Razvoj mobilnog 3D skenera za mapiranje prostora

Strahija, Ivan

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Ivan Strahija

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:
Doc. dr. sc. Filip Šuligoj, mag. ing. mech.

Student:
Ivan Strahija

Zagreb, 2024.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem mentoru doc. dr. sc. Filipu Šuligoju na danim savjetima i pomoći pri izradi diplomskog rada.

Posebno zahvaljujem svojoj obitelji te kolegama i prijateljima koje sam upoznao kroz studij i ranije na motivaciji i podršci tokom cijelog akademskog putovanja.

Ivan Strahija

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
Proizvodno inženjerstvo, inženjerstvo materijala, industrijsko inženjerstvo i menadžment, mehatronika i robotika, autonomni sustavi i računalna inteligencija

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 24 - 06 / 1**	
Ur.broj: **15 - 24 -**	

DIPLOMSKI ZADATAK

Student: **Ivan Strahija** JMBAG: 0035219259

Naslov rada na hrvatskom jeziku: **Razvoj mobilnog 3D skenera za mapiranje prostora**

Naslov rada na engleskom jeziku: **Development of a mobile 3D scanner for area mapping**

Opis zadatka:

U ovom diplomskom radu istražuje se razvoj mobilnog 3D skenera kroz primjenu kombinacije SBC (eng. Single Board Computer) Nvidia Jetson Nano i Intel RealSense D435i dubinske kamere s IMU-om (eng. Internal Meauserment Unit). Cilj rada je integrirati ove napredne tehnologije kako bi se stvorilo učinkovito, mobilno rješenje za 3D skeniranje specifično usmjereno na robusno mapiranje prostora, a koje će se testirati u Regionalnom centru izvrsnosti za robotske tehnologije (CRTA).

Zadaci rada uključuju:

- Analizu i optimizaciju parametara za dohvat oblaka točaka.
- Integraciju IMU senzora za poboljšano pozicioniranje ključno za stvaranje točnih 3D modela prostora.
- Konfiguraciju SBC-a za brzu obradu podataka u realnom vremenu uz korištenje CUDA jezgara.
- Razvoj prototipa mobilnog 3D skenera koji će biti testiran unutar CRTA-e, verificirajući njegovu sposobnost detaljnog i efikasnog mapiranja unutrašnjeg prostora.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
9. svibnja 2024.	11. srpnja 2024.	15. – 19. srpnja 2024.

Zadatak zadao: Predsjednik Povjerenstva:

Doc. dr. sc. Filip Šuligoj Prof. dr. sc. Ivica Garašić

SADRŽAJ

1. UVOD ... 1
1.1. Definicija problema ... 1
1.2. Metodologija rada ... 2
1.3. Elementi sustava ... 3
1.3.1. NVIDIA Jetson Xavier NX Developer Kit ... 3
1.3.2. Intel RealSense D435i ... 4
2. PROGRAMSKO OKRUŽENJE ... 6
2.1. Linux Ubuntu ... 6
2.2. *Robot Operating System (ROS)* ... 7
2.3. OpenCV (Open Source Computer Vision Library) ... 8
3. PREGLED ALGORITAMA ZA DETEKCIJU I DESKRIPCIJU ZNAČAJKI ... 10
3.1. ORB (Oriented FAST and Rotated BRIEF) ... 10
3.2. GFTT (Good Features To Track) ... 12
3.3. FREAK (Fast Retina Keypoint) ... 12
3.4. SIFT (Scale-Invariant Feature Transform) ... 14
3.5. Usporedba metoda detekcije i deskripcije značajki ... 14
4. PREGLED METODA 3D REKONSTRUKCIJE PROSTORA ... 19
4.1. ICP (Iterative Closest Point) ... 19
4.2. RTAB-Map (Real-Time Apperance-Based Mapping) ... 20
4.2.1. Loop-closure detection ... 23
4.3. ORB-SLAM (Oriented FAST and Rotated BRIEF SLAM) ... 24
5. IZRADA SKUPA PODATAKA ZA VALIDACIJU ALGORITAMA ... 28
6. IZVOĐENJE SLAM ALGORITAMA ... 31
6.1. RTAB-Map ... 31
6.2. ORB-SLAM2 ... 36
6.3. Usporedba provedenih *SLAM* algoritama ... 38
6.4. Mogućnosti poboljšanja sustava ... 40
7. ZAKLJUČAK ... 41

POPIS SLIKA

Slika 1. CAD model i realizirani sklop razvijenog vizijskog sustava 2
Slika 2. NVIDIA Jetson Xavier NX Developer Kit [1] 3
Slika 3. Intel RealSense D435i [2] 4
Slika 4. Sučelje Intel RealSense Viewer aplikacije 5
Slika 5. Spajanje pinova za pokretanje Jetson Xavier NX računala u *recovery mode* načinu rada 6
Slika 6. Sučelje *NVIDIA SDK Manager* aplikacije 7
Slika 7. Prikaz komunikacije čvorova unutar ROS sustava [6] 8
Slika 8. Prikaz diskretizirane kružnice dobivene korištenjem Bresengamovog algoritma [8] 11
Slika 9. Detektirane značajke korištenjem ORB algoritma [10] 12
Slika 10. Detektirane značajke korištenjem *GFTT* algoritma [11] 12
Slika 11. Emulacija rada ljudskog oka binarnim operacijama [12] 13
Slika 12. Promjena veličine *Gaussian* kernela po obrascu retine [12] 13
Slika 13. Prikaz detekcije značajki i objekata koršitenjem SIFT algoritma [13] 14
Slika 14. Usporedba detekcije ključnih značajki korištenjem *FAST, GFTT* i *SIFT* algoritama 16
Slika 15. Uparene značajke dobivene korištenjem *BRIEF* algoritma 17
Slika 16. Uparene značajke dobivene korištenjem *FREAK* algoritma 17
Slika 17. *RTAB-Map* algoritam [18] 21
Slika 18. Blok dijagram *RTAB-Map* ROS node-a [19] 22
Slika 19. Loop-closure detection [22] 24
Slika 20. Moduli *ORB-SLAM2* sustava i način procesiranja ulaza [24] 25
Slika 21. Usporedba *ORB-SLAM* i *ORB-SLAM2* algoritama kod zadatka estimiranja trajektorije [24] 25
Slika 22. Glavni elementi *ORB-SLAM3* sustava [25] 26
Slika 23. Sučelje *ORB-SLAM2 paketa* 27
Slika 24. Struktura seta podataka korištenog za evaluaciju *SLAM algoritama* 29
Slika 25. Par slika dohvaćen stereo infracrvenim kamerama 30
Slika 26. Ukupan broj ključnih značajki detektiran korištenjem *RTAB-Map* algoritma 33
Slika 27. Uparivanje trenutno detektiranih s ranije zabilježenim ključnim značajkama 33
Slika 28. Promjena broja spremljenih *Key Frameova* 34
Slika 29. Ukupno prijeđen put tokom procesa mapiranja 34
Slika 30. Sučelje *RTAB-Map Database Viewer* aplikacije 35
Slika 31. Sučelje osnovne *RTAB-Map* aplikacije tokom i nakon mapiranja prostora 35
Slika 32. Prikaz spremljenog oblaka točaka u programu *Meshlab* 36
Slika 33. Mapa prostora prije provedbe *loop-closure* mehanizma 38
Slika 34. Mapa prostora nakon provedbe *loop-closure* mehanzima 38

POPIS TABLICA

Tablica 1. Tehničke specifikacije NVIDIA Jetson Xavier NX Developer Kit 3
Tablica 2. Tehničke specifikacije za Intel RealSense D435i 5
Tablica 3. Usporedba algoritama za detekciju ključnih značajki 15
Tablica 4. Vrijeme potrebno za deskripciju i uparivanje značajki korištenjem *BRIEF* algoritma 16
Tablica 5. Vrijeme potrebno za deskripciju i uparivanje značajki korištenjem *FREAK* algoritma 16
Tablica 6. Usporedba *RTAB-Map* i *ORB-SLAM2* algoritama 39

POPIS KRATICA

Oznaka	**Opis**
SBC	*engl. Single Board Computer*
SLAM	*engl. Simultaneous localization and mapping*
CRTA	Regionalni centar izvrsnosti za robotske tehnologije
LiDAR	*engl. Light Detection and Ranging*
IMU	*engl. Inertial measurement unit*
FPS	*engl. Frames per second*
ROS	*engl. Robot Operating System*
CAD	*engl. Computer-aided design*
SDK	*engl. Software development kit*
ORB	*engl. Oriented FAST and rotated BRIEF*
FAST	*engl. Features from Accelerated Segment Test*
BRIEF	*engl. Binary Robust Independent Elementary Features*
GFTT	*engl. Good Features To Track*
FREAK	*engl. Fast Retina Keypoint*
SIFT	*engl. Scale-Invariant Feature Transform*
ICP	*engl. Iterative closest point*
RANSAC	*engl. Random sample consensus*
RTAB-Map	*engl. Real-Time Appearance-Based Mapping*
RGB-D	*engl. Red Green Blue-Depth*
BoW	*engl. Bag-of-words*
FAB-MAP	*engl. Fast Appearance-Based Mapping*
ASTRO	*engl. Autonomous System for Teaching Robotics*

SAŽETAK

Sposobnost orijentacije i navigacije u nepoznatom prostoru jedan je od osnovnih problema koje je potrebno riješiti za uspješnu integraciju autonomnih sustava u ljudsku svakodnevnicu. U diplomskom radu predstavljene su i analizirane različite *SLAM* (*Simultaneous Localization and Mapping*) metode te je razvijen prijenosni vizijski sustav koji obavlja funkcije dohvaćanja podataka i izvođenja *SLAM* algoritama. Kao osnova vizijskog sustava korišten je *NVIDIA Jetson Xavier NX Developer Kit* - kompaktno računalo velike procesorske snage koje koristi CUDA jezgre za brzu obradu podataka. Za dohvaćanje slike i podataka s inercijske mjerne jedinice korištena je *Intel RealSense D435i* kamera. Za navedene komponente izrađeno je i kućište koje omogućava njihovu laku montažu na već postojeće robotske sustave te time predstavlja jednostavnu mogućnost proširenja njihove funkcionalnosti i povećanja stupnja autonomnosti. Predstavljeni su i osnovni algoritmi iz *OpenCV* biblioteke za detekciju i deskripciju značajki na fotografijama na kojima se baziraju implementirani *SLAM* algoritmi. Za testiranje razvijenog sustava snimljen je skup podataka u prostoru laboratorija *Regionalnog centra izvrsnosti za robotske tehnologije (CRTA)*. Usporedbom performansi algoritama i dobivenih rezultata vidljivo je kako se integracijom vizualne odometrije s podacima koje bilježi inercijska mjerna jedinica poboljšava kvaliteta mapiranja i lokalizacije u prostoru.

Ključne riječi: *SLAM, NVIDIA Jetson, Intel RealSense,* autonomni sustavi

SUMMARY

The ability to orientate and navigate in an unfamiliar space is one of the fundamental problems that has to be solved if autonomous systems were to be successfully integrated into our everyday life. This thesis introduces and analyzes various *SLAM* (*Simultaneous Localization and Mapping*) methods and also presents a portable vision system for data collection and execution of SLAM algorithms. The *NVIDIA Jetson Xavier NX Developer Kit* - a compact, high-performance computer utilizing CUDA cores for fast data processing serves as the foundation of the vision system. The *Intel RealSense D435i* camera is used for image acquisition and data collection from the inertial measurement unit. A custom housing for these components has been designed, facilitating easy mounting on existing robotic systems and providing a straightforward way of enhancing their functionality and increasing their level of autonomy. Basic algorithms from the *OpenCV* library for feature detection and description in images, which underpin the implemented *SLAM* algorithms, are also presented. For testing the developed system, a dataset was recorded in the *Regional Centre of Excellence for Robotic Technology* (*CRTA)* laboratory. A comparison of the algorithms' performance and the obtained results show how integrating visual odometry with data recorded by the inertial measurement unit improves the quality of mapping and localization in the environment.

Key words: *SLAM, NVIDIA Jetson, Intel RealSense,* autonomous systems

1. UVOD

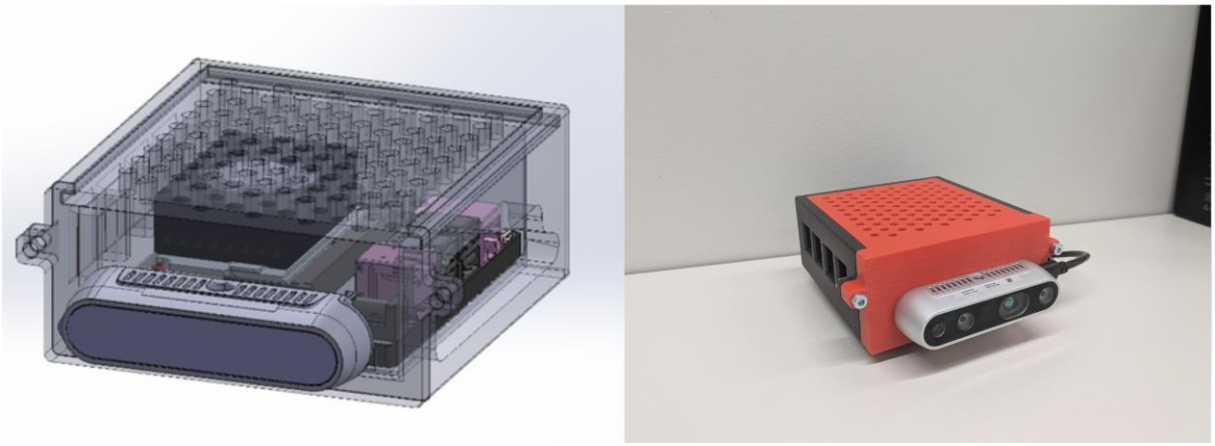
1.1. Definicija problema

Razvijanje i implementacija algoritama za simultano mapiranje prostora i lokalizaciju (*SLAM - Simultaneous Localization and Mapping*) aktualan je zadatak u području autonomne navigacije i robotike. Napredak tehnologije te povećanje dostupnosti i pristupačnosti senzora potrebnih za akviziciju značajki 3D prostora istovremeno je potaknuo i ubrzan rad na razvoju *SLAM* algoritama te njihovoj praktičnoj implementaciji. Kao jedan od najvećih problema autonomnih sustava naglašava se otežana mogućnost navigacije i pozicioniranja u nepoznatim ili brzo promjenjivim okruženjima. Bez postojećeg modela prostora ili drugačijeg zapisa (karte, koordinate) informacija o prostoru u kojem se nalazi, vozilo ili robot ne može se pouzdano kretati, navigirati ili izvršavati potrebnu zadaću. *SLAM* algoritmi stoga omogućuju stvaranje 3D modela prostora u stvarnom vremenu, istovremeno s kretanjem, te na taj način donekle simuliraju i čovjekovo ponašanje u novoj okolini i stvaranje mentalne mape prostora koja se zatim kontinuirano nadopunjuje s novim informacijama i značajkama simultano s kretanjem kroz prostor.

Od izrazite je važnosti i mogućnost točnog određivanja vlastite pozicije autonomnih sustava kako bi se eliminirala mogućnost sudara s okolinom ili nemogućnost pronalaska željene trajektorije kretanja. Budući da *SLAM* algoritmi kao ulaz koriste podatke sa više senzora (kamera, LiDAR, IMU), kombinacijom i provjerom podataka zaprimljenih sa senzora smanjuje se mogućnost pogreške te povećava pouzdanost i točnost mapiranja i lokalizacije unutar nepoznatog prostora. Novija istraživanja pokazuju mogućnosti i prednosti strojne obrade podataka te integracije umjetne inteligencije u procese obrade slike i podataka dobivenih s ostalih senzora. *SLAM* se u kontekstu mobilne robotike i autonomnih sustava predstavlja kao ključan aspekt omogućavanja autonomne navigacije i kretanja u nepoznatim prostorima pružajući rješenje za precizno pozicioniranje i korekciju eventualnih pogrešaka te prilagodbu dinamičkim okruženjima u kakvima se trenutno očekuje da autonomni sustavi funkcioniraju točno i dosljedno.

1.2. Metodologija rada

U sklopu diplomskog rada razvija se vlastito rješenje za 3D mapiranje prostora u vidu lako prenosivog sustava *NVIDIA Jetson* računala s *Intel RealSense D435i* kamerom. Zamišljeno je da se sustav može po potrebi lako montirati na već postojeće robote te tako proširiti mogućnosti njihove uporabe i omogućiti sposobnost autonomnog kretanja unutar radnog prostora. Potrebno je postaviti programsko okruženje na *NVIDIA Jetson* računalu što uključuje postavljanje operativnog sustava te instalaciju potrebnih biblioteka i modula. Za implementaciju i testiranje algoritama koriste se već postojeće biblioteke uključujući ROS (Robot Operating System) s dodatnim modulima, Open3D biblioteku za manipulaciju podacima u 3D sustavu te Intel RealSense SDK za akviziciju podataka s *RealSense D435i* kamere. Korištenje ove kamere, koja posjeduje i IMU (Inertial Measurment Unit) jedinicu, omogućava prikupljanje dodatnog skupa podataka s navedene inercijske mjerne jedinice te njihovom integracijom sa stereo slikom dodatno poboljšanje odometrije. Na zabilježenom setu podataka testirani su različiti *SLAM* algoritmi te se kroz modifikaciju parametara pokušala postići optimalna struktura za rad algoritama u danim uvjetima. Izvršena je konačna evaluacija te usporedba performansi svakog od korištenih algoritama. Na Slici 1. prikazan je CAD model te realizirani sklop razvijenog vizijskog sustava.



Slika 1. CAD model i realizirani sklop razvijenog vizijskog sustava

1.3. Elementi sustava

1.3.1. NVIDIA Jetson Xavier NX Developer Kit

NVIDIA Jetson Xavier NX Developer Kit je računalo velike snage procesiranja namijenjeno za primjernu u ugradbenim sustavima. Izrazito je pogodno za razvoj aplikacija koje karakterizira obrada velike količine podataka u kratkom vremenskom periodu. *Jetson Xavier NX Developer Kit* prikazan je na Slici 2.

Slika 2. NVIDIA Jetson Xavier NX Developer Kit [1]

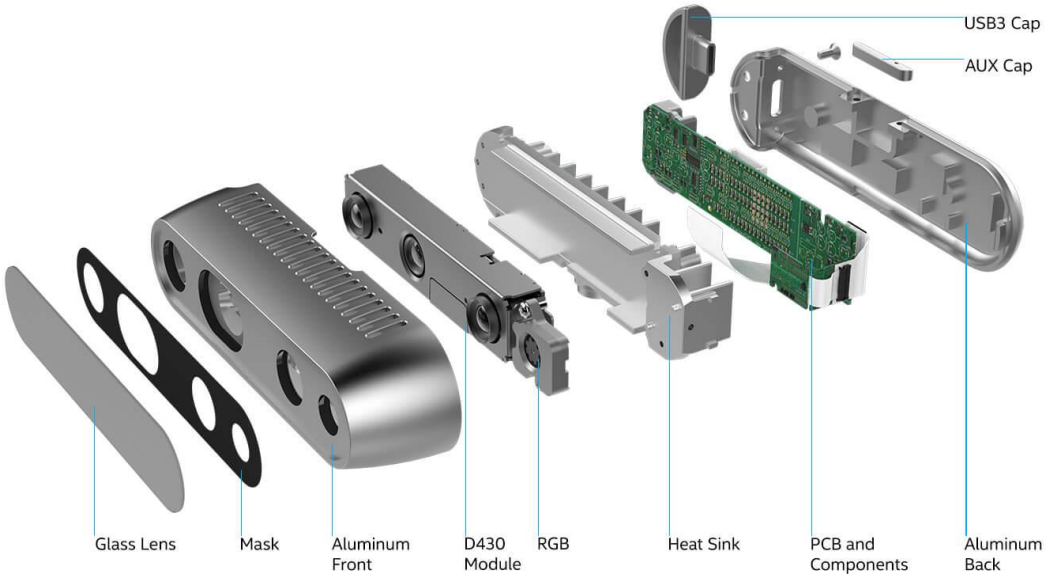
Tablica 1. prikazuje tehničke specifikacije za *NVIDIA Jetson NX Developer Kit*.

Tablica 1. Tehničke specifikacije NVIDIA Jetson Xavier NX Developer Kit

CPU	6-core NVIDIA Carmel ARMv8.2
GPU	NVIDIA Volta, 384 CUDA cores
RADNA MEMORIJA	8GB 128-bit LPDDR4x 59.7GB/s
POHRANA PODATAKA	microSD
POVEZIVOST	Gigabit Ethernet, M.2 Key E (WiFi/BT), M.2 Key M (NVMe)
VIDEO IZLAZ	HDMI, Display Port
DIMENZIJE	103mm x 90.5mm x 34mm

1.3.2. Intel RealSense D435i

Intel RealSense D435i je uređaj za mapiranje prostora koji se sastoji od RGB senzora, dvije dubinske kamere, Inertial Measurment Unit-a (IMU) te infracrvenog projektora. Eksplodirani prikaz *Intel RealSense D435i* uređaja prikazan je na Slici 3.



Slika 3. Intel RealSense D435i [2]

RGB senzor služi za dohvaćanje boja te time poboljšava interpretaciju i realnu vizualizaciju prostora. RGB senzor dohvaća sliku rezolucije 1920x1080 piksela do 30 puta u sekundi.

Dvije dubinske kamere služe za dohvaćanje informacija u udaljenosti pojedinih objekata sa slike odnosno pružaju mogućnost stereo vida. Dohvaćaju sliku rezolucije 1280x720 piksela do 90 puta u sekundi.

Inertial Measurement Unit (IMU) se sastoji od žiroskopa i akcelerometra. Akcelerometar mjeri ubrzanje duž tri osi te tako pruža mogućnost preciznog praćenja i bilježenja pokreta samog uređaja. Žiroskop pruža informaciju o kutnim brzina rotacije uređaja te nadopunjuje podatke koje daje akcelerometar i omogućava bolju stabilizaciju slike te praćenje promjena orijentacije i rotacije uređaja.

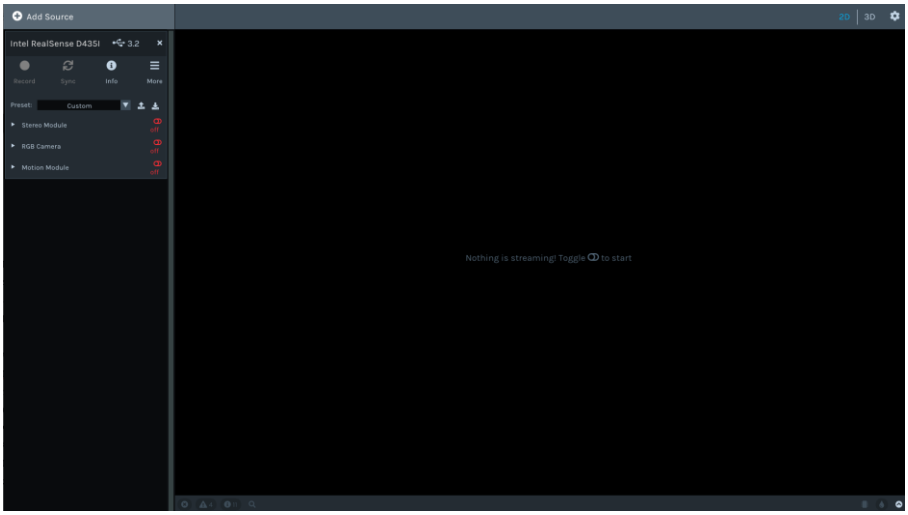
Infracrveni projektor emitira infracrvenu svjetlost te na taj način omogućuje robusnije i preciznije dohvaćanje podataka o dubini prostora te poboljšava rad dubinskih kamera.

Bitne tehničke specifikacije za *Intel RealSense D435i* prikazane su u Tablici 2.

Tablica 2. Tehničke specifikacije za Intel RealSense D435i

RGB REZOLUCIJA	1920x1080 PX
RGB FRAME RATE	30 FPS
RGB FIELD OF VIEW	69° × 42°
DEPTH REZOLUCIJA	1280x720 PX
DEPTH FRAME RATE	90 FPS
DEPTH FIELD OF VIEW	87° × 58°
SNAGA IC PROJEKTORA	360 mW
USB SUČELJE	USB 3.1
VANJSKE DIMENZIJE	90 mm x 25 mm x 25 mm

Intel je za *RealSense* kamere razvio i širok spektar razvojnih alata i knjižnica koje omogućavaju jednostavno postavljanje i korištenje same kamere te akvizicije podataka sa pojedinih senzora. Ovaj uređaj kompatibilan je s mnogim operativnim sustavima i programskim okruženjima (Linux, ROS, Windows) te je stoga često dio mnogih projekata iz područja strojnog vida. Za postavljanje *Intel RealSense D435i* uređaja u ovom radu korišten je Intel RealSense SDK 2.0 te aplikacija *Intel RealSense Viewer* čije je sučelje prikazano na Slici 4.



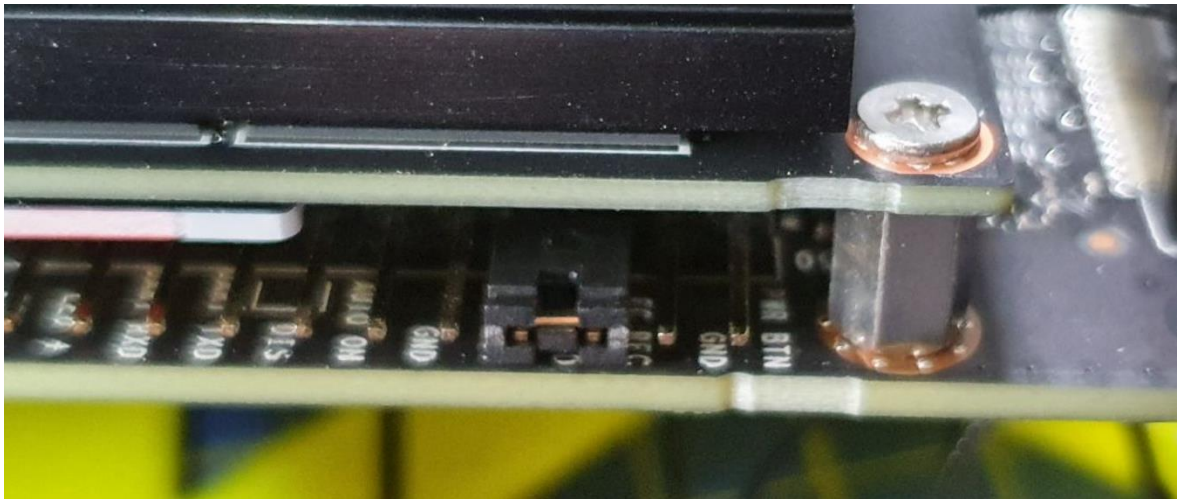
Slika 4. Sučelje Intel RealSense Viewer aplikacije

2. PROGRAMSKO OKRUŽENJE

2.1. Linux Ubuntu

Prvo službeno izdanje Linux *Ubuntu* operacijskog sustava objavljeno je 2004. godine pod kodnim imenom *Warty Warthog.* Samo ime operacijskog sustava ima korijen u staroafričkoj riječi *'oŏ'boŏntoō'* koja se najčešće prevodi kao sintagma 'čovječanstvo drugima'. Ime dakle sugerira *open-source* orijentaciju samog operacijskog sustava te želju da se istom omogući implementacija, korištenje i razvoj diljem svijeta. Za implementaciju *SLAM* algoritama u ovom radu korišten je *Ubuntu 20.04* poznat i pod kodnim imenom *Focal Fossa* izdan 2020. godine [3].

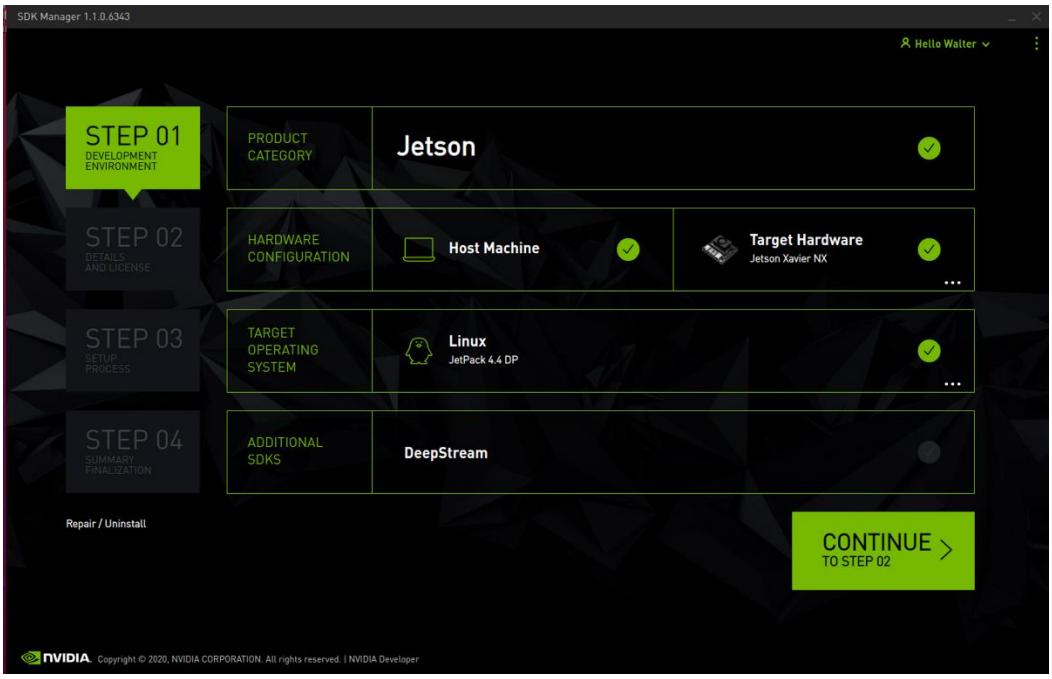
NVIDIA Jetson platforma bazirana je na *Ubuntu* operacijskom sustavu. Kao što je ranije navedeno, za implementaciju *SLAM* algoritama u ovom radu korišteno *je JetPack SDK 5.1.2* programsko okruženje koje je temeljeno na Linux Kernelu 5.10 i *Ubuntu 20.04* operacijskom sustavu. Za postavljanje programskog okruženja na korištenom *Jetson Xavier NX* računalu korišten je službeni *NVIDIA SDK Manager*. Kako bi se omogućilo postavljanje operacijskog sustava kroz *SDK Manager* potrebno je spojiti *Jetson* uređaj u *recovery mode-u* s računalom na kojem je prethodno instaliran *NVIDIA SDK Manager*. Za pokretanje *Jetson* uređaja u navedenom režimu rada potrebno je kratko spojiti pinove 9 i 10 koji se nalaze ispod ventilatora što je prikazano na Slici 5.



Slika 5. Spajanje pinova za pokretanje Jetson Xavier NX računala u *recovery mode* načinu rada

Nakon spajanja *Jetson* uređaja s računalom putem USB sučelja potrebno je kroz *SDK Manager* odabrati željenu verziju programskog okruženja koje želimo postaviti na *Jetson* modulu te odabrati dodatne programske pakete i knjižnice koje pritom želimo instalirati. Zatim

je potrebno pokrenuti proces preuzimanja, kreiranja i postavljanja odabranih programskih paketa. Nakon što je isto dovršeno potrebno je odspojiti kratko spojene pinove te pokrenuti *Jetson* uređaj u normalnom načinu rada te dovršiti postavljanje operativnog sustava. Prikaz *ssučelja NVIDIA SDK Manager* aplikacije vidljiv je na Slici 6.



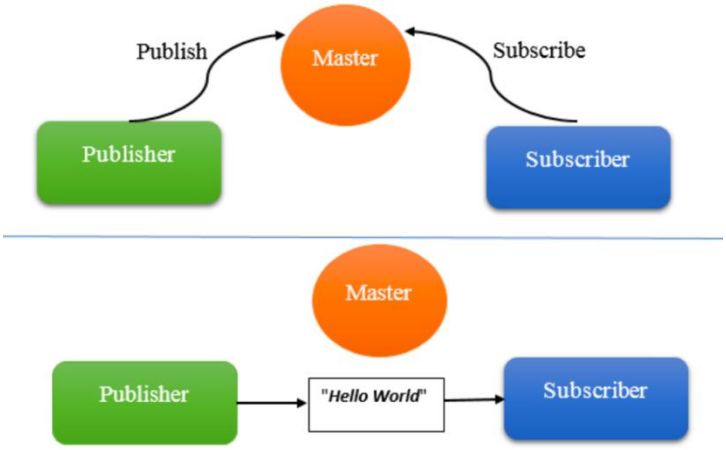
Slika 6. Sučelje *NVIDIA SDK Manager* aplikacije

2.2. *Robot Operating System (ROS)*

Robot Operating System definiran je kao softversko okruženje koje čine javno dostupne biblioteke koda za upravljanje robotskim sustavima. ROS je distribuirani okvir procesa (engl. *nodes*) koji omogućuje pojedinačno izrađenim izvršnim datotekama okvir povezivanja tijekom njihova zajedničkog izvođenja. Ti procesi mogu biti grupirani u pakete i stogove (engl. *packages, stacks*) što kodu omogućuje jednostavno dijeljenje i distribuciju. ROS paket predstavlja koherentnu zbirku izvršnih (*engl. executable*) i ostalih (*engl. supporting*) potrebnih datoteka koje imaju određenu namjenu [4]. Za inicijalizaciju ROS paketa potrebno je kreirati i postaviti radni direktorij (*engl. workspace*) iz kojeg će se pozivati naredbe i procesi. Ovakav dizajn, od razine pojedinačnih datoteka do razine zajednice omogućuje neovisne odluke kod razvoja i implementacije, no sve integrira kroz svoju infrastrukturu [5]. Standardizacijom naredbi i načina komunikacije ROS programski paket omogućio je brz razvoj i kolaboraciju znanstvenika i entuzijasta diljem svijeta koji istovremeno mogu koristiti postojeće funkcionalnosti sustava te implementirati i dodavati vlastite nadogradnje. Glavna motivacija

iza ovog javno dostupnog robotskog sustava jest mogućnost upotrebe postojećih algoritama u novim istraživanjima.

Procesiranje podataka u ROS-u odvija se u čvorovima koji obavljaju funkcije objavljivanja (*engl. publish*), slaganja (*engl.* multiplex) i primanja (*engl. subscribe*) poruka s aktuatora, senzora i ostalih elemenata robotskog sustava. Interakcija između čvorova odvija se koristeći i servise koji rade na principu pitanja i odgovora. Za povezivanje čvorova u funkcionalnu cjelinu i omogućavanje njihove komunikacije potrebna je inicijalizacija glavnog poslužitelja (*engl. master*). Sama komunikacija između čvorova unutar ROS-a izvršava se kroz slanje poruka (*engl. messages*) na predviđene teme *(engl. topics*). Prije nego što se pojedinom čvoru omogući objavljivanje na neku temu, isti šalje detalje o temi (ime teme, tip podataka) prema glavnom poslužitelju. On zatim provjerava postoje li drugi čvorovi koji su pretplaćeni na navedenu temu, te ako postoje, s pretplaćenim čvorovima dijeli podatke koje je objavio *publisher* [6]. Jednostavan prikaz principa komunikacije čvorova u ROS sustavu prikazan je na Slici 7.



Slika 7. Prikaz komunikacije čvorova unutar ROS sustava [6]

Za razvoj i implementaciju rješenja mapiranja i lokalizacije u 3D prostoru u ovom radu korištena je *ROS Noetic Ninjemys* [7] distribucija koja je objavljena 2020. godine.

2.3. OpenCV (Open Source Computer Vision Library)

Open Source Computer Vision Library javno je dostupna biblioteka za razvoj aplikacija računalnog vida i strojnog učenja. Kao i *ROS,* razvijen je s ciljem pružanja otvorene infrastrukture za primjenu algoritama računalnog vida i ubrzao implementaciju istih u komercijalno dostupnim proizvodima. Biblioteka sadrži više od 2500 ustaljenih te *state-of-the-art* optimiziranih algoritama koji primjenu pronalaze u raznim područjima kao što su identifikacija i praćenje objekata, prepoznavanje lica i praćenje trajektorije kamere. U ovom

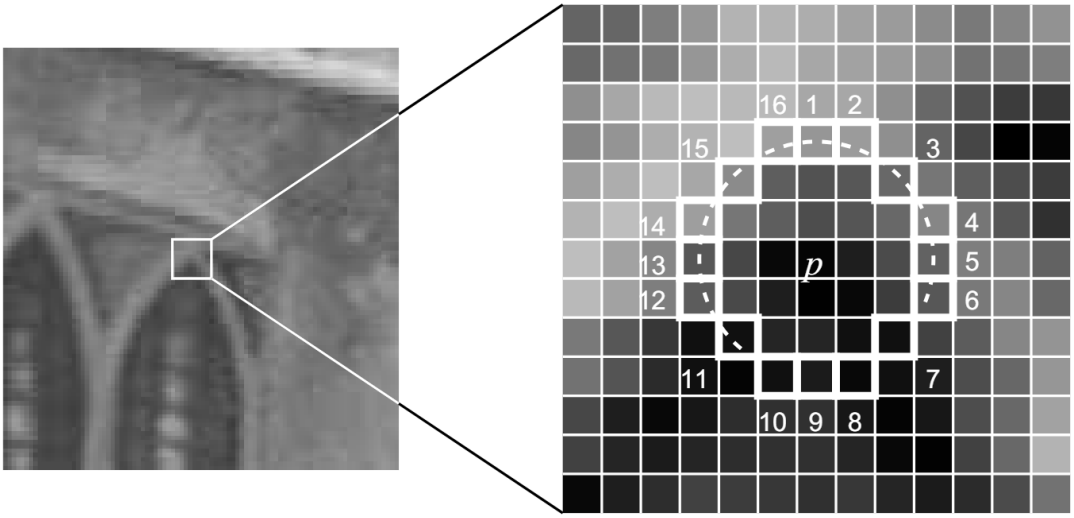
radu su za odometriju i mapiranje prostora korišteni paketi koji koriste detekcijske i deskripcijske funkcije iz *OpenCV* biblioteke – *ORB (Oriented FAST and Rotated BRIEF), GFTT (Good Features to Track), FREAK (Fast Retina Keypoint) i SIFT (Scale-Invariant Feature Transform).*

3. PREGLED ALGORITAMA ZA DETEKCIJU I DESKRIPCIJU ZNAČAJKI

U ovom poglavlju bit će predstavljeni principi rada algoritama za detekciju i deskripciju značajki fotografija na kojima se temelji većina *SLAM* algoritama. Detekcija značajki uključuje identifikaciju interesnih točaka u slici, poput rubova, kutova ili jedinstvenih tekstura, koje mogu biti prepoznate iz različitih kutova i u različitim uvjetima osvjetljenja. Deskripcija značajki podrazumijeva stvaranje jedinstvenih vektorskih prikaza tih točaka, omogućavajući usporedbu i prepoznavanje istih točaka u različitim slikama. Ovi procesi su ključni za SLAM jer omogućavaju algoritmu da povezuje slike snimljene iz različitih perspektiva, čime se gradi konzistentna i točna mapa okruženja. Oni omogućuju robusnost algoritama te osiguravaju funkcionalnost i za slučajeve promjena u osvjetljenju, rotaciji i veličini predmeta koji se nalaze u okolini sustava. Dva *SLAM* algoritma koja će biti uspoređena kasnije baziraju svoj rad na *FAST, GFTT* i *SIFT* algoritmima za detekciju značajki te *BRIEF* i *FREAK* algoritmima za deskripciju. Navedeni algoritmi dio su standardne *OpenCV* biblioteke.

3.1. ORB (Oriented FAST and Rotated BRIEF)

Kao što i samo ime nalaže, *ORB* kombinira prednosti dvije postojeće metode - *FAST (Features from Accelerated Segment Test)* i *BRIEF (Binary Robust Independent Elementary Features)* kako bi pružio robusno i učinkovito rješenje za problem izdvajanja i uparivanja značajki s fotografija. *FAST* detektor značajki je algoritam koji brzo i efikasno identificira rubove provođenjem logičkih operacija za svaki od piksela na fotografiji. Proces kojim se analizira svaki piksel sastoji se od uspoređivanja svjetline (*engl. brightness*) sa svakim od piksela koji se nalazi na diskretiziranoj kružnici određenog radijusa (najčešće 3) koja se definira korištenjem Bresenhamovog algoritma. Prikaz diskretizirane kružnice dobivene korištenjem Bresenhamovog algoritma vidljiv je na Slici 8.



Slika 8. Prikaz diskretizirane kružnice dobivene korištenjem Bresengamovog algoritma [8]

Na tako definiranoj kružnici nalazi se 16 piksela te se njihova svjetlina redom uspoređuje s početno definiranim referentnim pikselom. Kako bi se ubrzao rad algoritma, metodom početne provjere samo 4 piksela s kružnice potvrđuje se ili eliminira mogućnost da je početni piksel definiran kao rub. Ako barem 3 od 4 izabrana piksela prolaze prag razlike u svjetlini nastavlja se s analizom svih 16 piksela. Ako barem 12 od tih piksela prelazi prag razlike svjetline definirani se piksel označuje kao rub odnosno točka interesa. Za različite potrebe može se mijenjati broj diferencirajućih piksela kako bi se definirao rub, no eksperimentalnim putem pokazalo se kako se u tom slučaju pojavljuje velik broj točaka interesa te je samim time potrebno više računalne snage (*engl. computational power*) kako bi se kasnije pronađene točke interesa pratile i potvrdile na cijelom skupu fotografija. *BRIEF (Binary Robust Independent Elementary Features)* je deskriptor koji ORB algoritam koristi za opis pronađenih značajki pomoću *FAST* algoritma. Prije procesa identifikacije značajki algoritam provodi uglađivanje (*engl. smoothing*) fotografija kako bi se smanjila osjetljivost deskriptora na šum. Deskriptor numerički opisuje i sprema pronađene vizualne značajke u obliku vektora. Za pronalaženje istih značajki na različitim fotografijama potrebno je provesti usporedbu dobivenih vektora značajki, a isti se uspoređuju koristeći najčešće metode izračuna razlike udaljenosti između krajnjih točaka vektora, kosinusa kuta između vektora te skalarnog produkta dvaju vektora [9]. Slika 9 pokazuje ulaznu fotografiju te detektirane značajke korištenjem ORB algoritma.

Slika 9. Detektirane značajke korištenjem ORB algoritma [10]

3.2. GFTT (Good Features To Track)

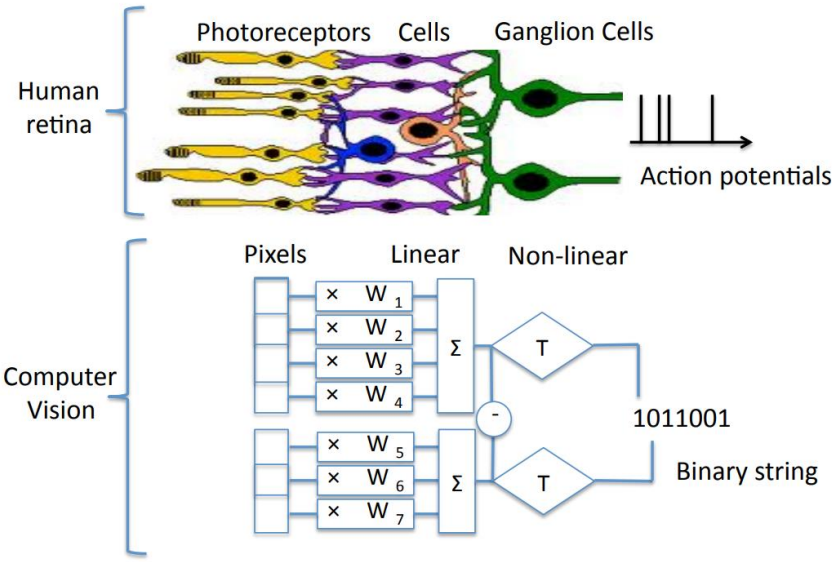
Good Features To Track je metoda pronalaženja i praćenja značajki koju su Jianbo Shi i Carlo Tomasi predstavili u svom istoimenom radu objavljenom 1994. godine. Metodu koju su predstavili testirali su na raznim scenarijima i pokazali njezinu uspješnost. Iako njihova metoda ne rješava sve probleme kod pronalaska dobrih značajki za praćenje, primjerice praćenje svijetle točke na sjajnoj površini, pokazali su da se primjenom lokalnih metoda pronalaska značajki može smanjiti broj točaka čije praćenje nije lako izvedivo [11]. Na Slici 10. prikazana je ulazna slika te detektirane dobre točke za praćenje korištenjem *GFTT* algoritma.

Slika 10. Detektirane značajke korištenjem *GFTT* algoritma [11]

3.3. FREAK (Fast Retina Keypoint)

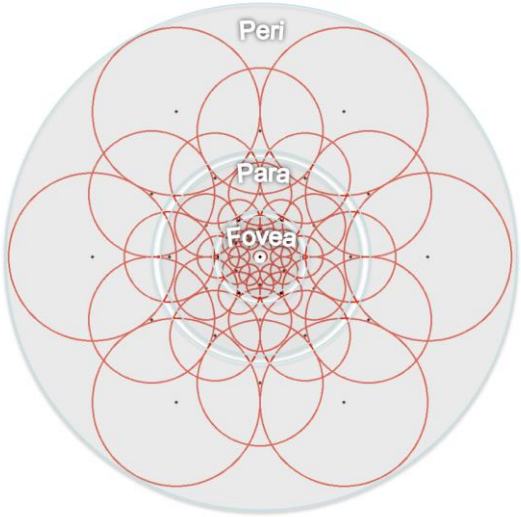
Kao što i ime nalaže, inspiraciju za razvoj ove metode deskripcije ključnih točaka njeni autori pronašli su u ljudskom oku. Kako je ranije navedeno, postojanje šuma na fotografijama uvelike otežava deskripcije ključnih točaka, a postavljanje i korištenje jedinstvenog *Gaussian*

filtera za uglađivanje cijele slike nije jasno definirano za sve moguće slučajeve te ne daje uvijek zadovoljavajuće rezultate. Iz istog je razloga nastala i motivacija za kopiranje topografije ljudskog oka i njegova načina eliminiranja šuma u računalni algoritam. Način emulacije ljudskog oka algoritmom koji se sastoji od binarnih operacija prikazan je na Slici 11.



Slika 11. Emulacija rada ljudskog oka binarnim operacijama [12]

Eksperimentalni su rezultati zatim kasnije pokazali kako ideja o korištenju različitih veličina *Gaussian* filtera po obrascu retine koji je prikazan na Slici 12. vodi do boljih rezultata kod opisivanja pronađenih značajki za praćenje.



Slika 12. Promjena veličine *Gaussian* kernela po obrascu retine [12]

Podaci dobiveni evaluacijom na primjerima iz stvarnog svijeta pokazali su kako *FREAK* algoritam daje bolje rezultate nego suvremeni algoritmi za deskripciju ključnih točaka te isto radi uz manji utrošak računalne snage. Daljnji razvoj ovog koncepta predviđa se u smjeru prepoznavanja objekata [12].

3.4. SIFT (Scale-Invariant Feature Transform)

Scale-Invariant Feature Transform (SIFT), kao što i nalaže njegovo ime, algoritam je za detekciju značajki koje pokazuju invarijantnost na sklairanje, translaciju i rotaciju fotografije te djelomičnu invarijantnost na promjene osvjetljenja i 3D projekcije. Detekcija navedenih značajki provodi se vise stupanjskom filtracijom te identifikacijom stabilnih točaka u prostoru. Zatim se stvaraju ključevi koji dopuštaju lokalne geometrijske deformacije kroz prikaz gradijenata zamagljene (*engl. blurred*) fotografije u višestrukim ravnima i skalama. Stvoreni ključevi koriste se kao ulaz u metodu indeksiranja koja identificira kandidate za značajke. Konačna verifikacija svakog pronađenog kandidata provodi se kroz traženjem reziduala metodom najmanjih kvadrata. Eksperimentalni rezultati istraživanja pokazali su mogućnost detekcije objekata na slikama u vremenima kraćim od 2 sekunde [13]. Prikaz detekcije značajki te objekata korištenjem *SIFT* metode prikazan je na Slici 13.

Slika 13. Prikaz detekcije značajki i objekata koršitenjem SIFT algoritma [13]

3.5. Usporedba metoda detekcije i deskripcije značajki

Za usporedbu navedenih metoda detekcije i deskripcije značajki izrađena je kratka skripta imena *compare.py* koja se nalazi na repozitoriju u prilogu rada. U skripti se konstruiraju i pozivaju navedene *OpenCV* metode detekcije i deskripcije značajki te se njihove performanse evaluiraju na setu fotografija. Korištenjem ugrađene *time* biblioteke mjeri se vrijeme potrebno za provedbu svake od metoda detekcije i deskripcije značajki. Za provedbu usporedbe metoda

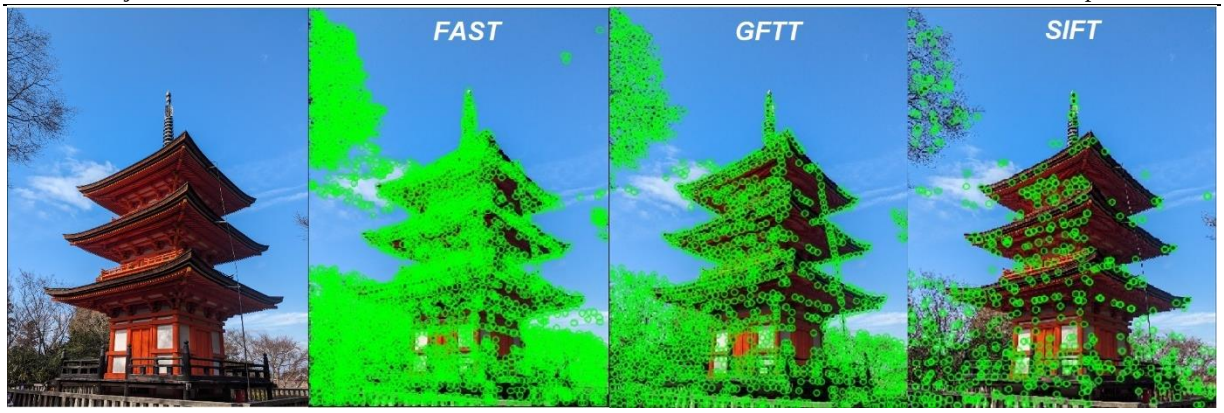
detekcije ključnih značajki korišten je set od 5 različitih fotografije dok su za provedbu usporedbe deskripcije i kasnijeg pronalaženja istih značajki na različitim fotografijama korištene dvije fotografije slikane s gotovo iste točke u prostoru uz malu promjenu orijentacije. Korištene fotografije također se nalaze na repozitoriju navedenom u prilogu rada.

Provođenjem usporedbe metoda detekcija značajki na setu od 5 fotografija izvorne rezolucije 3024 x 4032 piksela skaliranih na rezoluciju 403 x 302 piksela. Za svaku su fotografiju metode detekcije provedene 5 puta te su u Tablici 3. prikazani prosječni rezultati mjerenja. Valja naglasiti kako *FAST* metoda nema ugrađenu mogućnost ograničavanja broja pronađenih značajki već se istu po potrebi može naknadno samostalno implementirati, a navedena metoda detekcije pronalazi velik broj ključnih značajki. Za *GFTT* algoritam kod ove usporedbe postavljen je maksimalan broj od 5000 ključnih značajki budući da je broj pronađenih značajki bio niži od tog. Tako je postavljeno kako bi se brzina izvođenja algoritma mogla bolje usporediti s ostalima. Za *SIFT* algoritam nije postavljen maksimalan broj pronađenih ključnih značajki.

Tablica 3. Usporedba algoritama za detekciju ključnih značajki

	Broj pronađenih ključnih značajki			**Vrijeme izvođenja algoritma**		
Fotografija	***FAST***	***GFTT***	***SIFT***	***FAST***	***GFTT***	***SIFT***
1.jpg	2830	1451	547	0.00059 *s*	0.00308 *s*	0.01910 *s*
2.jpg	2182	1026	306	0.00040 *s*	0.00268 *s*	0.01822 *s*
3.jpg	9516	4404	693	0.00131 *s*	0.00362 *s*	0.01974 *s*
4.jpg	4311	1804	702	0.00079 *s*	0.00302 *s*	0.01879 *s*
5.jpg	5200	2462	961	0.00092 *s*	0.00342 *s*	0.01934 *s*

Dobiveni rezultati potvrđuju kako FAST algoritam uistinu najbrže detektira ključne značajke na fotografijama. Također je vidljivo kako isti konzistentno detektira i najveći broj ključnih značajki. Isto je povezano te se može zaključiti kako jednostavniji selekcijski kriterij i algoritam dovode do većeg broja pronađenih značajki u kraćem vremenu. Upravo suprotno, kod rezultata dobivenih korištenjem SIFT algoritma je vidljivo kako je za pronalazak najmanjeg broja ključnih značajki bilo potrebno najviše vremena. Bitno je znati kako se broj pronađenih značajki nije mijenjao između pojedinih testova, a vrijeme izvođenja algoritama variralo je u manjim intervalima. Na Slici 14. prikazane su sve detektirane ključne značajke s druge fotografije iz skupa testnih podataka korištenjem *FAST, GFTT* i *SIFT* algoritma.



Slika 14. Usporedba detekcije ključnih značajki korištenjem *FAST, GFTT* i *SIFT* algoritama

Za usporedbu deskriptivnih metoda ključnih značajki korištene su dvije fotografije. Prvo su na njima detektirane ključne značajke korištenjem već analiziranih *FAST, GFTT* i *SIFT* algoritama. Zatim je korištenjem *BRIEF* i *FREAK* algoritama na svakoj od fotografija provedena operacija deskripcije pronađenih značajki za svaki od algoritama detekcije. Kao posljednji korak evaluacije provedeno je uparivanje značajki kojima se dobiveni deskriptori podudaraju. U Tablici 4. prikazana su vremena potrebna za deskripciju značajki *BRIEF* algoritmom te vremena potrebna za uparivanje opisanih značajki na paru fotografija. U Tablici 5. prikazana su vremena potrebna za deskripciju značajki *FREAK* algoritmom te vremena potrebna za uparivanje opisanih značajki na paru fotografija. Za vremena potrebna za deskripciju uzet je prosjek vremena potrebnih za deskripciju značajki na svakoj od dviju fotografija.

Tablica 4. Vrijeme potrebno za deskripciju i uparivanje značajki korištenjem *BRIEF* algoritma

Vrijeme potrebno za deskripciju (*BRIEF*)			**Vrijeme potrebno za uparivanje (*BRIEF*)**		
FAST	*GFTT*	*SIFT*	*FAST*	*GFTT*	*SIFT*
0.01506 *s*	0.00167 *s*	0.00201 *s*	0.07302 *s*	0.00102 *s*	0.00212 *s*

Tablica 5. Vrijeme potrebno za deskripciju i uparivanje značajki korištenjem *FREAK* algoritma

Vrijeme potrebno za deskripciju (*FREAK*)			**Vrijeme potrebno za uparivanje (*FREAK*)**		
FAST	*GFTT*	*SIFT*	*FAST*	*GFTT*	*SIFT*
0.02096 *s*	0.00186 *s*	0.00233 *s*	0.09359 *s*	0.00106 *s*	0.00251 *s*

Iz ovih je podataka jasno vidljivo kako pronalazak većeg broja ključnih značajki kao posljedicu ima veća vremena potrebna za deskripciju i uparivanje istih. Iz navedenog je razloga kod korištenja *FAST* algoritma za detekciju ključnih značajki važno implementirati vlastitu metodu za odabir najboljih značajki s kojima će se nastaviti proces deskripcije i uparivanja značajki. Također treba naglasiti kako kod korištenja *FREAK* algoritma za deskripciju s značajkama detektiranim *SIFT* algoritmom često dolazi do pogrešnog uparivanja značajki. Iz tog razloga savjetuje se ne kombiniranje ove dvije metode kod procesa traženja sličnih značajki na fotografijama i njihovog uparivanja. Na Slici 15. prikazano je 15 najbolje rangiranih uparenih značajki korištenjem *BRIEF* algoritma za deskripciju značajki pronađenih korištenjem *FAST*, *GFTT* i *SIFT* algoritama. Na Slici 16. prikazano je 15 najbolje rangiranih uparenih značajki korištenjem *FREAK* algoritma za deskripciju značajki pronađenih korištenjem *FAST*, *GFTT* i *SIFT* algoritama.

Slika 15. Uparene značajke dobivene korištenjem *BRIEF* algoritma

Slika 16. Uparene značajke dobivene korištenjem *FREAK* algoritma

Budući da se *SLAM* algoritmi korišteni u ovom radu baziraju upravo na implementaciji uspoređenih metoda detekcije i deskripcije ključnih značajki, iz navedenih se rezultata može pretpostaviti kako će korištenje *FAST* metode detekcije rezultirati pronalaskom većeg broja ključnih značajki što zatim uzrokuje duže vrijeme za deskripciju istih te zauzeće većeg memorijskog prostora. Kako bi se smanjilo potrebno vrijeme za procesuiranje pronađenih značajki potrebno je optimalno postavljanje parametara detekcije te ograničavanje

maksimalnog broja pronađenih ključnih značajki odnosno njihovo filtriranje nakon što su pronađene. Valja uočiti kako su vremena potrebna za detekciju i deskripciju pronađenih značajki za sve korištene metode dovoljno mala za implementaciju u realnom vremenu te omogućavaju procesuiranje dovoljnog broja slika u sekundi za korištenje kod mapiranja i lokalizacije u prostoru. Problem detekcije istovjetnih značajki na setu podataka kod provođenja mapiranja i lokalizacije drugačiji je od ovdje prikazane usporedbe jedino u vidu usporedbe trenutne slike sa svim slikama dotad zabilježenim u bazi sustava te je potrebno razviti algoritme koji selektivno spremaju i ažuriraju bazu fotografija kako bi ista zauzimala što manje prostora, a pritom bila dovoljno opširna kako bi mogla kvalitetno opisati cijelo okruženje u kojem se sustav nalazi.

4. PREGLED METODA 3D REKONSTRUKCIJE PROSTORA

4.1. ICP (Iterative Closest Point)

ICP (Iterative Closest Point) algoritam predstavlja ključni alat u području računalne grafike i metode rekonstrukcije trodimenzionalnih prostora. Algoritam je razvijen s idejom preciznog poravnavanja trodimenzionalnih oblaka točaka (*engl. Point Cloud*). Kako i samo ime algoritma govori, iterativnom metodom algoritam kroz nekoliko koraka minimizira razliku između dvaju zadanih skupova točaka te ih na taj način poravnava u 3D prostoru. Razvoj i implementacija ovog algoritma ključna je za probleme kao što su mapiranje prostora, skeniranje objekata te lokalizacija u nepoznatom prostoru.

Teorijska osnova *ICP* algoritma utemeljena je na minimizaciji kvadratne pogreške između dva skupa točaka, a isto se postiže korištenjem metode najmanjih kvadrata. Kroz svoj su rad Paul J. Besl i Neil D. McKay pokazali da algoritam uspješno konvergira pod uvjetom da je početna pogreška, odnosno razlika u poziciji i orijentaciji dva oblaka točaka, dovoljno mala [14]. Matematičke osnove i teoremi konvergencije korišteni pri razvoju *ICP* algoritma detaljno su raspisani u navedenom radu. Kako je ranije navedeno, algoritam iterativno prolazi kroz par koraka kako bi poravnao zadane oblake točaka. Prvo se za svaku točku iz početnog skupa podataka pronalazi najbližu točku u referentnom skupu. Zatim se računa optimalna metoda transformacije, koja uključuje translaciju i rotaciju, s ciljem minimizacije kvadratne pogreške između danih točaka. Ta se transformacija zatim primjenjuje na čitav skup podataka i proces se ponavlja sve dok se pogreška, odnosno razlika u translaciji i rotaciji točaka ne smanji ispod željene razine ili dok se ne dosegne zadani maksimalan broj iteracija. Iterativni pristup za cilj ima pronalazak globalnog minimuma pogreške, no ako je početna pogreška izrazito velika, postoji mogućnost zapinjanja u lokalnim minimumima. Iz tog razloga razvili su se i drugi algoritmi koji imaju zadaću početnog (šturog) poravnavanja oblaka točaka kako bi ICP algoritam imao veću mogućnost konvergiranja prema globalnom minimumu pogreške.

Kao neke od glavnih prednosti razvijenog *ICP* algoritma Paul J. Besl i Neil D. McKay u svom radu iz 1992. navode mogućnost poravnavanja oblaka točaka bez potrebe za njihovim pretprocesiranjem, mogućnost konvergencije uz prihvatljivu razinu vektorskog šuma do 10% veličine samog objekta te neovisnost konvergencije o obliku oblaka točaka [14].

Za implementaciju razvijenog algoritma u stvarnim situacijama bilo je potrebno poboljšati nekoliko aspekata rada samog algoritma kao što su optimizacija brzine, poboljšanje robusnosti te mogućnost paralelnog ili distribuiranog procesiranja podataka (korištenjem GPU-a ili distribuiranih računalnih mreža).

U svom radu iz 2001. godine, S. Rusinkiewicz i M.Levoy navode mogućnost optimizacije u 6 faza rada *ICP* algoritma; odabir skupa točaka, uparivanje točaka, ponderiranje odgovarajućih parova, odbacivanje određenih parova pri uparivanju, mjerenje pogreške te minimizaciji izmjerene pogreške [15]. Naposlijetku predlažu optimiziranu verziju *ICP* algoritma koja rješava problem registracije dva oblaka točaka u milisekundama te navode kako se buduća istraživanja trebaju fokusirati na povećanje stabilnosti i robusnosti registracije.

Za poboljšanje robusnosti *ICP* algoritma bitan je razvoj metoda za prepoznavanje, procesiranje i ignoriranje *outliera,* odnosno podataka u oblaku točaka koji se znatno razlikuju od ostalih podataka u skupu. Do njihove pojave najčešće dolazi zbog grešaka u prikupljanju podataka, lošeg osvijetljenja ili pojave šuma. Iz tog razloga dolazi do razvoja metoda poput RANSAC-a (Random Sample Consensus) koje služe za identifikaciju i eliminaciju nepouzdanih vrijednosti iz zabilježenog skupa podataka. Glavni cilj RANSAC algoritma je izdvojiti skup podataka koji najbolje odgovara modelu dobivenom korištenjem svih dostupnih podataka iz oblaka točaka [16].

Kao daljnji razvoj *ICP* metode predstavlja se Color ICP; varijanta *ICP* algoritma koja uključuje podatke o boji pojedine točke kod poravnavanja 3D modela. Kao dodatni atribut, odnosno dodatna metrika po kojoj je moguće provesti registraciju oblaka točaka, usporedba boje doprinosi preciznijem poravnanju oblaka točaka. Iako ova metoda zahtjeva obradu većeg broja podataka, porastom procesorske moći s vremenom postaje očito kako je danas bolje koristiti *Color ICP* s obzirom na minimalnu razliku u vremenu izvođenja algoritma te velika poboljšanja koja isti donosi u vidu točnosti i robusnosti registracije oblaka točaka [17].

4.2. RTAB-Map (Real-Time Apperance-Based Mapping)

RTAB-Map (*Real-Time Appearance-based mapping*) je 2013. godine predstavljen kao *loop-closure detection approach* metoda bazirana na mehanizmu upravljanja memorijom u radu tadašnjeg studenta Mathieu Labbea. *Loop-closure detection approach* definira se kao metoda provjere trenutnog okruženja i određivanja radi li se o već poznatoj lokaciji ili pak o novoj. S porastom postojeće mape prostora raste i vrijeme potrebno za usporedbu trenutnog okruženja s već zabilježenim. Prezentirana *RTAB-Map* metoda bazirana je na čuvanju najnovijih i najčešće pojavljivanih lokacija u radnoj memoriji (*WM – Working Memory*) te njih koristi za *loop-closure detection,* dok ostale lokacije prebacuje i sprema u dugoročnu memoriju (*LTM – Long Term Memory*) [18]. Prikaz *RTAB-Map* algoritma kao *loop-closure detection approach* metode dan je na Slici 18.

Algorithm 1 RTAB-Map

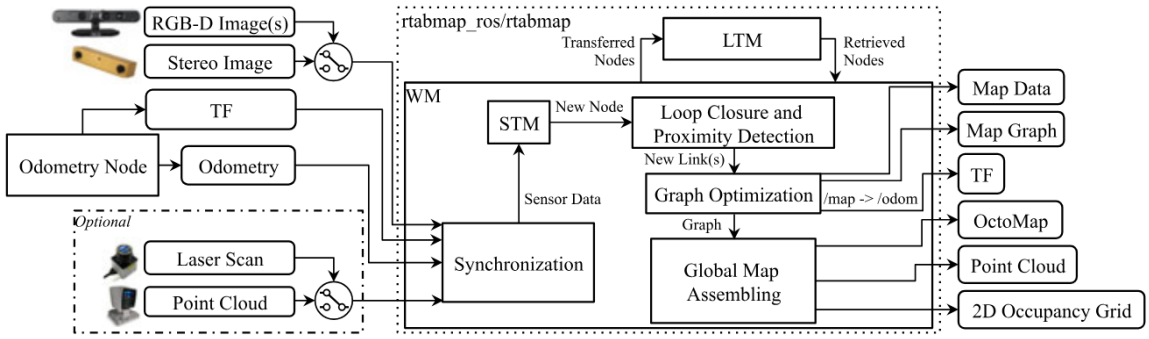
$time \leftarrow$ TIMENOW() ▷ TIMENOW() returns current time
$I_t \leftarrow$ acquired image
$L_t \leftarrow$ LOCATIONCREATION(I_t)
if z_t (of L_t) is a bad signature (using T_{bad}) **then**
    Delete L_t
else
    Insert L_t into STM, adding a neighbor link with L_{t-1}
    Weight Update of L_t in STM (using $T_{\text{similarity}}$)
    if STM's size reached its limit (T_{STM}) **then**
        Move oldest location of STM to WM
    end if
    $\boldsymbol{p}(S_t|L^t) \leftarrow$Bayesian Filter Update in WM with L_t
    Loop Closure Hypothesis Selection ($S_t = i$)
    if $S_t = i$ is accepted (using T_{loop}) **then**
        Add loop closure link between L_t and L_i
    end if
    Join $trash$'s thread ▷ Thread started in TRANSFER()
    RETRIEVAL(L_i) ▷ LTM → WM
    $pTime \leftarrow$ TIMENOW() $- time$ ▷ Processing time
    if $pTime > T_{\text{time}}$ **then**
        TRANSFER() ▷ WM → LTM
    end if
end if

Slika 17. *RTAB-Map* algoritam [18]

Pseudoalgoritam započinje inicijalizacijom vremena, a zatim slijedi preuzimanje nove slike u datom trenutku te njeno alociranje. Zatim se prema vremenu koje je prošlo od bilježenja prošle slike odlučuje treba li trenutno spremljenu sliku odbaciti i spremiti novu ili s trenutnom slikom krenuti dalje u postupak obrade. Ako je uvjet zadovoljen slika se dodaje u kratkotrajnu memoriju (*engl. Short Term Memory, STM*) te se ažuriraju vrijednosti težina u kratkotrajnoj memoriji korištenjem praga sličnosti. Ako je nakon dodavanja nove slike kratkotrajna memorija dosegla svoj limit najstarija slika iz kratkotrajne memorije premješta se u radnu memoriju (*engl. Working Memory, WM*). Tada se korištenjem Bayesianovog filtra ažuriraju vrijednosti podudaranja u radnoj memoriji te se provjerava *loop-closure* hipoteza za trenutnu sliku s već zabilježenim slikama. Ako se hipoteza zadovolji dodaje se potvrdna *loop-closure* veza između trenutne slike i slike s kojom se trenutna slika podudara. Zatim se dohvaća lokacija slike iz dugotrajne memorije (*engl. Long Term Memory. LTM*) s kojom je trenutna slika uparena te se računa vrijeme koje je bilo za izvršavanje koraka od trenutka zabilježavanja trenutne slike. Ako vrijeme obrade premašuje zadano maksimalno vrijeme obrade tada se podaci iz radne memorije prenose u dugotrajnu memoriju.

Daljnjim radom i razvojem kao *open-source* paketa za simultanu lokalizaciju i mapiranje prostora (*SLAM*), *RTAB-Map* postao je nezavisna *C++* biblioteka te paket unutar *Robot Operating System-a* (*ROS*). Trenutno se najčešće primjenjuje za online procesiranje, mapiranje

i lokalizaciju mobilnih robota u nepoznatim prostorima, mapiranje u više navrata (problem inicijalnih uvjeta ili problem otetog robota) i robusnu odometriju [19]. Na Slici 18. prikazan je blok dijagram *RTAB-Map ROS* node-a.



Slika 18. Blok dijagram *RTAB-Map* ROS node-a [19]

Za mapiranje, lokalizaciju i odometriju *RTAB-Map* može koristiti podatke primljene s raznih senzora; RGB-D kamera, stereo kamera, IMU-a i LiDAR-a te isto omogućuje precizno praćenje pozicije i mapiranje u nepoznatom okruženju. Odometrija je definirana kao estimacija vlastite pozicije u prostoru i vremenu proizašla iz interpretacije podataka dobivenih sa senzora. Kako bi se mogla ispravno odrediti vlastita pozicija ujedno je potrebno poznavati i pozicije objekata u okolini. Integracija podataka s više senzora pruža poboljšanu preciznost i pouzdanost mapiranja i lokalizacije. Ključno je naglasiti kako se algoritam izvodi u realnom vremenu te time omogućava primjenu u raznim aplikacijama kao što su autonomna navigacija ili proširena stvarnost (AR).

Za odometriju *RTAB-Map* koristi podatke dobivene od IMU-a, LiDAR-a, enkodera (na primjer s kotača autonomnog vozila) te vizualnom odometrijom. Za vizualnu odometriju koriste se *SIFT, GFTT* i *FREAK metode*.

Unatoč svojim prednostima, *RTAB-Map* i dalje ne predstavlja idealno rješenje za potrebe mapiranja velikih površina s dinamičkim okruženjem u realnom vremenu. Sama kompleksnost i količina podataka koju algoritam obrađuje zahtjeva određeno vrijeme procesiranja te je stoga brzina autonomnog vozila ili robota koje mapira nepoznati prostor ograničena brzinom procesiranja zabilježenih podataka o okolini. Također, korištenjem senzora kratkog dosega postoji mogućnost da autonomno vozilo ili robot zastane u mjestu s kojeg senzori ne dohvaćaju dovoljno značajki kako bi se mogli lokalizirati (bijeli zid, crna površina, površina bez teksture, dugi koridori itd.) te na taj način izgubi mogućnost lokalizacije [19].

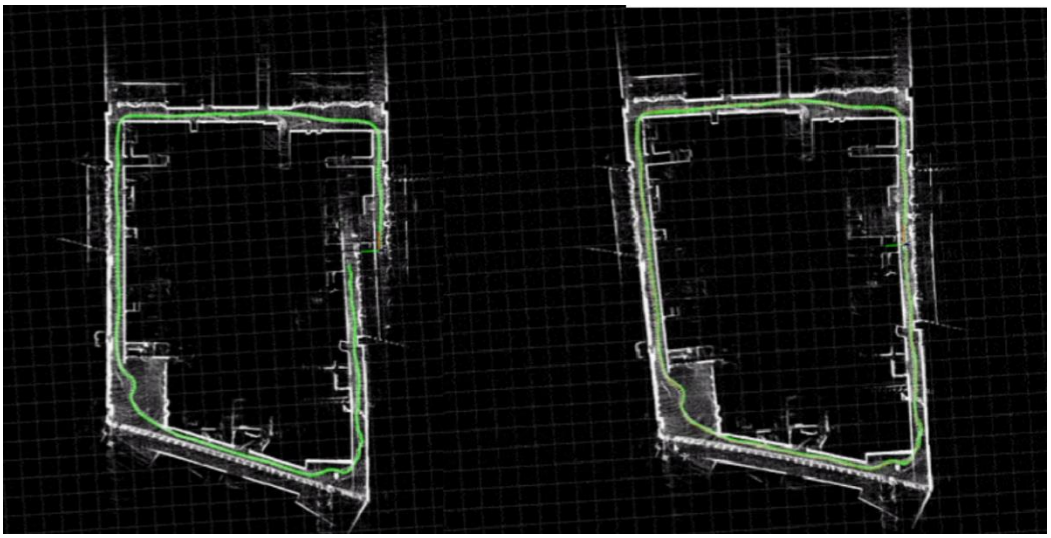
Za daljnji razvoj *RTAB-Map* sustava predlaže se dodavanje mogućnosti automatskog prilagođavanja parametara mapiranja i optimizacije resursa kako bi se unaprijedile performanse algoritma u dinamičkim okruženjima.

4.2.1. Loop-closure detection

Loop-closure detection, odnosno metoda detekcije zatvorene petlje ključna je kod rekonstrukcije mape prostora. Upravo ona omogućava autonomnom sustavu da prepozna svoje trenutno okruženje te na temelju ranije zabilježenih značajki objekata koji se u njemu nalaze ažurira i definira svoju lokaciju i orijentaciju u prostoru. Metode detekcije zatvorene petlje mogu biti bazirane na podacima zabilježeni kamerom ili podacima očitanim s LiDAR-a. U sklopu ovog rada bit će predstavljene i implementirane metode korištenjem podataka zabilježenih samo Intel RealSense kamerom, no kod njihove implementacije moguća je i integracija podataka dobivenih korištenjem LiDAR-a. LiDAR (Light Detection and Ranging) je naziv za senzor koji vraća podatke o udaljenosti okolnih točaka korištenjem emitiranja i registriranja laserskih zraka te mogućom kombinacijom dobivenih informacija s GPS podacima. Razvoj LiDAR sustava započeo je kratko nakon razvoja lasera. Prvi sistem koji je koji je računao udaljenosti pojedinih točaka u okolini mjerenjem vremena potrebnog da se odašiljana svjetlosna zraka vrati do početne točke razvijen je 1961. godine i nazvan je *Colidar - coherent light detecting and ranging* [20]. Metode bazirane na setu podataka koje bilježi LiDAR direktno rade sa modelima oblaka točaka.

Metode bazirane na podacima zabilježenim kamerom fokusiraju se na detekciju i deskripciju značajki opisanim u poglavlju 3. Tako zabilježene i opisane značajke zatim se uspoređuju i traži se njihovo podudaranje na pojedinim slikama. Kako bi se vizualne značajke lakše, brže i uz korištenje manje resursa mogle uspoređivati koristi se njihova reprezentacija kroz vektore dobivene metodama deskripcije. Bitno je naglasiti kako i dalje većina metoda koristi usporedbu novih vektora s onima zabilježenim u ranije definiranom rječniku (*engl. vocabulary*) što odgovara i ljudskom načinu prepoznavanja predmeta – ako prvi put vidimo objekt čije nam ime ili opis nisu poznati, nećemo ga biti u mogućnosti definirati jer njegovom usporedbom s našom bazom podataka (odnosno znanjem i iskustvom stečenim ranije kroz život) ne nalazimo nikakve podudarnosti. Ovakav pristup podudaranja deskriptorskih značajki s onima zabilježenim u prije definiranom rječniku naziva se *Bag Of Words (BoW)*[21]. Budući da se autonomni sustavi često nalaze u dinamičkim i brzo promjenjivim okruženjima takav pristup nije uvijek najbolji. Iz istog razloga razvijeni su algoritmi poput *FAB-MAP-a (Fast*

Appearance Based Mapping) koji pokazuju bolje performanse u dinamičkim okruženjima. Prikaz zatvaranja petlje mapiranog prostora vidljiv je na Slici 19.

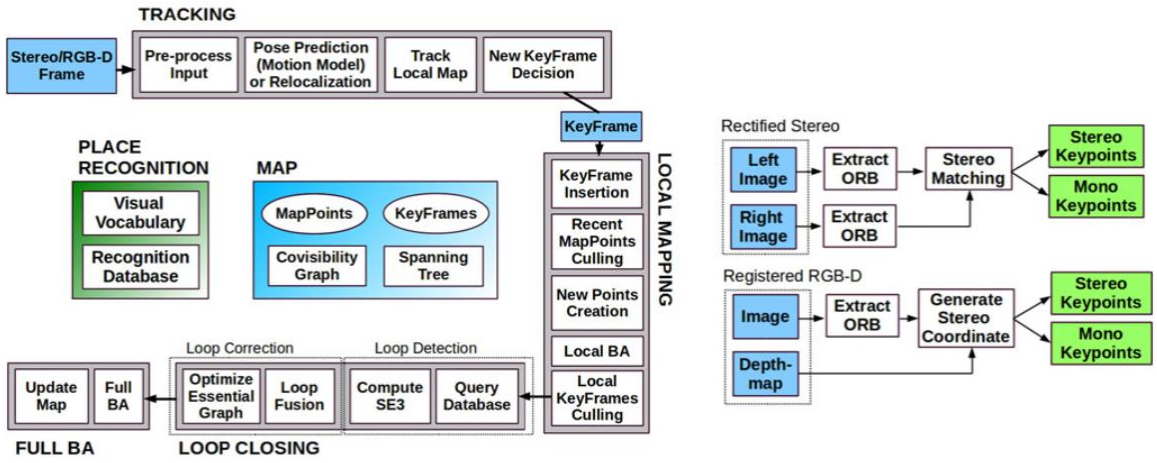


Slika 19. Loop-closure detection [22]

4.3. ORB-SLAM (Oriented FAST and Rotated BRIEF SLAM)

ORB-SLAM (*Oriented FAST and Rotated BRIEF SLAM*) je metoda za mapiranje prostora prvenstveno razvijena za monokularne kamere kao projekt istraživača s Univerzidad Zaragoza. Kao što i ime nalaže, za detekciju i deskripciju ključnih točaka koristi ranije opisane *FAST* i *BRIEF* algoritme. Koristi strategiju 'opstanka najboljih' kod izbora ključnih točaka i slika za rekonstrukciju 3D modela te time osigurava robusnost, a stvorena mapa se proširuje samo ako se okruženje mijenja [23]. Svi razvijeni *ORB-SLAM* algoritmi također implementiraju *loop-closure detection* princip opisan u prošlom poglavlju, no za isto koriste značajke pronađene *FAST* i opisane *BRIEF* algoritmom.

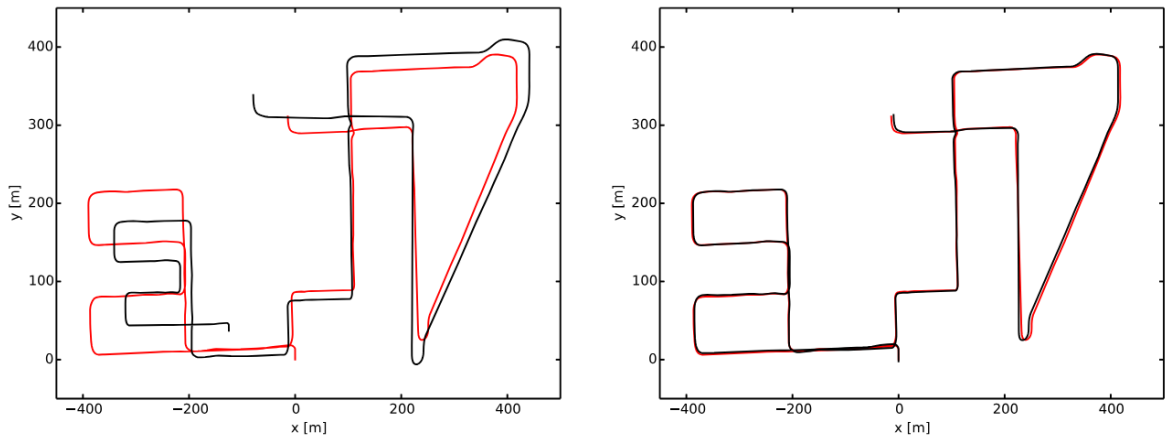
Daljnjim radom i unapređenjem algoritama Raul Mur-Artal i Juan D. Tardos dovršavaju *ORB-SLAM2* metodu za mapiranje prostora koja koristi sve značajke i prednosti *ORB-SLAM* metode te dodaje podršku za stereo i RGB-D kamere [24]. Na Slici 20. prikazani su moduli *ORB-SLAM2* sustava (lijevo) te način procesiranja ulaza (desno).



Slika 20. Moduli *ORB-SLAM2* sustava i način procesiranja ulaza [24]

Kao jedna od prednosti razvijenog *SLAM* algoritma navodi se mogućnost procesiranja podataka i izvođenje procesa lokalizacije i mapiranja prostora u realnom vremenu na procesorskim jedinicama (CPU) koje su česte u svakodnevnoj uporabi. U svom su radu algoritam testirali s raznim javno dostupnim setovima podataka na računalu pokretano Intel Core i7-4790 procesorom s 16GB RAM memorije [24].

Raul Mur-Artal i Juan D. kroz rad pokazuju i kako je uvođenje podrške za stereo i RGB-D kamere uvelike poboljšalo performanse odometrije što se najviše očitovalo u gotovo potpunoj eliminaciji *drifta* na velikim zavojima kod praćenja zadane trajektorije. Usporedba rezultata *ORB-SLAM* i *ORB-SLAM2* algoritma kod estimiranja trajektorije dana je na Slici 21. Crvena trajektorija predstavlja stvarnu putanju dok crna putanja predstavlja estimiranu putanju.

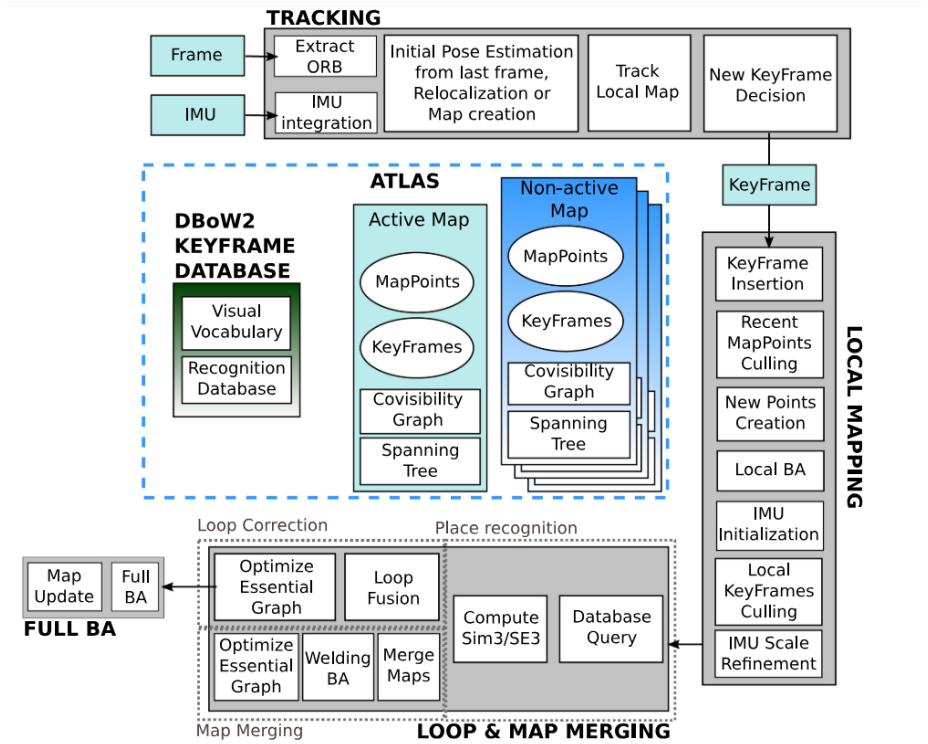


Slika 21. Usporedba *ORB-SLAM* i *ORB-SLAM2* algoritama kod zadatka estimiranja trajektorije [24]

Kao i kod *RTAB-Map*-a, *ORB-SLAM2* također je osjetljiv na brze promjene u sceni ili nagle pokrete kamere. Uz to, algoritam se oslanja na detekciju *ORB* značajki koje se najbolje

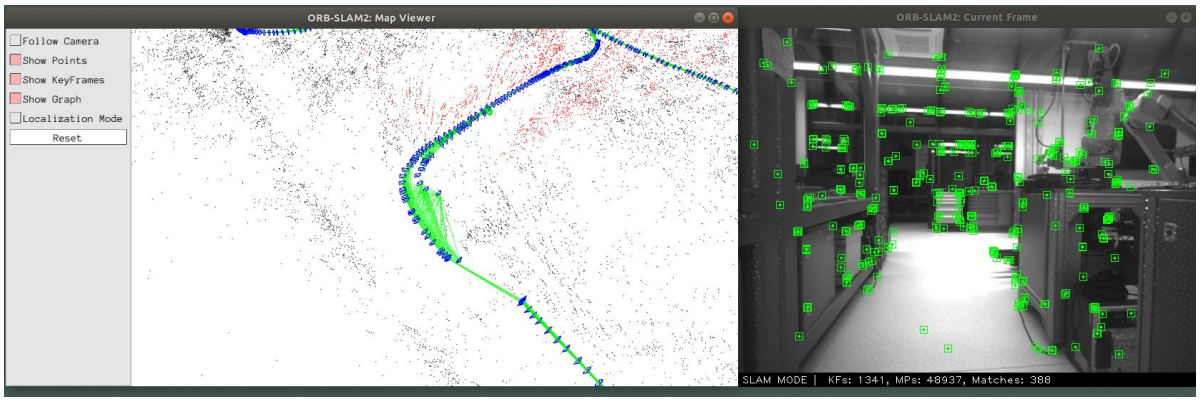
uočavaju kod scena s bogatom teksturom. Stoga do zaglavljivanja ili dobivanja rezultata manje preciznosti može doći ako se sustav nađe u prostoru u kojem je teško detektirati *ORB* značajke. To su prostori poput praznih bijelih zidova, staklene površine, dugi prazni hodnici i površine bez teksture.

Uskoro je uslijedio i razvoj *ORB-SLAM3* sustava koji uključuje integraciju podataka dobivenih sa IMU jedinice kod procesa mapiranja i lokalizacije. Također se kao napredak u odnosu na *ORB-SLAM2* sustav navodi i povećanje robusnosti. Valja napomenuti kako je i ovaj sustav prvi uveo mogućnost korištenja bilo koje kamere ili sustava kamera za provođenje algoritma. Potrebno je samo dodati kameru te upisati parametre koji je određuju te će se *SLAM* algoritam moći izvoditi [25]. Na Slici 22. prikazani su glavni elementi *ORB-SLAM3* sustava.



Slika 22. Glavni elementi *ORB-SLAM3* sustava [25]

Budući da podržava mapiranje bazirano na setu podataka zabilježenim stereo kamerom te ne prihvaća podatke koje bilježi inercijska mjerna jedinica, za usporedbu performansi s *RTAB-Map* algoritmom u ovom radu korišten je *ORB-SLAM2*. Slika 23. prikazuje sučelje *ORB-SLAM2* paketa.



Slika 23. Sučelje *ORB-SLAM2 paketa*

5. IZRADA SKUPA PODATAKA ZA VALIDACIJU ALGORITAMA

Zbog specifičnosti formata podataka koji svaki od korištenih *SLAM* algoritama zahtjeva potrebno je razviti strategiju prikupljanja i prilagođavanja relevantnog skupa za testiranje algoritama. Kako je *RTAB-Map* dio *ROS* programskog okruženja, isti predviđa rad s *.bag* formatom, dok *ORB-SLAM2* zahtjeva segmentaciju zabilježenih slika u određene direktorije i bilježenje vremena (*engl. timestamp*) u kojem je svaka od slika zabilježena te kreiranja odgovarajuće *.txt* datoteke koja povezuje određene slike s određenim trenutkom. Za zadovoljavanje navedenih kriterija odabrano je rješenje inicijalnog bilježenja skupa podataka u *.bag* formatu te izrade kratke *Python* skripte koja služi za prilagođavanje tako zabilježenih podataka formatu koji odgovara *ORB-SLAM* algoritmu.

Za kreiranje *.bag* datoteke korištena je *makebag.launch* skripta dostupna na repozitoriju u prilogu rada. Skripta poziva postojeću *rs_camera.launch* datoteku koja je sastavni dio instaliranog *Intel Realsense ROS* paketa. Kroz skriptu se zatim prilagođavaju potrebni parametri kamere kao što su širina, visina te brzina dohvaćanja slike. Također se omogućuje i dohvaćanje podataka sa žiroskopa i akcelerometra te se korištenjem metode linearne interpolacije isti podaci bilježe u jednu *ROS* temu kako bi bili dostupni za kasniju upotrebu kod provedbe *RTAB-Map* algoritma. Pokretanjem navedene skripte počinje inicijalizacija kamere te započinje objavljivanje svih tema koje su navedene u *rs_camera.launch* skripti. Za potrebu provođenja *RTAB-Map* i *ORB-SLAM2* algoritama potrebno je spremiti samo sljedećih 6 tema:

/camera/color/camera_info

/camera/color/image_raw

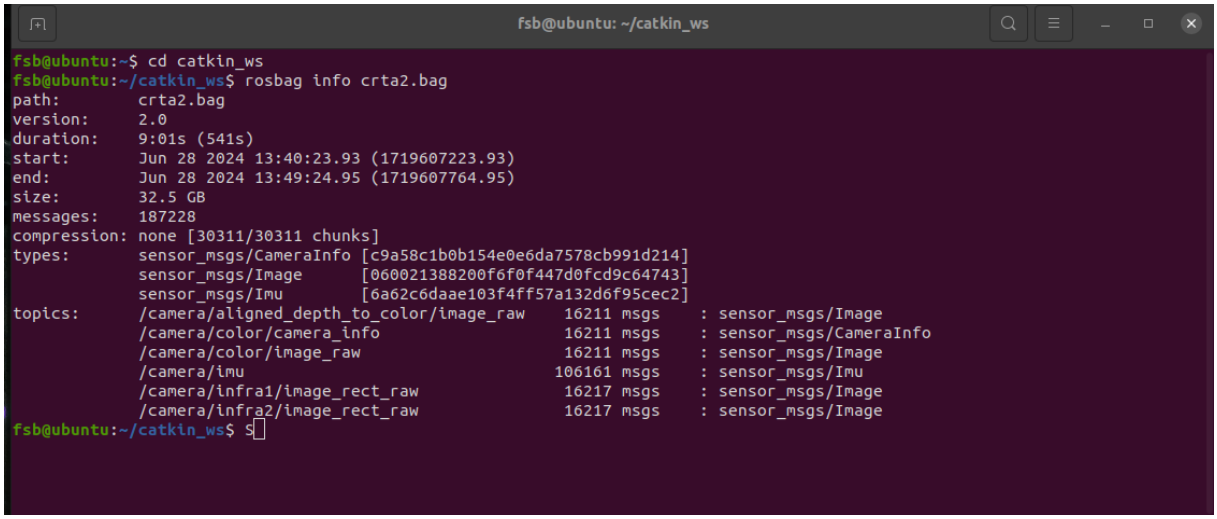
/camera/aligned_depth_to_color/image_raw

/camera/imu

/camera/infra1/image_rect_raw

/camera/infra1/image_rect_raw.

Nakon završetka snimanja moguće je provjeriti strukturu dobivenog skupa podataka korištenjem naredbe *rosbag info.* Prikaz strukture seta podataka korištenog za evaluaciju *SLAM* algoritama prikazan je na Slici 24.



Slika 24. Struktura seta podataka korištenog za evaluaciju *SLAM algoritama*

Kako bi se dobiveni set podataka spremljen u *.bag* formatu prilagodio za provedbu *ORB-SLAM2* algoritma izrađena je *convert.py* skripta dostupna na repozitoriju navedenom u prilogu rada. Za navedeni algoritam potrebno je sortirati fotografije koje bilježe lijeva i desna infracrvena kamera u zasebne direktorije te izraditi *timestamps.txt* datoteku koja služi za povezivanje slika zabilježenih u istom vremenskom trenutku. Također je potrebno izraditi i *.yaml* datoteku u kojoj su navedeni parametri korištene kamere. Za dobivanje intrinzičnih i ekstrinzičnih podataka moguće je koristiti već gotove *OpenCV* metode kalibracije ili kod *Intel RealSense* kamera kalibraciju i očitanje parametara provesti kroz sučelje *Intel RealSense SDK* aplikacije. Radi dodatne validacije parametara u ovom radu provedene su obje metode te su potrebni parametri zapisani u *my_realsense.yaml* datoteci dostupnoj na repozitoriju. Direktorij *ORB_SLAM_DATA* u kojem se nalaze spremljene fotografije i datoteke za provedbu *ORB-SLAM* algoritma treba biti strukturiran na sljedeći način:

```
ORB_SLAM_DATA/
        DATASET_NAME/
                CONFIG.yaml
                timestamps.txt
                images/mav0/
                cam0/data/      * sadrži fotografije lijeve infracrvene kamere
                cam1/data/      * sadrži fotografije desne infracrvene kamere
```

Kreirani skup podataka snimljen je u prostoru laboratorija CRTA-e koji je zbog svog rasporeda pogodan za provjeru *loop-closure* funkcija *SLAM* algoritama. Za evaluaciju algoritama dohvaćane su slike rezolucije 640 *x* 480 piksela kako veličina snimljenog skupa podataka ne bi zauzimala preveliku količinu prostora. Na Slici 25. prikazan je jedan par fotografija dohvaćenih stereo infracrvenim kamerama.

Slika 25. Par slika dohvaćen stereo infracrvenim kamerama

6. IZVOĐENJE SLAM ALGORITAMA

U ovom poglavlju bit će prikazani rezultati provođenja *RTAB-Map* i *ORB-SLAM2* na pripremljenom setu podataka čija je struktura prikazana u prošlom poglavlju. Kao glavni parametri usporedbe bit će predstavljeni broj zabilježenih ključnih točaka (*engl. keypoints)* na cijelom skupu podataka, broj spremljenih ključnih kadrova (*engl. key frames*), broj pronađenih *loop-closure* sekvenci, potrebno vrijeme za procesuiranje skupa podataka te opterećenost CPU i GPU jedinica tokom provedbe algoritama. Zbog različitosti izlaznih podataka koje pružaju *RTAB-Map* i *ORB-SLAM2* algoritmi nije moguće provesti detaljniju usporedbu navedenih *SLAM* metoda te se kao daljnja mogućnost nadogradnje sustava predlaže razvoj algoritma za evaluaciju dobivenih podataka iz obje metode.

6.1. RTAB-Map

Za pokretanje *RTAB-Map* algoritma kreirana je *myrtabmap.launch* skripta za pokretanje u *ROS* programskom okruženju dostupna na repozitoriju u prilogu rada. Skripta pokreće *RTAB-Map* algoritam pozivanjem *rtabmap.launch* skripte koja je sastavni dio instaliranog *ROS RTAB-Map* paketa. Zatim se navode teme i argumenti za izvođenje algoritma. Valja navesti s kojih tema *RTAB-Map* želi dohvaćati podatke te kako će te teme biti mapirane na unutarnje varijable algoritma. Isto je prikazano na sljedećem isječku koda iz *myrtabmap.launch* skripte.

```
<arg name="rtabmap_args" value="--delete_db_on_start"/>
<arg name="depth_topic" value="/camera/aligned_depth_to_color/image_raw"/>
<arg name="rgb_topic" value="/camera/color/image_raw"/>
<arg name="camera_info_topic" value="/camera/color/camera_info"/>
<arg name="imu_topic"  value="/rtabmap/imu"/>
<param name="wait_imu_to_init" value="true"/>
```

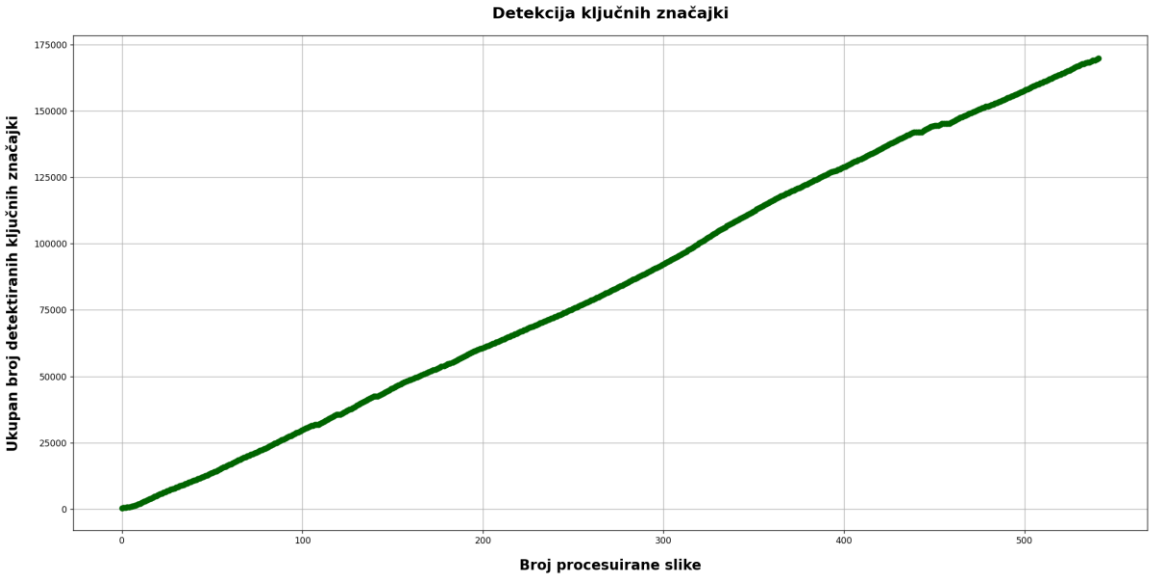
Kako bi se podaci s inercijske mjerne jedinice ispravno formatirali za njihovu uporabu tokom izvođenja algoritma potrebno je postaviti *Madgwick* filter koji se koristi za pretvaranje informacija s akcelerometra i žiroskopa u jedinstveni orijentacijski kvaternion. Za implementaciju istog u *ROS-u* je razvijen posebni paket pod imenom *imu_filter_madgwick* [26]. U ranije spomenutoj skripti za pokretanje *RTAB-Map* algoritma isti se aktivira te se njegovi parametri i argumenti definiraju sa sljedećim skupom naredbi.

```
<node pkg="imu_filter_madgwick" type="imu_filter_node" name="imu_filter_node"
output="screen">
        <param name="use_mag" value="false"/>
        <param name="publish_tf" value="false"/>
        <param name="world_frame" value="enu"/>
        <remap from="/imu/data_raw" to="/camera/imu"/>
        <remap from="/imu/data" to="/rtabmap/imu"/>
</node>
```

Po potrebi je moguće i postaviti definiciju transformacije iz položaja kamere u položaj robota, no za ovaj primjer mapiranja i lokalizacije korišten je standardni položaj koordinatnog sustava kamere.

Nakon pozivanja *myrtabmap.launch* skripte i pokretanja odgovarajuće *.bag* datoteke kreće izvođenje samog algoritma. Za provođenje cijelog postupka mapiranja danog skupa podataka bilo je potrebno 271 sekundi, odnosno manje nego što je samo trajanje *.bag* datoteke. Ta činjenica pokazuje mogućnost korištenja *RTAB-Map* paketa za mapiranje i lokalizaciju u nepoznatom prostoru u stvarnom vremenu što je jedna od njegovih glavnih komparativnih prednosti. Samo sučelje sustava pruža mogućnost spremanja dobivene baze podataka i 3D modela prostora te ispis parametara na grafovima. Iz statističkih podataka je vidljivo kako je za kreiranje mape i vlastite baze podataka iskorištena samo 541 slika, odnosno jedna slika po sekundi trajanja videozapisa. Za praćenje rada algoritma stoga je konstruirano nekoliko grafova koji najbolje pokazuju performanse sustava te pružaju mogućnost usporedbe s drugim algoritmima.

Kao jedan od bitnih pokazatelja rada SLAM algoritma treba navesti ukupan broj detektiranih ključnih značajki na cijelom skupu podataka. Kako broj spremljenih značajki u memoriji raste tako raste i potrebno vrijeme i potrebna računalna snaga za usporedbu trenutno detektiranih značajki s onima zapisanim u bazi podataka. Kod izvođenja *RTAB-Map* algoritma ne dolazi do prevelikog opterećenja sustava ni kod velikog broja zabilježenih ključnih značajki te se isti izvodi u realnom vremenu. Na Slici 26. prikazano je kretanje ukupnog broj detektiranih ključnih značajki na cijelom skupu podataka. S cijelog skupa podataka ukupno je spremljeno 169889 ključnih značajki.

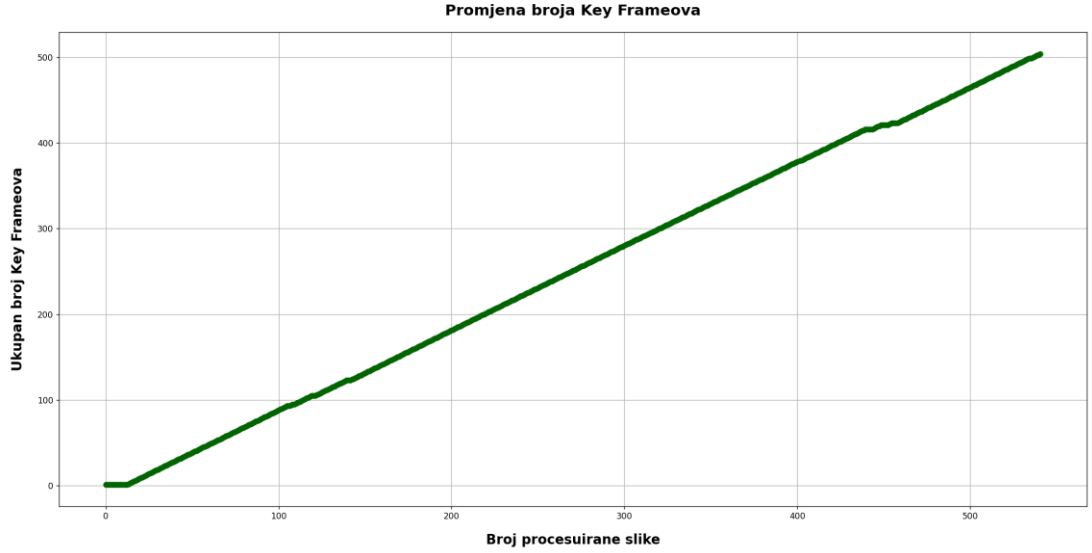


Slika 26. Ukupan broj ključnih značajki detektiran korištenjem *RTAB-Map* algoritma

Kada se sustav nađe u okolini koja je već ranije zabilježena u trenutnu bazu podataka dolazi do detektiranja trenutne pozicije na temelju usporedbe trenutno pronađenih ključnih značajki s onima koje su ranije zabilježene. Prikaz uparivanja trenutnih s već spremljenim značajkama vidljiv je na Slici 27.

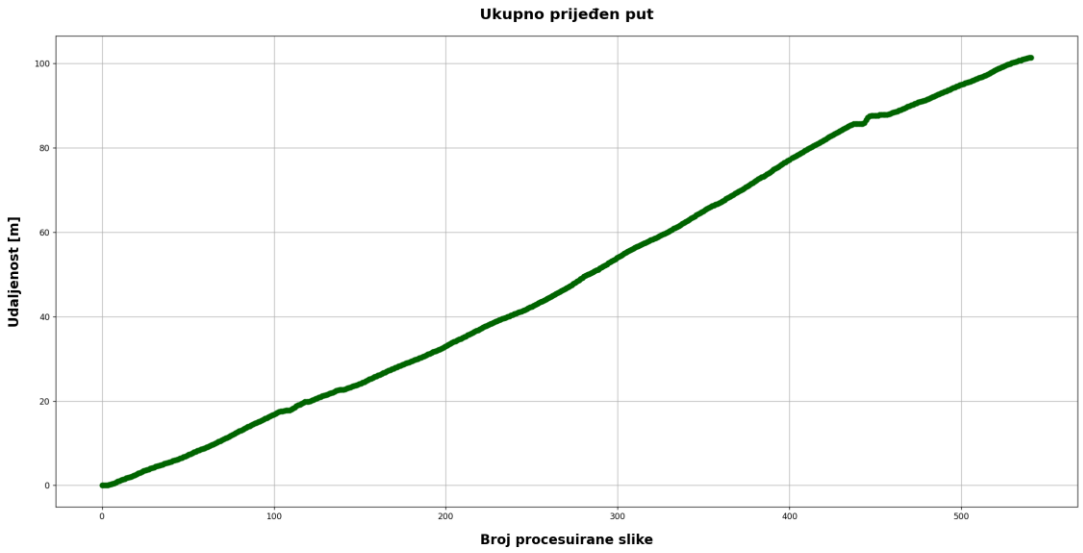
Slika 27. Uparivanje trenutno detektiranih s ranije zabilježenim ključnim značajkama

Valja naglasiti kako algoritam nije spremio svaki od procesuiranih *frameova* kao ključni, već samo određen broj, odnosno 504 za korišteni skup podataka. One kod kojih je pronađena sličnost s prethodno zabilježenim *frameovima*, kao kod primjera na Slici 27., algoritam nije spremio kao nove. Na Slici 28. prikazano je kretanje ukupnog broja *Key Frameova*.



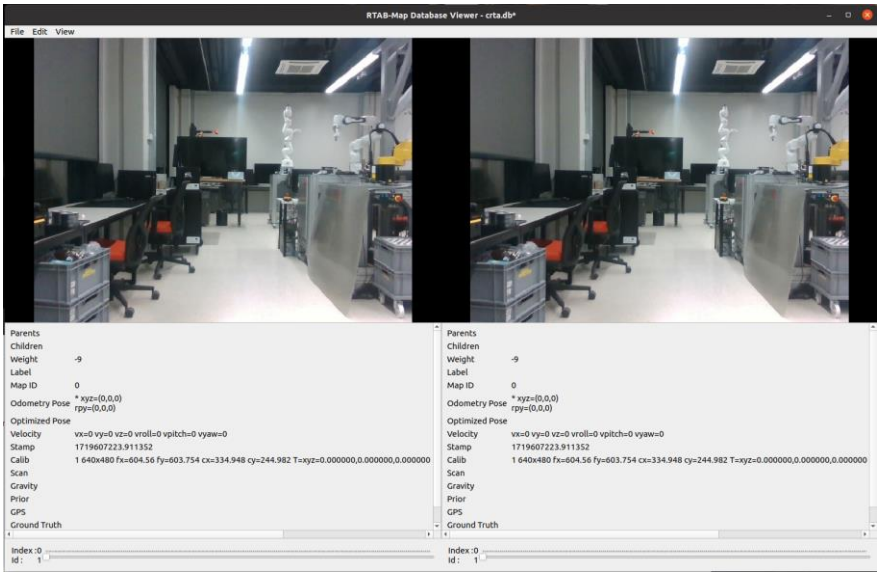
Slika 28. Promjena broja spremljenih *Key Frameova*

Još jedan zanimljiv podatak koji pruža *RTAB-Map* paket jest pregled ukupne pređene udaljenosti. Za slučajeve primjene ovog paketa na već postojećim mobilnim sustavima ovaj podatak mogao bi se usporediti s podacima koje bilježe njihovi enkoderi na kotačima ili slično te na taj način dodatno provjeriti validnost algoritma. Na Slici 29. prikazan je ukupan put koji je pređen za vrijeme mapiranja prostora te isti iznosi 101,4 metra.



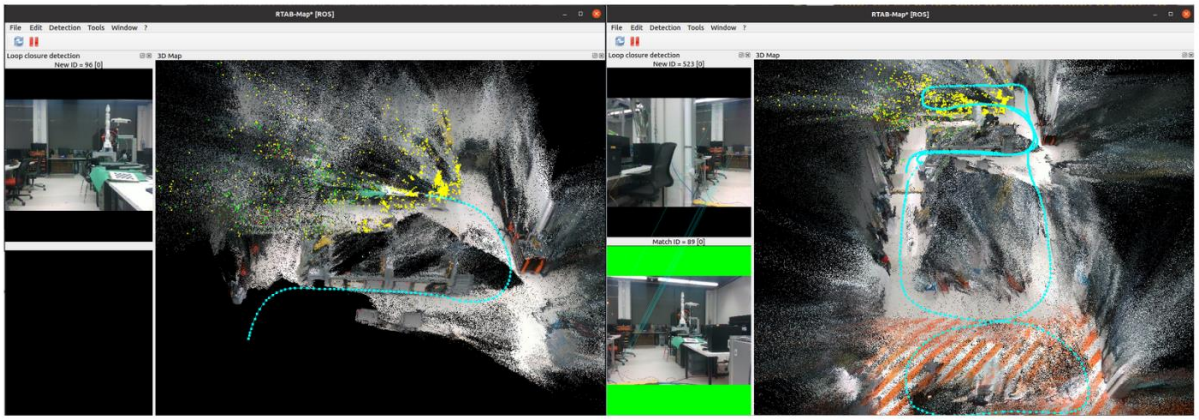
Slika 29. Ukupno prijeđen put tokom procesa mapiranja

Nakon što je dovršena obrada podataka *RTAB-Map* pruža mogućnost spremanja sakupljenje baze podataka u *.db* formatu koji pruža dodatnu mogućnost provjere zabilježenih podataka. Isti se otvara pozivanjem ugrađene *RTAB-Map* naredbe *rtabmap-databaseViewer* uz unošenje imena željene datoteke. Na Slici 30. prikazano je sučelje *RTAB-Map Database Viewer-a.*



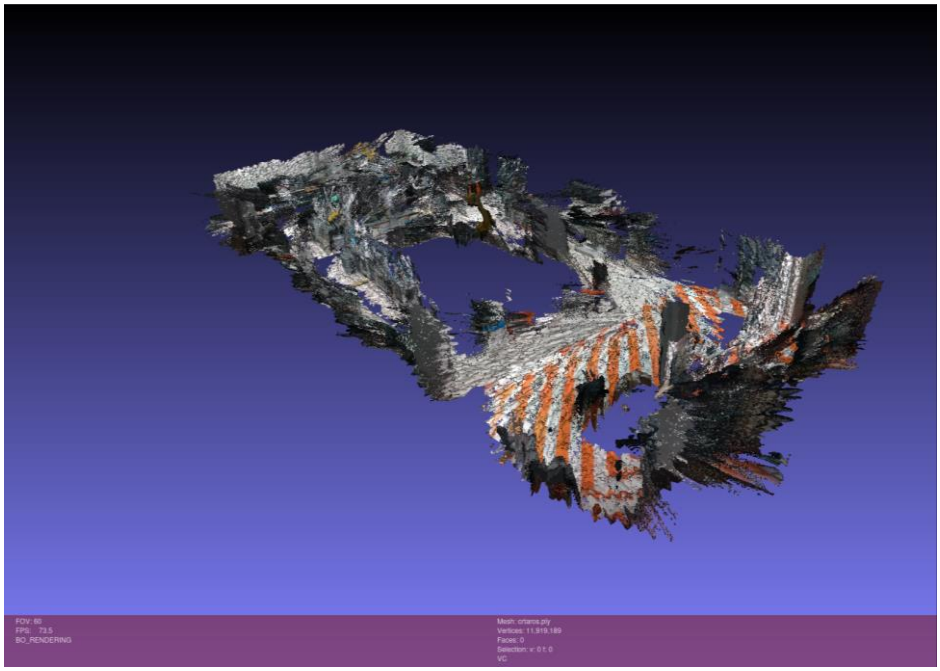
Slika 30. Sučelje *RTAB-Map Database Viewer* aplikacije

Kroz sučelje osnovnog programa moguće je i za vrijeme same provedbe algoritma vidjeti vizualiziranu putanju koju prolazi kamera te oblak točaka koji sprema. Na taj način moguće je provjeriti ispravnost mapiranja u stvarnom vremenu što je velika prednost kod implementacije ovakvog algoritma u rad autonomnih sustava. Na Slici 31. prikazano je sučelje osnovne *RTAB-Map* aplikacije tokom i po završetku mapiranja laboratorija. Svijetlo plavom bojom označena je putanja kamere.



Slika 31. Sučelje osnovne *RTAB-Map* aplikacije tokom i nakon mapiranja prostora

Također je moguće dobiveni oblak točaka spremiti u *.ply, .pcb* ili *.obj* formatu te ga kasnije obrađivati i pregledavati u nekom od za to namijenjenih programa. Prije spremanja moguće je podesiti parametre i izvršiti jednostavniju obradu samog oblaka točaka. U ovom slučaju ostavljeni su predodređeni parametri te je oblak točaka spremljen u *.ply* formatu. Na Slici 32. prikazan je spremljeni oblak točaka u programu *Meshlab.*



Slika 32. Prikaz spremljenog oblaka točaka u programu *Meshlab*

6.2. ORB-SLAM2

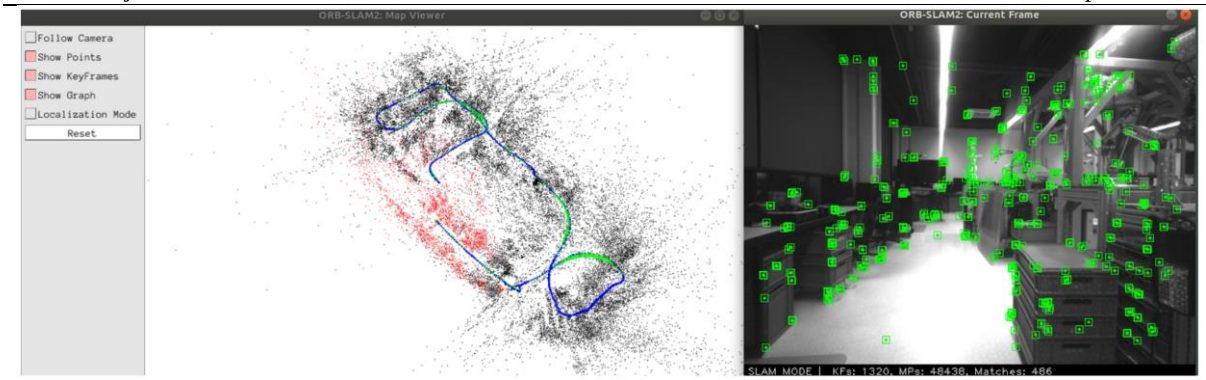
Za pokretanje *ORB-SLAM2* algoritma potrebno je pravilno organizirati direktorij s raspoređenim fotografijama koje su bilježile lijeva i desna infracrvena kamera kao što je pokazano u poglavlju 5. Nakon toga potrebno je iz terminala pozvati naredbu koja redom navodi putanju do datoteke s uputama za izvođenje algoritma (stereo ili mono verzija), putanju do datoteke koja sadrži inicijalni rječnik koji opisuje značajke, putanju do datoteke s parametrima korištene kamere, putanju do direktorije gdje se nalaze fotografije lijeve pa zatim desne infracrvene kamere te putanju do tekstualne datoteke u kojoj su navedena vremena preko kojih se povezuju lijeva i desna slika budući da je odgovarajući par slika okinut istovremeno. U nastavku je dana naredba za pokretanje *ORB-SLAM2* algoritma za zabilježen set podataka s *Intel RealSense D435i* kamerom čiji su parametri navedeni u *myrealsense.yaml* datoteci koja je dostupna na repozitoriju navedenom u prilogu rada.

```
./Stereo/run_stereo Vocabulary/ORBvoc.txt CameraParams/myrealsense.yaml
~/Datasets/crta2orb/mav0/cam0/data ~/Datasets/crta2orb/mav0/cam1/data Timestamps/crta.txt
```

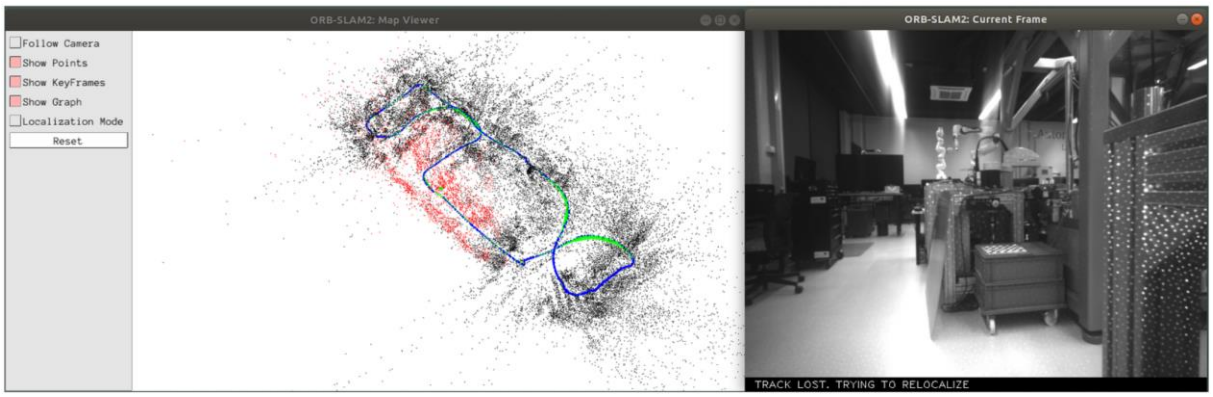
Nakon potvrde unosa naredbe u terminalu kreće izvođenje samog algoritma. Vrijeme potrebno za provođenje cijelog algoritma na zabilježenom setu podataka iznosi 664 sekunde što je duže nego što je bilo potrebno da se prikupi navedeni set podataka. Iz istog se može zaključiti kako izvođenje ovog algoritma u stvarnom vremenu može dovesti do poteškoća zbog nemogućnosti dovoljno brze obrade podataka. Na isto se može utjecati smanjenjem broja dohvaćenih slika u sekundi *(engl. frames per second, FPS*) ili smanjenjem rezolucije.

Za razliku od *RTAB-Map*a, *ORB-SLAM2* algoritam ne pruža velik broj izlaznih podataka. Točnije, moguće je samo iz grafičkog sučelja očitati broj ključnih značajki detektiranih na trenutnoj slici, ukupan broj detektiranih ključnih značajki te ukupan broj *key frameova.* Po završetku izvođenja algoritma broj ukupno detektiranih ključnih značajki iznosi 50671, a broj ukupno spremljenih *key frameova* iznosi 1419.

Kod izvođenja ovog algoritma moglo se primijetiti odstupanje kod bilježenja trenutne pozicije prije povratka u početnu točku i provođenja *loop-closure* sekvence. Budući da ovaj algoritam ne koristi podatke dobivene s inercijske mjerne jedinice moguće je da podaci dobiveni vizualnom odometrijom nisu dovoljni kako bi pozicija kamere bila točno određena u svakom trenutku. Isto ne predstavlja problem u slučaju snimanja mape prostora prije nego što se u istom nađe neki autonomni sustav budući da se po završetku algoritma navedena greška ispravi korištenjem *loop-closure* mehanizma, no kod mapiranja i lokalizacije u realnom vremenu ovakav rad mogao bi predstavljati problem. Na Slici 33. prikazana je mapa koju je algoritam stvorio trenutak prije, a na Slici 34. mapa dobivena nakon provedbe *loop-closure* mehanizma. Iz iste je vidljiva poveća greška kod aproksimacije pozicije prije provedbe navedenog mehanizma.



Slika 33. Mapa prostora prije provedbe *loop-closure* mehanizma



Slika 34. Mapa prostora nakon provedbe *loop-closure* mehanzima

6.3. Usporedba provedenih *SLAM* algoritama

Za provedbu usporedbe navedenih algoritama bit će uzeti samo oni podaci koje je moguće dobiti za oba implementirana algoritma. Također valja navesti kako postoji bitna razlika i u samom radu algoritama s obzirom na tip podataka koji koriste. Dok *ORB-SLAM2* algoritam za odometriju koristi samo podatke dobivene iz stereo ili mono kamera, *RTAB-Map* uz mogućnost korištenja podataka dobivenih s kamera pruža i mogućnost integracije podataka koje bilježi inercijska mjerna jedinica. U ovom je radu korištena kamera koja sama posjeduje inercijsku mjernu jedinicu, no moguće je razviti i sustav koji inercijsku mjernu jedinicu, odnosno žiroskop i akcelerometar ima odvojeno od vizijskog sustava. Sama integracija podataka s inercijske mjerne jedinice u teoriji već povećava točnost odometrije te je za očekivati bolje performanse *RTAB-Map* algoritma. U Tablici 6. dana je usporedba osnovnih podataka koje je bilo moguće prikupiti kod izvođenja *RTAB-Map* i *ORB-SLAM2* algoritama.

Tablica 6. Usporedba *RTAB-Map* i *ORB-SLAM2* algoritama

	RTAB-Map	***ORB-SLAM2***
VRIJEME IZVOĐENJA	271 *s*	664 *s*
MAKSIMALNO OPTEREĆENJE CPU JEDINICE	64 %	100 %
MAKSIMALNO OPTEREĆENJE GPU JEDINICE	72 %	75 %
UKUPAN BROJ KLJUČNIH KADROVA	504	1419
UKUPAN BROJ KLJUČNIH ZNAČAJKI	169889	50671

Već se iz podataka o vremenu izvođenja može lako zaključiti kako *RTAB-Map* algoritam pokazuje znatno bolje performanse kod provođenja na zabilježenom setu podataka. Također je vidljivo kako je, iako je zabilježen manji broj ključnih kadrova (*engl. key frames*), ukupan broj zabilježenih ključnih značajki znatno veći kod *RTAB-Map* algoritma. Stoga slijedi kako, uz uvjet da se ne preoptereti sustav, bilježenje većeg broja značajki ne utječe negativno na performanse algoritma. Budući da korišteni *NVIDIA Jetson Xavier NX* koristi *CUDA* jezgre za brzo procesuiranje podataka, a *OpenCV* biblioteka izgrađena je s *CUDA* podrškom (*cmake -D WITH_CUDA=ON*, vidljivo je kako oba algoritma kod izvođenja podjednako opterećuju GPU.

Treba naglasiti i prethodno spomenutu činjenicu da je kod mapiranja prostora ORB-SLAM2 algoritmom u jednom trenutku došlo do nakupljanja veće greške prije nego što je izvršen loop-closure mehanizam, a što je vidljivo na Slici 34. Kod provedbe *RTAB-Map* algoritma do sličnog odstupanja nije došlo ni u jednom trenutku. Budući da *RTAB-Map* koristi fuziju vizualnih podataka s podacima zaprimljenim od inercijske mjerne jedince, točnost izvođenja samog algoritma je povećana.

Iz dobivenih rezultata očituje se prednost korištenja *SLAM* algoritama koji za odometriju koriste fuziju podataka s više senzora, u ovom slučaju povezivanjem podataka sa stereo slike s podacima koje šalje IMU *RTAB-Map* daje znatno bolje rezultate bez pojavljivanja greške u trenutno estimiranoj poziciji.

6.4. Mogućnosti poboljšanja sustava

S ciljem osiguravanja mogućnosti bolje usporedbe performansi algoritama od velikog bi značaja bila mogućnost dohvaćanja većeg broja relevantnih podataka kod izvođenja *ORB-SLAM2* algoritma. Budući da je isti dostupan pod *open-source* licencom na *GitHub* repozitoriju [27] analiza i adaptacija osnovnog koda aplikacije te pružanje mogućnosti pristupa većem broju podataka mogao bi biti jedan od smjerova za unapređenje mogućnosti usporedbe algoritama.

Također, budući da CRTA laboratorij već koristi *OptiTrack* tehnologiju za praćenje pokreta, kao dodatna mogućnost validacije *SLAM* algoritama predstavlja se i usporedba stvarne trajektorije kamere (odnosno ona koju bilježi *OptiTrack* sustav) s trajektorijom dobivenom provođenjem algoritama. Na taj način mogla bi se konkretno i objektivno ispitati točnost pozicioniranja i zabilježene trajektorije. Uz to, mobilni roboti *ASTRO (Autonomous System for Teaching RObotics)* razvijeni u CRTA laboratoriju već su opremljeni *NVIDIA Jetson* računalom, *Intel RealSense* kamerom te potrebnim markerima za *OptiTrack* lokalizaciju. Kao sljedeći korak nadogradnje sustava stoga se predlaže implementacija *SLAM* algoritama na postojeću konfiguraciju *ASTRO* robota te simultano korištenje flote robota za mapiranje i lokalizaciju prostora.

7. ZAKLJUČAK

Orijentacija i navođenje u nepoznatoj okolini i dalje predstavlja fundamentalni iskorak koji je potrebno napraviti kako bi se postigla potpuna autonomija robotskih i mobilnih sustava. Budući da se od robotskih sustava očekuje izuzetna točnost i osiguranje sigurnosti u njihovoj radnoj okolini, *SLAM* algoritmi, koji pružaju mogućnost stvaranja preciznih 3D modela i dinamičkih okolina, nude osiguranje navedenih zahtjeva kroz kontinuiranu prilagodbu već zabilježenih podataka te estimaciju i korekciju stvarne pozicije sustava u realnom vremenu. ZA povećanje pouzdanosti i točnosti mapiranja i navigacije koriste se podaci s više različitih senzora kao što su kamere, LiDAR i IMU. Navedeni senzori već su duže vrijeme javno dostupni pa se tako ubrzao i razvoj *SLAM* algoritma kako na istraživačkim projektima tako i u industriji.

U ovom radu prikazan je i uspoređen rad dvaju *SLAM* algoritama otvorenog pristupa (*engl. open source*) koji svoj rad baziraju na različitom tipu ulaznih podataka te različitim algoritmima obrade istih. *RTAB-Map* kao ulaz može zaprimati podatke s kamera, LiDAR-a i IMU-a dok *ORB-SLAM2* algoritam radi samo s podacima zaprimljenih s kamera. Kroz rad je potvrđena premisa i pokazana prednost korištenja fuzije podataka s više senzora kod mapiranja i lokalizacije u nepoznatom prostoru. Implementirani *RTAB-Map* algoritam pokazao je veću točnost te u kraćem vremenu proveo proces mapiranja prostora. Kako se korištenje ovih algoritama predviđa za mobilne sustave, razvijeno rješenje za mapiranje prostora izvedeno je korištenjem *NVIDA Jetson Xavier NX* računala malih dimenzija koje je lako moguće montirati na već postojeće sustave. Predložena je i jedna mogućnost montaže s kućištem za *Jetson* računalo na koje je moguće pričvrstiti i korištenu *Intel RealSense D435i* kameru.

Kao idući korak u implementaciji i validaciji ovakvog sustava predlaže se postavljanje *SLAM* algoritama na *ASTRO (Autonomous System for Teaching RObotics)* mobilne robote razvijene u CRTA-i koji već koriste *NVIDA Jetson* računala te su opremljeni *Intel RealSense* kamerama. Uz to, isti su opremljeni i *OptiTrack* sustavom za praćenje pokreta pa bi se pozicija koju estimira *SLAM* algoritam mogla uspoređivati s njihovom stvarnom pozicijom u prostoru, Također je moguće implementirati i algoritme za simultano mapiranje prostora koje bi u tom slučaju mogla provoditi cijela flota robota.

LITERATURA

[1] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/ Pristupano 25.6.2024.

[2] https://www.intelrealsense.com/depth-camera-d435i/ Pristupano 25.6.2024.

[3] https://ubuntu.com/about Pristupano 25.6.2024.

[4] O'Kane J.M., A Gentle Introduction to ROS, 2014.

[5] ROS/Introduction - ROS Wiki Pristupano 25.6.2024.

[6] Introduction to ROS | VNAV (mit.edu) Pristupano 25.6.2024.

[7] http://wiki.ros.org/noetic Pristupano 25.6.2024.

[8] Rosten E., Drummond T., Machine learning for high-speed corner detection, 2006.

[9] https://www.baeldung.com/cs/image-processing-feature-descriptors Pristupano 25.6.2024.

[10] https://docs.nvidia.com/vpi/algo_orb_feature_detector.html Pristupano 25.6.2024.

[11] Shi Jianbo, Tomasi C., Good Features to Track, 1994.

[12] Alahi A., Ortiz R., Vandergheynst P., FREAK: Fast Retina Keypoint, *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[13] Lowe D., Object Recognition from Local Scale-Invariant Features, 1999.

[14] Best P.J., A method for registration of 3-D shapes, 1992.

[15] Rusinkiewicz, S., Levoy, M.. Efficient variants of the ICP algorithm. Proceedings Third International Conference on 3-D Digital Imaging and Modeling. 2001.

[16] Baker S., Matthews, I. Lucas-Kanade 20 years on: A unifying framework, 2004.

[17] Park J., Zhou Qian-Yi, Koltun V., Colored Point Cloud Registration Revisited, 2017.

[18] Labbe M., Michaud F., Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation. *IEEE Transactions on Robotics, 29(3),* 2013.

[19] Labbé M., Michaud F., RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation, *Journal of Field Robotics,* 2018.

[20] Stitch M. L., Woodburry E. J., Morse J. H., Optical ranging system uses laser transmitter, 1961.

[21] Galvez-Lopez D., Real-Time Loop Detection with Bags of Binary Words, 2011.

[22] https://www.thinkautonomous.ai/blog/loop-closure/ Pristupano 25.6.2024.

[23] Mur-Artal R., Montiel J. M. M., Tardos J. D., ORB-SLAM: A Versatile and Accurate Monocular SLAM System, *IEEE Transactions on Robotics, 31(5),* 2015.

[24] Mur-Artal R., Tardos J. D., ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras, *IEEE Transactions on Robotics, 33(5)*, 2017.

[25] Campos C., Elvira R., Rodriguez J. J. G., Montiel J. M., D Tardos, J., ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM, *IEEE Transactions on Robotics,* 2021.

[26] https://wiki.ros.org/imu_filter_madgwick Pristupano 29.6.2024

[27] https://github.com/raulmur/ORB_SLAM2 Pristupano 29.6.2024

PRILOZI

I. *GitHub* repozitorij - IvanStrahija/JetsonXavierNX_SLAM: Diplomski rad - Razvoj mobilnog 3D skenera za mapiranje prostora (github.com)