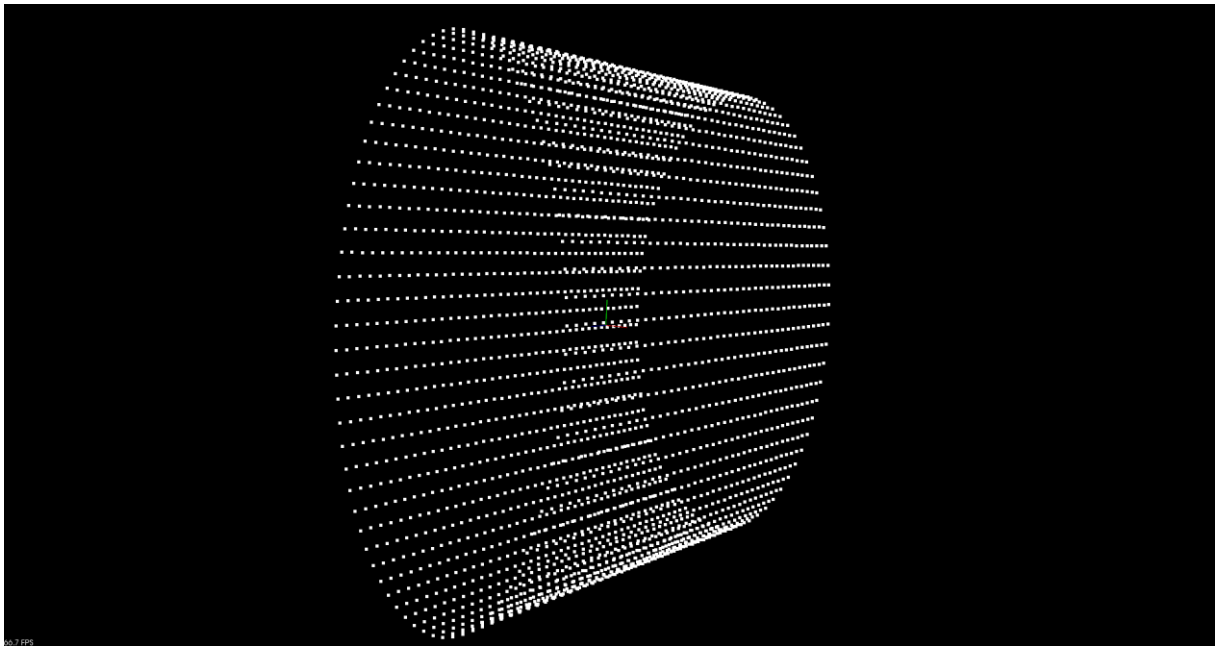
2.2. Prikaz točke i oblaka točaka u koordintatnom sustavu

Kao što je već napomenuto, točka se u svom koordinatnom sustavu opisuje X, Y i Z koordinatama. Za takvu se točku kaže da se nalazi u Euklidovom 3D prostoru.



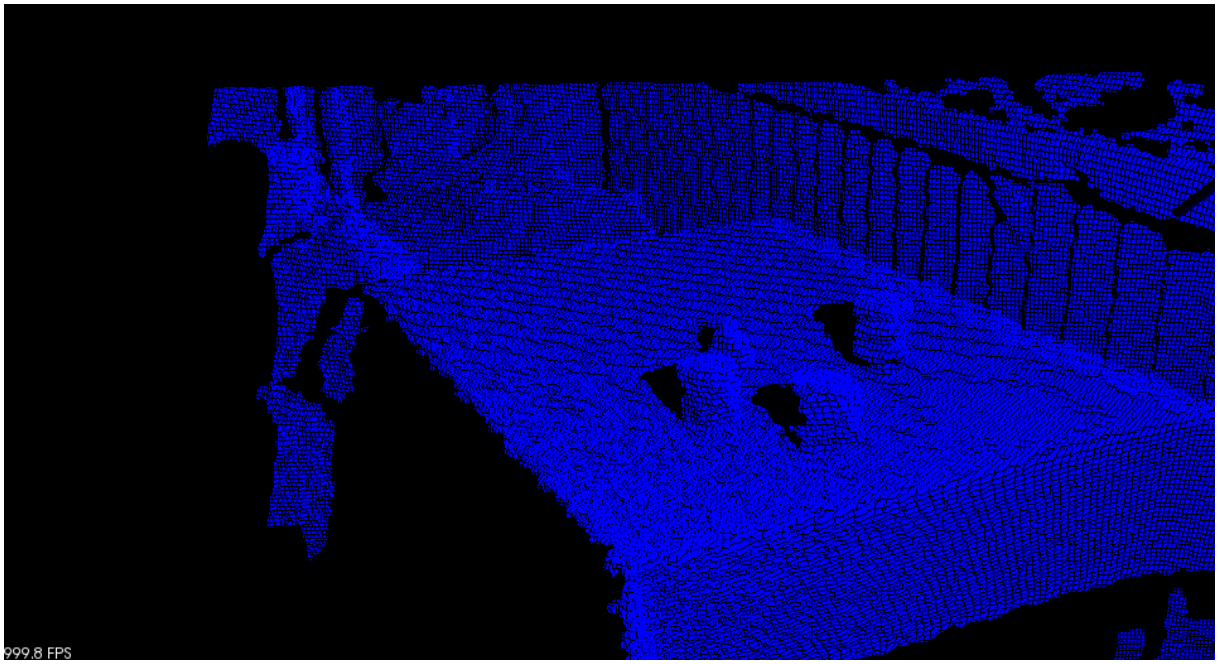
Slika 2 Točka u XYZ koordinatnom sustavu

Slika X prikazuje točku u svom koordinatnom sustavu te iz nje nije teško zaključiti kako će izgledati i oblak točaka. Oblak točaka na kraju će biti vektor P_x koji se sastoji od točaka p_i, a svaka točka će imati svoje koordinate X, Y i Z.



Slika 3 Oblak točaka P_{xyz}

Iz slike je lako zaključiti da je situacije iz stvarnog svijeta moguće prikazivati pomoću oblaka točaka.



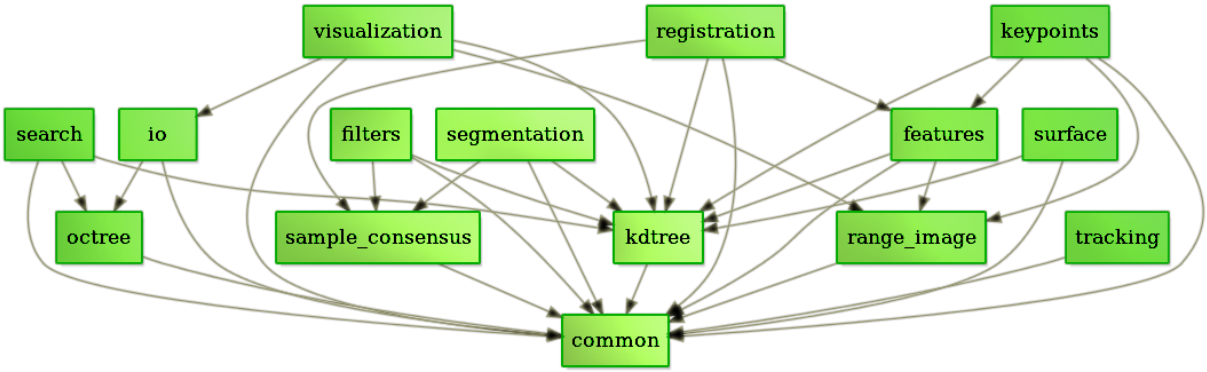
Slika 4 Oblak točaka $P_{realna\ situacija}$

2.3. Pcl (Point cloud library) [2]

Point Cloud Library predstavlja opsežnu knjižnicu algoritama koja se koristi u svrhe obrade i analize oblaka točaka prikupljenih pomoću raznih uređaja (MS Kinect i slično). Sastoji se od raznih modula koji omogućuju filtriranje oblaka, estimaciju značajki,

rekonstrukciju površine itd. Pisana je u C++ programskom jeziku te se tako i implementira. Knjižnica je zamišljena jednako kao i OpenCV knjižnica algoritama za obradu 2D slike.

Slika 5 Point Cloud Library logo



Slika 6 Struktura PCL-a

Da bi se uporaba pojednostavnila knjižnica je podijeljena u dvanaest modula:

- **pcl_filters**- modul koji sadrži algoritme za uklanjanje šumova iz oblaka točaka
- **pcl_features**- modul koji sadrži algoritme i mehanizme za određivanje 3D značajki u oblacima točaka
- **pcl_keypoints**- modul koji sadrži osnovne algoritme i mehanizme za estimaciju ključnih točaka (točaka od interesa) i opisivača
- **pcl_registration**- modul koji sadrži algoritme za registraciju oblaka točaka
- **pcl_kdtree**- modul koji sadrži funkcionalnost i implementaciju *kd* stabla
- **pcl_octree**- modul koji sadrži metode za stvaranje hijerarhijskog stabla
- **pcl_segmentation**- modul koji sadrži algoritme za segmentaciju oblaka u raspoznatljive elemente

- **pcl_sample_consensuz**- modul koji sadrži metode poput Sample Consensus i modele poput ravnina i cilindara
- **pcl_surface**- modul koji sadrži metode za rekonstrukciju originalne površine pomoću oblaka točaka
- **pcl_recognition**- modul koji sadrži algoritme za prepoznavanje objektata
- **pcl_io**- modul koji sadrži klase i funkcije za čitanje i pisanje oblaka točaka sa datoteka ili perifernih uređaja
- **pcl_visualization**- modul koji sadrži metode za brzu vizualizaciju oblaka točaka

2.4. Ms kinect [3]

Samo ime Kinect predstavlja liniju senzora za prepoznavanje ljudskih pokreta proizvedena u tvrtci Microsoft. Prva generacija senzora proizvedena je 2010. Godine za Xbox 360 igraču konzolu, dok je druga generacija proizvedena i za Windows operativni sustav 2012. Godine. Tvrtka Microsoft izdala je i SDK (Software Development Kit) koji omogućuje razvoj aplikacija vezanih za Kinect. Aplikacije je moguće pisati u C++, C# i Visual Basic programskim jezicima na bilo kojem sučelju (IDE).

Slika 7 MS Kinect

2.4.1. Tehnologija i izvedba senzora

Kinect je, u suštini, duguljasto horizontalno kučište spojeno na malo motorizirano postolje dizajnirano za pozicioniranje ispod ili iznad ekrana. Sastoji se od RGB kamere, senzora dubine i polja mikrofona. U sklopu rada biti će opisan samo senzor dubine (depth sensor) jer se ne koriste oblaci točaka sa značajkom boje.

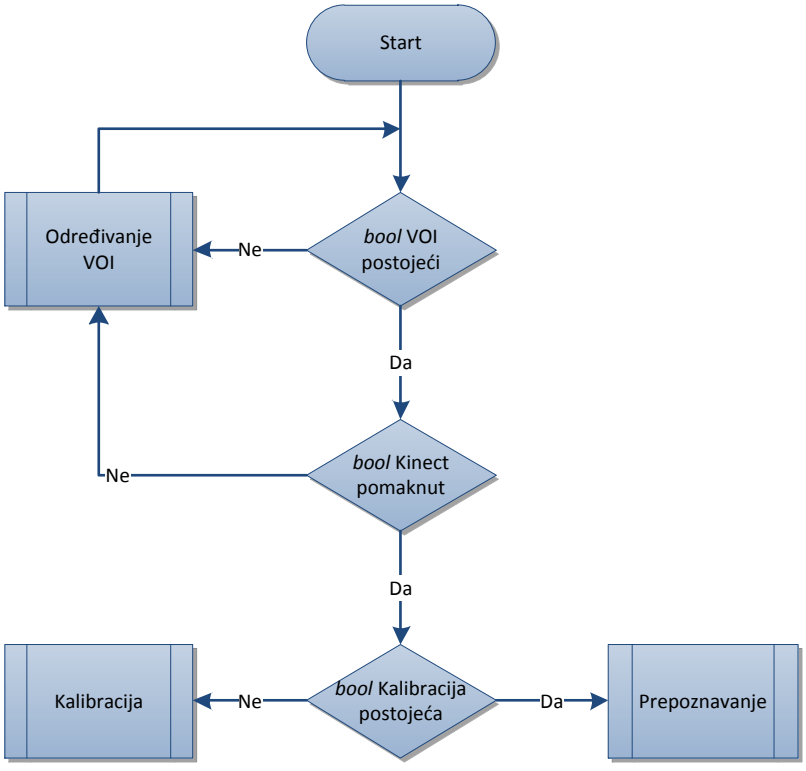
Senzor dubine sastoji se od infracrvenog laserskog topa kombiniranog sa monokromnim CMOS senzorom za akviziciju signala. Algoritam za određivanje dubine je TOF (Time-Of-Flight algoritam) koji mjeri vrijeme potrebno da signal sa lasera dođe do CMOS senzora.

Slika 8 MS Kinect opis komponenata

2.5. Idejno rješenje

Idejno rješenje problema biti će prikazano dijagramom toka te će se razlagati u zasebne sustave. Svaki podsustav opisati će se kroz teoriju i implementaciju pomoću PCL knjižnice algoritama.

Sustav je podjeljen u tri glavna segmenta koji opisuju aplikaciju.



Dijagram 1 Idejno rješenje

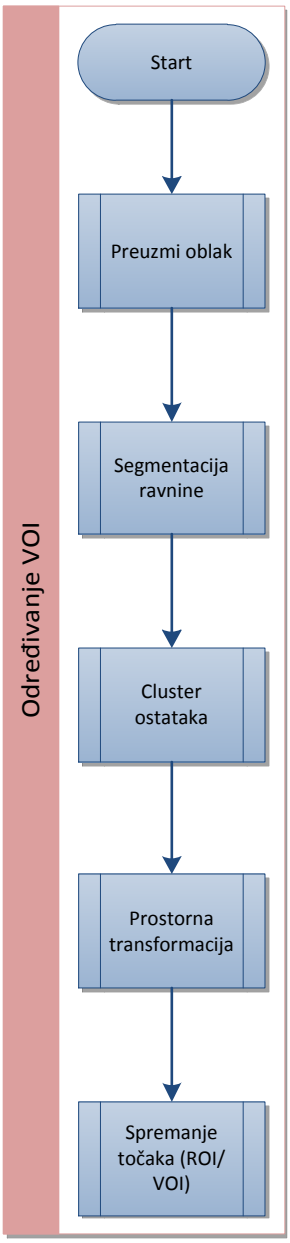
Na dijagramu su prikazana tri glavna segmenta koji će se u obraditi u slijedećim poglavljima:

- Određivanje VOI (Volume Of Interest)

- Kalibracija
- Prepoznavanje

2.5.1. Određivanje VOI (Volume Of Interest)

Određivanje područja od interesa odvija se u pet glavnih podfunkcija kako je prikazano na dijagramu toka:

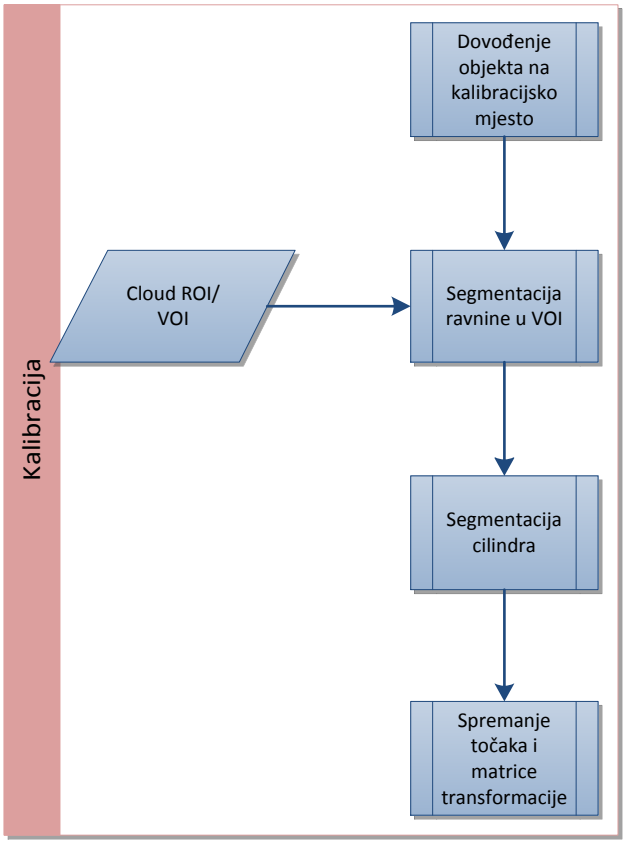


Dijagram 2 Idejno rješenje (Određivanje VOI)

2.5.2. Kalibracija

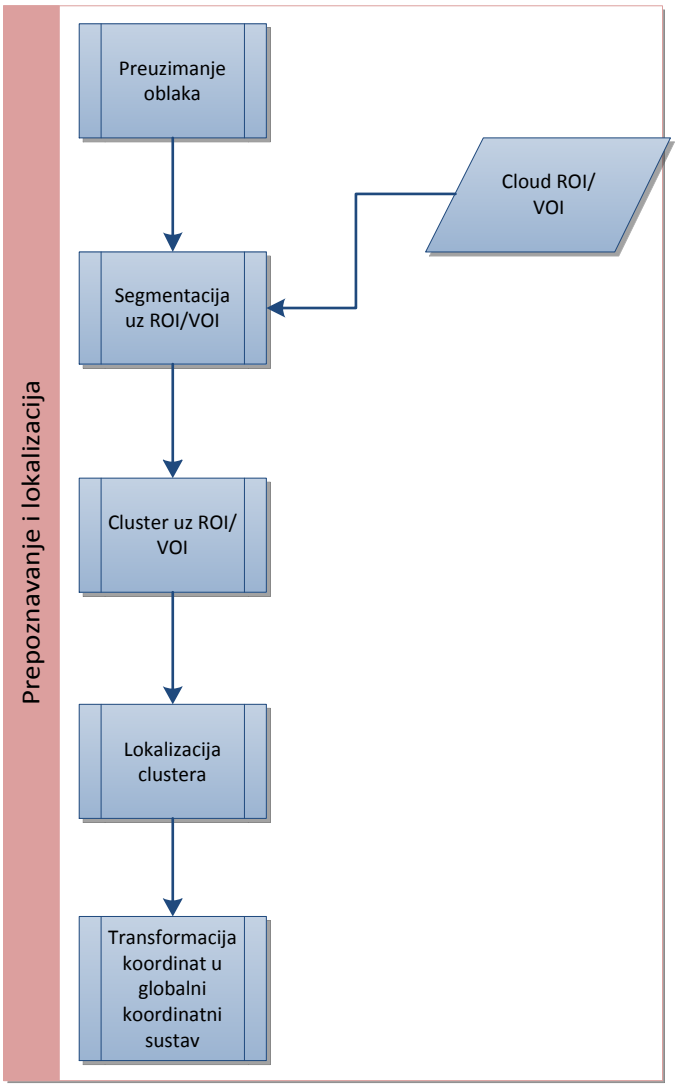
Kalibracija se odvija uz pomoć robota koji u trenutku kada je potrebno donosi kalibracijski objekt na mjesto kalibracije. Poziciija objekta i mjesta kalibracije je

unaprijed određena. Robot je sa računalom povezan preko TCP/IP veze, no radi jednostavnosti se komunikacija neće opisivati u dijagramu toka.



Dijagram 3 Idejno rješenje (Kalibracija)

2.5.3. Prepoznavanje i lokalizacija



Dijagram 4 Idejno rješenje (Prepoznavanje i lokalizacija)

3. KONCEPT SUSJEDSTVA [7]

S obzirom da se, u suštini, ovaj rad bavi obradom točaka u istom koordinatnom sustavu pomoću računala, potrebno je odrediti značajke koje će biti smislene računalu za daljnju obradu i proračun.

U ovom slučaju radi se o velikoj količini podataka, jako usko zbijenih u Kartezijskom sustavu. S obzirom na činjenicu da se svaka točka obrađuje posebno, nema smisla uzimati u obzir cijeli oblak P prilikom svakog prolaska kroz petlju. Zato se u cijelu priču uvodi koncept susjedstva točaka.

Određivanje susjedstva ili provođenje algoritma najbližeg susjeda q u oblaku P oko točke p, generalno ne predstavlja problem i ne zahtjeva kompliciran matematički aparat, no kao što je već napomenuto, algoritam je potrebno provesti za svaku točku u oblaku. Zbog nepoznatog broja ulaznih podataka algoritam najbližeg susjeda spada u vremenski nedeterminističke algoritme. Vrijeme izvršavanja algoritma ovisiti će o broju točaka i o odabiru veličine susjedstva od strane korisnika. S ciljem da se algoritam izvodi pomoću računala, osmišljeni su algoritmi aproksimacije najbližeg susjeda koji računaju na prihvatljivu pogrešku $e \geq 0$ što znači da za svaku pronađenu točku u susjedstvu p_i^k, omjer udaljenosti između nje i pravog najbližeg susjeda q^k je najviše $1+e$, a jednadžba glasi:

$$\left\| p_i^k - p_q \right\|_x \leq (1 - \epsilon) \left\| q^k - p_q \right\|_x \qquad (1.1)$$

Svrha prihvačanja takve pogreške je ubrzavanje algoritma koji se izvodi na računalu ponavljajući se u jako kratkom vremenskom rasponu.

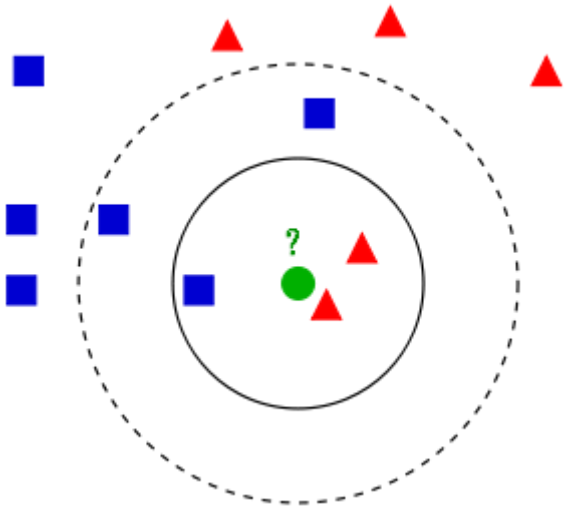
Generalno, postoji dva načina kako bi se algoritam ubrzao, tj. Smanjio broj točaka za pretragu:

- Određivanje *k* najbližih susjeda u oblaku točaka
- Određivanje najbližeg susjeda unutar radijusa *r*

Uz to koriste se i *kd*-stabla koja usmjeravaju algoritam u pravom smjeru prije nego se uopće počne izvoditi.

3.1. Algoritam najbližeg susjeda

U grubo, algoritam spada u najjednostavniji oblik kasifikacije objekata, baziran na naučenim primjerima u prostoru značajki. Spada u najjednostavniji algoritam jer je objekt klasificiran večinskim udjelima najbližih susjeda, gdje *k* označava broj najbližih susjeda koji ulaze u proces klasifikacije.



Slika 9 Algoritam najbližeg susjeda (k=3 i k=5)

Na slici se jasno vidi princip na kojemu funkcionira algoritam. Prostor značajki sastoji se od crvenih trokuta i plavih kvadrata. Zeleni krug označava značajku koja još nije klasificirana. Također se vidi i utjecaj broja *k*. Ukoliko je *k* jedna broju 3, značajka će biti klasificirana kao trokut jer u području od tri najbliža susjeda oko nepoznate značajke ima najviše crvenih trokuta, ali ako je *k* uvećan na 5, značajka će biti klasificirana kao plavi kvadrat. Takav se princip odlučivanja naziva *majority vote*. Broj *k* je obično mali broj, ali uvijek neparan, kako bi se izbjegao jednak broj najbližih susjeda koji posjeduju ista svojstva u prostoru značajki. Iz slike je jasno da je ovakav algoritam jednostavan za implementaciju u aplikacije vezane uz strojni vid.

Pseudokod za osnovni *k*-NN algoritam

***Ulaz**: $P = \{(x_1, c_1), \dots, (x_N, c_k)\}; \dot{X} = (\dot{x}_1, \dots, \dot{x}_n);$*

***for** svaka klasificirana instanca (x_i, c_j), i=(1, ... , N); j=(1, ... , k);*

Izračunaj udaljenost $d(x_i, \dot{X})$;

Poredaj $d(x_i, \dot{X})$ od najniže do najveće;

Odaberi K najbližih instanci $\dot{X}$*; Pridjeli ih u* D_x^K*;*

Pridjeli $\dot{X}$ *klasi koja se najviše pojavljuje u* D_x^K*;*

No, postoje ograničenja na ovakav algoritam, tj. preveliki zahtjevi. Naime, SURF deskriptor poprima oblik vektora koji u sebi sadrži brojčane vrijednosti koje na kraju čine polje podataka. U poglavlju o implementaciji SURF algoritma prikazano je na dijagramu kako je iz slike moguće pronaći do 270 točaka od kojih svaka posjeduje svoj deskriptor. Iz toga se zaključuje da na slikama postoji *n* klasa koje potrebno uspoređivati iz slike u sliku, što drastično smanjuje kvalitetu aplikacije i povećava vrijeme računanja parova točaka, tj. povezivanja točaka od interesa. Očito treba pronaći kvalitetnije rješenje koje će osigurati kvalitetno povezivanje točaka, brzo i uz malu pogrešku.

Predloženo je dosta nadogradnji na osnovni *k*-NN algoritam, no u sklopu rada će se raspravljati o metodi zvanoj *algoritam brzog aproksimiranog najbližeg susjeda.*

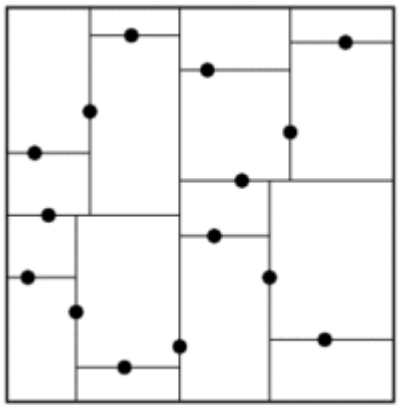
3.1.1. Algoritam brzog aproksimiranog najbližeg susjeda (ANN)

U računalnom smislu, najzahtjevniji dio aplikacija koje uključuju strojni vid i vizualno prepoznavanje je algoritam najbližeg susjeda u prostoru brojnih dimenzija. Za takav prostor ne postoji algoritam najbližeg susjeda koji je efektivniji od jednostavne linearne potrage. No, kako je takva potraga prezahtjevna za aplikacije strojnog vida, nastaje potreba za algoritmom koji približno računa najbližeg susjeda, koji ponekad vrača pogrešne rezultate. Takav je algoritam puno brži i rezultati koje vraća su blizu optimalnim, bez obzira na grešku koju radi.

Napisano je puno algoritama koji poboljšavaju djelovanje osnovnog *k*-NN algoritma, ali za potrebe ovoga rada bilo je dovoljno proučiti samo *randimzed k-d tree* algoritam.

3.1.2. k-d tree algoritam

U računalnim znanostima, *k-d tree* algoritam podrazumijeva strukturu za podjelu prostora za organizaciju točaka u *k*-dimenzionalnom prostoru. To je binarno "stablo" u kojemu je svaki čvor *k*-dimenzionalna točka.



Slika 10 Izgled k-d stabla za zadani set točaka

Očito je da će *Nearest Neighbor* algoritam funkcionirati puno bolje ukoliko se napravi ovakvo particioniranje prostora. No, prije nego što se opiše uporaba *k-d stabla* na *Nearest Neighbor* algoritmu, potrebno je opisati samu konstrukciju algoritma i particioniranje prostora.

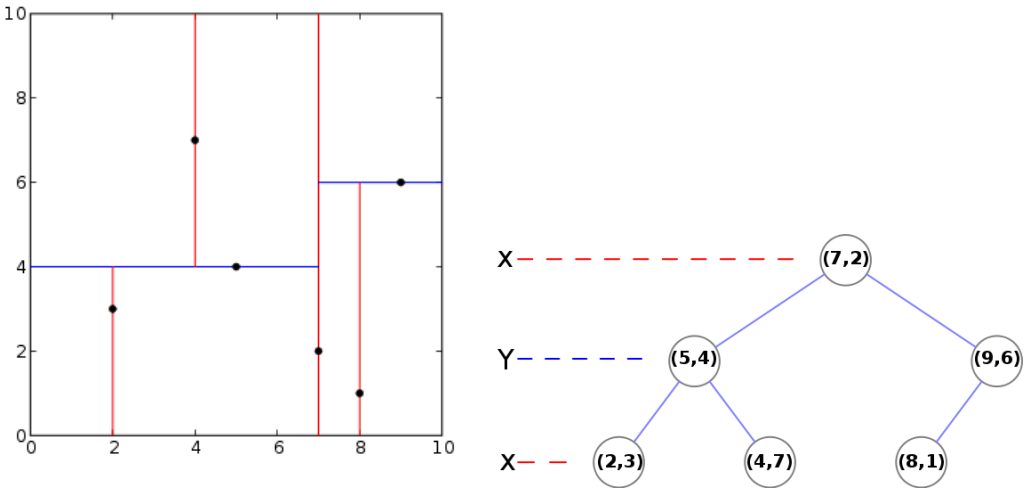
3.1.2.1. Konstrkcija k-d stabla

Na slici je opisan izgled stabla za zadani set točaka. Vertikalnim i horizontalnim linijama je opisano particioniranje prostora po ordinati i apscisi. Vidljivo je da se prvo odabire os po kojoj će se prostor dijeliti, a zatim se odabire središnja točka na toj osi (*median point*). To je prvi čvor u stablu oko kojega se provodi ostatak particioniranja. S obzirom na to da postoji mnogo načina za podjelu ravnina, postoji i mnogo načina za konstrukciju *k-d stabla.* Odabire se kanonska metoda konstrukcije stabla:

- Pregledavanjem stabla u smjeru, prema dolje, promatramo osi korištene za odabir diobenih ravnina
- Točke se pridjeljuju odabirom srednjih točaka na osi

Ovom se metodom dolazi do balansiranog stabla u kojem je svaki čvor- list jednake udaljenosti od korijenskog čvora.

Treba napomenuti, da nije potrebno odabrati središnju točku, ali u tom slučaju stablo više nije balansirano.

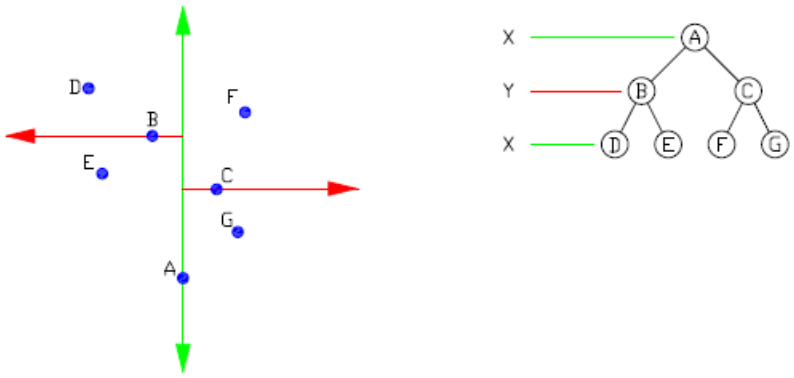


Slika 11 Particioniranje prostora (lijevo) i prikaz k-d stabla (desno) za zadani set točaka: P=[(2,3), (5,4), (9,6), (4,7), (8,1)]

3.1.2.2. Nearest Neighbor algoritam modificran k-d stablom

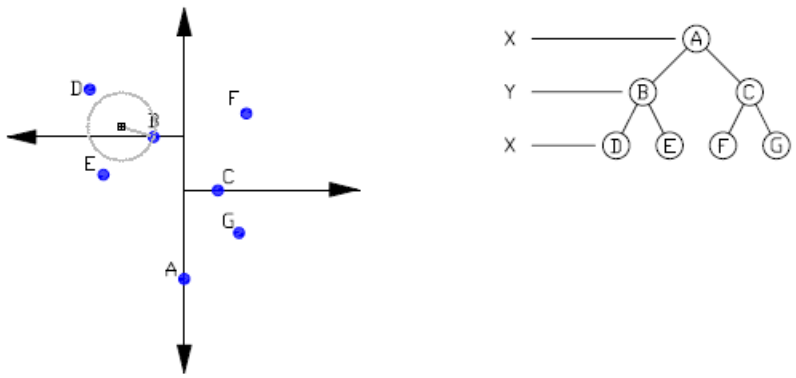
Algoritam najbližeg susjeda koji koristi *k-d stabla* nastoji pronaći točku na stablu koja je najbliža zadanoj točki za klasifikaciju. Ovim je postupkom znatno ubrzan osnovni *k-NN* algoritam jer se na ovaj način brzo i efektivno može zanemariti veliki dio prostora.

Na slici je prikazan prostor sa zadanim točkama i stvorenim *k-d stablom*.



Slika 12 Euclid-ov prostor sa zadanim točkama i k-d stablom

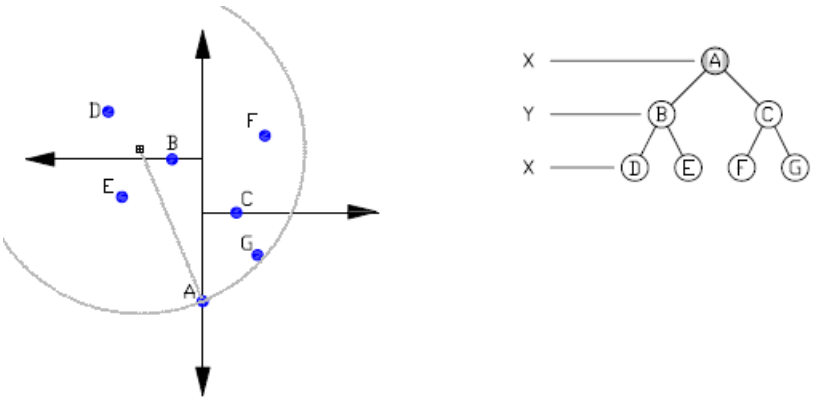
U prostor se uvrštava točka koju želimo klasificirati te joj se pridodaje udaljenost do najbližeg susjeda, označena kao sfera, koja je na početku nepoznata.



Slika 13 Prikaz prostora sa unesenom točkom za klasifikaciju

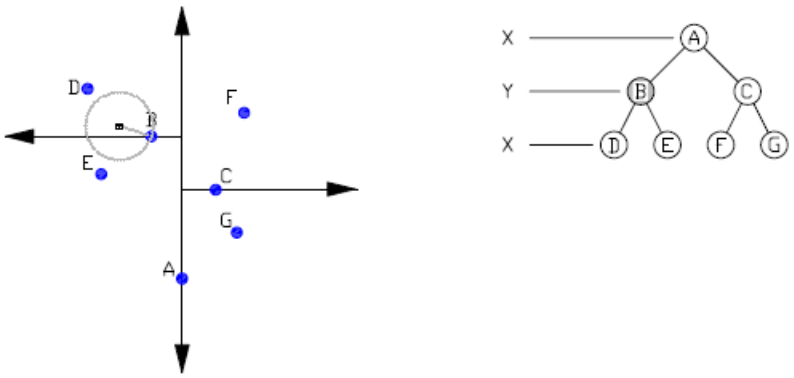
Način na koji algoritam funkcinira odvija se u nekoliko koraka:

- Počevši sa korijenskim čvorom, algoritam se po stablu kreće rekurzivno prema dolje



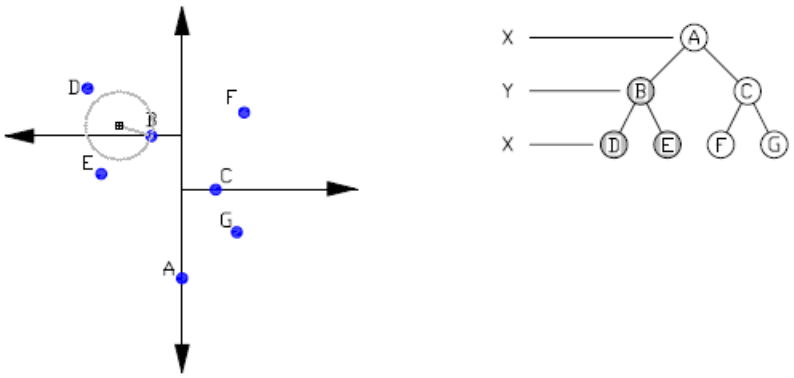
Slika 14 Algoritam započinje pretragu od korijenskog čvora (desno); Sfera presjeca sva područja, niti jedno se u ovom trenutku ne može izostaviti (lijevo)

- Kada se dođe do lisnog čvora, sprema taj čvor kao "*trenutno najbolji*"



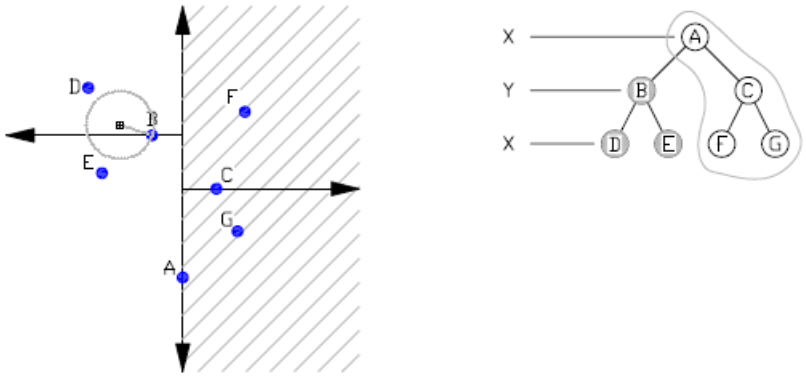
Slika 15 Odabrana točka B ima manju udaljenost od A i sprema se kao "trenutno najbolja"

- Algoritam proširuje rekurziju stabla, provodeći slijedeće korake:
  - Ukoliko je trenutni čvor bliži zadanoj točki od "*trenutno najboljeg*" čvora, taj čvor tada postaje najbolji i sprema se kao takav



Slika 16 Provjerava se da li su D i E bliži zadanoj točki od B; U ovom slučaju nisu pa se izbacuju iz pretrage

  - Provjerava se može li biti točaka na drugoj strani ravnine koje su bliže zadanoj točki nego "*trentno najbolja*". To se postiže presjecanjem ravnine sa sferom oko točke koja ima radijus jednak trenutnoj najbližoj udaljenosti
    - Ukoliko sfera presjeca ravninu, moguće je da postoje bliže točke na drugoj strani ravnine, pa se algoritam mora pomicati i po drugoj polovici toga djela stabla tražeći bliže točke, na isti način kao i prethodno
    - Ukoliko sfera ne presjeca ravninu, cijeli dio stabla može se izbrisati iz pretrage



Slika 17 Sfera ne presjeca ravninu (lijevo) i cijeli desni dio stabla može se ukloniti iz pretrage (desno)

- Kada se završi proces za korijenski čvor , algoritam je završen

U ovakvom se algoritmu koriste kvadratne udaljenosti, da bi se izbjeglo računanje korijena u Euclid-ovom prostoru.

Treba napomenuti da ovakav algoritam još uvijek nije dovoljno dobar za problem sustava sa brojnim dimenzijama, kakav predstavlja SURF algoritam. Naime, što je više točaka, to je više i dimenzija u sustavu i algoritam je drastično usporen.

3.1.2.3. Nasumična k-d stabla

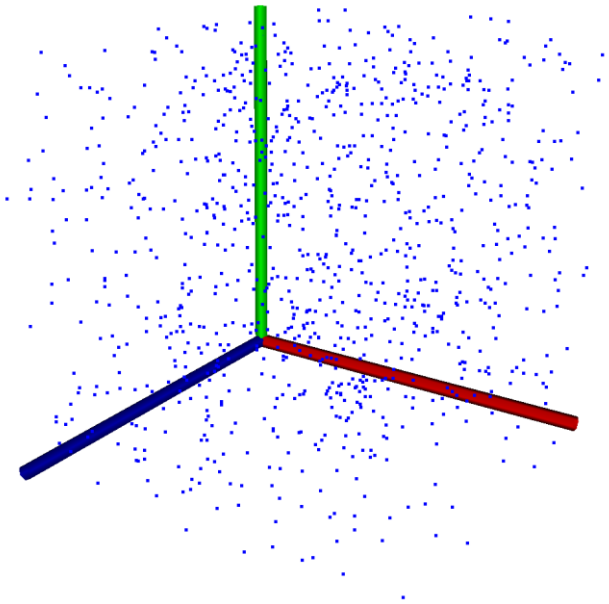
Zbog problema velike dimenzionalnosti koji predstavlja SURF algoritam, potrebno je primjeniti bolji način na koji će funkcionirati *k-d tree* algoritam. Naime, u višedimenzionalnom prostoru, *k-d stablo* će biti poprilično veliko i *Nearest Neighbor* algoritam će ga pretraživati cijelo dok ne pronađe najbližeg susjeda zadanoj točci. Taj proces predstavlja manu jer algoritam treba u realnom vremenu pratiti objekt u prostoru, a što dulje traje obrada podataka, kvaliteta praćenja postaje slabija.

Predložen je algoritam *nasumičnih k-d stabla* u kojemu se stvaraju višebrojna stabla. Originalna stabla dijele podatke na pola, pri svakom stupnjevanju. Algoritam nasumičnih stabla bira dimenziju dijeljenja ravnine nasumično u prvih *D* dimenzija u kojima podaci imaju najveću varijancu.

Pri pretraživanju stabala održava se jedan red sa prioritetom na svim nasumičnim stablima kako bi se pretraga mogla poredati prema povećavanju udaljenosti prema svakoj granici. Stupanj aproksimacije određuje se pretraživanjem određenog broja čvorova, pri čemu se pretraga završava i određuje se najbolji član kao par dviju točaka.

4. KONCEPT SUSJEDSTVA (IMPLEMENTACIJA POMOĆU PCL)

Osim algoritama za obradu oblaka točaka, PCL objedinjuje i algoritme za strojno učenje u što spada i algoritam najbližeg susjeda. U ovom će se poglavlju predstaviti implementacija algoritma na primjeru nasumično generiranih točaka.



Nasumično generirani oblak točaka

Na primjeru će se prikazati *nearest neighbor* pretraga u slučaju *k* pretrage i radijus pretrage.

Da bi se započelo sa pretragom, potrebno je prvo definirati klasu *kdtreeFLANN*.

4.1. Kratki opis klase KdTreeFLANN

Da bi se klasa uopće počela koristiti potrebno je uključiti ju u projekt:

```
#include <pcl/kdtree/kdtree_flann.h>
```

U sklopu rada opisati će se samo funkcije koje klasa sadrži, no koristiti će se samo dvije funkcije koje su od značaja.

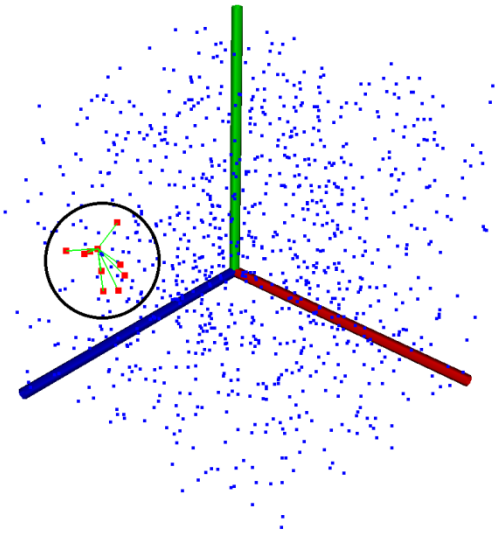
Tabela 1 Member funkcije klase *kdtree*

Tip varijable	*Member* funkcija
	KdTreeFLANN (bool sorted=true)
	Konstruktor za **KdTreeFLANN**

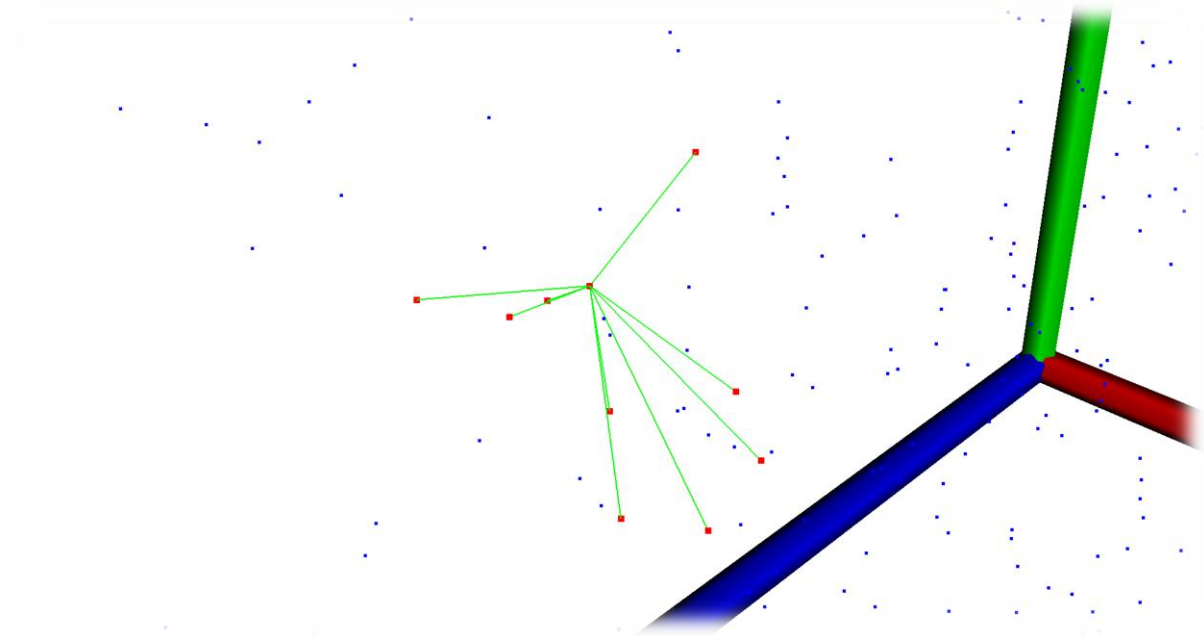
	KdTreeFLANN (const **KdTreeFLANN**< **PointT** > &k)
	Instanca za kopiranje konstruktora
KdTreeFLANN< **PointT** > &	**operator=** (const **KdTreeFLANN**< **PointT** > &k)
	Instanca za kopiranje operatora
void	**setEpsilon** (float eps)
	Postavljanje dozvoljene greške ϵ
void	**setSortedResults** (bool sorted)
Ptr	**makeShared** ()
virtual	**~KdTreeFLANN** ()
	Destruktor za **KdTreeFLANN**
void	**setInputCloud** (const **PointCloudConstPtr** &cloud, const **IndicesConstPtr** &indices=**IndicesConstPtr**())
	Instanca za postavljanje ulaznog oblaka
int	**nearestKSearch** (const **PointT** &point, int k, std::vector< int > &k_indices, std::vector< float > &k_sqr_distances) const
	Pretraga za K najbližih susjeda
int	**radiusSearch** (const **PointT** &point, double radius, std::vector< int > &k_indices, std::vector< float > &k_sqr_distances, unsigned int max_nn=0) const
	Pretraga unutar sfere određenog radijusa

4.2. Pretraga u slučaju *k* najbližih susjeda

U slučaju pretrage *k* najbližih susjeda, potrebno je koristiti funkciju *nearestKSearch*. Za pravilno korištenje funkcije potrebno je definirati točku oko koje će se tražiti najbliži susjedi (*const **PointT** &point*), parametar *k* koji određuje koliko najbližih susjeda se traži (*int k*), vektor u koji se spremaju indeksi točaka ulaznog oblaka (*std::vector< int > &k_indices*) i vektor u koji se spremaju udaljenosti pronađenih točaka od točke oko koje se traže susjedi (*std::vector< float > &k_sqr_distances*).



Slika 18 Rezultat pretrage sa K = 10

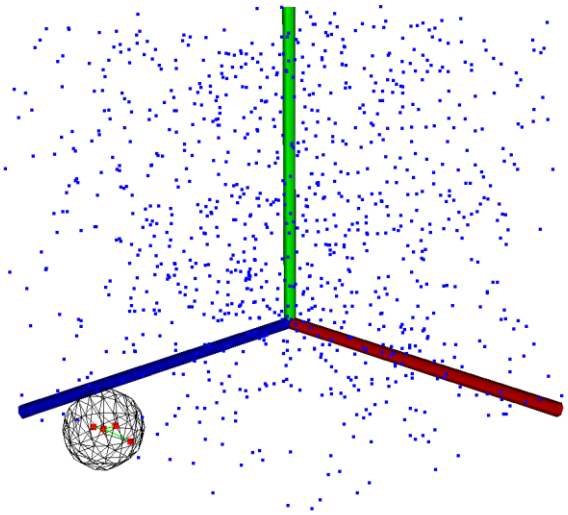


Slika 19 Rezultat pretrage sa K = 10 (povećani prikaz)

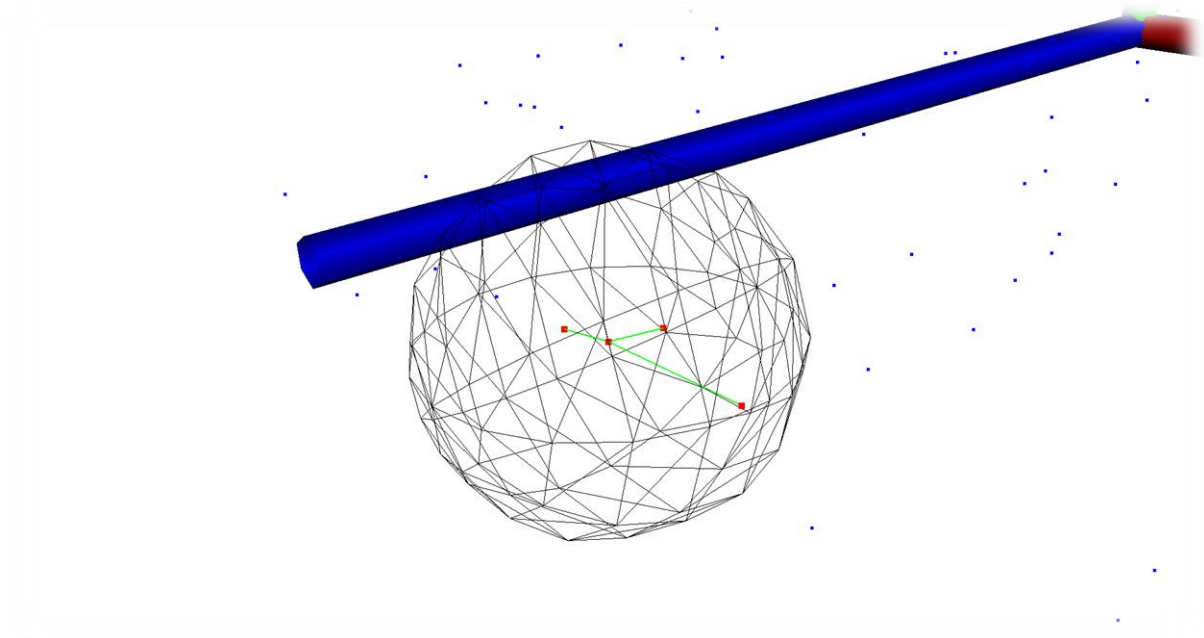
4.3. Pretraga u slučaju odabranog radijusa pretrage

Da bi se pravilno koristio tip pretrage unutar sfere, potrebno je koristiti funkciju *radiusSearch*. Parametri koje je potrebno unijeti su točka oko koje će se tražiti najbliži susjedi (*const* ***PointT*** *&point*), radijus pretrage (*double radius*), vektor u koji se spremaju indeksi točaka ulaznog oblaka (*std::vector< int > &k_indices*), vektor u koji se spremaju

udaljenosti pronađenih točaka od točke oko koje se traže susjedi (*std::vector< float > &k_sqr_distances*) i maksimalni željeni broj susjeda (*unsigned int max_nn=0*).



Slika 20 Rezultati pretrage sa R = 100



Slika 21 Rezultati pretrage sa R = 100 (povećani prikaz)

5. ESTIMACIJA NORMALA I ZAKRIVLJENOSTI POVRŠINE [7]

Jednom određeno, susjedstvo točaka P^k oko značajne točke p može se iskoristiti za estimaciju lokalne značajke koja opisuje geometriju površine oko točke p. Prvi problem predstavlja estimacija orijentacije površine u koordinatnom sustavu, tj. estimacija normale. One su bitno svojstvo površina i često se koriste u računalnoj grafici.

Makar postoje razne metode za estimaciju normala, najjednostavnija je ugađanje ravnine prvog reda. Problem određivanja normale aproksimira se problemom estimacije normale na tangentu ravnine. Ravnina je prikazana kao točka $\boldsymbol{x}$, a vektor normale sa $\vec{\boldsymbol{n}}$, dok je udaljenost od točke $p_i \in P^k$ do ravnine definirana kao

$$d_i = (p_i - \boldsymbol{x}) \cdot \vec{n} \quad (5.1)$$

Vrijednosti $\boldsymbol{x}$ i $\vec{n}$ računaju se metodom najmanjih kvadrata tako da je $d_i = 0$.

Uzevši u obzir

$$\boldsymbol{x} = \bar{\boldsymbol{p}} = \frac{\mathbf{1}}{k} \cdot \sum_{\boldsymbol{i=0}}^{k} \boldsymbol{p_i} \quad (5.2)$$

Kao centroidu oblaka P^k, rješenje za $\vec{\boldsymbol{n}}$ dobiva se analizom svojstvenih vrijednosti i svojstvenih vektora matrice kovarijance $C \in \mathbb{R}^{3x3}$ oblaka P^k

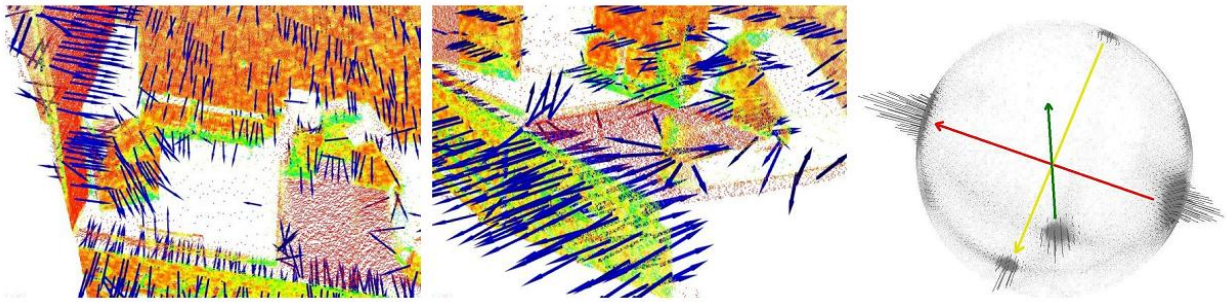
$$C = \frac{1}{k}\sum_{i=1}^{k} \xi_i \cdot (\boldsymbol{p}_i - \vec{p}) \cdot (p_i - \vec{p})^T, C \cdot \overrightarrow{v_j} = \lambda_j \cdot \overrightarrow{v_j}, j \in \{0, 1, 2\}$$

Izraz ξ_i mogući težinski faktor za $\boldsymbol{p}_i$ i obično je jednak 1. C je simetrična pozitivno semidefinitna matrica i njene svojstvene vrijednosti i svojstveni vektori su realni brojevi $\lambda_j \in \mathbb{R}$. Svojstveni vektori $\overrightarrow{v_j}$ tvore ortogonalni okvir, koji odgovara komponentama oblaka. Ukoliko je $0 \leq \lambda_0 \leq \lambda_1 \leq \lambda_2$, svojstveni vektor $\overrightarrow{v_0}$ koji odgovara najmanjoj

svojstvenoj vrijednosti λ_0 je aproksimacija $+\vec{n} = \{n_x, n_y, n_z\}$ ili $-\vec{n}$. Alternativno se $\vec{n}$ može predstaviti kao par kuteva (φ, θ) u sfernom koordinatnom sustavu

$$\varphi = arctan\left(\frac{n_z}{n_y}\right), \theta = arctan\frac{\sqrt{\left(n_y^2 + n_z^2\right)}}{n_x} \quad (5.4)$$

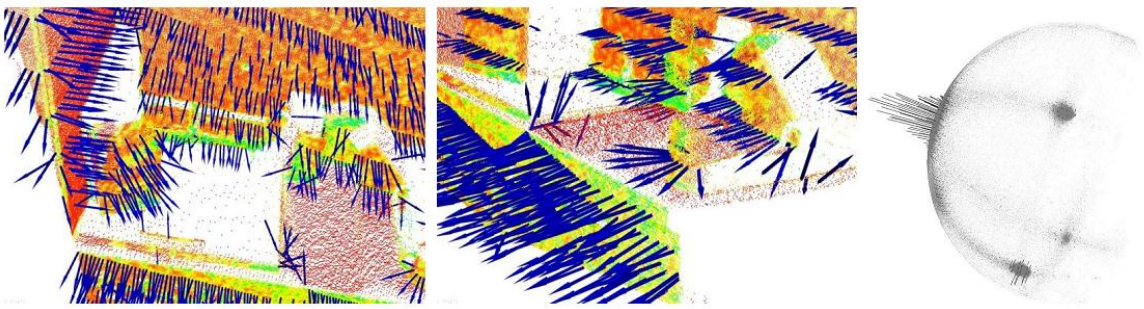
U pravilu, ne postoji matematički način da se odredi pravilna orijentacija normale $\vec{n}$. Orijentacija normale izračunata kako je prikazano je dvosmislena i nije pravilno usmjerena u odnosu na oblak točaka.



Slika 22 Loša orijentacija normala

Rješenje problema je jednostavno ukoliko je poznato ishodište kordinatnog sustava (gledište (*eng. viewpoint*)) v_p. Da bi se sve normale orijentirale prema gledištu moraju zadovoljavati jednadžbu

$$\overrightarrow{n_i} \cdot \left(v_p - p_i\right) > 0 \quad (5.5)$$



Slika 23 Kvalitetna orijentacija normala

U situacijama kada nisu poznate informacije o gledištu, problem se je puno teži za rješiti. Jedno od mogućih rješenja je modeliranje dosljednosti kao problem optimizacije grafa. Ideja se sastoji od promatranja dvije točke u prostoru p_i i p_j koje pripadaju glatkoj površini i geometrijski su blizu. Normale tih dviju točaka su jednako orijentirane ukoliko zadovoljavaju uvjet

$$n_i \cdot n_j \approx 1 \qquad (5.6)$$

Takva formulacija rješava problem, no samo kada je oblak točaka gust i uzorkovan je po glatkoj površini. Prema navedenoj jednadžbi svaku je točku $p_i \in P$ potrebno modelirati kao čvor. Točka koju se prvu bira je nasumična točka koja sigurno ima točno orijentiranu normalu. Na kraju se postavlja uvjet

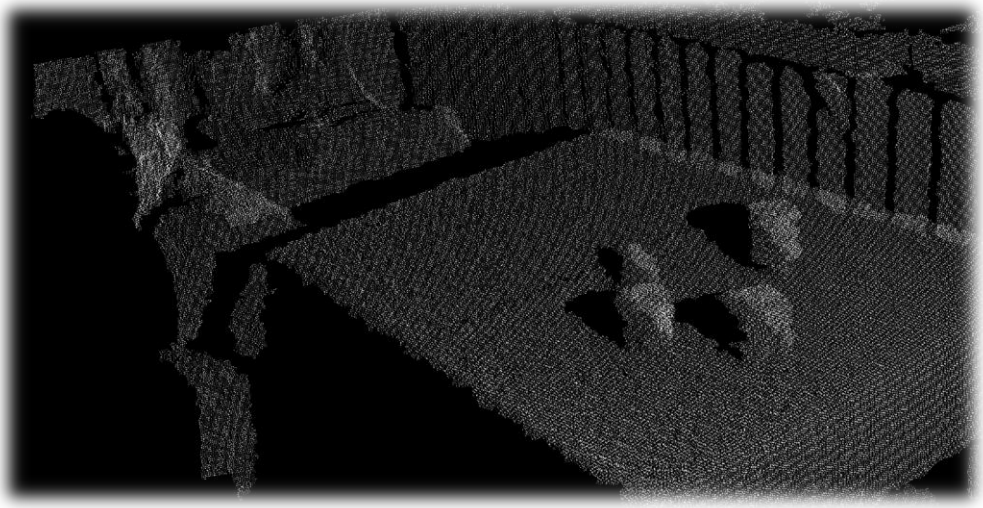
$$n_i \cdot n_j < 0 \rightarrow n_j = -n_j \qquad (5.7)$$

Problem zakrivljenosti površine u određenoj točki rješava se uporabom svojstvenih vrijednosti λ_j matrice kovarijance C. Ukoliko je $\lambda_0 = \min(\lambda_j)$ tada je varijacija oko točke *p* procijenjena izrazom

$$\sigma_p = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \qquad (5.8)$$

6. ESTIMACIJA NORMALA I ZAKRIVLJENOSTI POVRŠINE (IMPLEMENTACIJA POMOĆU PCL)

Određivanje normala pomoću PCL knjižnice algoritama zahtjeva korištenje klase *NormalEstimation*. Rezultati će biti prikazani na eksperimentalnom postavu.



Slika 24 Eksperimentalni postav

6.1. Kratki opis klase *NormalEstimation*

Da bi se klasa mogla početi koristiti potrebno ju je uključiti u projekt pomoću:

```
#include <pcl/features/normal_3d.h>
```

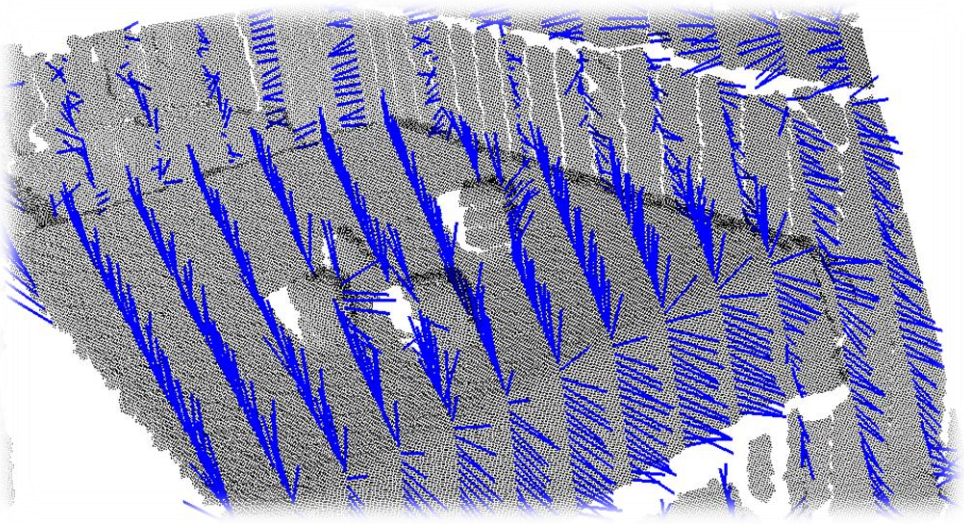
Tabela 2 Member funkcije klase *NormalEstimation*

Tip varijable	*Member* funkcija
	NormalEstimation ()
	Konstruktor
virtual	**~NormalEstimation** ()
	Destruktor
void	**computePointNormal** (const **pcl::PointCloud**< PointInT > &cloud, const std::vector< int > &indices, Eigen::Vector4f &plane_parameters, float &curvature)

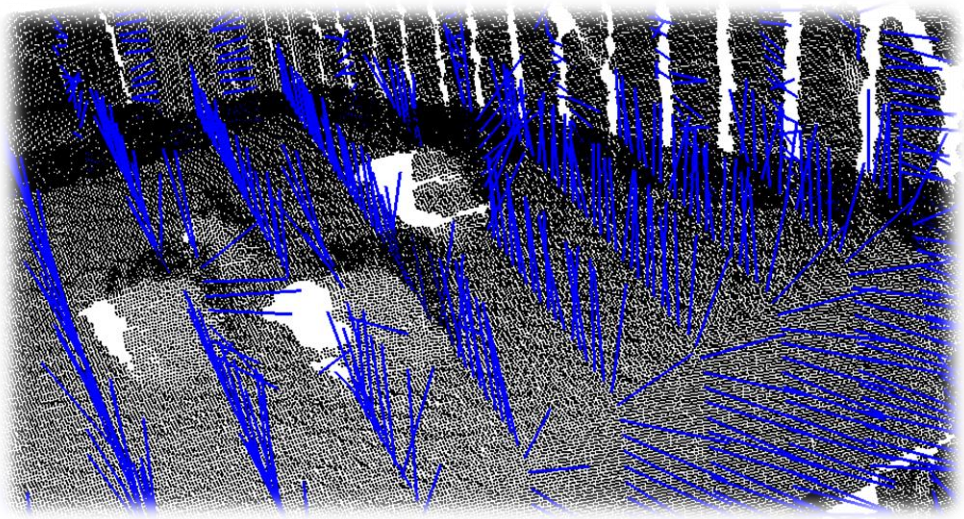
	Računanje normale u zadanoj točki
void	**computePointNormal** (const **pcl::PointCloud**< PointInT > &cloud, const std::vector< int > &indices, float &nx, float &ny, float &nz, float &curvature)
	Računanje normale u zadanoj točki
virtual void	**setInputCloud** (const **PointCloudConstPtr** &cloud)
	Ulazni oblak točaka na kojemu se vrši estimacija
void	**setViewPoint** (float vpx, float vpy, float vpz)
	Postavi gledište
void	**getViewPoint** (float &vpx, float &vpy, float &vpz)
	Dohvati gledište
void	**useSensorOriginAsViewPoint** ()
	Koristi ishodište senzora kao gledište

6.2. Rezultati estimacije normala pomoću PCL

Rezultati algoritma prikazani su za oblak točaka prikupljenog sa eksperimentalnog postava.



Slika 25 Rezultati estimacije normal pomoću klase *NormalEstimation*



Slika 26 Rezultati estimacije normal (povećani prikaz)

7. SEGMENTACIJA RAVNINE [7]

U suštini, potraga za određenim strukturama ili uzorcima u oblaku točaka daje kvalitetne rezultate ovisno o dva parametra:

- Složenost strukture koja se traži
- Razina prisutnog šuma

Oba dva parametra utječu na svojstva proračuna u pretrazi. Kako bi se uštedilo vrijeme, algoritmi koji ugađaju modele su ugrađeni u generatore heurističke hipoteze poput RANSAC-a, koji priskrbljuje gornje granice na broju iteracija koje treba provesti kako bi se postigla određena vjerojatnost za uspjeh.

U idealizirnom svijetu postojala bi baza podataka pojednostavljenih modela (poput CAD modela) za sve postojeće objekte, koji bi zamijenili većinu točaka u oblaku. Pod pretpostavkom da algoritmi rade jako dobro, problem segmentacije površina i pojednostavljenja prikupljenih setova podataka bi bio rješen. Na žalost, to se rijetko događa i zato je jedna od najpopularnijih tehnika pojednostavljenja podataka ugađanje 3D geometrijskih primitiva. U ovom slučaju, radi se o jednostavnoj geometrijskoj primitivi koju je jednostavno za izračunati i može aproksimirati prikupljene podatke unutar tolerancija. Primjeri takvih primitiva su ravnine, cilindri, sfere i konusi.

Da bi se razumjelo zašto je segmentacija od velikog značaja potrebno je razmotriti konačne ciljeve. Cijeli se problem može vidjeti na primjeru mobilnog robota koji u prostoru pokušava izbjegavati prepreke. Kada bi se njegov virtualni okoliš sastojao od točaka bilo bi potrebno proračunati mnogo udaljenosti od svojih ekstremiteta do svih točaka u prostoru kako bi zaključio da li je u stanju kolizije ili ne. Ukoliko se točke koje pripadaju ravnim površinama zamjene 2D poligonima, to bi drastično smanjilo broj prolazaka kroz računsku petlju. Računalo više ne bi trebalo računati sa svim točkama, već samo sa jednom.

Još jedan primjer nalazi se u rješavanju problema prihvata (grasping). Ukoliko robotski manipulator treba prihvatiti čašu ili limenku (ili bilo kakav cilindrični objekt), estimacija točnih pozicija prihvata ili strategija na šumovitom setu podataka bila bi prezahtjevna za računalo. Zamjenom oblaka točaka sa jednim od modela pojednostavila bi problem.

U ovom diplomskom radu pričati će se najviše o segmentaciji ravnine, tj. Segmentaciji podloge na kojoj se objekti nalaze.

Da bi se provela segmentacija ravnine, algoritam treba izvršiti nekoliko koraka:

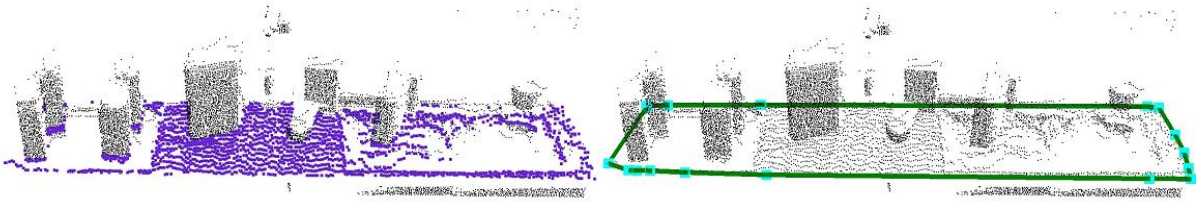
- Nasumično izabrati tri ne-kolinearne točke $\{\boldsymbol{p_i}, \boldsymbol{p_j}, \boldsymbol{p_k}\}$ u oblaku P
- Izračunati koeficijente modela pomoću tri izabrane točke

$$ax + by + cz + d = 0 \qquad (7.1)$$

  Izračunati udaljenosti svih $p \in P$ do modela ravnine (a, b, c, d)
- Izbrojati broj točaka $p^* \in P$ čije udaljenosti *d* do modela ravnine spadaju unutar $0 \leq |d| \leq |d_t|$, gdje je d_t granica koju odabire korisnik

Svaki set točaka p^* se sprema i gore navedeni koraci se ponavljaju *k* iteracija. Nakon što se program zaustavi set s najvećim brojem unutarnjih točaka (*inliers*) je odabran kao najbolja podloga za model ravnine.

Gore navedeni koraci za estimaciju modela ravnine predstavljaju teoretski pristup za segmentaciju. U praksi je moguće naići i na vertikalne ravnine gdje se mora postaviti i uvjet okomitosti na željenu os. Za *tabletop* algoritme to je *Z* os kako bi algoritam pronašao samo horizontalne.



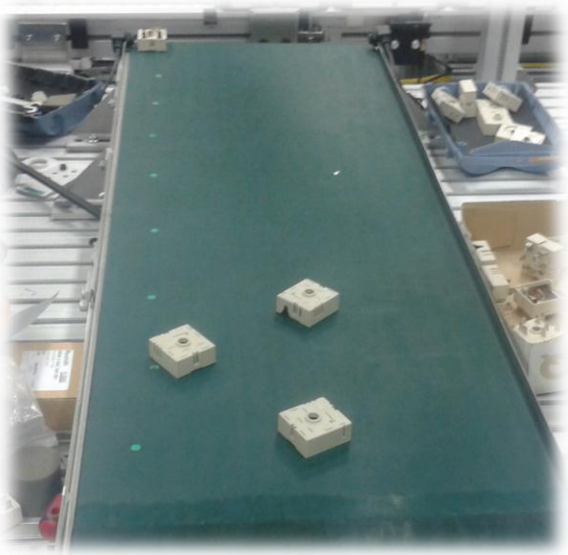
Slika 27 Primjer segmentacije ravnine

8. SEGMENTACIJA RAVNINE (IMPLEMENTACIJA POMOĆU PCL) [4]

Da bi se pomoću PCL omogućila segmentacija objekata, potrebno je iskoristiti klasu *SACSegmentation*. Klasa sama po sebi sadrži mnoge modele za segmentaciju, no u ovom poglavlju koristi se samo model ravnine (*ax+by+cz+d=0*).

Da bi se što lakše ispitao algoritam naprevljen je eksperimentalni postav koji ličina pokretnoj traci u laboratoriju fakulteta. Kamera je postavljena tako da su joj izraženi rubovi podloge koju se segmentira.

Slika 28 Eksperimentalni postav



Slika 29 Pokretna traka u laboratoriju fakulteta

8.1. Kratki opis klase SACSegmentation

Da bi se klasa mogla početi koristiti potrebno ju je uključiti u projekt pomoću:

```
#include <pcl/segmentation/sac_segmentation.h>
```

Tabela 3 Member funkcije klase *SACSegmentation*

	SACSegmentation (bool random=false)
	Konstruktor
virtual	**~SACSegmentation** ()
	Destruktor
void	**setModelType** (int model)
	Tip modela koji će se koristiti
int	**getModelType** () const
	Tip SAC modela
SampleConsensusPtr	**getMethod** () const
	Dohvati pokazivač za SAC metodu
SampleConsensusModelPtr	**getModel** () const
	Dohvati pokazivač za SAC model
void	**setMethodType** (int method)
	Sample consenzus metoda

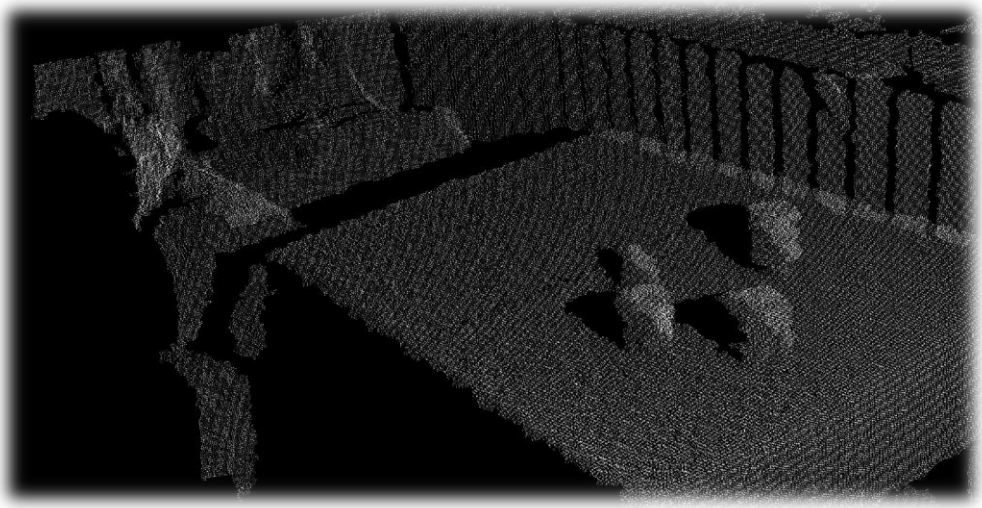
int	**getMethodType** () const
	Dohvati tip *sample consensus* metode
void	**setDistanceThreshold** (double threshold)
	Granica modela
double	**getDistanceThreshold** () const
	Dohvati granicu modela
void	**setMaxIterations** (int max_iterations)
	Postavi maksimalni broj iteracija
int	**getMaxIterations** () const
	Dohvati maksimalni broj iteracija prije odustajanja
void	**setProbability** (double probability)
	Postavi vjerojatnost
double	**getProbability** () const
	Dohvati vjerojatnost
void	**setOptimizeCoefficients** (bool optimize)
	Postavi u *true* ukoliko je potrebna dodatna optimizacija
bool	**getOptimizeCoefficients** () const
	Dohvati parametre optimizacije

void	**setRadiusLimits** (const double &min_radius, const double &max_radius)
	Odredi minimalni i maksimalni radijus objekta (za cilindre, sfere ili konuse)
void	**getRadiusLimits** (double &min_radius, double &max_radius)
	Dohvati minimalni i maksimalni radijus
void	**setSamplesMaxDist** (const double &radius, **SearchPtr** search)
void	**getSamplesMaxDist** (double &radius)
void	**setAxis** (const Eigen::Vector3f &ax)
	Postavi os na koju će objekt biti okomit
Eigen::Vector3f	**getAxis** () const
	Dohvati os na koju će objekt biti okomit
void	**setEpsAngle** (double ea)
double	**getEpsAngle** () const

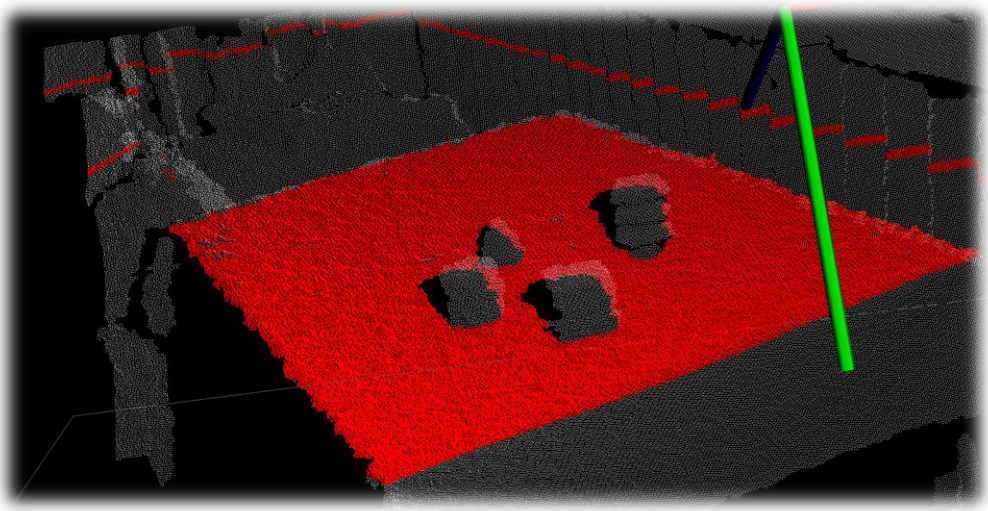
virtual void	segment (PointIndices &inliers, ModelCoefficients &model_coefficients)
	Metoda za segmentaciju modela

8.2. Rezultati segmentacije ravnine

Rezultati algoritma prikazati će se za oblak točaka prikupljeog sa eksperimentalnog postava.



Slika 30 Oblak točaka prikupljenog sa eksperimentalnog postava



Slika 31 Rezultati segmentacije ravnine

Na slici su prikazani rezultati segmentacije ravnine koji pokazuju da u kontroliranoj okolini algoritam radi kvalitetno. No, u nekoj drugoj okolini potrebno je dodatno podesiti

okolinu, kameru ili oboje. U laboratoriju fakulteta bilo je potrebno dodavati „zamke“ koje će usmjeriti algoritam na izraženu ravninu sa jasnim granicama.

Slika 32 Podešavanje ravnine u laboratoriju fakulteta

Na slici je vidljivo da su postavljena dva predmeta koji za algoritam predstavljaju dvije dodatne ravnine okomite na onu koju se segmentira.

8.3. Funkcija u C++ programskom jeziku

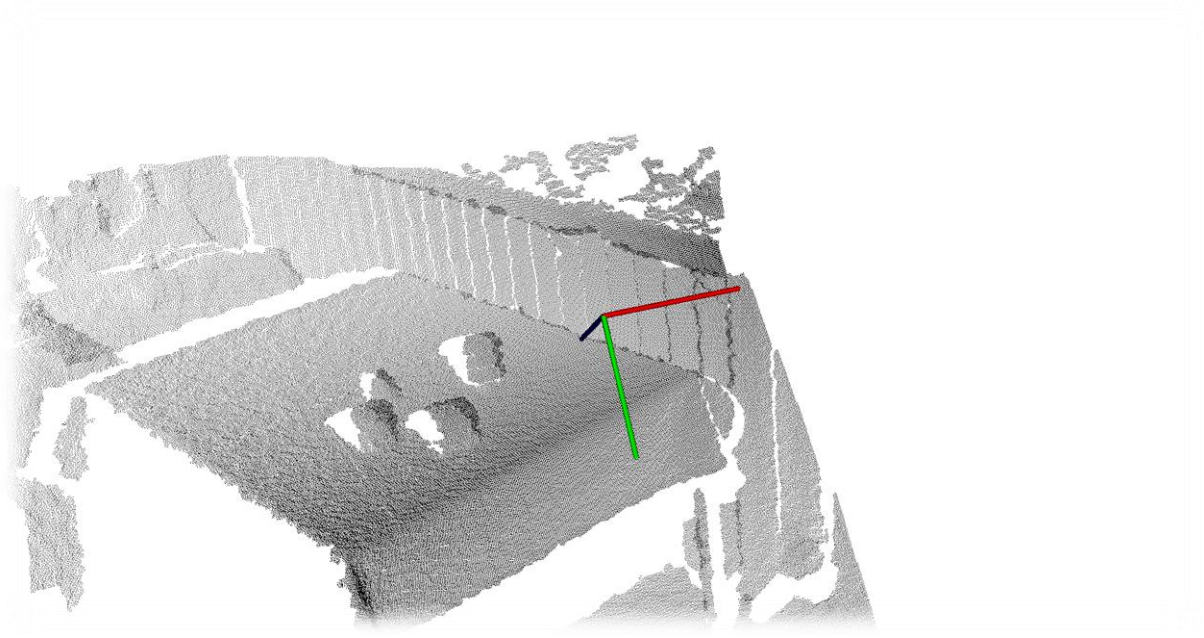
```
void segmentacija_ravnine(pcl::SACSegmentation<pcl::PointXYZ> segmentacija,
	pcl::ExtractIndices<pcl::PointXYZ> extract,
	pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_in,
	pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_out,
	pcl::ModelCoefficients::Ptr coeff,
	pcl::PointIndices::Ptr indeksi,
	double threshold,
	int max_it,
	bool setneg){

		// Zadavanje parametara segmentacije
		segmentacija.setInputCloud(cloud_in);
		segmentacija.setOptimizeCoefficients (true);
		segmentacija.setModelType(pcl::SACMODEL_PLANE);
		segmentacija.setMethodType(pcl::SAC_RANSAC);
		segmentacija.setDistanceThreshold(threshold);
		segmentacija.setMaxIterations(max_it);
		segmentacija.segment(*indeksi, *coeff);
		extract.setInputCloud(cloud_in);
		extract.setIndices(indeksi);
		extract.setNegative(setneg);
		extract.filter(*cloud_out);
}
```

9. GRUPIRANJE OBJEKATA (CLUSTERING) [7]

Prema potrebama algoritma grupiranje podataka ima zadatak dijeljenja neorganiziranog oblaka u manje djelove u svrhu postizanja veće točnosti i kraćeg vremena same obrade.

Jednostavan pristup grupiranja podataka je implementacija podpodjele pomoću 3D mreže prostora koristeći kocke fiksnih dimenzija, tj. Octree struktura podataka. Takva reprezentacija podataka je vrlo brza i korisna je u situacijama gdje je potrebna ili volumetrička reprezentacija zauzetog prostora ili pri aproksimaciji podataka različitim strukturama u rezultiranim kockama.

Unatoč prednostima, metoda 3D mreže ima smisla samo kada je potrebna podjela s jednakim razmakom. U slučajevima kada su grupirani podaci različitih dimenzija, potreban je složeniji algoritam. Da bi se teorija bolje shvatila, pretpostaviti će se da postoji oblak točaka koji predstavlja površinu sa različitim objektima, različitih veličina na njoj. Takvi se primjeri mogu naći u kuhinji, radnom stolu, laboratoriju, na industrijskoj pokretnoj traci itd.



Slika 33 Površina sa objektima

Da bi se postiglo kvalitetno grupiranje podataka u neorganiziranoj cjelini, sustav prvo treba znati što je to *grupa točaka* (*cluster*) i što ga dijeli odr drugih grupa u istom

koordinatnom sustavu. Da bi se to postiglo, potrebno je matematički definirati grupu točaka u istom koordinatnom sustavu.

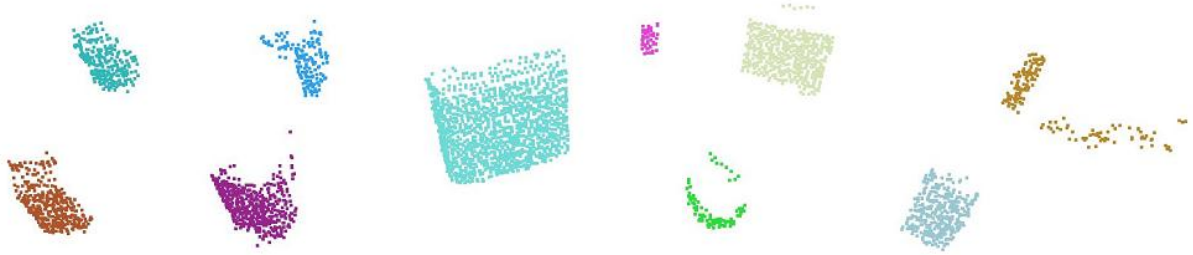
Grupa točaka $O_i = \{p_i \in P\}$ je različita od druge grupe $O_j = \{p_j \in P\}$ ukoliko je

$$min\|p_i - p_j\|_2 \geq d_{th} \qquad (9.1)$$

Gdje je d_{th} maksimalna nametnuta granica udaljenosti. Gore navedena jednadžba nalaže da ukoliko je minimalna udaljenost između setova točaka $p_i \in P$ i $p_j \in P$ veća nego zadana udaljenost, točke p_i spadaju grupi točaka O_i dok set točaka p_j spada grupi točaka O_j.

Sa stajališta implementacije, bitno je znati kako će se minimalna udaljenost estimirati. Jedno rješenje je iskoristiti algoritam najbližeg susjeda proširen kd stablom. Tako će algoritam grupiranja dobiti slijedeći oblik:

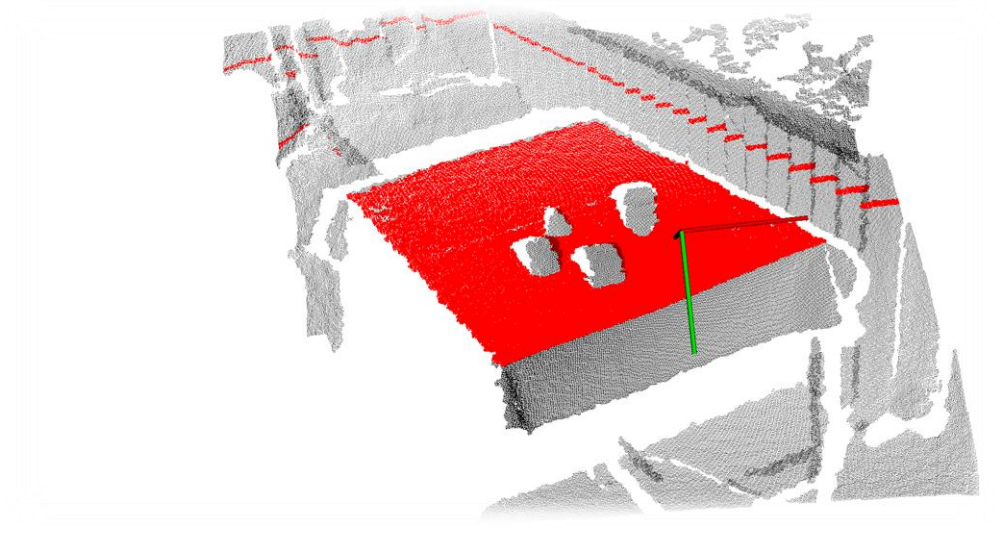
- Stvoriti *kd* reprezentaciju ulaznog oblaka
- Postaviti praznu listu *C* za grupe, i listu točaka koje treba provjeravati *Q*
- Za svaku točku $p_i \in P$ provesti slijedeće korake:
  - Dodaj p_i u *Q*
  - Za savaku točku $p_i \in Q$ provedi slijedeće korake:
    - Traži set P_i^k sastavljen od susjeda točke p_i u sferi radijusa $r < d_{th}$
    - Za svaki susjed $p_i^k \in P_i^k$ pregledaj da li je točka već obrađena i ukoliko nije dodaj je u *Q*
  - U trenutku kada je lista svih točaka u *Q* obrađena, dodaj *Q* u listu grupa *C* i postavi *Q* ponovno na praznu listu
- Algoritam prestaje kada su sve točke $p_i \in P$ obrađene



Slika 34 Primjer grupiranih oblaka točaka

10. GRUPIRANJE OBJEKATA (IMPLEMENTACIJA POMOĆU PCL) [6]

Grupiranje objekata u oblaku točaka se vrši pomoću klase *EuclideanClusterExtraction*. U ovom poglavlju opisati će se grupiranje oblaka točaka dobivenog segmentacijom. Svrha takve obrade je filtriranje ostatka ravnine kako bi se dobila najveća površina koja opisuje podlogu na kojoj se nalaze objekti za prepoznavanje. U kasnijem poglavlju algoritam grupiranja koristiti će se za lokalizaciju objekata u prostoru.



Slika 35 Ulazni oblak (segmentirana površina označena je crvenom bojom)

10.1. Kratki opis klase *EuclideanClusterExtraction*

Da bi se klasa pravilno koristila u C++ programskom jeziku, potrebno ju je inicijalizirati pomoću:

```
#include <pcl/segmentation/extract_clusters.h>
```

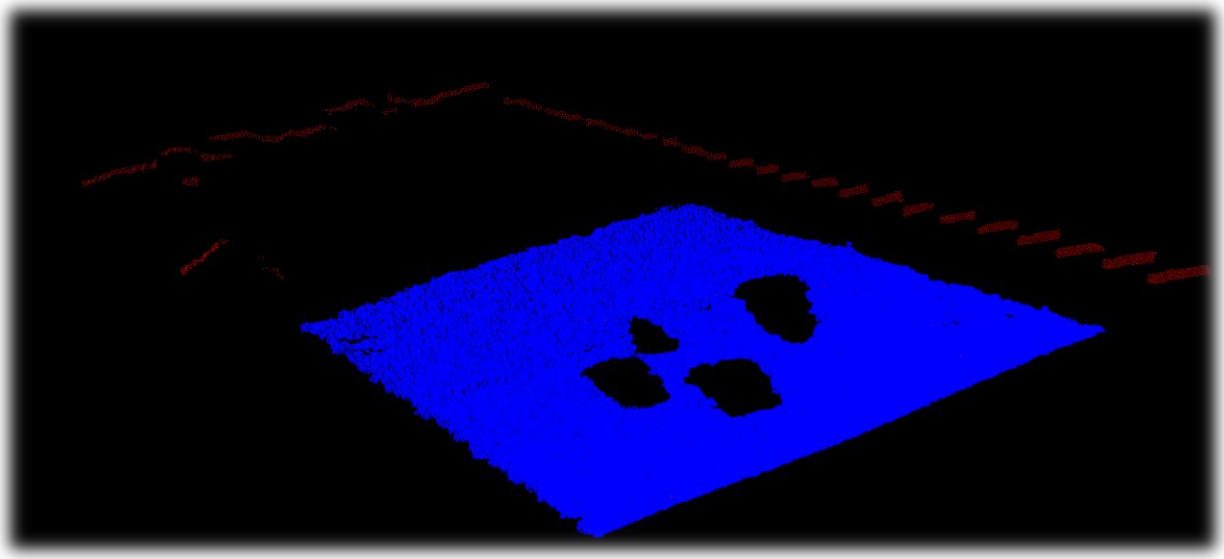
Tabela 4 Member funkcije *EuclideanClusterExtraction* klase

Tip varijable	*Member* funkcija
	EuclideanClusterExtraction ()
	Konstruktor
void	setSearchMethod (const KdTreePtr &tree)
	Pokazivač na metodu pretrage

KdTreePtr	getSearchMethod () const
	Dobavi pokazivač za metodu pretrage
void	setClusterTolerance (double tolerance)
	Tolerancija grupe
double	getClusterTolerance () const
	Dobavi toleranciju grupe
void	setMinClusterSize (int min_cluster_size)
	Postavi minimalnu veličinu grupe
int	getMinClusterSize () const
	Dobavi minimalnu veličinu grupe
void	setMaxClusterSize (int max_cluster_size)
	Postavi maksimalnu veličinu grupe
int	getMaxClusterSize () const
	Dobavi maksimalnu veličinu grupe
void	extract (std::vector< PointIndices > &clusters)
	Započinje grupiranje oblaka

10.2. Rezultati algoritma grupiranja

Rezultat algoritma grupiranja prikazan je na eksperimentalnom postavu.



Slika 36 Rezultat algoritma grupiranja (plavom bojom označena pronađena površina; crvenom bojom označene su ostale grupe koje se odbacuju)

10.3. Funkcija u C++ programskom jeziku

```
vector<pcl::PointCloud<pcl::PointXYZ>::Ptr>
clustering(pcl::search::KdTree<pcl::PointXYZ>::Ptr tree,
	pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_in,
	pcl::EuclideanClusterExtraction<pcl::PointXYZ> ece,
	double cl_tol,
	int min_cl_size)
{
	tree->setInputCloud(cloud_in);

	vector<pcl::PointIndices> cl_ind;

	ece.setClusterTolerance(cl_tol);
	ece.setMinClusterSize(min_cl_size);
	ece.setSearchMethod(tree);
	ece.setInputCloud(cloud_in);
	ece.extract(cl_ind);

	pcl::PCDWriter writer;

	int z = 0;
	vector<pcl::PointCloud<pcl::PointXYZ>::Ptr> container;
	for (std::vector<pcl::PointIndices>::const_iterator it = cl_ind.begin (); it != 
cl_ind.end (); ++it)
  {
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_cluster (new 
pcl::PointCloud<pcl::PointXYZ>);
    for (std::vector<int>::const_iterator pit = it->indices.begin (); pit != it-
>indices.end (); pit++)
	cloud_cluster->points.push_back (cloud_in->points[*pit]);
	cloud_cluster->width = cloud_cluster->points.size ();
	cloud_cluster->height = 1;
	cloud_cluster->is_dense = true;

	std::cout << "PointCloud representing the Cluster: " << cloud_cluster->points.size
() << " data points." << std::endl;
	std::stringstream pp;
	pp << "cloud_ROI_cluster_" << z << ".pcd";
```

```
      writer.write<pcl::PointXYZ> (pp.str (), *cloud_cluster, false);
         container.push_back(cloud_cluster);
      z++;
  }
       return(container);
       delete[] &container;
       delete[] &cl_ind;
}
```

11. PROSTORNA TRANSFORMACIJA (IMPLEMENTACIJA U C++ PROGRAMSKOM JEZIKU)

Cilj određivanja prostorne transformacije je određivanje homogene matrice transformacije q^{4x4}. Struktura matrice glasi:

$$q = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (11.1)$$

Prije računanja matrice treba napomenuti da se radi o određivanju orijentacije oblaka točaka za koji nije poznata orijentacija u prostoru već samo pozicija.

Postupak određivanja matrice transformacije jedan je od ključnih koraka za dobivanje VOI (Volumena od interesa).

Postupak određivanja transformacije je:

1. Računanje centroide oblaka (težišta)
2. Računanje normalizirane matrice kovarijance
3. Računanje svojstvenih vektora
4. Uvrštavanje svojstvenih vektora u matricu transformacije *q*

11.1. Implementacija algoritma za računanje transformacije u C++ programskom jeziku

Računanje transformacije u C++:

```
Eigen::Vector4f centroida;
     pcl::compute3DCentroid(*cloud_segmentirano, centroida);
     Eigen::Matrix3f kovarijanca;
     pcl::computeCovarianceMatrixNormalized(*cloud_segmentirano, centroida, kovarijanca);
     Eigen::SelfAdjointEigenSolver<Eigen::Matrix3f>              eigen_solver(kovarijanca,
Eigen::ComputeEigenvectors);
     Eigen::Matrix3f eigDx = eigen_solver.eigenvectors();
     eigDx.col(2) = eigDx.col(0).cross(eigDx.col(1));
     Eigen::Matrix4f p2w(Eigen::Matrix4f::Identity());
     p2w.block<3,3>(0,0) = eigDx.transpose();
     p2w.block<3,1>(0,3) = -1.f * (p2w.block<3,3>(0,0) * centroida.head<3>());
```

U navedenom programu matrica *q* opisana je varijablom *p2w* (*Points2World*)

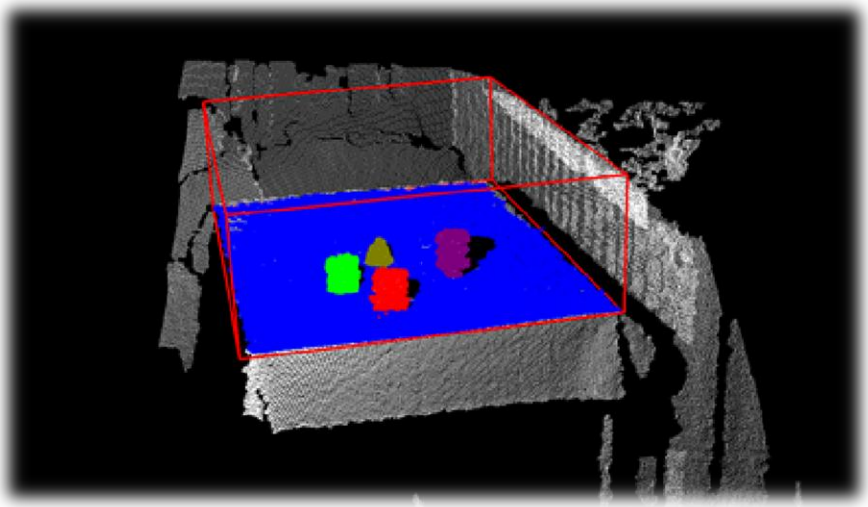
11.2. Implementacija određivanja VOI u C++ programskom jeziku

Računanje VOI u C++:

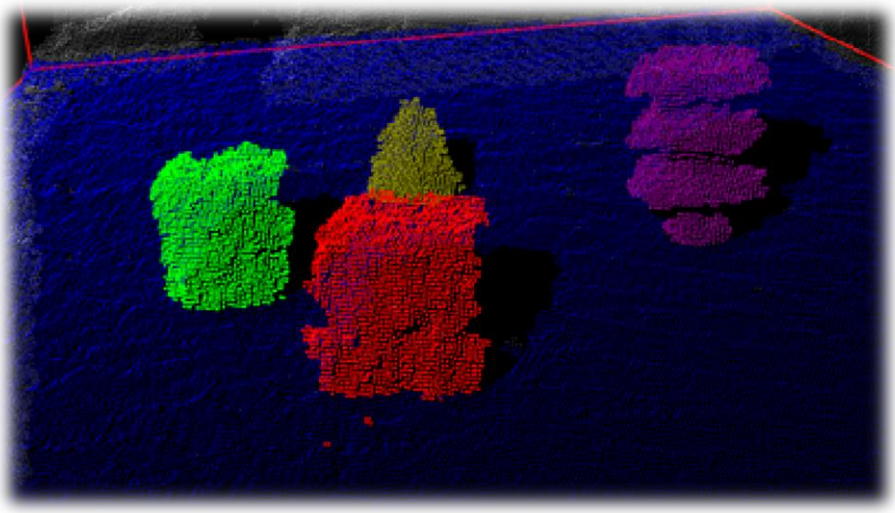
```
      Eigen::Vector4f centroida;
      pcl::compute3DCentroid(*cloud_segmentirano, centroida);
      Eigen::Matrix3f kovarijanca;
      pcl::computeCovarianceMatrixNormalized(*cloud_segmentirano, centroida, kovarijanca);
      Eigen::SelfAdjointEigenSolver<Eigen::Matrix3f>              eigen_solver(kovarijanca,
Eigen::ComputeEigenvectors);
      Eigen::Matrix3f eigDx = eigen_solver.eigenvectors();
      eigDx.col(2) = eigDx.col(0).cross(eigDx.col(1));
      Eigen::Matrix4f p2w(Eigen::Matrix4f::Identity());
      p2w.block<3,3>(0,0) = eigDx.transpose();
      p2w.block<3,1>(0,3) = -1.f * (p2w.block<3,3>(0,0) * centroida.head<3>());
         pcl::PointCloud<pcl::PointXYZ> cPoints;
      pcl::transformPointCloud(*cloud_segmentirano, cPoints, p2w);
         pcl::PointXYZ tockicamin;
         pcl::PointXYZ tockicamax;
         pcl::getMinMax3D(cPoints, tockicamin, tockicamax);
         float visina_VOI;
         cout << "Kolika ce biti visina VOI?" << endl;
         cin >> visina_VOI;
         tockicamax.x = visina_VOI;
         cloud_min_max->points.push_back(tockicamin);
         cloud_min_max->points.push_back(tockicamax);
      const    Eigen::Vector3f    mean_diag    =    0.5f*(tockicamax.getVector3fMap()    +
tockicamin.getVector3fMap());
      const Eigen::Quaternionf qfinal(eigDx);
      const Eigen::Vector3f tfinal = eigDx*mean_diag + centroida.head<3>();
```

11.2.1. Konačni rezultati nakon provedbe svih navedenih algoritama

Slijedeća slika prikazuje konačni rezultat nakon provedbe svih navedenih algoritama (estimacija normala, segmentacija, clustering...).



Slika 37 Prikaz rezultata (segmentacija, VOI, grupiranje oblaka točaka u objekte i lokalizacija)



Slika 38 Prikaz rezultata provedenih algoritama (povećani prikaz)

12. ZAKLJUČAK

U ovome radu objedinjeni su algoritmi obrade oblaka točaka, segmentacije te grupiranja. Sustav je povezan na robota putem TCP/IP veze, a predmeti su uspješno lokalizirani.

Sustav za akviziciju podataka MS Kinect pokazao se pouzdanim za istraživanje i razvoj algoritama za prepoznavanje objekata, no ta kamera nije zamišljena za implementaciju u industrije, već u domovima na igraćim konzolama i shodno tome posjeduje slabu rezoluciju (640x480; YxX). Senzor dubine je također jako grubo diskretiziran (podaci o dubini koje nudi su veličine 2^{11} bita) što rezultira velikim smetnjama na većim udaljenostima.

Knjižnica algoritama PCL pokazala se jako kvalitetnom, no u mnogo slučajeva nije jednostavna implementacija i zahtjeva veće znaje programskog jezika C++. Kako je sama knjižnica sastavljena od niza drugih knjižnica, potrebno je dodatno proučavanje kako bi se u potpunosti shvatio pristup korištenja PCL-a. Nadalje, sama knjižnica je još uvijek u razvoju i mnogo toga još ne radi kvalitetno (konstantno rušenje sustava zbog vizualizacije velikog seta točaka-> MS Kinect po jednom uzorkovanju prikuplja 640x480 točaka). Razvijeni algoritmi ne mogu se implementirati na nekim računalima zbog nekonzistentnosti konfiguracije računala. U ovome radu, prepoznavanje objekata je bilo zamišljeno učenjem pomoću CAD modela (*.stl*), ali ne postoji PCL podrška za takav pristup koja pouzdano radi. Za buduće projekte sa PCL-om potrebno je pričekati vreziju *2.0* u kojoj su ispravljene navedene greške.

Uporaba 3D vizijskih sustava pokazala se vrlo kvalitetnom u industrijskom okruženju. Prepoznavanje objekata pomoću 2D vizijskih sustava zahtjeva puno više pažnje i uporabe raznih filtara te razvoja raznih značajki. U slučaju uporabe oblaka točaka koji posjeduju samo informaciju o prostornim koordinatama (bez RGB vrijednosti) smislena je uporaba samo prostornih normala za opisivanje zakrivljenosti i ostalih svojstava površina što je pogodno za prepoznavanje objekata bez teksture. Od te se pretpostavke krenulo u izradu ovog rada. Nadalje, prihvat podataka bez teksture (prihvat samo prostornih koordinata) automatski je izbacio šumove poput sjena i lošeg osvjetljenja što je povoljno kako za industrijska i laboratorijska tako i za stohastička okruženja.

13. LITERATURA

[1] http://pointclouds.org/

[2] http://pointclouds.org/documentation/

[3] http://pointclouds.org/documentation/tutorials/openni_grabber.php#openni-grabber

[4] http://pointclouds.org/documentation/tutorials/planar_segmentation.php#planar-segmentation

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

[6] http://pointclouds.org/documentation/tutorials/cluster_extraction.php#cluster-extraction

[7] Radu Bogdan Rusu; Semantic 3D Object Maps for Ever yday Manipulation in Human Living Environments; Institut für Informatik der Technischen Universität München; München; 2009.

[8] Aitor Aldoma; 3D Object Recognition and 6DOF Pose Estimation; 2013.

14. PRILOG

(1) Implementacija aplikacije u C++ programskom jeziku (*Klijent*)

(2) Implementacija aplikacije u Fanuc Karel programskom jeziku (*Server*)

(1) Implementacija aplikacije u C++ programskom jeziku

```
#include <pcl/io/openni_grabber.h>
#include <pcl\io\io.h>
#include <pcl\io\pcd_io.h>
#include <pcl\visualization\interactor.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl\visualization\pcl_visualizer.h>
#include <pcl/point_types.h>
#include <pcl/visualization/interactor_style.h>
#include <pcl\filters\passthrough.h>
#include <iostream>
#include <pcl\point_cloud.h>
#include <pcl/sample_consensus/sac_model_plane.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>
#include <pcl/filters/extract_indices.h>
#include <boost/thread/thread.hpp>
#include <pcl/common/common_headers.h>
#include <pcl/features/normal_3d.h>
#include <pcl/console/parse.h>
#include <pcl/segmentation/extract_clusters.h>
#include <pcl/point_cloud.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <pcl\common\common.h>
#include <vector>
#include <ctime>
#include <pcl\common\transforms.h>
#include <Eigen\src\StlSupport\StdVector.h>
#include <pcl\surface\mls.h>
// TCP/IP Socket
#include <iostream>
#include <string>
#include <ws2tcpip.h>

#pragma comment (lib, "Ws2_32.lib")
#pragma comment (lib, "Mswsock.lib")
#pragma comment (lib, "AdvApi32.lib")

#define DEFAULT_BUFLEN 512
#define DEFAULT_PORT "6161"
#define DEFAULT_IP "192.168.123.27"

pcl::PointCloud<pcl::PointXYZ>::Ptr cloud2 (new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr                cloud_segmentirano                (new
pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_min_max (new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr                cloud_kalibracijski                (new
pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr               cloud_kalibracijski1               (new
pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_ROIx (new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_ROIy (new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_ROIz (new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_cluster (new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_centroide (new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_cil (new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_cil2 (new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::PointXYZ>::Ptr cilindar_pos (new pcl::PointCloud<pcl::PointXYZ>);
pcl::PointCloud<pcl::Normal>::Ptr normale_cilindra (new pcl::PointCloud<pcl::Normal>);
pcl::PointCloud<pcl::Normal>::Ptr normale (new pcl::PointCloud<pcl::Normal>);
pcl::PointCloud<pcl::PointNormal>::Ptr mls_points;
```

```
pcl::Grabber* bla;
pcl::Grabber* bla1;

bool saveCloud(false);
bool copyCloud(true);
bool mouseClicked(false);

int text_id = 0;

boost::mutex cloud_mutex;

Eigen::Matrix4f matrica_transformacije;

pcl::visualization::PCLVisualizer* viewer1;

char k[4];
int home_pos = 140;
int LO_DIPL_KAL = 130;
int LO_DIPL_KALB = 150;
int LO_DIPL_KALC = 170;
int LO_DIPL_KALD = 180;
int home_pos_call_x = 160;
int home_pos_call_y = 161;
int home_pos_call_z = 162;
int pos_call_x = 170;
int pos_call_y = 171;
int pos_call_z = 172;
char p1[64];
// Varijable za home poziciju
float home_pos_x;
float home_pos_y;
float home_pos_z;

float pos_x;
float pos_y;
float pos_z;


void
keyboardEventOccurred(const pcl::visualization::KeyboardEvent& event,
                                        void* nothing)
{
	if (event.getKeySym() == "space" && event.keyDown()){
		saveCloud = true;
	}
}
// Segmentacija
void segmentacija_ravnine(pcl::SACSegmentation<pcl::PointXYZ> segmentacija,
	pcl::ExtractIndices<pcl::PointXYZ> extract,
	pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_in,
	pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_out,
	pcl::ModelCoefficients::Ptr coeff,
	pcl::PointIndices::Ptr indeksi,
	double threshold,
	int max_it,
	bool setneg){

		// Zadavanje parametara segmentacije
		segmentacija.setInputCloud(cloud_in);
		segmentacija.setOptimizeCoefficients (true);
		segmentacija.setModelType(pcl::SACMODEL_PLANE);
		segmentacija.setMethodType(pcl::SAC_RANSAC);
		segmentacija.setDistanceThreshold(threshold);
		segmentacija.setMaxIterations(max_it);
```

```
            segmentacija.segment(*indeksi, *coeff);
            extract.setInputCloud(cloud_in);
            extract.setIndices(indeksi);
            extract.setNegative(setneg);
            extract.filter(*cloud_out);
}
// Clustering
vector<pcl::PointCloud<pcl::PointXYZ>::Ptr>
clustering(pcl::search::KdTree<pcl::PointXYZ>::Ptr tree,
       pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_in,
       pcl::EuclideanClusterExtraction<pcl::PointXYZ> ece,
       double cl_tol,
       int min_cl_size)
{
       tree->setInputCloud(cloud_in);

       vector<pcl::PointIndices> cl_ind;

       ece.setClusterTolerance(cl_tol);
       ece.setMinClusterSize(min_cl_size);
       ece.setSearchMethod(tree);
       ece.setInputCloud(cloud_in);
       ece.extract(cl_ind);

       pcl::PCDWriter writer;

       int z = 0;
       vector<pcl::PointCloud<pcl::PointXYZ>::Ptr> container;
       for (std::vector<pcl::PointIndices>::const_iterator it = cl_ind.begin (); it !=
cl_ind.end (); ++it)
  {
    pcl::PointCloud<pcl::PointXYZ>::Ptr                cloud_cluster                (new
pcl::PointCloud<pcl::PointXYZ>);
    for  (std::vector<int>::const_iterator  pit  =  it->indices.begin  ();  pit  !=  it-
>indices.end (); pit++)
      cloud_cluster->points.push_back (cloud_in->points[*pit]);
      cloud_cluster->width = cloud_cluster->points.size ();
      cloud_cluster->height = 1;
      cloud_cluster->is_dense = true;

      std::cout << "PointCloud representing the Cluster: " << cloud_cluster->points.size
() << " data points." << std::endl;
      std::stringstream pp;
      pp << "cloud_ROI_cluster_" << z << ".pcd";
      writer.write<pcl::PointXYZ> (pp.str (), *cloud_cluster, false);
         container.push_back(cloud_cluster);
      z++;
  }
       return(container);
       delete[] &container;
       delete[] &cl_ind;
}
```

```
//Pasthrough filter
 void passthrough_filter(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_in,
       pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_out,
```

```
        pcl::PointCloud<pcl::PointXYZ>::Ptr cloudminmax,
        pcl::PassThrough<pcl::PointXYZ> pass_x,
        pcl::PassThrough<pcl::PointXYZ> pass_y,
        pcl::PassThrough<pcl::PointXYZ> pass_z,
        pcl::PointCloud<pcl::PointXYZ>::Ptr ROIx,
        pcl::PointCloud<pcl::PointXYZ>::Ptr ROIy,
        pcl::PointCloud<pcl::PointXYZ>::Ptr ROIz){


         cout << "passthroughx" << endl;
         //pcl::PassThrough<pcl::PointXYZ> pass_x;
         pass_x.setInputCloud(cloud_in);
         cout << "passthroughx1" << endl;
         pass_x.setFilterFieldName("x");
         cout << "passthroughx2" << endl;
         pass_x.setFilterLimits(cloudminmax->points[0].x, cloudminmax->points[1].x);
         cout << "passthroughx3" << endl;
         pass_x.filter(*ROIx);


         cout << "passthroughy" << endl;
         //pcl::PassThrough<pcl::PointXYZ> pass_y;
         pass_y.setInputCloud(cloud_ROIx);
         cout << "passthroughy1" << endl;
         pass_y.setFilterFieldName("y");
         cout << "passthroughy2" << endl;
         pass_y.setFilterLimits(cloudminmax->points[0].y, cloudminmax->points[1].y);
         cout << "passthroughy3" << endl;
         pass_y.filter(*ROIy);

         cout << "passthroughz" << endl;
         //pcl::PassThrough<pcl::PointXYZ> pass_z;
         pass_z.setInputCloud(cloud_ROIy);
         cout << "passthroughz1" << endl;
         pass_z.setFilterFieldName("z");
         cout << "passthroughz2" << endl;
         pass_z.setFilterLimits(cloudminmax->points[0].z, cloudminmax->points[1].z);
         cout << "passthroughz3" << endl;
         pass_z.filter(*ROIz);

         cout << "copycloudpass" << endl;
         pcl::copyPointCloud(*cloud_ROIz, *cloud_out);

         cout << "ciscenje" << endl;
         cloud_ROIx->clear();
         cloud_ROIy->clear();
         cloud_ROIz->clear();

 }


// Mouse Event
void mouseEventOccurred (const pcl::visualization::MouseEvent &event,
                         void* viewer_void)
{
  pcl::visualization::PCLVisualizer                         viewer                         =
*static_cast<pcl::visualization::PCLVisualizer*> (viewer_void);
  if (event.getButton () == pcl::visualization::MouseEvent::RightButton &&
      event.getType () == pcl::visualization::MouseEvent::MouseButtonRelease)
  {
    std::cout << "Right mouse button released at position (" << event.getX () << ", " <<
event.getY () << ")" << std::endl;
       mouseClicked = true;
       cout << mouseClicked << endl;
```

```
    char str[512];
    sprintf (str, "text#%03d", text_id ++);
      viewer.addText("clicked here", event.getX (), event.getY (), str);
  }
}

 class SimpleOpenNIViewer
 {
   public:
     SimpleOpenNIViewer () : viewer ("PCL OpenNI Viewer") {}

     void cloud_cb_ (const pcl::PointCloud<pcl::PointXYZ>::ConstPtr &cloud)
     {
         viewer.showCloud (cloud);
          if (saveCloud){
                 cloud_mutex.lock();
                 cloud2->clear();
               pcl::copyPointCloud(*cloud, *cloud2);
               copyCloud = false;
               saveCloud = false;
               cloud_mutex.unlock();
          }

     }

        void run (pcl::Grabber* k)
     {
       k = new pcl::OpenNIGrabber();
       boost::function<void (const pcl::PointCloud<pcl::PointXYZ>::ConstPtr&)> f =
         boost::bind (&SimpleOpenNIViewer::cloud_cb_, this, _1);

          boost::signals2::connection c = k->registerCallback (f);
       k->registerCallback (f);
       k->start ();
          viewer.registerKeyboardCallback(keyboardEventOccurred);

       while (!viewer.wasStopped())
       {
         //boost::this_thread::sleep (boost::posix_time::seconds (1));
               if(copyCloud == false) break;
          }

          k->stop();
          //if (k->isRunning() == true){
          k->~Grabber();
          //delete(k);

     }
        pcl::visualization::CloudViewer viewer;
 };

 // Eigen
 namespace Eigen{
template<class Matrix>
void write_binary(const char* filename, const Matrix& matrix){
    std::ofstream out(filename,ios::out | ios::binary | ios::trunc);
    typename Matrix::Index rows=matrix.rows(), cols=matrix.cols();
    out.write((char*) (&rows), sizeof(typename Matrix::Index));
    out.write((char*) (&cols), sizeof(typename Matrix::Index));
    out.write((char*) matrix.data(), rows*cols*sizeof(typename Matrix::Scalar) );
    out.close();
}
template<class Matrix>
void read_binary(const char* filename, Matrix& matrix){
```

```
    std::ifstream in(filename,ios::in | std::ios::binary);
    typename Matrix::Index rows=0, cols=0;
    in.read((char*) (&rows),sizeof(typename Matrix::Index));
    in.read((char*) (&cols),sizeof(typename Matrix::Index));
    matrix.resize(rows, cols);
    in.read( (char *) matrix.data() , rows*cols*sizeof(typename Matrix::Scalar) );
    in.close();
}
}

 //TCP/IP Send data
 void send_data(int iResult, int ConnectSocket)
{
	// Ispis koordinata na ekranu
	printf("Koordinate: %s\n", k);

	// Slanje koordinata (buffera)
	iResult = send(ConnectSocket, k, (int)strlen(k), 0);
	if (iResult == SOCKET_ERROR) {
		printf("Greška kod slanja: %d\n", WSAGetLastError());
		closesocket(ConnectSocket);
		WSACleanup();
	}

	printf("Poslano bajtova: %ld\n", iResult);

}


 int main ()
 {
	 k[0] = 0;
	 p1[0] = 0;
	 int strng;
	 viewer1 = new pcl::visualization::PCLVisualizer("3D Viewer");
	 HWND hWnd = (HWND)viewer1->getRenderWindow()->GetGenericWindowId();
	 delete viewer1;
	 DestroyWindow(hWnd);
	 bool nastavak;
	 if (pcl::io::loadPCDFile<pcl::PointXYZ> ("Cloud_MIN_MAX.pcd", *cloud_min_max) ==
-1) //* load the file
	 {
		PCL_ERROR ("Couldn't read file Cloud_MIN_MAX.pcd \n");
		//return (-1);
		nastavak = true;
	 }
	 else
	 {
		nastavak = false;
		std::cout << "Loaded "
				<< cloud_min_max->width * cloud_min_max->height
				<< " data points from test_pcd.pcd with the following
fields: "
			<< std::endl;
		Eigen::read_binary("p2w.dat", matrica_transformacije);
		cout << "Matrica transformacije za cloud je:" << endl;
		cout << matrica_transformacije << endl;
	 }

	 // MIsc op
	 cloud_min_max->points[0].y = cloud_min_max->points[0].y + 0.1;
	 cloud_min_max->points[1].y = cloud_min_max->points[1].y - 0.1;

	 //TCP/IP Socket inicijalizacija
```

```
	WSADATA wsaData;
	SOCKET ConnectSocket = INVALID_SOCKET;
	struct addrinfo *result = NULL, *ptr = NULL, hints;
	char *sendbuf = k;
	char recvbuf[DEFAULT_BUFLEN];
	int iResult;
	int recvbuflen = DEFAULT_BUFLEN;

	iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
	if (iResult != 0) {
		cout << "Greska kod inicijalizacije WinSocka. Greska: " << iResult << endl;
		return 1;
	}

	ZeroMemory( &hints, sizeof(hints) );
	hints.ai_family = AF_UNSPEC;
	hints.ai_socktype = SOCK_STREAM;
	hints.ai_protocol = IPPROTO_TCP;

	// definiranje IP adrese i porta
	iResult = getaddrinfo(DEFAULT_IP, DEFAULT_PORT, &hints, &result);
	if ( iResult != 0 ) {
		cout << "Greska kod IP adrese ili porta. Greska: " << iResult << endl;
		WSACleanup();
		return 1;
	}

	// spajanje na zadanu IP adresu
	for(ptr=result; ptr != NULL ;ptr=ptr->ai_next) {

		// kreiranje socketa za konekciju
		ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype,
			ptr->ai_protocol);
		if (ConnectSocket == INVALID_SOCKET) {
			cout << "Greska kod kreiranja socketa. Greska: %ld\n" << WSAGetLastError() << endl;
			WSACleanup();
			return 1;
		}

		// uspostavljanje veze
		iResult = connect( ConnectSocket, ptr->ai_addr,
			(int)ptr->ai_addrlen);
		if (iResult == SOCKET_ERROR) {
			closesocket(ConnectSocket);
			ConnectSocket = INVALID_SOCKET;
			continue;
		}
	break;
	}

	freeaddrinfo(result);

	if (ConnectSocket == INVALID_SOCKET) {
		cout << "Nemoguce je ostvariti vezu sa serverom!" << endl;
		WSACleanup();
		return 1;
	}

	iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
		if ( iResult > 0 ){
			printf("Primljeno bajtova: %d\n", iResult);
			strng = std::stoi(recvbuf);;
```

```
		}
		else if ( iResult == 0 ){
			printf("Zatvaranje veze\n");
		}
		else{
			printf("Greska kod primanja: %d\n", WSAGetLastError());
		}

	//cout << "String je: " << strng << endl;

		if (strng = 100){
			cout << "Ocito smo spojeni!!!" << endl;
			_itoa_s(home_pos, p1, 10);
			strcat(k, p1);
			send_data(iResult, ConnectSocket);
			k[0] = 0;
			cout << "Otisao sam u HOME!!!" << endl;
			iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
			if ( iResult > 0 ){
				printf("Primljeno bajtova: %d\n", iResult);
				strng = std::stoi(recvbuf);
			}
			else if ( iResult == 0 ){
				printf("Zatvaranje veze\n");
			}
			else{
				printf("Greska kod primanja: %d\n", WSAGetLastError());
			}
			if (strng = 141){
				cout << "Dosao sam u HOME!!!" << endl;
			}
		}
		// Traženje pozicije
		// Traženje HOME pozicije

		p1[0] = 0;
		_itoa_s(home_pos_call_x, p1, 10);
		strcat(k, p1);
		send_data(iResult, ConnectSocket);
			k[0] = 0;
			cout << "Posalji mi home koord" << endl;
			iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
			if ( iResult > 0 ){
				printf("Primljeno bajtova: %d\n", iResult);
				home_pos_x = std::stof(recvbuf);
			}
			else if ( iResult == 0 ){
				printf("Zatvaranje veze\n");
			}
			else{
				printf("Greska kod primanja: %d\n", WSAGetLastError());
			}

			cout << "Home pos x je: " << home_pos_x << endl;

			Sleep(5);

			p1[0] = 0;
		_itoa_s(home_pos_call_y, p1, 10);
		strcat(k, p1);
		send_data(iResult, ConnectSocket);
			k[0] = 0;
			cout << "Posalji mi home koord" << endl;
			iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
```

```
                        if ( iResult > 0 ){
                                printf("Primljeno bajtova: %d\n", iResult);
                                home_pos_y = std::stof(recvbuf);
                        }
                        else if ( iResult == 0 ){
                                printf("Zatvaranje veze\n");
                        }
                        else{
                                printf("Greska kod primanja: %d\n", WSAGetLastError());
                        }

                        cout << "Home pos y je: " << home_pos_y << endl;

                        Sleep(5);

                        p1[0] = 0;
                _itoa_s(home_pos_call_z, p1, 10);
                strcat(k, p1);
                send_data(iResult, ConnectSocket);
                        k[0] = 0;
                        cout << "Posalji mi home koord" << endl;
                        iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
                        if ( iResult > 0 ){
                                printf("Primljeno bajtova: %d\n", iResult);
                                home_pos_z = std::stof(recvbuf);
                        }
                        else if ( iResult == 0 ){
                                printf("Zatvaranje veze\n");
                        }
                        else{
                                printf("Greska kod primanja: %d\n", WSAGetLastError());
                        }

                        cout << "Home pos z je: " << home_pos_z << endl;

                        Sleep(5);


        /* recvbuf[0] = 0;
         k[0] = 0;*/

         //...
         SimpleOpenNIViewer v;
         //bool nastavak = true;
         int c = 0;
         std::stringstream plpl;
         // Varijable za segmentaciju
         pcl::ModelCoefficients::Ptr     koeficijenti1     (new     pcl::ModelCoefficients),
koeficijenti_cilindra (new pcl::ModelCoefficients);
         pcl::PointIndices::Ptr unutarnji1 (new pcl::PointIndices);
         pcl::ExtractIndices<pcl::PointXYZ> extract1;
         pcl::SACSegmentation<pcl::PointXYZ> segmentacija1;
         // Varijable za clustering
         pcl::search::KdTree<pcl::PointXYZ>::Ptr                 drvo                 (new
pcl::search::KdTree<pcl::PointXYZ>);
         pcl::EuclideanClusterExtraction<pcl::PointXYZ> ec0;
         vector<pcl::PointCloud<pcl::PointXYZ>::Ptr> konteiner;
         std::vector<int> indeksi111;
         // Varijable za passThrough
         pcl::PassThrough<pcl::PointXYZ> pass_x;
         pcl::PassThrough<pcl::PointXYZ> pass_y;
         pcl::PassThrough<pcl::PointXYZ> pass_z;
         // Varijable za estimaciju normala
         pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> estimacija_normala;
```

```
        // Varijable za segmentaciju cilindra
        pcl::SACSegmentationFromNormals<pcl::PointXYZ, pcl::Normal> seg_cil;
        pcl::search::KdTree<pcl::PointXYZ>::Ptr              drvo_normale              (new
pcl::search::KdTree<pcl::PointXYZ>);
        pcl::search::KdTree<pcl::PointXYZ>::Ptr                drvo_mls                (new
pcl::search::KdTree<pcl::PointXYZ>);
        pcl::MovingLeastSquares<pcl::PointXYZ, pcl::PointNormal> mls;
        pcl::PointIndices::Ptr unutarnji_cil (new pcl::PointIndices);
        pcl::ExtractIndices<pcl::PointXYZ> extract_cil;
        pcl::ExtractIndices<pcl::Normal> extract_norm;
        Eigen::Vector4f centroida_cilindar;
        pcl::PointXYZ tez_cil;
        // Varijable MISC
        // Fill in the cloud data
        cloud_min_max->width    = 2;
        cloud_min_max->height   = 1;
        cloud_min_max->is_dense = false;
        //cloud_min_max->points.resize (cloud_min_max->width * cloud_min_max->height);
        Eigen::Vector4f centroida;
        std::vector<Eigen::Vector4f,            Eigen::aligned_allocator<Eigen::Vector4f>>
centroide;
        pcl::PointXYZ centroida_tocka;
        //std::vector<pcl::PointXYZ> centroide_tocke;
        //Eigen::Vector4f centroida2;
        double start;
        double stop;
        double gotovo;
        std::vector<int> indices;
        cout << "Pocetak programa" << endl;
        while(nastavak){
               int odl_VOI;
               int p;
               cin >> p;

               switch(p){
               case 1: //Potreni Viewer
                      if (!cloud2->empty()){
                             cloud2->clear();
                      }
                      cout << "Kopiraj si novi oblak!" << endl;
                      v.run(bla);
                      copyCloud = true;
                      saveCloud = false;
                      break;
               case 2: //Pogledaj cloud
                      c = c + 1;
                      plpl << "Cloud" << c;
                      cout << "Otvaram Viewer!" << endl;
                      viewer1 = new pcl::visualization::PCLVisualizer("3D Viewer");
                      viewer1->addPointCloud(cloud2, "cloud2");
                      cloud_mutex.lock();
                      while(!viewer1->wasStopped()){
                             viewer1->spin();
                      }
                      viewer1->removeAllPointClouds();
                      hWnd = (HWND)viewer1->getRenderWindow()->GetGenericWindowId();
                      delete viewer1;
                      DestroyWindow(hWnd);
                      cloud_mutex.unlock();
                      //cloud2->clear();
                      break;
               case 3: //Napravi segmentaciju, clustering, transformaciju i ROI/VOI
                      // Segmentacija
                      start = pcl::getTime();
```

```
                    segmentacija_ravnine(segmentacija1,        extract1,        cloud2,
cloud_segmentirano, koeficijenti1, unutarnji1, 0.01, 100, false);
                    stop = pcl::getTime();
                    gotovo = stop - start;
                    cout << "Segmentacija gotova u: " << gotovo << " sekundi" << endl;

pcl::removeNaNFromPointCloud(*cloud_segmentirano,*cloud_segmentirano, indices);
                    indices.clear();
                    // Clustering
                    konteiner = clustering(drvo, cloud_segmentirano, ec0, 0.02, 100);
                    // Vizualizacija segmentacije
                    viewer1 = new pcl::visualization::PCLVisualizer("3D Viewer");
                    viewer1->addPointCloud(cloud2, "Oblak_ns");
                    viewer1->addPointCloud(konteiner[0], "oblak_s");
                    viewer1-
>setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_COLOR,   1,   0,   0,
"oblak_s");
                    viewer1-
>setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_POINT_SIZE,       2,
"oblak_s");
                    cloud_mutex.lock();
                    while(!viewer1->wasStopped()){
                           viewer1->spin();
                    }
                    viewer1->removeAllPointClouds();
                    hWnd = (HWND)viewer1->getRenderWindow()->GetGenericWindowId();
                    delete viewer1;
                    DestroyWindow(hWnd);
                    cloud_mutex.unlock();
                    cout << "Jesi zadovoljan segmentacijom?" << endl;
                    cin >> odl_VOI;
                    switch(odl_VOI){
                    case 1:
                           cloud_segmentirano->clear();
                           pcl::copyPointCloud(*konteiner[0], *cloud_segmentirano);
                           break;
                    case 0:
                           konteiner.clear();
                           break;
                    }
                    break;
             case 4: //Izađi iz aplikacije
                    cout << "Exit!!" << endl;
                    nastavak = false;
                    break;
             }
       }
       if (!cloud_segmentirano->empty()){
       // Transformacija
     Eigen::Vector4f centroida;
     pcl::compute3DCentroid(*cloud_segmentirano, centroida);
     Eigen::Matrix3f kovarijanca;
     pcl::computeCovarianceMatrixNormalized(*cloud_segmentirano, centroida, kovarijanca);
     Eigen::SelfAdjointEigenSolver<Eigen::Matrix3f>              eigen_solver(kovarijanca,
Eigen::ComputeEigenvectors);
     Eigen::Matrix3f eigDx = eigen_solver.eigenvectors();
     eigDx.col(2) = eigDx.col(0).cross(eigDx.col(1));
     Eigen::Matrix4f p2w(Eigen::Matrix4f::Identity());
     p2w.block<3,3>(0,0) = eigDx.transpose();
     p2w.block<3,1>(0,3) = -1.f * (p2w.block<3,3>(0,0) * centroida.head<3>());
        pcl::PointCloud<pcl::PointXYZ> cPoints;
     pcl::transformPointCloud(*cloud_segmentirano, cPoints, p2w);
        pcl::PointXYZ tockicamin;
        pcl::PointXYZ tockicamax;
```

```
        pcl::getMinMax3D(cPoints, tockicamin, tockicamax);
        float visina_VOI;
        cout << "Kolika ce biti visina VOI?" << endl;
        cin >> visina_VOI;
        tockicamax.x = visina_VOI;
        cloud_min_max->points.push_back(tockicamin);
        cloud_min_max->points.push_back(tockicamax);
     const    Eigen::Vector3f    mean_diag    =    0.5f*(tockicamax.getVector3fMap()    +
tockicamin.getVector3fMap());
     const Eigen::Quaternionf qfinal(eigDx);
     const Eigen::Vector3f tfinal = eigDx*mean_diag + centroida.head<3>();

        int VOI;
        cout << "Jesi li zadovoljan sa ROI/VOI?" << endl;
        cin >> VOI;
        cout << "Size" << endl;
        cout << cloud_min_max->size() << endl;
        cout << "Matrica p2w:" << endl;
        cout << p2w << endl;
        switch(VOI){
        case 1:
               pcl::io::savePCDFile("Cloud_MIN_MAX.pcd", *cloud_min_max);
               cout << "Sejvano u -> Cloud_MIN_MAX.pcd" << endl;
               Eigen::write_binary("p2w.dat", p2w);
               break;
        case 0:
               cout << "Pokreni aplikaciju ponovo!!!" << endl;
               break;
        }
        }
        else
        {
               cout << "Preci cu na clustering ili kalibraciju stroja" << endl;
        }
        int odl;
        cout << "clustering ili kalibracija?" << endl;
        cin >> odl;
        Eigen::Matrix4f matrica_transformacije_inv = matrica_transformacije.inverse();
        switch(odl){
        case 1:
               p1[0] = 0;
               _itoa_s(LO_DIPL_KAL, p1, 10);
               strcat(k, p1);
               send_data(iResult, ConnectSocket);
               k[0] = 0;
               cout << "Otisao sam po cilindar!!!" << endl;
               do{
               iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
               if ( iResult > 0 ){
                      printf("Primljeno bajtova: %d\n", iResult);
                      strng = std::stoi(recvbuf);;
               }
               else if ( iResult == 0 ){
                      printf("Zatvaranje veze\n");
               }
               else{
                      printf("Greska kod primanja: %d\n", WSAGetLastError());
               }
               if (strng = 131){
                      cout << "Donio sam cilindar!!!" << endl;
               }
               }
               while (strng != 131);
```

```
p1[0] = 0;
_itoa_s(home_pos, p1, 10);
strcat(k, p1);
send_data(iResult, ConnectSocket);
k[0] = 0;
cout << "Otisao sam u HOME!!!" << endl;
do{
iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
if ( iResult > 0 ){
	printf("Primljeno bajtova: %d\n", iResult);
	strng = std::stoi(recvbuf);;
}
else if ( iResult == 0 ){
	printf("Zatvaranje veze\n");
}
else{
	printf("Greska kod primanja: %d\n", WSAGetLastError());
}
if (strng = 141){
	cout << "Dosao sam u HOME!!!!" << endl;
}
}
while (strng != 141);

cout << "Kalibracija" << endl;
cout << "Uzmi cloud" << endl;
v.run(bla);
cout << "sada sam kod copy" << endl;
copyCloud = true;
cout << "sada sam kod save" << endl;
saveCloud = false;
cout << "sada sam kod transformacije" << endl;
pcl::transformPointCloud(*cloud2, *cloud2, matrica_transformacije);
cout << "sada sam kod pass" << endl;
//passthrough_filter(cloud_kalibracijski,             cloud_kalibracijski1,
cloud_min_max, pass_x, pass_y, pass_z, cloud_ROIx, cloud_ROIy, cloud_ROIz);
//PassThrough
//pcl::PassThrough<pcl::PointXYZ> pass_x;
pass_x.setInputCloud(cloud2);
pass_x.setFilterFieldName("x");
pass_x.setFilterLimits(cloud_min_max->points[0].x,           cloud_min_max-
>points[1].x);
pass_x.filter(*cloud_ROIx);

//pcl::PassThrough<pcl::PointXYZ> pass_y;
pass_y.setInputCloud(cloud_ROIx);
pass_y.setFilterFieldName("y");
pass_y.setFilterLimits(cloud_min_max->points[0].y,           cloud_min_max-
>points[1].y);
pass_y.filter(*cloud_ROIy);

//pcl::PassThrough<pcl::PointXYZ> pass_z;
pass_z.setInputCloud(cloud_ROIy);
pass_z.setFilterFieldName("z");
pass_z.setFilterLimits(cloud_min_max->points[0].z,           cloud_min_max-
>points[1].z);
pass_z.filter(*cloud_ROIz);

cout << "sada sam kod transf" << endl;
pcl::transformPointCloud(*cloud_ROIz,               *cloud_kalibracijski1,
matrica_transformacije_inv);
cout << "sada sam kod segmentacije" << endl;
// Segmentacija ravnine
start = pcl::getTime();
```

```
                segmentacija_ravnine(segmentacija1,     extract1,     cloud_kalibracijski1,
cloud_segmentirano, koeficijenti1, unutarnji1, 0.01, 100, false);
                stop = pcl::getTime();
                gotovo = stop - start;
                cout << "Segmentacija ravnine gotova u " << gotovo << " sekundi!!!" <<
endl;

                // Segmentacija cilindra
                //          ->Estimacija normala
                start = pcl::getTime();
                estimacija_normala.setInputCloud(cloud_kalibracijski1);
                estimacija_normala.setSearchMethod(drvo_normale);
                estimacija_normala.setKSearch(50);
                estimacija_normala.compute(*normale_cilindra);
                stop = pcl::getTime();
                gotovo = stop - start;
                cout << "Estimacija normala gotova u " << gotovo << " sekundi!!!" << endl;
                //          <-Estimacija normala
                cout << "Broj normala u oblaku je: " << normale_cilindra->size() << endl;
                extract1.setInputCloud(cloud_kalibracijski1);
                extract1.setNegative(true);
                extract1.setIndices(unutarnji1);
                extract1.filter(*cloud_cil);
                cout << "velicina cloud_cil: " << cloud_cil->size() << endl;
                extract_norm.setNegative(true);
                extract_norm.setInputCloud(normale_cilindra);
                extract_norm.setIndices(unutarnji1);
                extract_norm.filter(*normale);
                cout << "vlicina normala je: " << normale->size() << endl;

                start = pcl::getTime();
                seg_cil.setOptimizeCoefficients(true);
                seg_cil.setModelType(pcl::SACMODEL_CYLINDER);
                seg_cil.setMethodType(pcl::SAC_RANSAC);
                seg_cil.setNormalDistanceWeight(0.1);
                seg_cil.setMaxIterations(1000);
                seg_cil.setDistanceThreshold(0.01);
                seg_cil.setRadiusLimits(0, 0.1);
                seg_cil.setAxis(Eigen::Vector3f(0, 0, 1));
                seg_cil.setInputCloud(cloud_cil);
                seg_cil.setInputNormals(normale);
                seg_cil.segment(*unutarnji_cil, *koeficijenti_cilindra);
                stop = pcl::getTime();
                gotovo = stop - start;
                cout << "Segmentacija cilindra gotova u " << gotovo << " sekundi!!!" <<
endl;
                cout << "Broj indeksa je: " << unutarnji_cil->indices.size() << endl;
                // Extract cilindar u cloud
                extract_cil.setInputCloud(cloud_cil);
                extract_cil.setIndices(unutarnji_cil);
                extract_cil.setNegative(false);
                extract_cil.filter(*cloud_cil2);
                cout << "Velicina cloud od cilindra je" << cloud_cil2->size() << endl;

                pcl::compute3DCentroid(*cloud_cil2, centroida_cilindar);

                tez_cil.x = centroida_cilindar.x();
                tez_cil.y = koeficijenti_cilindra->values[1];
                tez_cil.z = koeficijenti_cilindra->values[2];

                //tez_cil.z = centroida_cilindar.z();

                cilindar_pos->push_back(tez_cil);
```

```
p1[0] = 0;
_itoa_s(LO_DIPL_KALB, p1, 10);
strcat(k, p1);
send_data(iResult, ConnectSocket);
k[0] = 0;
cout << "Otisao sam natrag po cilindar!!!" << endl;
do{
      iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
      if ( iResult > 0 ){
              printf("Primljeno bajtova: %d\n", iResult);
             strng = std::stoi(recvbuf);;
      }
      else if ( iResult == 0 ){
             printf("Zatvaranje veze\n");
      }
      else{
             printf("Greska kod primanja: %d\n", WSAGetLastError());
      }
      if (strng = 151){
             cout << "Donio sam cilindar natrag!!!" << endl;
             }
      }
while (strng != 151);

//Odlazak u mjesto za trazenje pozicije

p1[0] = 0;
_itoa_s(LO_DIPL_KALC, p1, 10);
strcat(k, p1);
send_data(iResult, ConnectSocket);
k[0] = 0;
cout << "Otisao sam na kalibracijsko mjesto" << endl;
do{
      iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
      if ( iResult > 0 ){
              printf("Primljeno bajtova: %d\n", iResult);
             strng = std::stoi(recvbuf);;
      }
      else if ( iResult == 0 ){
             printf("Zatvaranje veze\n");
      }
      else{
             printf("Greska kod primanja: %d\n", WSAGetLastError());
      }
      if (strng = 171){
             cout << "Dosao sam!!!" << endl;
             }
      }
while (strng != 171);

// Trazenje pozicije
p1[0] = 0;
_itoa_s(home_pos_call_x, p1, 10);
strcat(k, p1);
send_data(iResult, ConnectSocket);
       k[0] = 0;
       cout << "Posalji mi koord x" << endl;
       iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
       if ( iResult > 0 ){
              printf("Primljeno bajtova: %d\n", iResult);
              pos_x = std::stof(recvbuf);
       }
       else if ( iResult == 0 ){
              printf("Zatvaranje veze\n");
```

```
		}
		else{
			printf("Greska kod primanja: %d\n", WSAGetLastError());
		}

		cout << "pos x je: " << home_pos_x << endl;

		Sleep(5);

		p1[0] = 0;
	_itoa_s(home_pos_call_y, p1, 10);
	strcat(k, p1);
	send_data(iResult, ConnectSocket);
		k[0] = 0;
		cout << "Posalji mi home koord y" << endl;
		iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
		if ( iResult > 0 ){
			printf("Primljeno bajtova: %d\n", iResult);
			pos_y = std::stof(recvbuf);
		}
		else if ( iResult == 0 ){
			printf("Zatvaranje veze\n");
		}
		else{
			printf("Greska kod primanja: %d\n", WSAGetLastError());
		}

		cout << "pos y je: " << home_pos_y << endl;

		Sleep(5);

		p1[0] = 0;
	_itoa_s(home_pos_call_z, p1, 10);
	strcat(k, p1);
	send_data(iResult, ConnectSocket);
		k[0] = 0;
		cout << "Posalji mi home koord" << endl;
		iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
		if ( iResult > 0 ){
			printf("Primljeno bajtova: %d\n", iResult);
			pos_z = std::stof(recvbuf);
		}
		else if ( iResult == 0 ){
			printf("Zatvaranje veze\n");
		}
		else{
			printf("Greska kod primanja: %d\n", WSAGetLastError());
		}

		cout << "pos z je: " << home_pos_z << endl;

		Sleep(5);

		//Vracanje s mjesta

	p1[0] = 0;
	_itoa_s(LO_DIPL_KALD, p1, 10);
	strcat(k, p1);
	send_data(iResult, ConnectSocket);
	k[0] = 0;
	cout << "Vracam se s kalibracijskog mjesta" << endl;
	do{
	      iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
	      if ( iResult > 0 ){
```

```
                        printf("Primljeno bajtova: %d\n", iResult);
                        strng = std::stoi(recvbuf);;
                }
                else if ( iResult == 0 ){
                        printf("Zatvaranje veze\n");
                }
                else{
                        printf("Greska kod primanja: %d\n", WSAGetLastError());
                }
                if (strng = 181){
                        cout << "Dosao sam!!!" << endl;
                        }
                }
            while (strng != 181);

            p1[0] = 0;
            _itoa_s(home_pos, p1, 10);
            strcat(k, p1);
            send_data(iResult, ConnectSocket);
            k[0] = 0;
            cout << "Idem doma" << endl;
            do{
                iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
                if ( iResult > 0 ){
                        printf("Primljeno bajtova: %d\n", iResult);
                        strng = std::stoi(recvbuf);;
                }
                else if ( iResult == 0 ){
                        printf("Zatvaranje veze\n");
                }
                else{
                        printf("Greska kod primanja: %d\n", WSAGetLastError());
                }
                if (strng = 141){
                        cout << "Dosao sam!!!" << endl;
                        }
                }
            while (strng != 141);


            break;
        case 2:
            cout << "Clustering" << endl;
            cout << "Uzmi cloud" << endl;
            v.run(bla);
            cout << "sada sam kod copy" << endl;
            copyCloud = true;
            cout << "sada sam kod save" << endl;
            saveCloud = false;
            cout << "sada sam kod transformacije" << endl;
            pcl::transformPointCloud(*cloud2, *cloud2, matrica_transformacije);
            cout << "sada sam kod pass" << endl;
            //passthrough_filter(cloud_kalibracijski,             cloud_kalibracijski1,
cloud_min_max, pass_x, pass_y, pass_z, cloud_ROIx, cloud_ROIy, cloud_ROIz);
            //PassThrough
            //pcl::PassThrough<pcl::PointXYZ> pass_x;
            pass_x.setInputCloud(cloud2);
            pass_x.setFilterFieldName("x");
            pass_x.setFilterLimits(cloud_min_max->points[0].x,          cloud_min_max-
>points[1].x);
            pass_x.filter(*cloud_ROIx);

            //pcl::PassThrough<pcl::PointXYZ> pass_y;
            pass_y.setInputCloud(cloud_ROIx);
```

```
		pass_y.setFilterFieldName("y");
		pass_y.setFilterLimits(cloud_min_max->points[0].y, cloud_min_max->points[1].y);
		pass_y.filter(*cloud_ROIy);

		//pcl::PassThrough<pcl::PointXYZ> pass_z;
		pass_z.setInputCloud(cloud_ROIy);
		pass_z.setFilterFieldName("z");
		pass_z.setFilterLimits(cloud_min_max->points[0].z, cloud_min_max->points[1].z);
		pass_z.filter(*cloud_ROIz);

		cout << "sada sam kod transf" << endl;
		pcl::transformPointCloud(*cloud_ROIz, *cloud_kalibracijski1, matrica_transformacije_inv);
		start = pcl::getTime();
		segmentacija_ravnine(segmentacija1, extract1, cloud_kalibracijski1, cloud_cluster, koeficijenti1, unutarnji1, 0.01, 100, true);
		stop = pcl::getTime();
		gotovo = stop - start;
		cout << "Segmentacija gotova u " << gotovo << " sekundi!!!" << endl;
		pcl::removeNaNFromPointCloud(*cloud_cluster, *cloud_cluster, indeksi111);
		konteiner = clustering(drvo, cloud_cluster, ec0, 0.02, 500);
		for (int i = 0; i < konteiner.size(); i++){
			pcl::compute3DCentroid(*konteiner[i], centroida);
			centroide.push_back(centroida);
			centroida_tocka.x = centroida.x();
			centroida_tocka.y = centroida.y();
			centroida_tocka.z = centroida.z();
			cloud_centroide->push_back(centroida_tocka);
		}
	}

	pcl::PointXYZ centar;
	centar.x = 0.0;
	centar.y = 0.0;
	centar.z = 0.0;
	pcl::visualization::PCLVisualizer viewer2;
	std::stringstream slsls;
	viewer2.addCoordinateSystem(0.1);
	switch(odl){
	case 1:
		/*viewer2.addPointCloud(cloud_kalibracijski1, "situacija_kalibracijska");*/
		viewer2.addPointCloud(cloud_segmentirano, "situacija_segmentirana");

	viewer2.setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_COLOR, 1, 0, 0, "situacija_segmentirana");

	viewer2.setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 2, "situacija_segmentirana");
		viewer2.addPointCloud(cloud_cil2, "cilindar");

	viewer2.setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_COLOR, 0, 1, 0, "cilindar");

	viewer2.setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 5, "cilindar");
		//viewer2.addCylinder(*koeficijenti_cilindra, "cilindar");
		viewer2.addPointCloud(cilindar_pos, "cilindar_pos");

	viewer2.setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_COLOR, 0, 0, 1, "cilindar_pos");
```

```
      viewer2.setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_POINT_
SIZE, 5, "cilindar_pos");
            break;
       case 2:
            viewer2.addPointCloud(cloud_kalibracijski1, "situacija kalibracijska");
            viewer2.addPointCloud(cloud_cluster, "klaster");

      viewer2.setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_COLOR,
1, 0, 0, "klaster");

      viewer2.setPointCloudRenderingProperties(pcl::visualization::PCL_VISUALIZER_POINT_
SIZE, 2, "klaster");
            for (int j = 0; j < cloud_centroide->size(); j++){
                  slsls << "Linija" << j;
                  viewer2.addLine(centar,  cloud_centroide->points[j],  0.0,  1.0,  0.0,
slsls.str());

      viewer2.setShapeRenderingProperties(pcl::visualization::PCL_VISUALIZER_LINE_WIDTH,
3, slsls.str());
            }
            break;
       }


 while (!viewer2.wasStopped()){
               viewer2.spin();
          }
       viewer2.removeAllPointClouds();

   return 0;
 }
```

(2) Implementacija aplikacije u Fanuc Karel programskom jeziku (*Server*)

```
PROGRAM LO_DIPL_MAIN
%NOLOCKGROUP
%NOPAUSE = ERROR + COMMAND + TPENABLE

VAR
i,n,tmp_int,STATUS:INTEGER
entry : INTEGER
file_var:FILE
vox_str:STRING[128]
n_bytes,STAT,prog_index:INTEGER
posi_cur:POSITION
posi_cur1:XYZWPR
posx, posy, posz:REAL
posxs, posys, poszs: STRING[128]
FINISHED:BOOLEAN
TCP : FILE

----------VANJSKE RUTINE----------
ROUTINE OPEN_FILE_(FILE_ : FILE; TAG_ : STRING) FROM LIB_FILE
ROUTINE CLOSE_FILE_(FILE_ : FILE; TAG_ : STRING) FROM LIB_FILE
ROUTINE WRITE_(STRING_ : STRING) FROM LIB_FILE
ROUTINE HANDSHAKING_(ID_ : STRING; TIP_: STRING) FROM LIB_FILE
----------

ROUTINE KAL1
BEGIN
        REPEAT
        READ TCP(vox_str::3)
        IF UNINIT(vox_str) THEN
        vox_str=''
        ENDIF
        IF (vox_str='130') THEN
                CALL_PROG('LO_DIPL_KAL', prog_index);
                WRITE TCP('131',CR);
                --FINISHED = TRUE;
        ENDIF
        IF (vox_str ='140') THEN
                CALL_PROG('LO_DIPL_HOME', prog_index)
                WRITE TCP('141', CR)
                --FINISHED = TRUE;
        ENDIF
        IF (vox_str ='150') THEN
                CALL_PROG('LO_DIPL_KALB', prog_index)
                WRITE TCP('151', CR)
                --FINISHED = TRUE;
        ENDIF
        IF (vox_str ='160') THEN
                posi_cur = CURPOS(0,0);
                posi_cur1 = posi_cur;
                posx=posi_cur1.x;
                CNV_REAL_STR(posx, 4, 5, posxs);
                WRITE TCP(posxs,CR);
                --FINISHED = TRUE;
        ENDIF
        IF (vox_str ='161') THEN
                posi_cur = CURPOS(0,0);
                posi_cur1 = posi_cur;
                posy=posi_cur1.y;
                CNV_REAL_STR(posy, 4, 5, posys);
                WRITE TCP(posys,CR);
                --FINISHED = TRUE;
        ENDIF
        IF (vox_str ='162') THEN
                posi_cur = CURPOS(0,0);
                posi_cur1 = posi_cur;
                posz=posi_cur1.z;
                CNV_REAL_STR(posz, 4, 5, poszs);
                WRITE TCP(poszs,CR);
                --FINISHED = TRUE;
        ENDIF
        IF (vox_str ='170') THEN
                CALL_PROG('LO_DIPL_KALC', prog_index)
                WRITE TCP('171', CR)
                --FINISHED = TRUE;
        ENDIF
```

```
        IF (vox_str ='180') THEN
                CALL_PROG('LO_DIPL_KALD', prog_index)
                WRITE TCP('181', CR)
                --FINISHED = TRUE;
        ENDIF

        UNTIL (FINISHED=TRUE)

END KAL1

BEGIN -- Main Prog

    --WRITE TPDISPLAY('Pokrenuo sam program... i inicijalizirati cu neke varijable',CR);
--  INICIJALIZACIJA TCP IP VEZE   ROBOT JE SERVER
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$OPER',0,STATUS);
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$STATE',0,STATUS) ; DELAY 20
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$COMMENT','MARKO',STATUS) ;
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$PROTOCOL','SM',STATUS) ;
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$REPERRS','FALSE',STATUS) ;
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$TIMEOUT',9999,STATUS) ;
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$PWRD_TIMEOUT',0,STATUS) ;
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$SERVER_PORT',6161,STATUS) ;   --port
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$STRT_PATH','192.168.123.27',STATUS) ; ---IP OD M3   . 192.168.123.27
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$STRT_REMOTE','192.168.123.27',STATUS) ;  --IP OD M3   . 192.168.123.27
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$PATH','192.168.123.27',STATUS) ;  --IP OD M3   . 192.168.123.27
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$remote','192.168.123.27',STATUS) ; --IP OD M3   . 192.168.123.27
    DELAY 10 ;
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$OPER',3,STATUS);
    DELAY 10;
    SET_VAR(entry,'*SYSTEM*','$HOSTS_CFG[8].$STATE',3,STATUS) ;
    DELAY 10 ;
        reconn_::    --
    DELAY 100;  CLOSE_FILE_(TCP,'S8:') ; DELAY 100;
    OPEN_FILE_(TCP,'S8:');

    DELAY(5000);

   WRITE TCP('100',CR);

   DELAY(2000);

   FINISHED = FALSE;

   KAL1;

   CLOSE_FILE_(TCP,'S8:');

END LO_DIPL_MAIN
```