Postprocesiranje rezultata 3D mapiranja pomoću ICP algoritma

Korić, Karmen

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:840901

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-27**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1244

POSTPROCESIRANJE REZULTATA 3D MAPIRANJA POMOĆU ICP ALGORITMA

Karmen Korić

Zagreb, siječanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1244

POSTPROCESIRANJE REZULTATA 3D MAPIRANJA POMOĆU ICP ALGORITMA

Karmen Korić

Zagreb, siječanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 2. listopada 2023.

ZAVRŠNI ZADATAK br. 1244

Pristupnica:	**Karmen Korić (0036533670)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Stjepan Bogdan

Zadatak: **Postprocesiranje rezultata 3D mapiranja pomoću ICP algoritma**

Opis zadatka:

U sklopu ovog zadatka potrebno je proučiti i primijeniti ICP algoritam za postprocesiranje rezultata 3D mapiranja građevinskog objekta. Objekt je mapiran korištenjem 3D LIDAR-a ugrađenog na bespilotnu letjelicu, a 3D oblak točaka generiran je pomoću SLAM paketa Cartographer. U svrhu povećanja kvalitete 3D oblaka točaka potrebno je razviti metodu za postprocesiranje rezultata SLAM-a koja će na temelju estimata trajktorije letjelice i podataka prikupljenih LIDAR-om izraditi kvalitetniji oblak točaka u odnosu na oblak točaka dobiven Cartographerom. Kvaliteta oblaka točaka procijenit će se mjerenjem disperzije točaka oko konstrukcijskih elemenata poznatih dimenzija poput stupova.

Rok za predaju rada: 26. siječnja 2024.

Uvod 2
1. 3D mapiranje i LiDAR senzor 3
2. Pregled izvornih podataka 5
2.1. Bag s LIDAR podacima 5
2.2. Bag s podatcima o putanji 6
2.3. URDF datoteka 7
3. Prikupljanje podataka i pretproscesiranje 8
3.1. Ekstrakcija LiDAR podataka 8
3.2. Ekstrakcija informacija o transformaciji 11
4. Transformacija LiDAR podataka 14
4.1. Čitanje podataka 14
4.2. Transformacija LiDAR podataka 16
5. ICP algoritam 22
5.1. ICP algoritam 22
5.2. VoxelGrid filter 26
6. Rezultati 28
Zaključak 31
Literatura 32
Sažetak 33
Summary 34
Skraćenice 35

Uvod

Pojava tehnika trodimenzionalnog (3D) mapiranja revolucionirala je razna područja, od robotike do praćenja okoliša, pružajući bogate i detaljne prostorne informacije. Posljednjih se godina tehnologija detekcije i dometa svjetla (LiDAR) pojavila kao moćan alat za snimanje 3D podataka visoke razlučivosti, omogućujući precizno mapiranje složenih okruženja. Ovaj se rad bavi postprocesiranjem rezultata 3D mapiranja, korištenjem algoritma Iterativne najbliže točke (ICP) za pročišćavanje podataka dobivenih od 3D LIDAR-a ugrađenog na bespilotnu letjelicu.

Primarni cilj ovog istraživanja je razviti metodologiju naknadne obrade za rezultate 3D mapiranja. Pristup uključuje ekstrakciju LiDAR podataka iz rosbag-ova, transformaciju pomoću informacija o putanji i primjenu ICP algoritma za usavršavanje. Krajnji cilj je generirati točan prikaz građevinskog objekta.

Ovaj se rad usredotočuje na implementaciju ICP algoritma kao ključne komponente u obradi podataka. Rad se posebno bavi integracijom LiDAR podataka s informacijama o putanji izdvojenim iz rosbag-ova i uključuje dodatne složenosti uvedene korištenjem Unified Robot Description Format (URDF) datoteka. Primjena ove metodologije rezultira sveobuhvatnim prikazom oblaka točaka građevinskog objekta.

U sljedećim odjeljcima ovog rada zadubit ćemo se u metodologiju, detaljno opisujući korake uključene u ekstrakciju LiDAR podataka, transformaciju putanje, modeliranje temeljeno na URDF-u i primjenu ICP algoritma. Osim toga, predstavit ćemo i analizirati rezultate dobivene ovim procesom, ističući učinkovitost predloženog pristupa u postizanju točnih i konsolidiranih rezultata 3D mapiranja.

Dok krećemo u ovo istraživanje, bitno je prepoznati značaj usavršavanja rezultata 3D mapiranja u dinamičkim scenarijima i potencijalni utjecaj na polja kao što su robotika, znanost o okolišu i šire.

1. 3D mapiranje i LiDAR senzor

Tehnologija 3D mapiranja je metoda koja se koristi za konstruiranje trodimenzionalnih prikaza objekata, površina i okoliša korištenjem tehnika prostornog mapiranja. Ovaj inovativni proces obuhvaća prikupljanje i obradu slika i podataka pomoću različitih senzora, softvera i metodologija. Ishod je precizan i realističan 3D model određenog područja, primjenjiv u raznim industrijama.

3D mapiranje nudi nekoliko ključnih prednosti u odnosu na 2D tehnologiju, pruža precizniji prikaz u usporedbi s 2D tehnologijom, omogućuje realističnije i intuitivnije predstavljanje objekata i okruženja. 3D modeli nude opsežnije informacije od 2D tehnologije, bilježe dodatne podatke i više detalja.

LiDAR je tehnologija slična radaru, koja koristi laser umjesto radio valova. Uključuje emitiranje laserskog svjetla iz izvora (odašiljača) i hvatanje reflektiranog svjetla od objekata pomoću prijemnika sustava. Vrijeme leta (TOF) se zatim koristi za izradu karte udaljenosti objekata.

Sl. 1.1 Ouster OS0-128

Senzor trenutno bilježi trodimenzionalne podatke, nazvane oblak točaka (point cloud), koji predstavlja "okvir" u kontekstu 3D LiDAR-a.

LiDAR i kamere imaju različite svrhe u otkrivanju objekata. Kamere snimaju boje, ali nemaju mogućnost mjerenja udaljenosti. Na slikama fotoaparata veliki objekt koji se nalazi daleko izgleda iste veličine kao mali objekt blizu fotoaparata. LiDAR, s druge strane, bilježi prostorne podatke, uključujući podatke o udaljenosti, obliku, volumenu i brzini objekata. To omogućuje sveobuhvatnije razumijevanje fizičkih karakteristika promatranih objekata. Standardni LiDAR sustav obično dostavlja podatke brzinom od oko 20 okvira u sekundi.

Oblaci točaka su trodimenzionalni prikazi prostornih podataka koji se sastoje od brojnih točaka u koordinatnom sustavu. Svaka točka u oblaku točaka nosi informacije o svojoj poziciji u XYZ prostoru, često izvedene iz različitih senzora poput LiDAR-a, fotogrametrije ili 3D skenera. Ove točke zajednički tvore detaljan i točan prikaz geometrije površine fizičkih objekata ili okruženja. Oblaci točaka nalaze široku primjenu u područjima kao što su geodezija, geoprostorno mapiranje, autonomna vozila, virtualna stvarnost i računalni vid. Njihova sposobnost vjernog hvatanja zamršenosti površina stvarnog svijeta čini oblake točaka vrijednima za zadatke kao što su prepoznavanje objekata, rekonstrukcija i analiza. Učinkoviti alati za obradu i vizualizaciju omogućuju istraživačima, inženjerima i profesionalcima da izvuku značajne uvide iz ogromne količine prostornih podataka kodiranih unutar oblaka točaka, pridonoseći napretku u različitim poljima.

2. Pregled izvornih podataka

Iskoristili smo tri primarna izvora podataka za ovaj rad.

LiDAR rosbag: Izdvojeni LiDAR podaci iz rosbag-a, koji obuhvaćaju neobrađene prostorne informacije snimljene tijekom faze prikupljanja podataka.

Rosbag s podatcima o putanji: Korištene informacije o putanji izvučene iz rosbag-a, služe kao ključna komponenta za točno usklađivanje LiDAR podataka u fazi postprocesiranja.

URDF datoteka: Uključena datoteka Unified Robot Description Format (URDF), pružajući detaljan opis bespilotne letjelice koja nosi LiDAR senzor.

2.1. Bag s LIDAR podacima

LiDAR podaci, pohranjeni u rosbag datoteci "2022-04-13-12-19-00.bag", pregledani su pomoću naredbe "rosbag info". Dobivene informacije uključuju trajanje snimljenih podataka tj. 1037 sekunda, veličina datoteke je 30.5 GB te sadrži 271594 poruka.

Podaci unutar rosbag datoteke uključuju različite vrste poruka, kao što su quaternion, vector3Stamped, odometry, imu, navSatFix i PointCloud2. Poruke su raspoređene po različitim temama, a svaka predstavlja određenu vrstu podataka.

Teme povezane s LiDAR senzorom uključuju '/os_cloud_node/imu' 103693 poruka i '/os_cloud_node/points' 10371 poruka što ukazuje na podatke imu i oblaka točaka.

Proces ekstrakcije LiDAR podataka uključuje ekstrakciju relevantnih informacija iz ovih tema, posebno teme '/os_cloud_node/points'.

Vrsta poruke 'sensor_msgs/PointCloud2' sastoji se od informacija zaglavlja, dimenzija (visina i širina), polja točaka koja opisuju strukturu svake točke, podatkovnog niza koji sadrži stvarne podatke oblaka točaka i dodatnih parametara.

Za izdvajanje LiDAR podataka potrebno je obraditi poruke iz teme '/os_cloud_node/points', a niz 'podataka' unutar svake poruke može se upotrijebiti za rekonstrukciju informacija oblaka točaka.

Nakon postupka ekstrakcije LiDAR podataka, generiran je značajan broj datoteka oblaka točaka u PLY formatu (10371 datoteka), pružajući detaljan prikaz snimljenih 3D prostornih informacija. Ove će se datoteke dalje koristiti u sljedećim koracima metodologije za naknadnu obradu pomoću ICP algoritma.

2.2. Bag s podatcima o putanji

Kako bi se osiguralo točno poravnanje i prostorna koherencija, podaci o putanji prošli su transformaciju za usklađivanje koordinatnog sustava LiDAR podataka te integraciju u naknadnu obradu.

Podatci o putanji dobiveni iz rosbag datoteke "estimated_trajectory.bag" su 14574 poruka snimljenih u 1035 sekundi. Podaci unutar rosbag datoteke uključuju dvije vrste poruka, 'geometry_msgs/TransformStamped' te tf2_msgs/TFMessage'. Tema '/tf' sadrži 7287 poruka tipa 'tf2_msgs/TFMessage', a tema 'trajectory_0' sadrži 7287 poruka tipa 'geometry_msgs/TransformStamped.'

Svaka 'TransformStamped' poruka uključuje standardne informacije zaglavlja, uključujući redni broj, vremensku oznaku i ID okvira, podatke o transformaciji, uključujući translaciju i rotaciju.

Translacija: Vektor u trodimenzionalnom prostoru (x, y, z).

Rotacija: Kvaternion koji predstavlja orijentaciju u prostoru (x, y, z, w).

Podaci o putanji predstavljaju transformacije između različitih okvira tijekom vremena. Tema '/tf' pruža tok transformacija, a tema 'trajectory_0' specifično opisuje transformacije tipa 'geometry_msgs/TransformStamped'.

Za korištenje ovih podataka, transformacije se mogu primijeniti na podatke LiDAR oblaka točaka, usklađujući ih s informacijama o procijenjenoj putanji. Transformacije s vremenskim oznakama pružaju dinamičko mapiranje kretanja LiDAR senzora, omogućujući točno prostorno poravnanje tijekom postprocesiranja.

U sljedećim koracima metodologije, ove transformacije s vremenskim oznakama koristit će se za integraciju informacija o putanji u podatke LiDAR, omogućavajući točniji i koherentniji prikaz 3D okruženja u globalnom koordinatnom sustavu.

2.3. URDF datoteka

LiDAR podaci su prošli još transformaciju za usklađivanje koordinatnog sustava letjelice i samog LiDAR senzora.

URDF definira robota pod nazivom "danieli-drone" i definira materijale te RGBA vrijednosti.

Robot se sastoji od nekoliko karika i zglobova, od kojih je svaki definiran posebnim vizualnim i geometrijskim svojstvima. Dvije značajne veze su "red/imu_link" i "os_sensor". Ove su veze povezane s "red/base_link" preko fiksnih spojeva ("red/imu_link_joint" i "red/ouster_link_joint"), uspostavljajući hijerarhijsku strukturu robota.

Ova URDF datoteka pruža detaljan opis letjelice koji nosi LiDAR senzor, specificirajući njegove vizualne komponente, geometriju i zglobne veze. Tijekom procesa 3D mapiranja, ovo modeliranje temeljeno na URDF-u olakšava točan prikaz pomažući u usklađivanju LiDAR podataka s procijenjenom putanjom.

Ova sveobuhvatna metodologija osigurava sustavan i točan pristup naknadnoj obradi rezultata 3D mapiranja.

3. Prikupljanje podataka i pretproscesiranje

3.1. Ekstrakcija LiDAR podataka

Ekstrakcija LiDAR podataka iz rosbag-a slijedi sustavan proces koji koristi prilagođeni C++ program.

ROS (Robot Operating System) je uslužni program koji izvlači PointCloud2 poruke iz rosbag datoteke, posebno poruke iz teme "/os_cloud_node/points". Glavna funkcija ovog dijela koda je pretvaranje i spremanje podataka PointCloud2 u PLY (Polygon File Format) datoteke.

Program koristi biblioteku ROS rosbag za otvaranje ulazne datoteke rosbag u načinu rada samo za čitanje. Implementirano je rukovanje iznimkama radi rješavanja mogućih pogrešaka u slučaju neuspjelog otvaranja rosbag-a.

```
rosbag::Bag input_bag;

try {
      input_bag.open(input_bag_path, rosbag::bagmode::Read);
} catch (rosbag::BagException& e) {
      ROS_ERROR("Failed to open the input bag file: %s", e.what());
      return 1;
}
```

U glavnoj funkciji kod iterira kroz poruke pomoću TypeQueryja kako bi filtrirao samo poruke "sensor_msgs/PointCloud2". Za svaku poruku PointCloud2 u temi "/os_cloud_node/points", ekstrahira poruku i konstruira putanju izlazne datoteke na temelju vremenske oznake poruke. Zatim se poziva funkcija savePointCloudToPLY za pretvaranje i spremanje poruke PointCloud2 u PLY datoteku.

```
rosbag::TypeQuery type_query("sensor_msgs/PointCloud2");

for (rosbag::MessageInstance const& m : rosbag::View(input_bag,
rosbag::TypeQuery(type_query))) {

      if (m.getTopic() == "/os_cloud_node/points") {

            sensor_msgs::PointCloud2::ConstPtr pc_msg =
      m.instantiate<sensor_msgs::PointCloud2>();

            if (pc_msg != nullptr) {

                  std::string output_file_path = output_directory +
            "/pointcloud_" + std::to_string(m.getTime().toSec()) +
            ".ply";
                  savePointCloudToPLY(pc_msg, output_file_path);

            }
      }
}
```

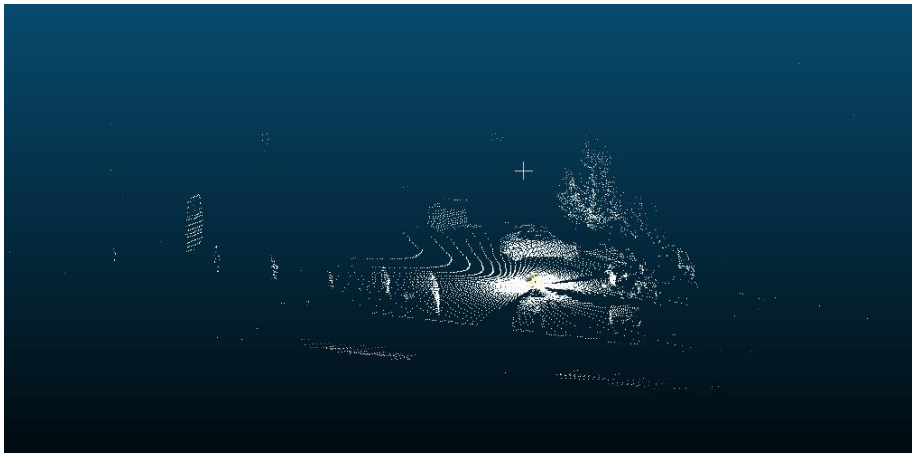
Poruka PointCloud2 pretvara se u PCL (Point Cloud Library) strukturu podataka oblaka točaka (pcl::PointCloudpcl::PointXYZI). Konvertirani oblak točaka se zatim sprema u datoteku PLY (Polygon File Format).

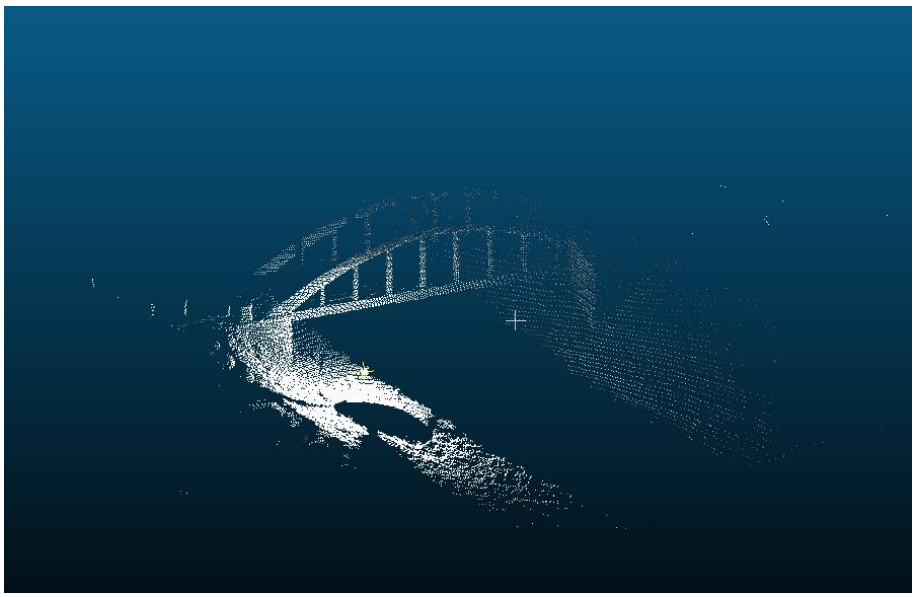
Spremljena PLY datoteka sadrži podatke o oblaku točaka, uključujući 3D koordinate i vrijednosti intenziteta (PointXYZI), što je čini prikladnom za naknadnu analizu ili vizualizaciju pomoću različitih alata za obradu oblaka točaka. ‘pcl::fromROSMsg’ olakšava konverziju ROS PointCloud2 poruka u format kompatibilan s PCL bibliotekom, dok ‘pcl::io::savePLYFileASCII’ omogućuje pohranjivanje obrađenih podataka oblaka točaka u široko korištenom PLY formatu datoteke. Ove funkcije spajaju ROS i PCL, omogućujući besprijekornu integraciju podataka oblaka točaka u PCL za daljnju analizu i vizualizaciju.

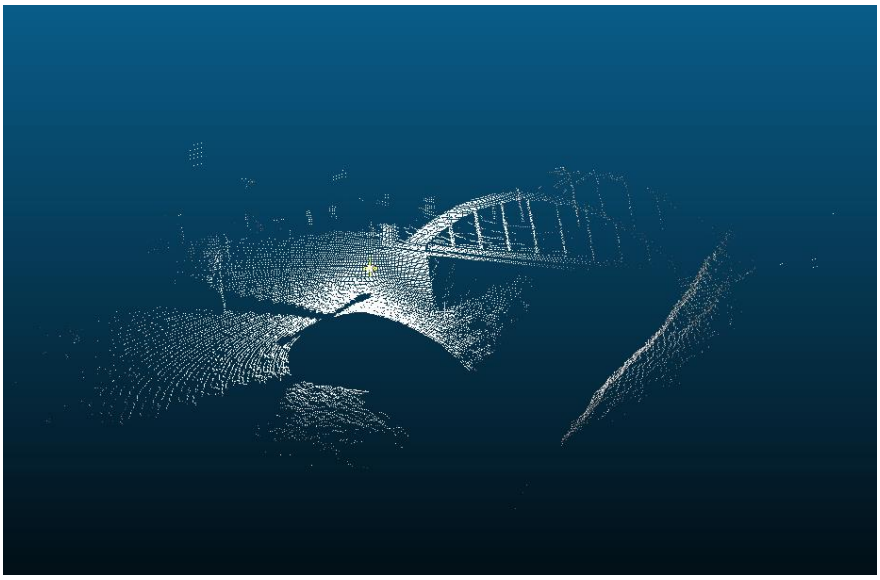
```
void savePointCloudToPLY(const sensor_msgs::PointCloud2::ConstPtr&
pc_msg, const std::string& output_file_path) {
```

```
        pcl::PointCloud<pcl::PointXYZI> pcl_cloud;
        pcl::fromROSMsg(*pc_msg, pcl_cloud);
        pcl::io::savePLYFileASCII(output_file_path, pcl_cloud);

}
```

Ova metodologija osigurava besprijekorno i učinkovito izdvajanje LiDAR podataka iz rosbaga, pružajući PLY datoteke s vremenskim oznakama koje se mogu dalje koristiti za 3D mapiranje i analizu. Modularni dizajn programa omogućuje prilagodbu i integraciju u veće kanale za obradu podataka.







Sl. 3.1 Primjeri izvučenih PLY file-ova

3.2. Ekstrakcija informacija o transformaciji

U procesu stvaranja sveobuhvatnog sustava 3D mapiranja, izdvajanje transformacijskih informacija ključno je za razumijevanje prostornih odnosa između različitih komponenti. Ovo potpoglavlje detaljno opisuje metodologiju i implementaciju prilagođenog C++ programa razvijenog za ekstrakciju TF (Transform) poruka iz snimljenog rosbag-a.

Program koristi biblioteku ROS rosbag za otvaranje ulazne datoteke rosbag u načinu rada samo za čitanje. Implementirano je na isti način kao kod ekstrakcije LiDAR podataka.

Iterirajući kroz sve poruke unutar rosbag-a, program selektivno izdvaja TF poruke povezane s temom "/tf". TF poruke, predstavljene tipom tf2_msgs::TFMessage, sadrže informacije o transformaciji. Za svaku ekstrahiranu TF poruku, program piše odgovarajuće zaglavlje i transformira informacije u izlaznu datoteku.

```
for (rosbag::MessageInstance const& m : rosbag::View(input_bag)) {

    if (m.getTopic() == "/tf") {

        tf2_msgs::TFMessage::ConstPtr tf_msg =
m.instantiate<tf2_msgs::TFMessage>();

        if (tf_msg != nullptr && !tf_msg->transforms.empty()) {

            output_file << "Header:\n" << tf_msg-
>transforms[0].header << "\n";
            output_file << "Transform:\n" << tf_msg-
>transforms[0].transform << "\n\n";

        }
    }
}
```

Izlazna datoteka je strukturirana tako da predstavlja zaglavlje svake TF poruke i transformacijske detalje u jasnom i organiziranom formatu. Ova struktura pomaže u kasnijim analizama i olakšava razumijevanje vremenske evolucije transformacija unutar snimljenih podataka.

Ovo poglavlje ističe značaj ekstrakcije TF poruka u kontekstu 3D mapiranja, pružajući čitateljima uvid u metodologiju i implementaciju prilagođenog C++ programa. Izdvojeni TF podaci služe kao temeljna komponenta za postizanje točnog prostornog usklađivanja i razumijevanje dinamičkih interakcija unutar snimljenog prostora

Primjer jedne transformacije:

```
Header:
seq: 7
stamp: 1649845175.391681800
frame_id: map

Transform:
translation:
  x: 0.0126018
  y: -0.00907322
  z: -0.000519015
rotation:
  x: 0.062991
  y: 0.990885
  z: -0.00203866
  w: 0.119059
```

4. Transformacija LiDAR podataka

Temelj našeg 3D mapiranja leži u pažljivoj obradi LiDAR podataka i integraciji transformacijskih poruka. Ovo poglavlje opisuje metodologiju korištenu za otvaranje LiDAR podataka, izdvajanje transformacijskih poruka i naknadnu primjenu tih transformacija na oblake LiDAR točaka.

4.1. Čitanje podataka

Početni korak uključuje čitanje transformacijskih poruka iz prethodno obrađene datoteke. Ove poruke, koje predstavljaju promjene u prostornoj orijentaciji i položaju tijekom vremena, ključne su za točno usklađivanje LiDAR podataka. Transformacijska struktura sažima informacije o vremenskoj oznaci, translaciji i rotaciji.

```
struct Transformation {
    double timestamp;
    double translation_x, translation_y, translation_z;
    double rotation_x, rotation_y, rotation_z, rotation_w;
};
```

Translation_x, translation_y, translation_z: Ove varijable predstavljaju komponente translacije transformacije. Oni pokazuju koliko je koordinatni sustav translatiran duž x, y, odnosno z osi.

Rotation_x, rotation_y, rotation_z, rotation_w: Ove varijable predstavljaju komponente rotacije transformacije. Obično se predstavljaju pomoću kvaternionske notacije, gdje rotation_x, rotation_y i rotation_z predstavljaju imaginarne komponente kvaterniona te određuju os rotacije, a rotation_w predstavlja skalarnu komponentu te količinu rotacije oko te osi.

Prilagođena funkcija, readTransformationsFromFile, analizira tekstualnu datoteku koja sadrži poruke transformacije. Izvlači vrijednosti translacije i rotacije s vremenskom oznakom, organizirajući ih u strukturu Transformation. Te se transformacije zatim pohranjuju u vektor za kasniju upotrebu.

LiDAR oblak točaka je pohranjen u PLY (Polygon File Format) datotekama unutar određenog direktorija. Program sustavno prolazi kroz ovaj direktorij, učitavajući svaku datoteku oblaka LiDAR točaka za daljnju obradu.

```
struct dirent* entry;
while ((entry = readdir(dir)) != nullptr) {
	if (entry->d_type == DT_REG) {
		filenames.push_back(entry->d_name);
	}
}
closedir(dir);

std::sort(filenames.begin(), filenames.end());

for (size_t i = 0; i < filenames.size(); i += 1) {
	const auto& filename = filenames[i];
	std::string fullPath = std::string(directoryPath) + filename;

	std::cout << "Processing file: " << fullPath << std::endl;
```

4.2. Transformacija LiDAR podataka

U ovom potpoglavlju ulazimo u zamršenost transformacije podataka LiDAR oblaka točaka pomoću ekstrahiranih parametara transformacije. Ovaj proces je bitan za postizanje prostorne koherencije i spajanje transformiranih podataka u sveobuhvatni 3D model.

Struktura Point enkapsulira prostorne koordinate (x, y, z) s dodatnim informacijama o intenzitetu.

Intensity varijabla predstavlja intenzitet povezan s točkom. Intenzitet je mjera jačine ili svjetline signala. U kontekstu LiDAR-a, intenzitet može predstavljati refleksiju ili snagu povratnog signala s površine na toj točki.

```
struct Point {
      float x, y, z, intensity;
};

tf2::Vector3 translation(0.0, 0.0, 0.0);
tf2::Quaternion rotation(0.0, 0.0, 0.0, 0.0);
tf2::Transform transform(rotation, translation);
```

Kod također postavlja parametre transformacije iz URDF-a te definira transformaciju predstavljenu objektom tf2::Transform. Komponenta translacije određena je kao tf2::Vector3(0.08, 0.0, -0.1578), postavljajući translaciju duž x, y, odnosno z osi. Komponenta rotacije definirana je pomoću tf2::Quaterniona pod nazivom rotation_urdf. Orijentacija je postavljena pomoću kutova Roll-Pitch-Yaw (RPY) s vrijednostima (3,161592, 0,27284, 0,0). Ovi parametri zajedno definiraju prostornu transformaciju i orijentaciju komponente robota.

```
tf2::Vector3 translation_urdf(0.08, 0.0, -0.1578);
```

```
tf2::Quaternion rotation_urdf;
rotation_urdf.setRPY(3.161592, 0.27284, 0.0);
tf2::Transform transform_urdf(rotation_urdf, translation_urdf);
```

Vremenske oznake također igraju ključnu ulogu u povezivanju LiDAR oblaka točaka s parametrima transformacije. Funkcija ‘extractTimestampFromFilename’ izdvaja vremenske oznake iz naziva datoteka koje se potom vezuju s njihovim odgovarajućim transformacijama.

Kod zatim provjerava je li pronađena važeća transformacija uspoređujući iterator (it) s krajem vektora transformacija. Ako je važeća transformacija prisutna, ona ažurira globalne varijable i globalnu transformaciju na temelju dohvaćenih komponenti translacije i rotacije. Informacije o transformaciji pohranjuju se u strukturu Transformation, a globalne varijable translacija i rotacija ažuriraju se u skladu s tim.

```
double timestamp = extractTimestampFromFilename(filename);

auto it = std::find_if(transformations.begin(),
transformations.end(),
[timestamp](const Transformation& t) { return t.timestamp ==
timestamp; });

if (it == transformations.end()) {
      it = std::min_element(transformations.begin(),
transformations.end(),
[timestamp](const Transformation& a, const Transformation& b) {
return std::abs(a.timestamp - timestamp) < std::abs(b.timestamp -
timestamp);
});
}

const Transformation& closestTransformation = *it;

if (it != transformations.end()) {

      translation.setX(it->translation_x);
```

```
        translation.setY(it->translation_y);
        translation.setZ(it->translation_z);

        rotation.setW(it->rotation_w);
        rotation.setX(it->rotation_x);
        rotation.setY(it->rotation_y);
        rotation.setZ(it->rotation_z);

        transform.setOrigin(translation);
        transform.setRotation(rotation);

    }
```

Potom se podaci LiDAR oblaka točaka se transformiraju pomoću dobivenih parametara transformacije.

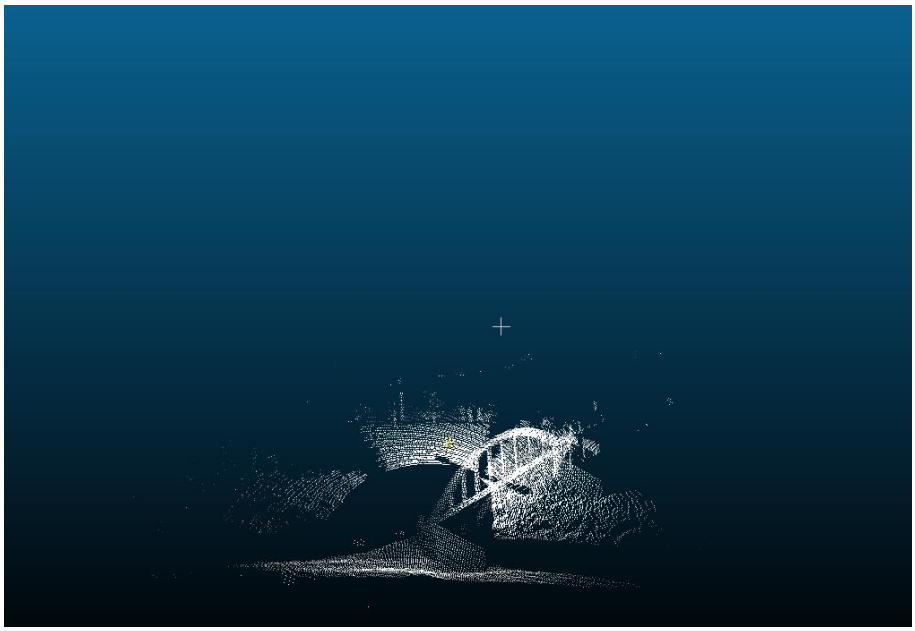
Funkcija transformPoint služi za primjenu trodimenzionalne geometrijske transformacije na objekt Point pomoću ponuđene tf2::Transform. Započinje pretvaranjem Kartezijevih koordinata ulazne točke u tf2::Vector3 pod nazivom input_vector. Nakon toga, transformacija se izvršava množenjem ulaznog vektora s transformacijskom matricom, što rezultira izlaznim_vektorom. Funkcija zatim ažurira koordinate izvornog Point objekta (x, y, z) transformiranim vrijednostima iz output_vector. Važno je da vrijednost intenziteta povezana s točkom ostaje nepromijenjena tijekom transformacije.

```
void transformPoint(Point& point, const tf2::Transform& transform)
{
    tf2::Vector3 input_vector(point.x, point.y, point.z);
    tf2::Vector3 output_vector = transform * input_vector;
    point.x = output_vector.x();
    point.y = output_vector.y();
    point.z = output_vector.z();
}
```

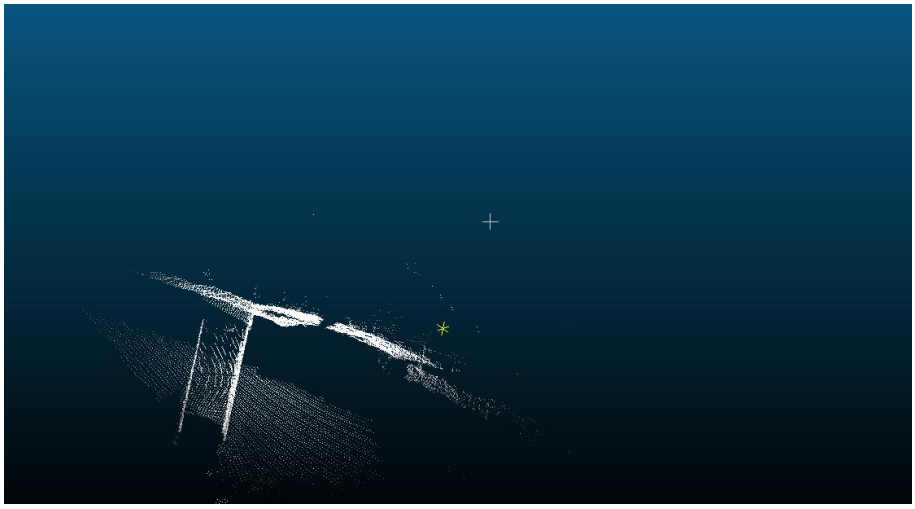
Nadalje kod obrađuje oblak točaka, fokusirajući se na uklanjanje točaka povezanih s tijelom drona. Kod čita svaki redak iz datoteke, izdvaja x, y, z i vrijednosti intenziteta za svaku točku u strukturu Point. Zatim provjerava jesu li koordinate točke izvan određenog raspona (-0,5 do 0,5) duž svake osi. Ako je bilo koji od uvjeta ispunjen, što ukazuje da točka nije unutar navedenog raspona, poziva se funkcija transformPoint primjenjujući dvije različite transformacije (transform i transform_urdf) na točku.

```
while (std::getline(file, line) && file >> point.x >> point.y >>
point.z >> point.intensity) {


      if (!(point.x >= -0.5 && point.x <= 0.5) || !(point.y >= -0.5
&& point.y <= 0.5) || !(point.z >= -0.5 && point.z <= 0.5)) {
            transformPoint(point, transform);
            transformPoint(point, transform_urdf);
            points.push_back(point);

      }
}
```

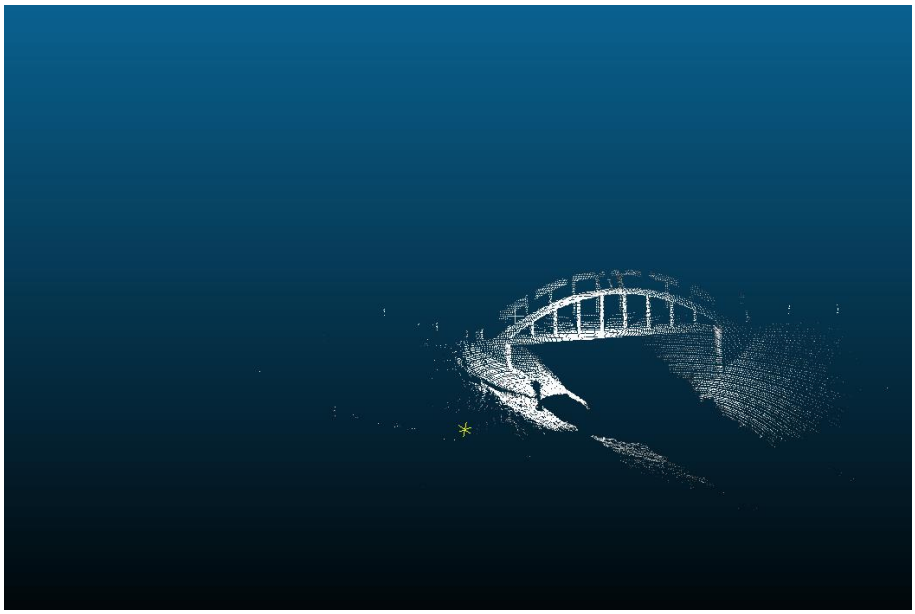
Ovo poglavlje pojašnjava proces transformacije primijenjen na LiDAR podatke. Integracija transformacija s vremenskim oznakama osigurava da LiDAR oblaci točaka odražavaju dinamičke promjene unutar okoline tijekom vremena. Dobiveni transformirani LiDAR podaci, pohranjeni u novoj PLY datoteci, postaju temelj za daljnju analizu, vizualizaciju i sveobuhvatno 3D mapiranje u sljedećim poglavljima.



Sl. 4.1 pointcloud_1649845310.104760.ply



Sl. 4.2 primijenjene globalne transformacije na pointcloud



Sl. 4.3 primijenjene globalne i urdf transformacije na pointcloud

5. ICP algoritam

5.1. ICP algoritam

Algoritam Iterativne najbliže točke (ICP) naširoko je korištena tehnika u području računarstva i robotike za poravnavanje i registriranje dva ili više skupova 3D točaka. Primarni cilj ICP-a je pronaći optimalnu transformaciju (translaciju i rotaciju) koja minimalizira razliku između dva oblaka točaka. Osobito je vrijedan u scenarijima gdje je bitna točna registracija oblaka točaka, kao što je robotika, 3D skeniranje, proširena stvarnost i računalna grafika.

Algoritam radi iterativno, pročišćavajući procjenu transformacije u svakoj iteraciji. Ključni koraci ICP algoritma uključuju odabir odgovarajućih točaka između dva oblaka točaka, procjenu transformacije koja poravnava te odgovarajuće točke i iterativno ažuriranje transformacije.

Tijekom svake iteracije, algoritam obično slijedi proces u dva koraka. Prvo, uspostavlja korespondenciju točaka između izvora i ciljnih oblaka točaka, često koristeći pretraživanje najbližeg susjeda ili druge mjere blizine. Ovaj korak ima za cilj identificirati parove točaka koje vjerojatno odgovaraju istoj fizičkoj značajki u oba oblaka točaka.

Zatim, algoritam procjenjuje transformaciju koja minimizira razliku između odgovarajućih točaka. Najčešća formulacija uključuje minimiziranje zbroja kvadrata udaljenosti između odgovarajućih točaka, što se često naziva metrikom udaljenosti od točke do točke ili točke do ravnine. Različite metode optimizacije, kao što je dekompozicija singularne vrijednosti (SVD) ili varijante poput Gauss-Newtonove metode, mogu se koristiti za pročišćavanje parametara transformacije.

ICP je svestran i može se prilagoditi različitim varijantama, uključujući varijante od točke do točke, točke do ravnine i robusne varijante za rukovanje odstupanjima i šumom u podacima.

Unatoč svojoj učinkovitosti, ICP je osjetljiv na početno usklađivanje, a njegova konvergencija ovisi o prirodi podataka i odabranoj metrici udaljenosti. Istraživači su predložili brojna poboljšanja i varijante, čineći ICP temeljnim algoritmom u domeni registracije oblaka točaka i 3D rekonstrukcije, značajno pridonoseći primjenama u rasponu od autonomne navigacije do medicinskog snimanja.

Navedeni kod implementira ICP algoritam koristeći biblioteku oblaka točaka (PCL) za izvođenje registracije između dva skupa 3D točaka predstavljenih kao vektori struktura Point. Funkcija performICP uzima dva skupa točaka, sourcePoints i targetPoints, i poravnava izvorne točke s ciljnim točkama pomoću ICP algoritma.

Kod počinje pretvaranjem ulaznih vektora Point struktura u PCL vrste oblaka točaka (PCLPointCloudType::Ptr) pomoću pomoćne funkcije toPCLPoint. Ova konverzija je neophodna jer PCL-ova implementacija ICP-a zahtijeva strukture podataka oblaka točaka.

```
PCLPointType toPCLPoint(const Point& point) {
	PCLPointType pclPoint;
	pclPoint.x = point.x;
	pclPoint.y = point.y;
	pclPoint.z = point.z;
	pclPoint.intensity = point.intensity;
	return pclPoint;
}

std::vector<Point> performICP(const std::vector<Point>&
sourcePoints, const std::vector<Point>& targetPoints) {

	PCLPointCloudType::Ptr sourceCloud(new PCLPointCloudType);
	PCLPointCloudType::Ptr targetCloud(new PCLPointCloudType);

	for (const auto& p : sourcePoints) {
		sourceCloud->push_back(toPCLPoint(p));
```

```
        }
        for (const auto& p : targetPoints) {
              targetCloud->push_back(toPCLPoint(p));
        }
```

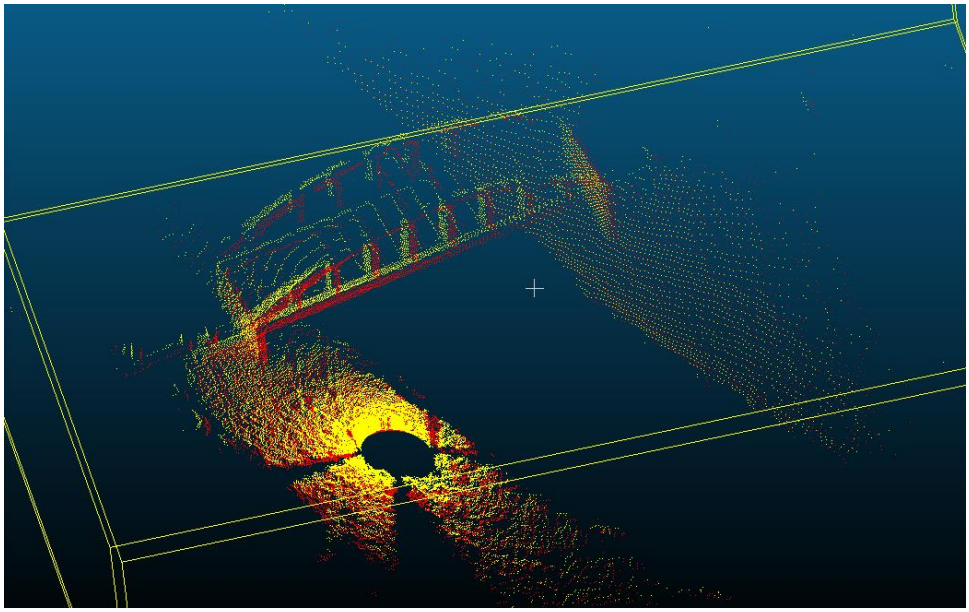
PCL ICP algoritam se zatim konfigurira s izvornim i ciljnim oblacima točaka i izvodi se poravnanje. Rezultirajući usklađeni oblak točaka pohranjuje se u finalCloud.

```
        pcl::IterativeClosestPoint<PCLPointType, PCLPointType> icp;
        icp.setInputSource(sourceCloud);
        icp.setInputTarget(targetCloud);

        PCLPointCloudType finalCloud;
        icp.align(finalCloud);
```

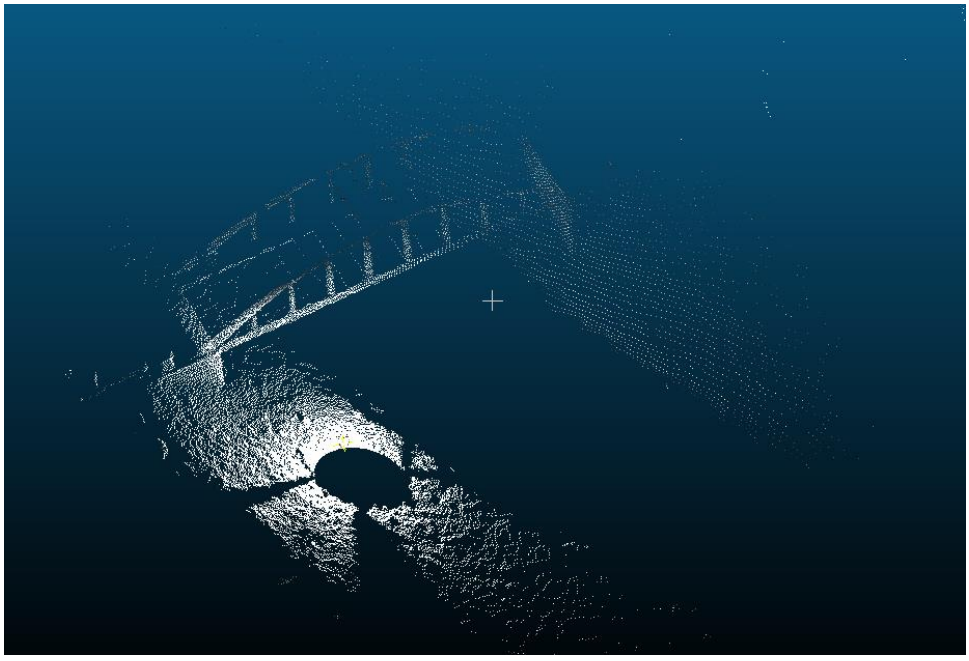
Naknadno, kod transformira originalni izvorni oblak točaka koristeći izračunatu transformaciju iz ICP poravnanja. Transformirani izvorni oblak zatim se pretvara natrag u vektor Point struktura pomoću funkcije fromPCLPoint.

```
        pcl::transformPointCloud(*sourceCloud, finalCloud,
        icp.getFinalTransformation());

        std::vector<Point> finalPoints;
        for (const auto& pclPoint : finalCloud.points) {
              finalPoints.push_back(fromPCLPoint(pclPoint));
        }

Point fromPCLPoint(const PCLPointType& pclPoint) {
      return Point{ pclPoint.x, pclPoint.y, pclPoint.z,
pclPoint.intensity };
}
```

Konačno, funkcija vraća vektor Point struktura koji predstavljaju izvorne točke nakon što su poravnani s ciljnim točkama pomoću ICP algoritma.



Sl. 5.1 sourceCloud i targetCloud



Sl. 5.2 finalCloud

Ovaj isječak koda ilustrira iterativni proces prikupljanja poravnatih oblaka točaka pomoću ICP algoritma. U svakoj iteraciji, usklađene točke koje proizlaze iz postupka registracije dodaju se akumuliranom oblaku točaka, označenom kao accumulatedPoints. Linija

```
acumulatedPoints.insert(accumulatedPoints.end(),
alignedPoints.begin(), alignedPoints.end());
```

dodaje vektor alignedPoints, koji predstavlja izvorne točke poravnate s ciljem, vektoru akumuliranih točaka. Ova akumulacija je ključna za pročišćavanje i poboljšanje cjelokupnog usklađivanja tijekom uzastopnih iteracija.

Naknadno, akumulirane točke postaju cilj za sljedeću iteraciju, a novi skup točaka, koji je izveden iz sljedećeg PLY file-a, služi kao izvorni oblak. Iterativna priroda ovog procesa dopušta nakupljanje usklađenih točaka za progresivno pročišćavanje transformacije između akumuliranog oblaka točaka i novog izvornog oblaka točaka. Ova iterativna registracija često se koristi u aplikacijama kao što je simultana lokalizacija i mapiranje (SLAM) ili 3D rekonstrukcija, gdje se višestruka skeniranja ili pogledi kombiniraju kako bi se izgradio sveobuhvatniji i točniji prikaz okoline.

5.2. VoxelGrid filter

Voxel Grid Filter široko je korištena tehnika u obradi oblaka točaka i 3D analizi podataka. Ova metoda filtriranja koristi se za smanjivanje i organiziranje gustog oblaka točaka grupiranjem obližnjih točaka u jedinstvene mrežne ćelije ili voxele. Osnovna ideja Voxel Grid Filtera je smanjiti gustoću podataka uz očuvanje ukupne strukture i karakteristika izvornog oblaka točaka. Podjelom 3D prostora u pravilne ćelije mreže, točke unutar svakog voxela se sjedinjuju, te se zadržava samo jedna reprezentativna točka, često središte voxela. Ovaj proces značajno smanjuje broj podatkovnih točaka, što dovodi do računalno učinkovitijeg i upravljivijeg skupa podataka za naknadnu analizu ili vizualizaciju.

Priloženi kod implementira Voxel Grid Filter za podatke oblaka točaka pomoću biblioteke oblaka točaka (PCL). Funkcija voxelGridFilter uzima vektor prilagođenih Point struktura i specificiranu veličinu lista voxel rešetke kao parametre. U početku se prilagođene točke pretvaraju u PCL oblak točaka (pcl::PointCloud<pcl::PointXYZI>) te se primjenjuje Voxel

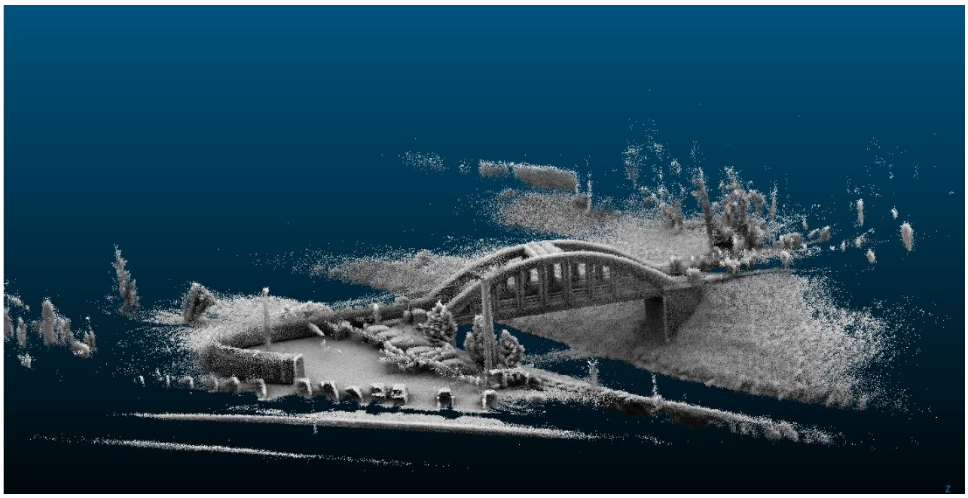
Grid Filter za smanjivanje oblaka točaka grupiranjem obližnjih točaka unutar svakog voxela definiranog veličinom lista. Filtrirani oblak točaka pohranjuje se u novi PCL oblak točaka (filteredCloud).

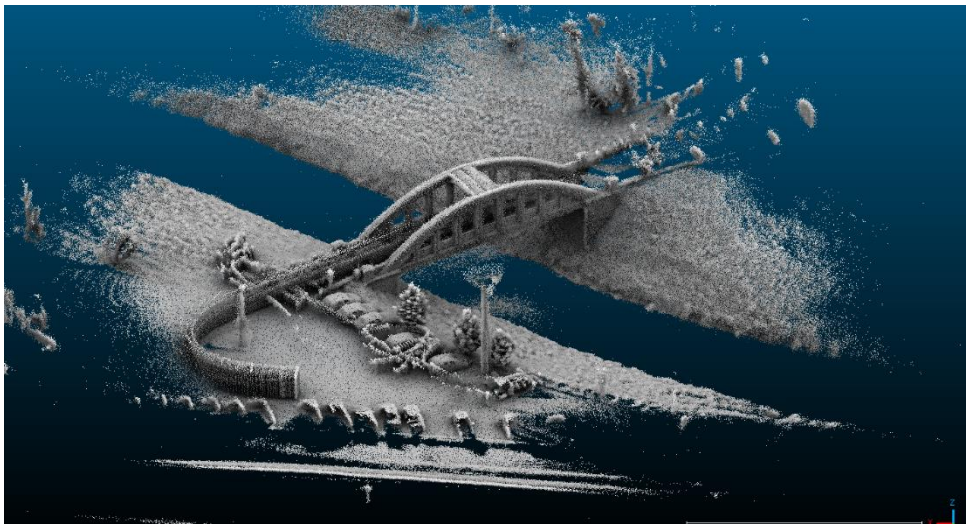
```
	pcl::VoxelGrid<pcl::PointXYZI> sor;
	sor.setInputCloud(cloud);
	sor.setLeafSize(leaf_size, leaf_size, leaf_size);

	pcl::PointCloud<pcl::PointXYZI>::Ptr filteredCloud(new
pcl::PointCloud<pcl::PointXYZI>);
	sor.filter(*filteredCloud);
```

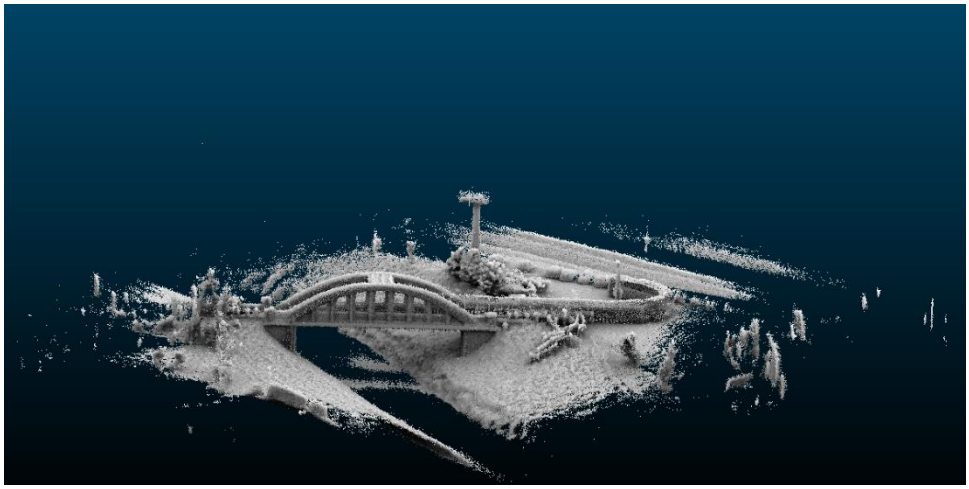
6. Rezultati

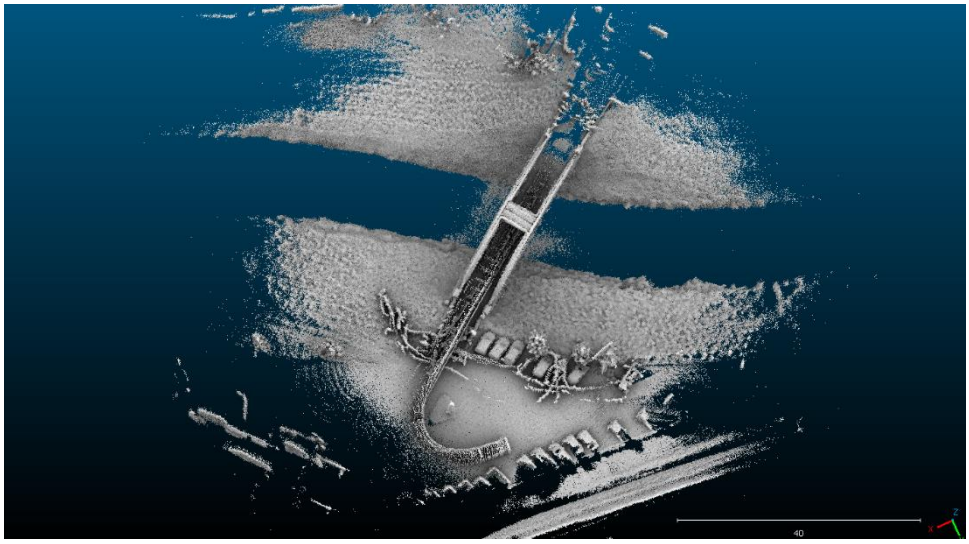
U ovom poglavlju predstavljam opipljive rezultate metodologije 3D mapiranja. Primjenom LiDAR obrade podataka, transformacije putanje i algoritma Iterativne najbliže točke (ICP), implementacija je uspješno dala rafinirani i usklađeni prikaz oblaka točaka. Nakon procesa ekstrakcije i transformiranja oblaka točaka, te poravnanja ICP algoritmom i akumuliranja svih oblaka točaka na sljedećim slikama je prikazan konačni oblak točaka. Ovaj vizualni prikaz služi kao dokaz učinkovitosti našeg pristupa i transformativne moći implementiranih algoritama.

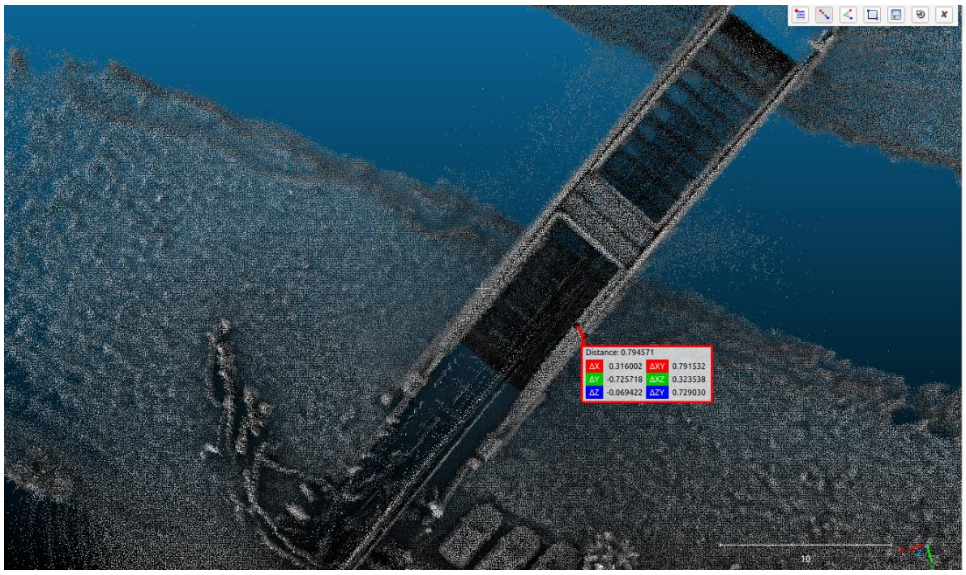
Sl. 6 prikaz finalnog oblaka točaka iz različitih perspektiva











Kako bismo poboljšali učinkovitost našeg algoritma za 3D mapiranje, moguće je implementirati mehanizam za filtriranje podataka prikupljenih tijekom kritičnih faza, kao što je tijekom polijetanja ili spuštanja drona. Isključivanjem točaka prikupljenih tijekom ovih dinamičkih stanja leta, cilj nam je ublažiti potencijalna izobličenja u oblaku točaka uzrokovana brzim promjenama u orijentaciji senzora. Osim toga, razmatranje uklanjanja podatkovnih točaka koje su previše udaljene od LiDAR senzora moglo bi dodatno optimizirati izvedbu algoritma. Točke iznad određenog praga mogu unijeti šum ili netočnosti, utječući na ukupnu kvalitetu oblaka točaka. Implementacija pristupa filtriranja temeljenog na udaljenosti mogla bi poboljšati robusnost algoritma davanjem prioriteta točkama unutar relevantne blizine LiDAR senzora. Ova poboljšanja postavljaju temelje za prilagodljiviji i otporniji algoritam 3D mapiranja, osiguravajući njegovu primjenjivost u različitim uvjetima okoline i scenarijima leta.

Zaključak

Ovaj je rad prošao kroz zamršenost 3D mapiranja uvodeći sveobuhvatan proces transformacije, besprijekorno integrirajući različite tehnologije. Putovanje je započelo metodologijom za obradu LiDAR podataka, otkrivajući proces ekstrakcije, transformaciju putanje i uključivanje modeliranja temeljenog na URDF-u. Istaknuta je ključna uloga algoritma Iterativne najbliže točke (ICP) u procesu obrade, prikazujući njegovu učinkovitost u usklađivanju prikupljenih podataka za precizno usklađivanje.

Istraživanje se nastavilo s izazovima rukovanja velikim skupovima podataka tijekom prikupljanja i predobrade podataka. Naglasak na obradi podataka o putanji za točno usklađivanje i uključivanje modeliranja temeljenog na URDF-u osvijetlili su put kroz ove izazove. Praktičnost metodologije je naglašena kroz isječke koda, gdje su korišteni ROS i PCL. Ovi isječci ilustrirali su ekstrakciju oblaka točaka i TF poruka i transformaciju LiDAR podataka na temelju ekstrahiranih transformacija.

Zalazeći dublje u kod, implementacija ICP algoritma zauzela je središnje mjesto, otkrivajući njegove zamršenosti, parametre, postavke, strategije integracije i postupke provjere valjanosti. Ovaj detaljni vodič osigurao je temeljito razumijevanje funkcionalnosti algoritma i njegovog značaja u usavršavanju 3D mapa.

U zaključku, rad kulminira naglašavanjem važnosti 3D mapiranja i transformacija, razjašnjavajući njihovu ulogu ne samo u tehnološkom napretku već i u primjenama kao što su robotika, praćenje okoliša i urbano planiranje. Ovo putovanje kroz kod pruža ne samo teoretske uvide, već i praktični putokaz za one koji se upuštaju u dinamično područje obrade prostornih podataka.

Literatura

[1] ROS PCL Documentation, poveznica: http://wiki.ros.org/pcl_ros

[2] ROS TF Documentation, poveznica: http://wiki.ros.org/tf

[3] ROS BAG Documentation, poveznica: http://wiki.ros.org/rosbag

[4] P.J. Besl, *A method for registration of 3-D shapes*, IEEE Transactions on Pattern Analysis and Machine Intelligence (1992)

[5] D. Chetverikov, *The Trimmed Iterative Closest Point algorithm,* 2002 International Conference on Pattern Recognition (2002)

Sažetak

Ovaj rad istražuje zamršeni proces 3D mapiranja pomoću različitih tehnologija i sveobuhvatnog procesa transformacije. Predstavlja metodologiju za obradu LiDAR podataka, detaljno opisujući proces ekstrakcije, transformaciju putanje i integraciju s modeliranjem temeljenim na URDF. Algoritam Iterativne najbliže točke (ICP) igra ključnu ulogu u obradi, usklađujući prikupljene podatke.

Prikupljanje podataka i prethodna obrada dalje naglašavaju izazove uključene u rukovanje velikim skupovima podataka, naglašavajući obradu podataka o putanji za točno usklađivanje i uključivanje modeliranja temeljenog na URDF-u. Korištenje ROS-a (Robot Operating System) i PCL-a (Point Cloud Library) u zadanim isječcima koda prikazuje ekstrakciju poruka oblaka točaka, ekstrakciju TF (Transform) poruka i transformaciju LiDAR podataka na temelju ekstrahiranih transformacija.

Implementacija ICP algoritma je detaljno objašnjena, pokrivajući parametre, postavke, integraciju s podacima LiDAR i postupke provjere valjanosti.

Rad zaključuje naglašavanjem važnosti takvog 3D mapiranja i transformacija uključujući robotiku, praćenje okoliša i urbano planiranje.

Summary

This paper explores the intricate process of 3D mapping using various technologies and a comprehensive transformation process. It introduces a methodology for LiDAR data processing, detailing the extraction process, trajectory transformation and integration with URDF-based modeling. The Iterative Closest Point (ICP) algorithm plays a key role in the processing, matching the collected data.

Data acquisition and preprocessing further highlight the challenges involved in handling large datasets, emphasizing the processing of trajectory data for accurate alignment and the inclusion of URDF-based modeling. Using ROS (Robot Operating System) and PCL (Point Cloud Library) in given code snippets shows extraction of point cloud messages, extraction of TF (Transform) messages and transformation of LiDAR data based on the extracted transforms.

The implementation of the ICP algorithm is explained in detail, covering parameters, settings, integration with LiDAR data, and validation procedures.

The paper concludes by emphasizing the importance of such 3D mapping and transformations including robotics, environmental monitoring and urban planning.

Skraćenice

LiDAR	*Light Detection and Ranging*
ICP	*Iterative Closest Point*
URDF	*Unified Robot Description Format*
ROS	*Robot Operating System*
PCL	*Point Cloud Library*
PLY	*Polygon File Format*