



UNIVERZITET U SARAJEVU  
ELEKTROTEHNIČKI FAKULTET  
ODSJEK ZA AUTOMATIKU I ELEKTRONIKU

---

**Implementacija algoritma za praćenja  
zidova i lokalizaciju X100 mobilnog robota  
na osnovu mjerena 3D LiDAR senzora**

---

ZAVRŠNI RAD  
- PRVI CIKLUS STUDIJA -

**Student:**  
**Azra Aladžuz**

**Mentor:**  
**Red. prof. dr Jasmin Velagić**

Sarajevo,  
septembar 2023.

## Sažetak

U ovom radu opisana je implementacija algoritma za praćenje zida i lokalizaciju putem mobilnog robota X100 opremljenog Velodyne 3D LiDAR senzorom. Cilj algoritma je omogućiti autonomnu navigaciju robota uz održavanje određene udaljenosti od zida te povratak u početnu poziciju. Algoritam koristi podatke iz Velodyne 3D LiDAR-a za detekciju i praćenje zidova. Robot X100 je opremljen potrebnom hardverskom i softverskom infrastrukturom za obradu LiDAR podataka i izvršavanje algoritma. U prvom dijelu rada pružene su teoretske osnove o ROS-u, mobilnom robotu X100, 3D LiDAR senzoru i lokalizaciji. Programiranje mobilnog robota je izvedeno u C++ stilu, a komunikacija s robotom ostvarena je putem robotskog operativnog sistema ROS. Dato je detaljno objašnjenje koda i upotrebljenih ROS paketa te su predstavljeni rezultati koji demonstriraju učinkovitost i preciznost implementiranog algoritma te sposobnost robota X100 za autonomno upravljanje i lokalizaciju.

## Abstract

This paper describes the implementation of an algorithm for wall tracking and localization using mobile robot X100 equipped with a Velodyne 3D LiDAR sensor. The goal of the algorithm is to enable autonomous navigation of the robot while maintaining a certain distance from the wall and returning to the initial position. The algorithm uses data from the Velodyne 3D LiDAR to detect and track walls. The X100 robot is equipped with the necessary hardware and software infrastructure for processing LiDAR data and executing algorithms. In the first part of the paper, the theoretical foundations of ROS, mobile robot X100, 3D LiDAR sensor and localization are provided. Programming of the mobile robot was done in C++ style, and communication with the robot was achieved through the ROS robot operating system. A detailed explanation of the code and used ROS packages is given, and results are presented that demonstrate the efficiency and precision of the implemented algorithm and the ability of the X100 robot for autonomous control and localization.

**Elektrotehnički fakultet, Univerzitet u Sarajevu**  
**Odsjek za Automatiku i elektroniku.**  
**Redovni prof. dr. Jasmin Velagić, dipl.el.ing**  
**Sarajevo, 20.9.2023.**

## Postavka zadatka završnog rada I ciklusa:

U radu je potrebno:

- Objasniti lokalizaciju mobilnog robota i problem praćenja zida na odgovarajućem rastojanju u toku kretanja mobilnog robota.
- Opisati osnovne pakete ROS-a koji će se koristiti u završnom radu.
- Implementirati algoritam koji omogućuje istovremeno praćenje zida i lokalizaciju mobilnog robota sa vraćanjem u početnu tačku unutar ROS-a korištenjem C++ programskog jezika.
- Verificirati i testirati algoritam u simulacijskom modu unutar Gazebo simulatora i eksperimentalno sa stvarnim modelom XMachine X100 mobilnim robotom.

### Polazna literatura:

- [1] J. Velagić, Mobilna robotika, Univerzitet u Sarajevu, Elektrotehnički fakultet, 2012.
- [2] I. Belkin, A. Abramenko and D. Yudin, “Real-Time Lidar-based Localization of Mobile Ground Robot,” Procedia Computer Science, vol. 186, pp. 440-448, 2021.
- [3] K. Bayer, Wall Following for Autonomous Navigation, Columbia University, 2012.

---

Red. prof. dr Jasmin Velagić, dipl. ing. el.

## **Izjava o autentičnosti radova**

### **Završni rad**

### **I ciklusa studija**

Ime i prezime: Azra Aladžuz

Naslov rada: Implementacija algoritma za praćenja zidova i lokalizaciju X100 mobilnog robota na osnovu mjerjenja 3D LiDAR senzora

Vrsta rada: Završni rad Prvog ciklusa studija

Broj stranica: 51

#### **Potvrđujem:**

- da sam pročitala dokumente koji se odnose na plagijarizam, kako je to definirano Statutom Univerziteta u Sarajevu, Etičkim kodeksom Univerziteta u Sarajevu i pravilima studiranja koja se odnose na I i II ciklus studija, integrirani studijski program I i II ciklusa i III ciklus studija na Univerzitetu u Sarajevu, kao i uputama o plagijarizmu navedenim na web stranici Univerziteta u Sarajevu;
- da sam svjesna univerzitetskih disciplinskih pravila koja se tiču plagijarizma;
- da je rad koji predajem potpuno moj, samostalni rad, osim u dijelovima gdje je to naznaceno;
- da rad nije predat, u cjelini ili djelimično, za stjecanje zvanja na Univerzitetu u Sarajevu ili nekoj drugoj visokoškolskoj ustanovi;
- da sam jasno naznačila prisustvo citiranog ili parafraziranog materijala i da sam se referirala na sve izvore;
- da sam dosljedno navela korištene i citirane izvore ili bibliografiju po nekom od preporučenih stilova citiranja, sa navođenjem potpune reference koja obuhvata potpuni bibliografski opis korištenog i citiranog izvora;
- da sam odgovarajuće naznačila svaku pomoć koju sam dobila pored pomoći mentora i akademskih tutora/ica.

Sarajevo, 20.9.2023.

Potpis:

---

Azra Aladžuz

# Sadržaj

<b>Popis slika</b>	<b>vi</b>
<b>1 Uvod</b>	<b>1</b>
1.1 Opis problema . . . . .	1
1.2 Pregled područja istraživanja . . . . .	1
1.3 Motivacija . . . . .	2
1.4 Zadaci i ciljevi istraživanja . . . . .	2
1.5 Struktura rada . . . . .	3
<b>2 Robotski operativni sistem - ROS</b>	<b>4</b>
2.1 Uvod . . . . .	4
2.2 Pojava ROS-a . . . . .	5
2.3 Dizajn ROS-a . . . . .	6
2.3.1 Čvorovi . . . . .	7
2.3.2 Teme . . . . .	7
2.3.3 Usluge/servisi . . . . .	7
2.4 ROS alati . . . . .	8
2.4.1 roscore . . . . .	8
2.4.2 rosrun . . . . .	8
2.4.3 roslaunch . . . . .	8
2.4.4 rostopic . . . . .	8
2.4.5 rviz . . . . .	8
<b>3 Mobilni roboti</b>	<b>9</b>
3.1 Uvod . . . . .	9
3.2 Podjela mobilnih robota . . . . .	10
3.3 X100 mobilni robot . . . . .	11
3.3.1 Dizajn i tehničke specifikacije . . . . .	11
3.3.2 Dodaci . . . . .	12
3.4 Aplikacije . . . . .	14
<b>4 LiDAR senzor</b>	<b>16</b>
4.1 Uvod . . . . .	16
4.2 Princip rada . . . . .	16
4.3 Tipovi LiDAR senzora . . . . .	17
4.4 Aplikacije . . . . .	18
4.5 Velodyne LiDAR . . . . .	19

<b>5 Lokalizacija</b>	<b>20</b>
5.1 Uvod . . . . .	20
5.2 Lokalizacija koja se oslanja na referentne tačke . . . . .	20
5.3 Odometrija . . . . .	21
5.3.1 Mjerenje odometrijskih pogrešaka . . . . .	23
5.4 Simultana lokalizacija i mapiranje SLAM . . . . .	23
<b>6 Implementacija algoritma za praćenje unutar ROS-a</b>	<b>24</b>
6.1 Uvod . . . . .	24
6.2 Paket laser_scan_matcher . . . . .	24
6.3 Paket tf . . . . .	26
6.4 Paket geometry_msgs . . . . .	28
6.5 Algoritam za upravljanje robotom . . . . .	29
<b>7 Testiranje algoritma</b>	<b>32</b>
7.1 Uvod . . . . .	32
7.2 Gazebo simulator . . . . .	32
7.3 Testiranje algoritma u okviru Gazebo okruženja . . . . .	33
7.4 Testiranje na stvarnom sistemu . . . . .	35
<b>8 Zaključak</b>	<b>38</b>
<b>9 Dodatak</b>	<b>39</b>
<b>Literatura</b>	<b>44</b>

# Popis slika

2.1	PR1 robot . . . . .	5
2.2	Rqt graf sa prikazanim temama i čvorovima . . . . .	6
3.1	X100 mobilni robot . . . . .	11
3.2	2D LiDAR senzor . . . . .	12
3.3	UR3e robotska ruka . . . . .	14
4.1	Mapa koju generiše LiDAR senzor . . . . .	17
4.2	3D mapa jezera George, SAD, izrađena korištenjem barimetrijskog LiDAR-a . . . . .	18
4.3	Velodyne Puck LiDAR senzor . . . . .	19
5.1	Elipse pogreške . . . . .	22
7.1	X100 unutar Gazebo okruženja . . . . .	33
7.2	X100 unutar Gazebo okruženja . . . . .	34
7.3	Generirana mapa unutar Rviz-a . . . . .	35
7.4	Generirana mapa okruženja unutar Rviz-a . . . . .	36
7.5	Generirana mapa okruženja unutar Rviz-a . . . . .	37

# Poglavlje 1

## Uvod

Uvodno poglavlje rada započet će kratkim opisom problema istraživanja, gdje su navedeni izazovi autonomnog praćenja zidova i određivanja lokacije mobilnog robota u toku kretanja (lokalizacija). Dat je uvid u temeljne motive za ovo istraživanje, te osvrт na postojeću literaturu i područje istraživanja rada. Na kraju je dat pregled strukture rada, gdje su navedena glavna poglavlja uz kratko objašnjenje njihovog sadržaja.

### 1.1 Opis problema

U području robotike i autonomne navigacije, sposobnost učinkovitog kretanja i interakcije s okolinom od velike je važnosti. Ovaj rad tretira problematiku implementacije algoritma dizajniranog za rješavanje zadatka praćenja zidova i lokalizacije pomoću mobilnog robota X100, opremljenog Velodyne 3D LiDAR senzorom. Temeljni problem usmjeren je na orkestriranje robotskog autonomnog slijedenja zidova na sigurnom odstojanju. Iako se ovo može, na prvi pogled, činiti jednostavnim, ono predstavlja višestruki izazov. Robot ne samo da mora održavati određenu udaljenost od zidova, već mora imati i sposobnost vraćanja u početni položaj. Postizanje ove razine autonomne preciznosti ključno je za veliki broj aplikacija u stvarnom svijetu, od zadatka nadzora i inspekcije do autonomne navigacije opće namjene u unutarnjim i vanjskim okruženjima. Mobilni robot X100, svestrana i sposobna platforma, odabran je za ovo istraživanje upravo zbog svojih hardverskih i softverskih komponenti koje omogućavaju učinkovitu obradu LiDAR podataka i izvršavanje algoritma praćenja i lokalizacije.

### 1.2 Pregled područja istraživanja

Robotika je svjedočila nevjerljivom napretku u posljednjih nekoliko desetljeća, a autonomna navigacija jedna je od njezinih najistaknutijih i najizazovnijih domena. Sposobnost robota da samostalno istražuje i upravlja nepoznatim okruženjima nije samo ključna za praktične primjene, već je i simbol tehnološkog napretka. Ključ za ovaj pothvat je razvoj robusnih algoritama koji mogu obraditi senzorne podatke, donositi odluke na inteligentan način i izvršavati precizne pokrete. Središnji dio ovog rada jeste robotski operativni sistem ROS, široko prihvaćen alat koji olakšava razvoj složenih robotskih sistema [1]. ROS pruža bogat skup alata, biblioteka i konvencija za izradu i upravljanje robotima, što ga čini nezamjenjivom platformom za ovaj rad. Istraživači diljem svijeta intenzivno su koristili ROS za različite primjene, od jednostavnih mobilnih robota do složenih čovjekolikih robota. Dobar uvid u evoluciju i razvoj ROS-a kroz vrijeme pruža [2]. Brojne studije razjasnile su arhitekturu i komponente ROS-a, uključujući

jući njegov komunikacijski sistem, upravljanje paketima i alate za vizualizaciju. Značajan dio istraživanja usmjeren je na korištenje ROS-a za platforme mobilnih robota, a upravo [3] daje uvod u programiranje i dizajn mobilnih robota korištenjem ROS-a. Ove studije s fokusom na mobilne robote, istražuju teme kao što su lokalizacija, mapiranje, planiranje puta i izbjegavanje prepreka. Kako ROS olakšava integraciju različitih senzora, uključujući LiDAR senzore, razvijeni su paketi i biblioteke za pokretače senzora, obradu podataka i vizualizaciju, ključni za zadatke percepcije i navigacije. Razumijevanje arhitekture ROS-a, te informacije, upute i tutorijale za sve aspekte ovog operativnog sistema pružaju [4] i [5]. Ono što je ključno za ovaj rad jesu područja mobilne robotike i lokalizacije, čiju teorijsku podlogu daju [6] i [7], dok osnovne informacije o X100 mobilnom robotu daje [8]. Mjereni podaci sa LiDAR senzora sastavni su dio algoritama za praćenje zidova i lokalizaciju. Tokom godina istraživanje na području korištenja LiDAR podataka za mapiranje, otkrivanje prepreka i lokalizaciju, znatno je napredovalo i uključuje literaturu [9], [10], [11], [12] i [13]. Iako postojeća literatura pruža dobar uvid u razvoj robotike temeljen na ROS-u, praćenje zidova i algoritme lokalizacije, postoji primjetan nedostatak u istraživanju koje se posebno bavi integracijom ROS-a, mobilnih robota poput X100 i Velodyne LiDAR senzora. Ovaj nedostatak služi kao značajna motivacija za ovaj rad, s ciljem implementacije novog algoritma pisanog upravo za ove komponente.

## 1.3 Motivacija

Autonomna navigacija predstavlja temelj moderne robotike i postaje neizostavan dio mnogih aplikacija. Bilo da se radi o industrijskoj automatizaciji, operacijama potrage i spašavanja ili istraživanju svemira, sposobnost robota da samostalno istražuju, upravljaju i komuniciraju s okolinom mnogo obećavaju u budućnosti. Ne samo da povećava učinkovitost, već i proširuje doseg u domene u kojima ljudska prisutnost može biti nepraktična ili nesigurna. U autonomnoj navigaciji, izazov praćenja zidova ili površina uz održavanje određene udaljenosti ostaje složen problem. Nadalje, osiguravanje da robot može pratiti svoj položaj i vratiti se u svoju početnu poziciju dodaje još jedan sloj složenosti. Postojeća rješenja često ne uspijevaju integrirati ROS, mobilne robote poput X100 i Velodyne LiDAR senzore za rješavanje ovih izazova. Ovaj rad motiviran je potrebom za razvojem algoritma koji iskorištava snagu ROS-a, mogućnosti robota X100 i maksimalno povećava potencijal Velodyne LiDAR senzora. Rezultirajući sistem omogućit će robotima autonomnu navigaciju, precizno praćenje zidova ili površina, istovremeno osiguravajući poštivanje unaprijed definiranih ograničenja udaljenosti i mogućnost ponovnog povratka u početnu tačku. Ukratko, rad je vođen raskrižjem robotike, ROS-a, mobilnih robota, LiDAR senzora i težnje za autonomnijom budućnošću. Baveći se ovim ključnim komponentama na kohezivan i inovativan način, nastoji se pridonijeti evoluciji robotike koja je u toku, nudeći novo rješenje za problem koji je dugo predstavljao izazov za ovo područje.

## 1.4 Zadaci i ciljevi istraživanja

Primarni cilj ovog rada je razviti i implementirati novi algoritam koji robotu X100 omogućava autonomnu navigaciju uz pridržavanje skupa definiranih kriterija. Zadatak jeste opremiti mobilnog robota X100, pojačanog preciznošću Velodyne 3D LiDAR senzora, sposobnošću autonomne navigacije duž zidova ili drugih površina uz zadržavanje određene udaljenosti od prepreka i, što je najvažnije, vraćanja u početni položaj. Također, potrebno je iskoristiti mogućnosti Velodyne 3D LiDAR senzora kako bi se poboljšala tačnost lokalizacije i percepcija okoline. To podrazumijeva razvoj algoritma koji omogućava robotu da razlikuje svoju okolinu, identificira

prikladne površine za navigaciju i dosljedno prilagođava svoju putanju kako bi održao unaprijed definiranu udaljenost od tih površina. Drugi temeljni aspekt rada uključuje postizanje precizne lokalizacije. Robot mora biti u stanju utvrditi svoj položaj unutar okoline što zahtijeva integraciju algoritama lokalizacije koji iskorištavaju podatke Velodyne 3D LiDAR senzora i odometrije.

## 1.5 Struktura rada

Osim prvog, uvodnog poglavlja, rad je u nastavku organiziran u sljedeća poglavlja:

U drugom poglavlju date su osnove ROS-a, pojašnjavajući njegove temeljne koncepte i arhitekturu. Zatim su istraženi principi dizajna koji ROS čine sveprisutnim izborom u istraživanju robotike, te su predstavljeni osnovni ROS alati.

Treće poglavlje posvećeno je mobilnom robotu X100, robotskoj platformi na kojoj je implementiran algoritam. Dat je sveobuhvatan pregled dizajna robota, ističući njegovu modularnu arhitekturu i mogućnosti integracije. Osim toga, istražene su razne aplikacije iz stvarnog svijeta koje prikazuju svestranost i korisnost platforme X100.

Četvrto poglavlje bavi se ključnom komponentom istraživanja, Velodyne LiDAR senzorom. U ovom poglavlju pojašnjena su temeljna načela LiDAR tehnologije. Fokus je zatim stavljen na Velodyne LiDAR senzor, gdje su istaknute njegove jedinstvene značajke i primjene.

Peto poglavlje daje osnovnu teorijsku podlogu o lokalizaciji i različitim tipovima lokalizacije, sa naglaskom na odometriju.

Šesto poglavlje sadrži implementaciju algoritma za praćenje zida i lokalizaciju. U ovom poglavlju detaljno je objašnjen razvijeni programski kod algoritma i pojašnjeni su ROS paketi koji su sastavni dio implementacije algoritma.

Sedmo poglavlje služi kao empirijska potvrda našeg rada. Ovdje su predstavljeni rezultati primjene algoritma na robotu X100 opremljenom Velodyne LiDAR senzorom.

U završnom poglavlju naveden je zaključak rada i izazovi koji su se javili tokom implementacije algoritma. Na kraju je dat uvid u potencijalni budući razvoj i aplikacije koje se mogu nadograđivati na ovo istraživanje.

## Poglavlje 2

# Robotski operativni sistem - ROS

### 2.1 Uvod

U svijetu robotike, uspješno upravljanje robotima zahtjeva efikasne i fleksibilne alate. Upravo tu ulogu ima ROS (Robot Operating System) - robotski operativni sistem koji je postao jedan od najpopularnijih i najutjecajnijih alata u oblasti robotskog programiranja.

ROS je paket za razvoj softvera otvorenog koda (open source) za robotske aplikacije i sam po sebi nije operativni sistem, već skup različitih okvira i sučelja za apstrakciju hardvera, niskorazinsku kontrolu robotskih sistema, implementaciju čestih i jednostavnih funkcionalnosti, razmjenu poruka između procesa i upravljanje paketima [1]. ROS nudi standardnu softversku platformu programerima u svim industrijama koja će ih voditi od istraživanja i izrade prototipa pa sve do implementacije i proizvodnje. Ovaj programski paket pruža alate i biblioteke potrebne za razvoj robotskih aplikacija, te se lako integrira s postojećim softverom. Budući da je otvorenog koda, omogućena je fleksibilnost odlučivanja gdje i kako koristiti ROS, kao i sloboda prilagođavanja vlastitim potrebama. ROS predstavlja normu za podučavanje robotike, kao i osnovu za većinu istraživanja na području robotike. Može se koristiti za širok spektar primjena u raznim domenama zahvaljujući svojoj fleksibilnosti, modularnosti i jednostavnosti integracije. Samo neka od područja primjene uključuju [1]:

- Industrijsku automatizaciju: omogućava bespriječnu integraciju robotskih ruku, hvaljki, senzora i druge opreme za automatizaciju, omogućavajući učinkovite i precizne proizvodne procese.
- Autonomna vozila: pruža robusnu platformu za fuziju senzora, percepciju, planiranje kretanja i kontrolu, omogućavajući vozilima navigaciju i sigurnu interakciju u složenim okruženjima.
- Istraživačku robotiku: često se koristi za eksperimentiranje i prototipiziranje novih robotskih algoritama i koncepata, te za razvoj inovativnih rješenja za različite robotske primjene, poput višerobotskih sistema i interakcije čovjeka i robota.
- Agrikultura: primjenjuje se u autonomnim poljoprivrednim vozilima, robotskom branju plodova i praćenju usjeva, te omogućava navigaciju kroz polja, analizu uvjeta tla i obavljanje ciljanog prskanja, smanjujući potrebu za ljudskim radom i optimizirajući upotrebu resursa.
- Svetarska i podvodna robotika: podvodni roboti opremljeni ROS-om mogu istraživati morske okoline, obavljati podvodne inspekcije i sudjelovati u oceanografskim istraživanjima.

njima, dok se u istraživanju svemira, ROS koristi za simulaciju i testiranje ponašanja robota.

S napretkom tehnologije i rastom ROS zajednice, za očekivati je da će se pojaviti još više revolucionarnih primjena koje koriste snagu ROS-a kako bi rješavale različite izazove i poboljšavale kvalitetu života pojedinaca i društva u cijelini.

## 2.2 Pojava ROS-a

Prije ROS-a, razvoj robota uključivao je mnogo suvišnih npora i prilagođenog softvera, ometajući napredak i saradnju na terenu. ROS se pojavio kao rješenje, pružajući otvoreno i standardizirano okruženje za istraživače i programere za dijeljenje koda, biblioteka i podataka, dodatno ubrzavajući napredak robotike. ROS vuče korijene iz ranih 2000-ih u Stanfordskom laboratoriju za umjetnu inteligenciju (SAIL) Univerziteta Stanford. Dok su radili na robotima za obavljanje zadataka manipulacije, studenti Eric Berger i Keenan Wyrobek primijetili su da mnoge njihove kolege koči raznolika priroda robotike: izvrstan programer softvera možda nema potrebno znanje o hardveru, i obratno. Ova dva studenta krenula su zatim u izradu osnovnog sistema koji bi bio polazna tačka za nadogradnju. Izgradili su PR1 kao hardverski prototip, slika 2.1, i dok su tražili sredstva za dalji razvoj, 2007. godine upoznali su Scotta Hassana, osnivača Willow Garage-a, istraživačkog centra s fokusom na proizvode robotike. Scottu je njihova ideja bila toliko zanimljiva da ju je odlučio finansirati i s njima pokrenuti osobni program robotike unutar Willow Garage-a. Kao nastavak PR1 robota nastao je PR2 robot i ROS kao softver za njegovo pokretanje. ROS projekt je postao toliko važan da su svi ostali projekti Willow Garage-a odbačeni i Willow Garage se koncentrirao samo na razvoj i širenje ROS-a. Početna verzija ROS-a (1.0) dočekana je sa oduševljenjem, a developeri diljem svijeta počeli su prihvataći ROS jer je nudio standardizirani okvir za izgradnju robotskih sistema. ROS je razvijan u Willow Garage-u oko 6 godina, sve dok se Willow nije ugasio 2014, kada OSRF (Open Source Robotics Foundation) preuzima primarni razvoj ROS-a.



**Slika 2.1:** PR1 robot [2]

Kako je ROS uzimao maha, developeri, zajedno sa stalno rastućom zajednicom, neprestano su

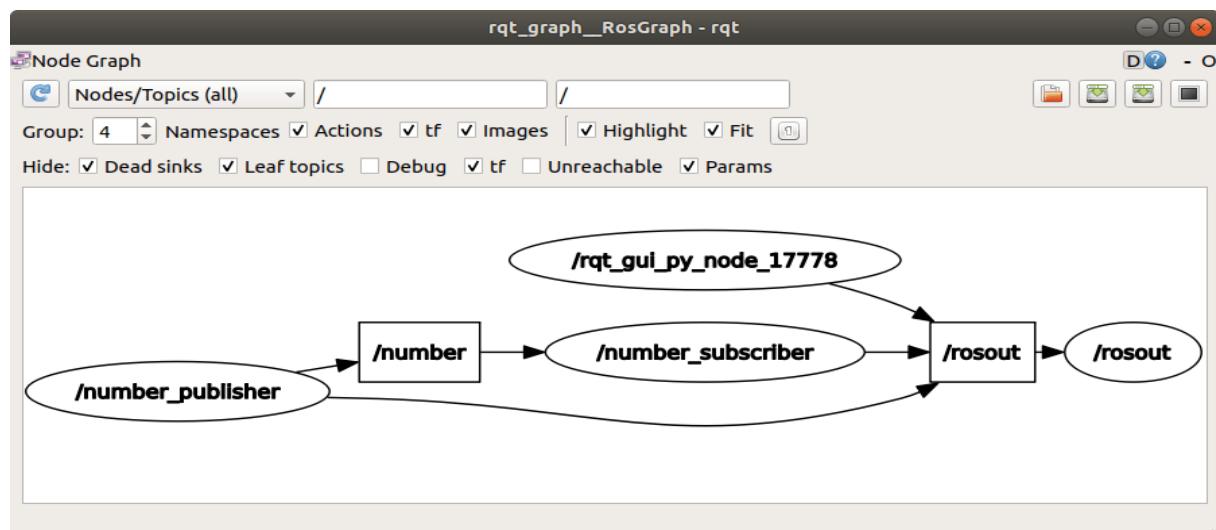
poboljšavali njegove mogućnosti. Podrška za razne programske jezike, uključujući C++, Python i Lisp, učinila ga je dostupnim širokom rasponu programera s različitim iskustvom i preferencijama. Štoviše, ROS se proširio kako bi se prilagodio različitim robotskim platformama, od robota na kotačima do dronova i humanoidnih robota.

Kako se ROS 1.0 nastavio razvijati, postalo je očito da određena ograničenja sprječavaju njegovu primjenjivost u velikim i sigurnosno kritičnim sistemima. Kao odgovor, pokrenut je razvoj ROS-a 2.0, s ciljem rješavanja ovih problema uz zadržavanje kompatibilnosti s prethodnim verzijama. ROS 2.0 je uveo značajke kao što su mogućnost rada u stvarnom vremenu, poboljšana sigurnost i bolja podrška za distribuirane sisteme. ROS 2.0 službeno je objavljen 2017. godine. Dok ROS 1.0 nastavlja napredovati, ROS 2.0 postepeno dobiva na snazi, posebno u aplikacijama koje zahtijevaju veće performanse, pouzdanost i mogućnosti industrijske razine. Uvođenje ROS 2.0 potaknulo je razvoj novih paketa i alata, dodatno proširujući opseg ROS aplikacija [2].

## 2.3 Dizajn ROS-a

ROS koristi decentraliziranu arhitekturu temeljenu na peer-to-peer mreži, koja omogućava distribuiranu obradu i komunikaciju između čvorova. Arhitektura ROS-a je usredotočena na čvorove, temeljne jedinice ROS-a, koji međusobno komuniciraju koristeći teme za strujanje podataka. Poruke definiraju strukture podataka koje se prenose putem tema, a paketi organiziraju kod i resurse pojedinačnih komponenti robotskog sistema temeljenog na ROS-u. Ova arhitektura kao rezultat daje vrlo fleksibilan i distribuiran pristup izgradnji robotskih aplikacija i omogućava jednostavnu integraciju različitih komponenti i algoritama.

Skup čvorova, tema, usluga i mastera često se naziva ROS graf. To je vizualizacija načina na koji su različite komponente robotskog sistema povezane i međusobno djeluju. GUI dodatak iz paketa alata Rqt, rqt graf, pruža upravo ovu mogućnost vizualizacije. Njegovim korištenjem moguće je vidjeti sve čvorove koji rade, kao i komunikaciju između njih. Čvorovi i teme prikazani su unutar svog imenskog prostora. Primjer rqt grafa dat je na slici 2.2. Čvorovi su prikazani kao elipse, a teme kao pravougaonici. Veze između čvorova i tema prikazane su strelicama. Smjer strelice označava smjer protoka podataka. U nastavku su objašnjeni ključni koncepti dizajna ROS-a [3].



Slika 2.2: Rqt graf sa prikazanim temama i čvorovima [4]

### 2.3.1 Čvorovi

Čvor, temeljna jedinica ROS-a, je izvršni proces koji obavlja specifične zadatke. To je neovisan proces koji obavlja određeni zadatak unutar robotskog sistema. Može se smatrati samostalnom izvršnom datotekom koja obavlja određenu funkciju. Primjeri zadataka koje čvorovi mogu obavljati uključuju obradu podataka senzora, kontrolu pokreta, percepцију, lokalizaciju, mapiranje i još mnogo toga. Svaki čvor posjeduje svoj jedinstveni naziv koji se povezuje na ROS Master prije nego što počne izvršavati bilo kakve radnje. ROS Master je središnja koordinirajuća komponenta koja pomaže čvorovima da otkriju jedni druge, upravljaju vezama i prate teme i usluge. Unutar različitih imenskih prostora može postojati više čvorova s različitim imenima, ili sam čvor može biti anoniman. U slučaju anonimnog čvora, uz njegovo ime će se generirati dodatni identifikator. Čvorovi su srž ROS programiranja, jer je većina ROS koda organizirana u vidu čvorova koji izvršavaju različite akcije na temelju informacija koje primaju od ostalih čvorova. Ova fleksibilnost i komunikacija između čvorova čini ih temeljnim gradivnim blokom unutar ROS ekosistema. Čvorovi međusobno komuniciraju pomoću ROS komunikacijske infrastrukture. Primarni komunikacijski mehanizmi su teme i usluge.

### 2.3.2 Teme

Teme su imenovane sabirnice kroz koje čvorovi razmjenjuju podatke. Nazivi tema moraju biti jedinstveni unutar svog imenskog prostora. Svakom temom se uspostavlja komunikacijski kanal, omogućavajući čvorovima da razmjenjuju informacije bez potrebe da znaju identitet ili lokaciju drugih čvorova. Da bi se poruka poslala na određenu temu, čvor mora prvo objaviti poruku na toj temi, dok se za primanje poruke mora preplatiti na temu. Kad čvor objavi poruku na određenoj temi, ta poruka se distribuira svim čvorovima koji su preplaćeni na tu temu. To znači da poruka može doseći više čvorova koji su zainteresirani za istu vrstu informacija. Ovaj koncept omogućava efikasno širenje informacija u velikim mrežama. Model "objavi-primi" (publish-subscribe) je anoniman, odnosno nijedan čvor u suštini ne zna koji drugi čvor šalje ili prima poruke na određenoj temi, osim što je svjestan primanja ili slanja poruke. Poruke mogu sadržavati različite sadržaje definirane od strane korisnika, kao što su podaci sa senzora, naredbe za upravljanje motorima, ažuriranja statusa, naredbe za aktuatore i mnoge druge. Uz sve to, "objavi-primi" model pruža skalabilnost i otpornost na kvarove. Ako neki čvorovi postanu nedostupni ili se pojave problemi u komunikaciji, mreža i dalje može funkcionirati jer poruke nisu usmjerene prema određenim čvorovima, već prema temama koje su neovisne o pojedinačnim čvorovima.

### 2.3.3 Usluge/servisi

Usluge ili servisi su još jedan ključni komunikacijski mehanizam koji omogućava interakciju između čvorova. Dok su teme korisne za emitovanje informacija na više čvorova, usluge olakšavaju komunikaciju od tačke do tačke. Usluga omogućava čvoru da zatraži određenu operaciju od drugog čvora i primi odgovor. Zbog toga se servisi često koriste za radnje koje imaju precizno definiran početak i kraj. Usluge u ROS-u slijede komunikacijski model klijent-poslužitelj. Čvorovi koji pružaju usluge poznati su kao poslužitelji, a čvorovi koji traže usluge poznati su kao klijenti. Usluge su identificirane jedinstvenim nazivima, sličnim temama. Konvencija imenovanja slijedi hijerarhijsku strukturu, što pomaže u organizaciji usluga na temelju njihove funkcionalnosti ili svrhe. Korištenjem servisa, čvorovi mogu efikasno sarađivati u ostvarenju kompleksnih funkcionalnosti, a istovremeno zadržavajući jasnu i predvidljivu dinamiku izvršenja.

## 2.4 ROS alati

ROS pruža širok raspon alata i biblioteka za pomoć programerima u stvaranju, simulaciji i kontroli robotskih aplikacija. Ovi alati su dostupni u paketima, slično kao i drugi algoritmi, i omogućavaju olakšano rukovanje robotskim izazovima. Osim toga, ovi alati omogućavaju programerima da bolje razumiju i upravljaju robotskim sistemima kroz intuitivno sučelje i funkcionalnosti koje nude. Njihova integracija u ROS ekosistem doprinosi većoj produktivnosti i efikasnosti u razvoju robotskih rješenja, što čini ROS popularnim okruženjem među istraživačima i inženjerima širom svijeta. Osnovni ROS alati opisani su u nastavku [5].

### 2.4.1 roscore

Alat roscore pruža temeljnu funkcionalnost za međusobnu komunikaciju ROS čvorova. Djeluje kao središnje čvorište za koordinaciju komunikacije između različitih ROS čvorova te inicijalizira ROS Master i pruža usluge imenovanja i registracije drugim ROS čvorovima. Bitno je navesti da, kako bi komunikacija između čvorova ispravno funkcionala, roscore mora biti pokrenut. Nakon njegovog pokretanja, prikazat će se informacije o ROS Master URI-ju i drugi bitni detalji i moguće je pokrenuti druge čvorove u zasebnim terminalima.

### 2.4.2 rosrun

Rosrun omogućava pokretanje pojedinačnih ROS čvorova navođenjem naziva njihovog paketa i naziva čvora. Osim toga, moguće je dodati argumente naredbi rosrun koji se zatim proslijeđuju navedenom čvoru.

### 2.4.3 roslaunch

Roslaunch je moćan alat koji se koristi za pokretanje i upravljanje više ROS čvorova, pokretanje datoteka i konfiguracija istovremeno. Omogućava definiranje i izvršavanje složenih sekvenci pokretanja koje otvaraju različite čvorove, postavljaju parametre, konfiguriraju imenske prostore i upravljaju varijablama okruženja. roslaunch pojednostavljuje proces pokretanja i upravljanja više ROS čvorova i konfiguracija, a posebno je koristan za stvaranje i održavanje složenih postavki i simulacija robota.

### 2.4.4 rostopic

Alat koji pruža informacije i olakšava interakciju s temama jeste rostopic. Ova naredba daje popis naziva tema s kojima je moguće komunicirati, informacije o određenoj temi, podatke koji se objavljaju u stvarnom vremenu i drugo.

### 2.4.5 rviz

Rviz je alat za 3D vizualizaciju koji omogućava grupisanje podataka senzora, modela robota i drugih 3D podataka u kombinirani prikaz. Korisnici mogu prilagoditi vizualizaciju za prikaz specifičnih podataka koji ih zanimaju. To uključuje postavljanje vrste vizualizacije, sheme boja, veličine i drugih parametara. Rviz pomaže u otklanjanju grešaka, praćenju i analizi ponašanja robota pružajući vizualnu povratnu informaciju u stvarnom vremenu.

# Poglavlje 3

## Mobilni roboti

### 3.1 Uvod

Mobilni roboti su uređaji koji posjeduju sposobnost kretanja unutar svoje okoline. Osim osnovne mobilnosti, ovi roboti karakteriziraju se i određenom autonomnošću, što znači da su u stanju donositi odluke i djelovati bez potrebe za konstantnom ljudskom interakcijom. Osim toga, mobilni roboti posjeduju inteligenciju koja im omogućava opažanje okoline, te reagovanje na nju na temelju prikupljenih informacija. Ova kombinacija mobilnosti, autonomnosti i inteligencije čini ih iznimno korisnim i raznolikim alatima u različitim aplikacijama, od istraživanja do primjena u industriji i medicini. Ključne značajke mobilnih roboata uključuju sposobnost kretanja kroz različite vrste terena, bilo da se radi o ravnem terenu, nepristupačnim okolinama ili urbanim sredinama. Osim toga, ovi roboti su opremljeni senzorima koji im omogućavaju da prikupljaju podatke o okolini, kao što su slike, zvukovi ili podaci o udaljenosti od objekata. Na temelju tih podataka, mobilni roboti donose odluke i izvršavaju zadatke, često prilagođavajući svoje akcije promjenjivim uvjetima.

S obzirom na sklopošku organizaciju, roboti se mogu razložiti na komponente [6]:

- Mehanizam koji omogućava robotu kretanje kroz njegovu okolinu, uključujući motore, prijenosnike i zupčanike potrebne za pokretanje robota.
- Računar ili skupina računara za upravljanje robotom.
- Skupina senzora putem kojih robot dobiva informacije o svojoj okolini.
- Komunikacijski sklopoli koji omogućavaju robotu komunikaciju s operaterom i drugim vanjskim računarima.

Zahtjevi za mobilnim robotom su sljedeći [6]:

- Mobilni robot mora donositi odluke na razmatran način.
- Moraju se prilagoditi okolini reagirajući na odgovarajući način.
- Mobilni robot mora biti sposoban anticipirati i upravljati neizvjesnim informacijama.
- Treba biti otporan i sposoban podnosi kvarove.
- Arhitektura mobilnog robota mora biti postupna i prilagodljiva.

Osim toga, mobilni roboti također trebaju:

- Demonstrirati visoku razinu autonomije u obavljanju svojih zadataka.
- Imati sposobnost učenja i prilagodbe na promjenjive uvjete okoline.
- Osigurati sigurnost za operatore i okolinu tokom svog djelovanja.
- Biti energetski učinkoviti kako bi produžili trajanje rada na jednom punjenju.
- Pružati mogućnost daljinskog upravljanja i nadzora kada je to potrebno.

Ovi zahtjevi čine mobilne robote vrlo složenim sistemima koji se moraju prilagoditi različitim scenarijima i izazovima u stvarnom svijetu.

## 3.2 Podjela mobilnih robota

Postoji nekoliko različitih kategorija mobilnih robota koje se razlikuju prema vrsti pogona, načinu kretanja, terenu na kojem se koriste, obliku, transportnom mediju koji koriste te stepenu autonomnosti. Te različite karakteristike u velikoj mjeri utječu na to koji će se sistem upravljanja i navigacije primijeniti. Jedna od najvažnijih podjela mobilnih robota temelji se na vrsti kretanja [6]:

- Roboti sa kotačima: Ovi roboti koriste kotače kao svoj glavni način kretanja. Kotači omogućavaju brzo i stabilno kretanje po ravnim površinama te su često korišteni u industrijskim okolinama i skladištima.
- Nožni roboti: Nožni roboti imaju noge ili ekstremite koji im omogućavaju da se kreću slično kao životinje ili ljudi. Ova vrsta robota često se koristi u istraživanju terena i neprijateljskim okruženjima gdje se zahtijeva veća pokretljivost.
- Gusjeničari: Gusjeničari koriste gusjenice umjesto kotača za kretanje. Ova vrsta mobilnih robota ima bolje prijanjanje na neravnim terenima, blatu i snijegu te se često koristi u vojnim, poljoprivrednim ili građevinskim aplikacijama.
- Zmijoliki (pužući) roboti: Zmijoliki roboti imaju tijelo koje je oblikovano poput zmije ili puža. Koriste se za kretanje kroz uske prostore, cijevi ili podzemne kanale gdje bi drugi roboti teško pristupili.

Još jedna značajna podjela jeste prema tipu pogona:

- diferencijalni pogon,
- sinhroni pogon,
- automobilski ili Ackermanov pogon,
- bicikl pogon,
- svesmjerni pogon (omnidirectional).

Osim ovih osnovnih podjela, mobilni roboti se također mogu klasificirati i prema drugim karakteristikama. Svaka od tih podjela pomaže u razumijevanju specifičnih potreba i izazova koji se odnose na upravljanje i navigaciju mobilnih robota u različitim kontekstima i primjenama.

### 3.3 X100 mobilni robot

Mobilni robot X100 predstavlja inovaciju u području autonomnih robota, dizajniranih za obavljanje različitih zadataka u različitim domenama. S porastom automatizacije i umjetne inteligencije, mobilni roboti postali su nezamjenjivi alati u raznim industrijskim područjima. Mobilni robot X100, slika 3.1, razvijen je od strane tvrtke XMachines. Kombinirajući napredne mogućnosti osjeta, percepcije i autonomne navigacije, X100 nudi svestrano rješenje za automatizaciju zadataka koje tradicionalno obavljaju ljudi. X100 predstavlja jako dobru mobilnu robotsku platformu za testiranje novih robotskih aplikacija i algoritama. Njegov API, aplikacijsko programsko sučelje, koji podržava ROS, čini ga svestranom platformom za istraživanje i razvoj u širokom rasponu primjena u koje spadaju poljoprivreda, miniranje, rudarstvo, građevina, dostavna robotika i slično.



Slika 3.1: X100 mobilni robot [8]

#### 3.3.1 Dizajn i tehničke specifikacije

Mobilni robot X100 odlikuje se elegantnim i kompaktnim dizajnom, što ga čini prikladnim za upotrebu u unutarnjim i vanjskim okruženjima. Karakterizira ga robusna čelična šasija, glatki kalibrirani pogon, pneumatski točkovi, te kompaktna veličina sa prijenosnim oblikom. Uz sam robot dolazi ugrađeni operativni sistem sa NVidia Xavier NX, namijenjenom za ugradive operacije mašinskog učenja i mašinske vizije. Za potrebe pokretanja ROS-a koristi se Ubuntu Linux. Baterije omogućavaju 4 sata nominalnog rada, a sa X100 isporučuju se dva seta baterija koje je moguće zamijeniti u samo nekoliko sekundi bez potrebe za isključivanjem robota. Ovim je omogućena produžena upotreba i nesmetan rad robota. Težina robota kao i nosivost iznose 75 kg, a maksimalna brzina 0.8 m/s. Jedinstven dizajn šasije robota omogućava korisniku postavljanje tereta na vrh, stražnju i prednju stranu robota bez ugrožavanja njegove stabilnosti. Šasija je dizajnirana s niskim težištem što pomaže u održavanju robota stabilnim i sigurnim s različitim veličinama tereta i konfiguracijama, čime je omogućena maksimalna fleksibilnost pri postavljanju tereta na robota. X100 robot sadrži niz najsavremenijih senzora, uključujući Li-

DAR, kamere, ultrazvučne senzore i inercijske mjerne jedinice (IMU), što mu omogućava tačnu percepciju okoline. Također, ima mogućnost dodavanja drugih senzora, navigacijske opreme, aktuatora i kamera. Manuelna kontrola ostvaruje se pomoću joysticka, korištenjem Bluetootha [8].

### 3.3.2 Dodaci

Mobilni robot X100 dizajniran je za obavljanje širokog spektra zadataka, a kako bi učinkovito obavljao te zadatke, robot je opremljen skupom senzora, aktuatora i kamera koje mu omogućavaju da percipira okolinu, samostalno se kreće i komunicira s okolinom i ljudima. U ovom dijelu rada će biti objašnjene ključne komponente koje pridonose funkcionalnosti mobilnog robota X100. Kako se tehnologija nastavlja razvijati, te će komponente vjerojatno doživjeti dalji napredak, čime će se još više poboljšati mogućnosti i potencijal mobilnog robota X100.

#### Senzori

Mobilni robot X100 integrira niz senzora koji omogućavaju robotu da opaža svoju okolinu, donosi informirane odluke i kreće se kroz složena okruženja. X100 opremljen je različitim senzorima poput:

- IMU (Inercijalna mjerna jedinica) se sastoji od akcelerometara i žiroskopa koji daju informacije o orijentaciji, ubrzanju i rotaciji robota. Ovi su podaci ključni za održavanje tačne navigacije i kontrole kretanja.
- Ultrazvučni senzori emituju visokofrekventne zvučne valove i mjere vrijeme potrebno da se zvuk reflektira od površinu. Ovi senzori su posebno korisni za otkrivanje prepreka iz neposredne blizine i izbjegavanje sudara.
- LiDAR (Light Detection and Ranging) senzori, slika 3.2, omogućavaju precizno otkrivanje i izbjegavanje prepreka, što će biti detaljnije obrađeno u jednom od narednih pogлавlja.



**Slika 3.2:** 2D LiDAR senzor [8]

## GPS

Mobilni robot X100, opremljen GPS-om, može učinkovito navigirati u vanjskim okruženjima. Primanjem signala s više GPS satelita, robot može odrediti svoje precizne geografske koordinate. Roboti s GPS-om poput X100 mogu se programirati da slijede unaprijed definirane putanje. Moguće je odrediti niz GPS koordinata do kojih robot treba navigirati, omogućavajući autonomnu navigaciju određenim rutama, što je posebno korisno u zadacima kao što su mapiranje i nadzor velikih područja. GPS podaci mogu se integrirati s podacima drugih senzora, kao što su LiDAR i IMU, kako bi se poboljšala tačnost mapiranja i lokalizacije. Robot može koristiti GPS informacije za usklađivanje svoje lokalne karte s globalnim koordinatama, pomažući u preciznoj navigaciji i pozicioniranju u velikim okruženjima. Međutim, važno je napomenuti da na GPS signale mogu utjecati faktori kao što su visoke zgrade, gusta vegetacija i prirodne prepreke. U okruženjima sa slabim GPS prijemom, tačnost položaja robota može se smanjiti. Osim toga, tačnost GPS-a općenito je veća na otvorenom nego u zatvorenom prostoru.

## Kamere

Kamere igraju ključnu ulogu u percepciji, interakciji i navigacijskim mogućnostima mobilnog robota X100. Više kamera visoke rezolucije pružaju vizualne podatke koji robotu omogućuju prepoznavanje objekata, čitanje znakova i tumačenje okoline, čime je omogućena i interakcija čovjeka i robota. RGB kamere hvataju vizualne informacije u boji, omogućavajući robotu identificiranje objekata, čitanje oznaka i navigaciju na temelju vizualnih znakova. Kamere za mjerjenje dubine pružaju informacije o udaljenosti do objekata u okruženju, olakšavajući precizno otkrivanje prepreka, navigaciju i 3D mapiranje. Pan-Tilt-Zoom (PTZ) kamere nude fleksibilnost za promjenu uglova gledanja i razine zumiranja. Ove kamere su posebno korisne za nadzor, daljinsko praćenje i inspekcijske zadatke.

## Aktuatori

Aktuatori su odgovorni za izvršavanje fizičkih pokreta robota i interakciju s okolinom. Mobilni robot X100 uključuje napredne pokretače koji mu omogućavaju precizno i agilno kretanje. Višesmjerni kotači omogućavaju robotu da se kreće u bilo kojem smjeru bez promjene orientacije, omogućavajući glatku i učinkovitu navigaciju. Neke verzije mobilnog robota X100 uključuju ruku manipulatora opremljenu raznim krajnjim efektorima, kao što su hvataljke ili alati. Ova ruka omogućava robotu obavljanje zadataka koji zahtijevaju manipulaciju, poput biranja i postavljanja predmeta. Na slici 3.3 je prikazana UR3e robotska ruka koju je moguće jednostavno ukorporirati sa X100 robotom. Razvijena je od strane Universal Robots tvrtke za pomoć u različitim zadacima uz održavanje visoke razine sigurnosti. Sa svojom kompaktnom veličinom, svestranošću i jednostavnim programiranjem, UR3e je vrlo prikladna za primjene u industrijama kao što su proizvodnja, zdravstvo i istraživanje, te za automatizaciju procesa u skućenim radnim prostorima.



**Slika 3.3:** UR3e robotska ruka [8]

## 3.4 Aplikacije

U nastavku će biti obrađene različite primjene mobilnog robota X100 u različitim sektorima, ističući kako on preoblikuje procese, povećava učinkovitost i pridonosi napretku modernih industrija. Primjene ovog robota uključuju, ali se ne ograničavaju na:

1. Poljoprivreda: Savremena poljoprivreda suočava se s izazovima u učinkovitom zadovoljavanju globalnih potreba za hranom. Robot X100 može se koristiti u preciznoj poljoprivredi, praćenju usjeva, primjeni gnojiva, pa čak i obavljanju zadataka poput branja voća. Iskorištavanjem svojih autonomnih mogućnosti navigacije i prikupljanja podataka, pridonosi povećanju prinosa, smanjenom rasipanju resursa i održivoj poljoprivrednoj praksi.
2. Praćenje i održavanje okoliša: U okruženjima gdje je ljudski pristup ograničen ili opasan, robot X100 može se koristiti za zadatke poput inspekcije cjevovoda, praćenja razine onečišćenja i procjene strukturalnog integriteta. Njegova sposobnost snalaženja po zahtjevnim terenima i ograničenim prostorima čini ga idealnim alatom za osiguranje sigurnosti i obavljanje rutinskih zadataka održavanja.
3. Proizvodnja i montaža: X100 igra ključnu ulogu u proizvodnim okruženjima, gdje su preciznost i učinkovitost najvažniji. Opremljen naprednim sposobnostima percepције i navigacije, može pomoći u zadacima na montažnoj traci, inspekcijskim kontrolama kvalitete i transportu dijelova. Ova aplikacija ubrzava proizvodne cikluse, održava dosljednu kvalitetu proizvoda i smanjuje vrijeme zastoja u proizvodnji.
4. Logistika i skladištenje: Mobilni robot X100 ističe se u zadacima rukovanja materijalom i navigacijom kroz dinamična okruženja za prijevoz robe. Optimizira upravljanje zalihama i procese distribucije, smanjujući ljudski rad i minimizirajući pogreške.

5. Zdravstvo i ugostiteljstvo: U zdravstvenim ustanovama, robot X100 nudi nezamjenjivu podršku. Može autonomno dostavljati lijekove, medicinske potrepštine i dokumente različitim odjelima, omogućavajući medicinskom osoblju da se usredotoči na brigu o pacijentima. Slično, u ugostiteljskom sektoru, robot može poboljšati iskustvo gostiju isporukom pogodnosti, narudžbama za poslužu u sobu i pružanjem informacija o pogodnostima objekta.

Važno je spomenuti da je mobilni robot X100 dizajniran za saradnju s ljudima, pridržavajući se sigurnosnih standarda kako bi se spriječila bilo kakva potencijalna šteta. Može učinkovito sarađivati s ljudskim radnicima, dijeleći radno opterećenje i rješavajući ponavljaće ili opasne zadatke, čime se poboljšava ukupna učinkovitost. Međutim, unatoč svojim sposobnostima, mobilni robot X100 također se suočava s određenim izazovima. Neki od njih uključuju:

- Trajanje baterije: kao i kod većine mobilnih robota, optimizacija trajanja baterije ostaje prioritet kako bi se poboljšala radna izdržljivost robota.
- Ljudska interakcija: Razvijanje intuitivnijih i prirodnijih sučelja za interakciju između čovjeka i robota dodatno će poboljšati korisničko iskustvo.

Pored ovoga, njegova izvedba u vrlo pretrpanim ili nestrukturiranim okruženjima može predstavljati problem, a njegova ovisnost o pouzdanoj bežičnoj komunikaciji izaziva zabrinutost u područjima s lošom vezom. Bez obzira na to, X100 ostaje izvanredno postignuće u polju robotike, nudeći napredne funkcionalnosti, prilagodljivost i svestranost.

# Poglavlje 4

## LiDAR senzor

### 4.1 Uvod

LiDAR predstavlja akronim za light detection and ranging, odnosno laser imaging, detection and ranging. To je napredna tehnologija koja koristi lasersku svjetlost za mjerjenje udaljenosti i stvaranje detaljnih trodimenzionalnih prikaza objekata i okruženja. Postoji još od 1960-ih, a posljednjih godina LiDAR senzori postigli su veliku popularnost zahvaljujući širokom rasponu primjena. Koristi se na području automatizacije, arheologije, poljoprivrede, kao i za kvantificiranje raznih atmosferskih komponenti.

### 4.2 Princip rada

LiDAR radi na principu mjerjenja vremena potrebnog za refleksiju laserske svjetlosti do transmitera i detektovanja varijacije njegove valne dužine. Ovaj senzor emituje laserske impulse prema ciljanom području i mjeri vrijeme koje je potrebno da se impuls vrati nakon odbijanja od objekta. Kombinacija brzine svjetlosti i potrebnog vremena omogućava proračun udaljenosti između senzora i objekta. Formula koja se koristi za ovaj proračun je sljedeća:

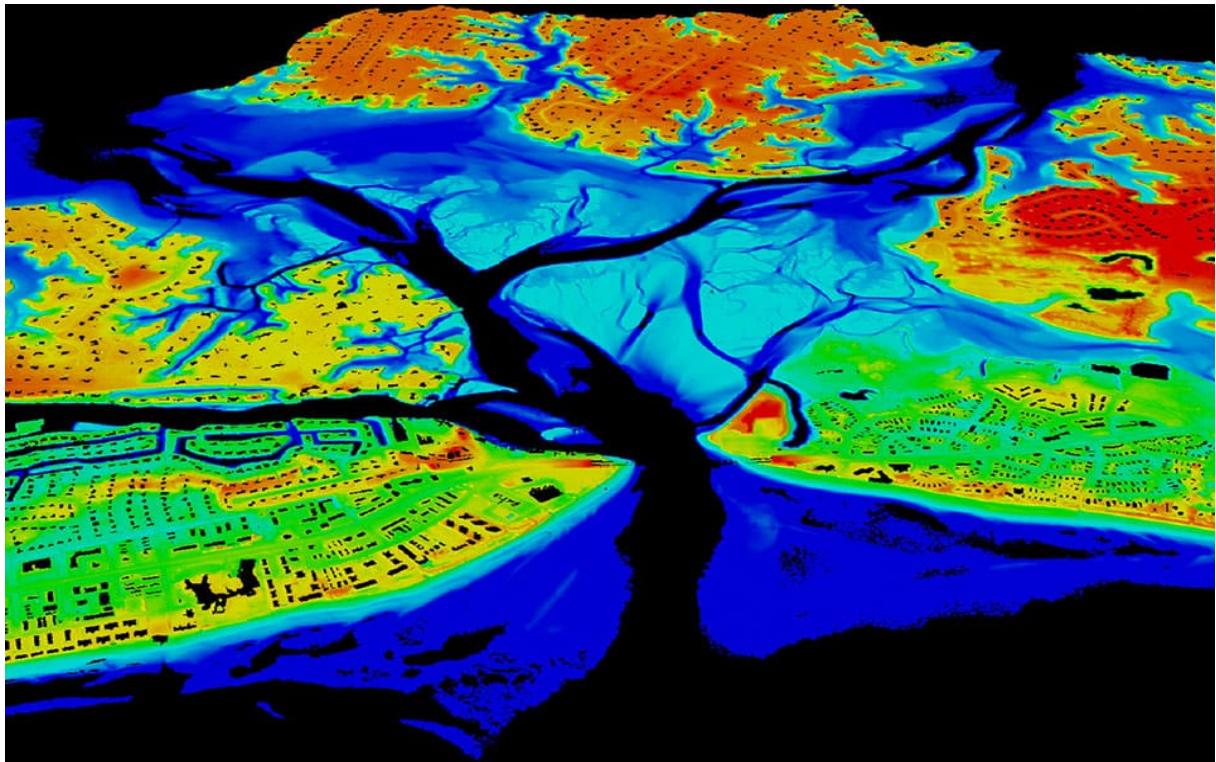
$$d = c \cdot \frac{t}{2} \quad (4.1)$$

gdje je

- d - udaljenost objekta
- c - brzina svjetlosti,
- t - vrijeme potrebno da svjetlost pređe put od lasera do objekta i nazad.

Jedan LiDAR sistem tipično se sastoji od laserskog skenera, senzora, GNSS (Globalni satelitski navigacijski sistem) prijemnika i inercijske mjerne jedinice (IMU). Laserski skeneri emituju hiljade zraka u sekundi na okolne objekte i na temelju svojstava reflektiranih zraka obavlja se proračun udaljenosti. Skeniranjem lasera preko vidnog polja i snimanjem udaljenosti pod različitim uglovima, formira se 3D oblak tačaka, pružajući detaljan prikaz okoline. Ovim je omogućeno generiranje preciznih, kvalitetnih i u nekim slučajevima trodimenzionalnih mapi okoline. Jedna ovakva mapa prikazana je na slici 4.1. LiDAR tehnologija omogućava prepoznavanje i karakterizaciju različitih materijala i svojstava zahvaljujući svojoj sposobnosti da

analizira povratni laserski signal. Određeni tipovi ovog senzora mogu proizvesti detaljna mapiranja fizičkih atributa s iznimnom preciznošću. Važno je napomenuti da različiti tipovi LiDAR sistema imaju svoje prednosti i ograničenja u prepoznavanju specifičnih materijala i svojstava. Također, preciznost identifikacije ovisi o parametrima poput snage laserskog impulsa, valne dužine, i reflektivnosti cilja.



Slika 4.1: Mapa koju generiše LiDAR senzor [9]

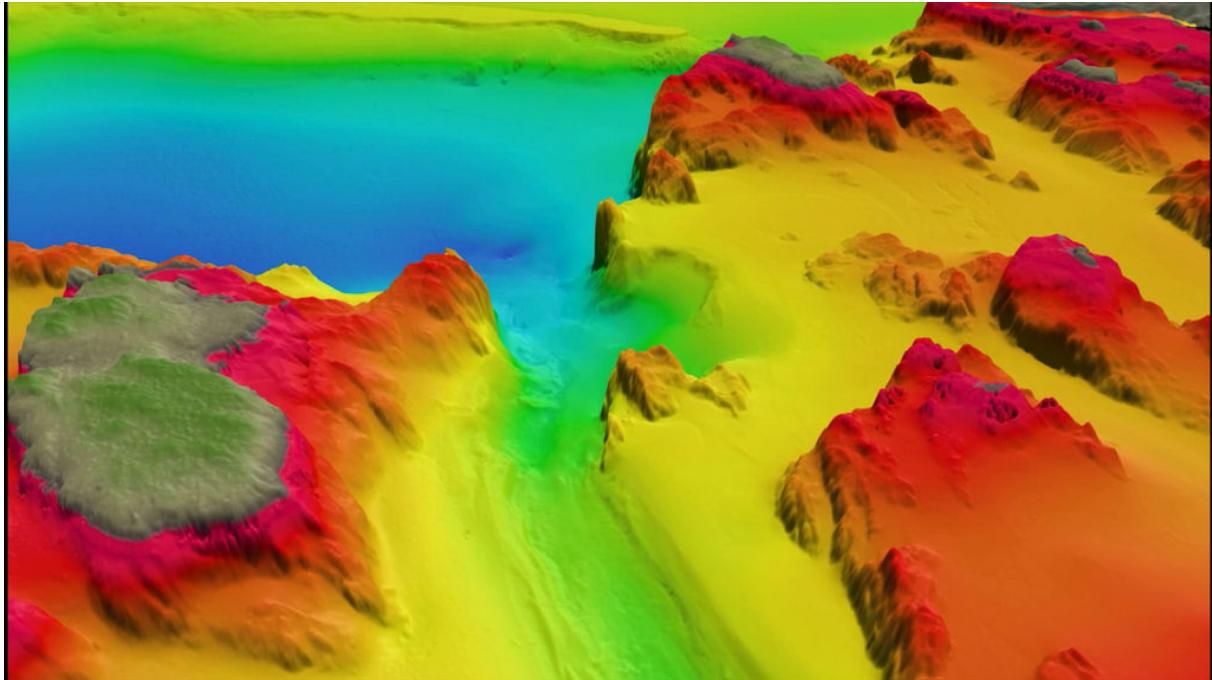
### 4.3 Tipovi LiDAR senzora

Kako je tehnologija LiDAR-a napredovala tokom godina, došlo je do razvoja različitih vrsta LiDAR sistema prilagođenih određenim aplikacijama i platformama. Kategorizacija LiDAR sistema može se temeljiti na njihovoj platformi, tehnologiji i metodi skeniranja.

Prema platformi se mogu klasificirati na sljedeći način [10]:

- **Zračni LiDAR:** Zračni LiDAR sistemi su dizajnirani tako da su usmjereni prema dolje i obično skeniraju prostorni ugao od  $180^\circ$ . Montiraju se na avione ili bespilotne letjelice, što ih čini idealnim za mapiranje i istraživanje velikih područja. Ovi sistemi generiraju model visine visoke rezolucije i dalje se dijele na dva tipa, topografski i batimetrijski. Topografski LiDAR se koristi za ispitivanje površine, dok se batimetrijski koristi za aplikacije prodora vode, obično za ispitivanje nadmorske visine i dubine vode istovremeno. Potencijal batimetrijskog LiDAR-a pokazuje slika 4.2, na kojoj je prikazana mapa izrađena korištenjem upravo ovog tipa LiDAR-a, kombinacijom detekcije iz zraka i nautike.
- **Terestrički LiDAR:** Terestrički LiDAR sistemi su postavljeni na tlu i daju detaljne 3D informacije o objektu ili okolini. Uglavnom izvode horizontalno skeniranje i obično pokrivaju  $360^\circ$  u 1D ili 2D. Koriste se za snimanje autocesta i željeznica, upravljanje objektima, stvaranje 3D modela za prostore, itd. Dijele se na dva tipa terestričkih LiDAR-a,

mobilni i statički. Kod mobilnih LiDAR-a, LiDAR je integriran unutar vozila, poput automobila ili vozova, a podaci se prikupljaju sa pokretnе platforme. Statički LiDAR-i mogu prikupljati podatke unutar građevina ili izvan njih i obično se koriste u arheologiji, ruderstvu i geodeziji.



Slika 4.2: 3D mapa jezera George, SAD, izrađena korištenjem barometrijskog LiDAR-a [11]

## 4.4 Aplikacije

Preciznost, sposobnost stvaranja detaljnih 3D modela i brzina prikupljanja podataka faktori su koji su omogućili širok spektar primjena LiDAR tehnologije u različitim sektorima:

- Autonomna vozila: LiDAR sistemi koriste se za osiguravanje  $360^\circ$  percepције okoline vozila te za identifikaciju i praćenje drugih vozila, pješaka i prepreka na cesti.
- Mapiranje i kartografija: LiDAR se koristi u stvaranju topografskih i batimetrijskih karata visoke rezolucije za geografske informacijske sisteme (GIS).
- Arheologija: Skeniranje tla i stvaranje detaljnih 3D modela pomaže u otkriću arheoloških nalazišta, poput zakopanih struktura i krajolika koje je inače teško otkriti.
- Meteorologija i atmosferska istraživanja: LiDAR se može koristiti za proučavanje atmosferskih pojava, poput mjerjenja svojstava oblaka i aerosola, kao i brzine i smjera vjetra u nižim slojevima atmosfere.
- Praćenje okoliša: LiDAR se može koristiti za praćenje promjena okoline, kao što su krčenje šuma, erozija zemljišta i promjene obale, a ima i značajnu ulogu u procjeni utjecaja prirodnih nepogoda poput potresa, poplava i klizišta.
- Poljoprivreda i šumarstvo: LiDAR može pružiti detaljne informacije o vegetaciji, visini stabala i strukturi krošnji, što predstavlja korisne podatke za šumarstvo i poljoprivredu.

- Upravljanje komunalnim uslugama i infrastrukturom: LiDAR se koristi za održavanje vodova, električnih mreža i druge infrastrukture stvaranjem detaljnih mapa za identificiranje potreba održavanja i potencijalne opasnosti.
- Urbanističko planiranje: LiDAR se koristi za prikupljanje podataka o visini zgrada, obližnjim objektima i infrastrukturi. Ovi podaci pomažu u izradi detaljnih prostornih analiza i donošenju boljih odluka o urbanom razvoju.

Ovo su samo neki od primjera mnogih primjena LiDAR tehnologije. Kako ona napreduje, njen potencijal za inovacije u raznim industrijama vjerojatno će se još više proširiti.

## 4.5 Velodyne LiDAR

Neke od vodećih firmi na području LiDAR tehnologije su Velodyne LiDAR, Innoviz Technologies, Luminar Technologies i Quanergy Systems. Velodyne LiDAR kompanija je osnovana 1983. godine i igrala je ključnu ulogu u razvoju senzora baziranih na LiDAR tehnologiji za različite primjene, uključujući autonomna vozila, industrijsku automatizaciju, robotiku, infrastrukturu, kartografiju, navigaciju i još mnogo toga. Sjedište Velodyne LiDARA je u San Joseu u Kaliforniji, a kompanija je poznata po razvoju LiDAR senzora različitih rezolucija i performansi, a senzor najveće vrijednosti na tržištu jeste Puck LiDAR senzor, slika 4.3. Odlikuju ga pouzdanost, energetska učinkovitost i kompaktnost, što ga čini pogodnim za različite aplikacije poput automobilske industrije, mapiranja i robotike. Princip na kojem se zasniva rad ovog senzora jeste dual return mod koji omogućava veoma precizno snimanje detalja okoline, poput vegetacije i infrastrukture. Zahvaljujući ovoj karakteristici, pomoću Velodyne Puck LiDAR senzora formiraju se precizne 3D slike koje omogućavaju unapređenje sigurnosti, učinkovitosti i slično.



Slika 4.3: Velodyne Puck LiDAR senzor [12]

# Poglavlje 5

## Lokalizacija

### 5.1 Uvod

Osnovni izazov u robotici leži u stjecanju znanja o položaju i orijentaciji robota u svakom trenutku, praktički znati gdje se on nalazi. Razumijevanje položaja i orijentacije robota ključno je za planiranje kretanja. Roboti moraju biti sposobni navigirati u različitim okruženjima, a za to je neophodno precizno pozicioniranje. Problem pozicioniranja robota može se razumjeti kroz dva osnovna pristupa: jaku i slabu lokalizaciju. Jaka lokalizacija zahtjeva precizno znanje o apsolutnom položaju robota u globalnom koordinatnom sistemu. Ovo je izazovno jer globalni koordinatni sistem često nije dostupan ili precizno poznat, a senzori i metode moraju osigurati pouzdano određivanje pozicije u takvim okruženjima. S druge strane, slaba lokalizacija ne zahtjeva apsolutno poznavanje položaja robota, već se temelji na odnosu prema prethodno posjećenim tačkama ili orijentirima u okolini. Ova metoda je korisna za robote koji se kreću u poznatom okruženju i samo trebaju održavati relativno znanje o svojoj poziciji kako bi izbjegli sudare i pravilno se kretali.

Postupci određivanja odnosno procjene položaja robota mogu se svrstati u dvije osnovne skupine: mjerjenje relativnog položaja (dead-reckoning) gdje spadaju odometrija (praćenje kretanja kotača) i inercijalna navigacija (praćenje ubrzanja i rotacije), te mjerjenje apsolutnog položaja. Mjerjenje apsolutnog položaja uključuje magnetske kompase, aktivne svjetionike, globalni pozicijski sistem, navigaciju temeljenu na orijentirima i slaganje modela. Ove metode se još nazivaju unutarnjim, koje se temelje na kretanju i silama koje robot generira, i vanjskim, koje koriste signale iz okoline za registraciju položaja robota [6]. Važno je napomenuti da kombinacija različitih metoda često donosi najbolje rezultate u praksi. Roboti često koriste kombinaciju senzora i algoritama kako bi ostvarili što preciznije pozicioniranje, ovisno o specifičnim zahtjevima okoline u kojoj se nalaze.

### 5.2 Lokalizacija koja se oslanja na referentne tačke

U nekim slučajevima, postavljaju se označke ili orijentiri s poznatim položajima u okolini. Roboti mogu koristiti mjerena senzora prema tim označama kako bi precizno odredili svoj položaj. Ove označke odnosno orijentiri mogu biti fizički objekti poput markera, svjetionika ili GPS satelita, ili mogu biti digitalne tačke koje se prepoznaju na osnovu senzora kao što su kamere ili senzori za prepoznavanje oblika. Orijentiri se odabiru tako da se što lakše identificiraju, te je potrebno da njihove karakteristike budu prethodno poznate i pohranjene u memoriju robota. Osnovni pristupi estimaciji pozicije temelje se na rješavanju geometrijskih ili trigonometrijskih

problema koji uključuju ograničenja na pozicije orijentira. Ovaj problem je povezan s procjenom pozicije orijentira u odnosu na nepomični senzor. Glavni faktori koji određuju upotrebu orijentira su: oblast na kojoj se orijentiri mogu detektirati, kakva je funkcionalna veza između mjerena orijentira i pozicije, i kako se manifestiraju pogreške. Orientiri mogu biti umjetni i prirodni. Umjetni orijentiri se postavljaju specifično za potrebe lokalizacije robota i obično se lako otkrivaju. Orientiri se također mogu podijeliti na pasivne i aktivne. Aktivni orijentiri su obično odašiljači koji emituju jedinstvene signale i raspoređeni su u okolini robota [6]. Ovaj pristup lokalizaciji igra ključnu ulogu u razvoju autonomnih sistema i mobilnih robota, omogućavajući im precizno navigiranje u različitim okruženjima.

Glavni problem u navigaciji temeljenoj na prirodnim orijentirima jeste detekcija i usklađivanje karakterističnih svojstava na temelju senzorskih ulaza. Ovaj kompleksan sistem navigacije općenito se sastoji od nekoliko ključnih komponenti:

- Senzor je odgovoran za detekciju prirodnih orijentira i razlikovanje njih od okolne podloge. Ovaj senzor koristi se za prikupljanje vizualnih informacija o okolini, uključujući oblike, boje i kontraste orijentira i njihove okoline.
- Postupak usklađivanja karakteristika koje su promatrane senzorom s mapom poznatih orijentira izuzetno je važan korak. Ovdje se koristi napredna računarska obrada kako bi se odredilo kako se promatrane karakteristike podudaraju s onima na mapi. Ovaj postupak često uključuje upotrebu algoritama za prepoznavanje oblika i uzoraka te praćenje pokazatelja koji pomažu u određivanju orijentacije.
- Nakon što se karakteristike usklade s mapom, slijedi postupak računanja položaja. To uključuje određivanje tačnog mjesta na temelju usklađenih podataka o orijentirima. Osim toga, sistem također procjenjuje pogrešku pozicioniranja kako bi se bolje razumjelo koliko je tačno određen položaj.
- Sistem može uključivati i kontinuirano praćenje položaja kako bi se ažurirao u stvarnom vremenu. Ovo praćenje može koristiti različite senzore, uključujući GPS, akcelerometre i žiroskope, kako bi se održala preciznost pozicioniranja, posebno tokom kretanja.

Detekcija je znatno jednostavnija s umjetnim orijentirima koji su pažljivo oblikovani kako bi postigli optimalan kontrast. Dodatna prednost umjetnih orijentira je da su njihove stvarne dimenzije (veličina) i oblik unaprijed poznati. Istraživači su razvili različite metode i tehnike za estimaciju položaja temeljene na različitim vrstama uzoraka ili oznaka na umjetnim orijentirima. Tačnost procesa lokalizacije ovisi o preciznosti geometrijskih parametara orijentira na slici, a ti parametri dalje ovise o relativnoj poziciji i ugлу između robota i orijentira. Važno je napomenuti da tačnost estimacije položaja obično opada s povećanjem relativne udaljenosti između robota i orijentira. Ova pojava je posebno izražena kod većih razmaka između objekata. U najčešće korištene senzore spadaju reflektori bar kodova za laserske skenere i kružni orijentiri. Ovi senzori se koriste za precizno mjerjenje udaljenosti i orijentacije, što dalje doprinosi tačnosti procesa lokalizacije.

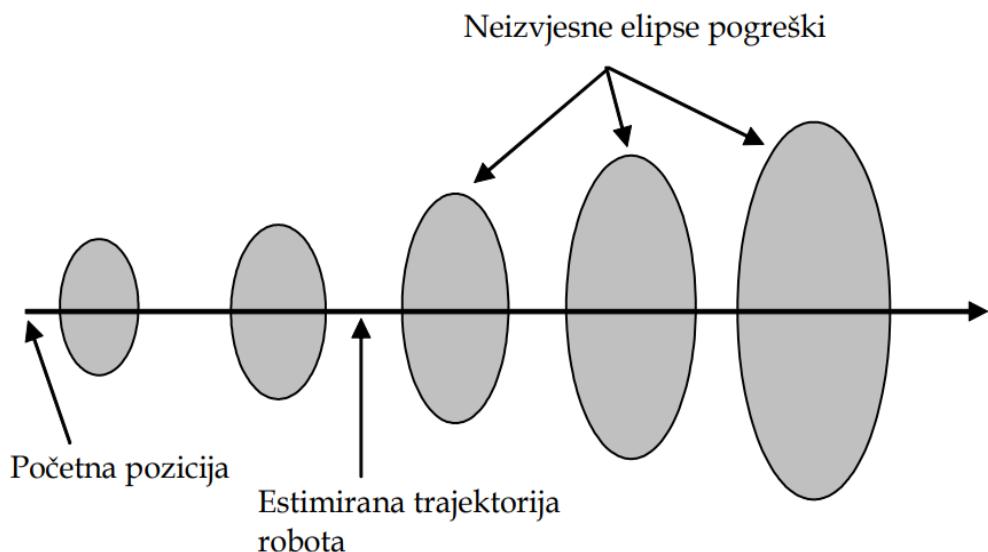
### 5.3 Odometrija

Odometrija je proces procjene položaja robota mjeranjem promjena u kretanju kotača. Ona podrazumijeva mjerjenje rotacije kotača kao funkcije vremena. Praćenjem broja okretaja kotača i pretpostavljajući da nema klizanja ili proklizavanja kotača, robot može procijeniti svoj položaj

u odnosu na početnu tačku. Optički inkrementalni enkoderi ugrađeni na pogonske kotače mobilnog robota pružaju diskretne inkrementalne podatke centralnom procesoru, koji neprestano ažurira položaj robota koristeći geometrijske jednadžbe. U idealnom slučaju, estimacija položaja robota može se dobiti integriranjem vektora brzina uz nulte početne uslove i u vremenskom intervalu od  $t_0$  do  $t_f$  [6]:

$$x = \int_{t_0}^{t_f} \frac{dx}{dt} dt \quad (5.1)$$

Međutim, odometrija ima tendenciju da nakuplja greške tokom vremena zbog klizanja kotača, hrapavosti podloge i diskretiziranog uzorkovanja pomaka kotača. Iz ovog razloga, odometrija se često koristi u kombinaciji s drugim metodama za preciznu lokalizaciju. Jedan od uzroka greške jeste činjenica da se odometrija oslanja na to da se rotacija kotača može translatirati u linijске pomake u odnosu na ravnu površinu po kojoj se kreće robot. Sve ove greške dijele se na dvije skupine: sistemske, koje nastaju iz faktora poput nejednakog polumjera kotača i konačne rezolucije enkodera, i nesistemske, koje nastaju zbog neravnog terena ili nailaska na neočekivane objekte [6]. Sistemske greške se konstantno akumuliraju i imaju veći doprinos kada se robot kreće po ravnim površinama. Nesistemske greške imaju veći doprinos na neravnim površinama i često se javljaju neočekivano. U svijetu mobilnih robota postoje mnogi algoritmi koji služe za procjenu nesigurnosti u položaju robota uzrokovanu odometrijskim mjeranjima. U ovim pristupima, svaki izračunati položaj robota prati karakteristična elipsa pogreške, slika 5.1, koja vizualno prikazuje regiju nesigurnosti oko stvarnog položaja robota. Obično, dimenzije ove elipse rastu kako robot putuje veće udaljenosti, odražavajući time povećanje nesigurnosti u poziciji uslijed inherentnih ograničenja odometrije. Kako bi se umanjila ova rastuća nesigurnost i poboljšala preciznost, često se koristi apsolutno pozicioniranje, poput GPS-a ili drugih referentnih tačaka. Kada se dobiju takvi apsolutni podaci o poziciji, nesigurnost se resetira, a veličina elipse pogreške se ponovno kalibrira. Ove tehnike estimacije pogrešaka temelje se na parametarskoj procjeni pogreške, koja se izvodi na temelju promatrana odometrijskih performansi robota. Parametri se uzimaju u obzir kako bi se korigirale sistemske greške u odometriji, čime se poboljšava tačnost estimacije položaja robota.



Slika 5.1: Elipse pogreške [6]

### 5.3.1 Mjerenje odometrijskih pogrešaka

Mjerenje odometrijskih pogrešaka je proces procjene i analize pogrešaka koje se javljaju u odometrijskom sistemu. Borenstein i Feng razvili su metode kvantitativnog mjerenja sistemskih i nesistemskih pogrešaka do ograničenog stepena. One se zasnivaju na modelu u kojem se dvije sistemske pogreške razmatraju kao dominantne. Ove dvije pogreške su [6]:

1. Pogreška zbog nejednakih promjera kotača:

$$E_d = \frac{D_R}{D_L} \quad (5.2)$$

gdje su  $D_R$  i  $D_L$  promjeri desnog i lijevog kotača.

2. Pogreška zbog nesigurnosti uslijed zakreta mobilne platforme:

$$E_b = \frac{b_{stvarno}}{b_{nominalno}} \quad (5.3)$$

gdje je  $b$  dužina mobilne platforme.

## 5.4 Simultana lokalizacija i mapiranje SLAM

SLAM (Simultaneous Localization and Mapping) je tehnika koja se koristi kako bi se istovremeno riješila dva osnovna problema:

- Lokalizacija: Robot mora znati gdje se nalazi u stvarnom vremenu.
- Mapiranje: Drugi problem je stvaranje mape okoline u kojoj se robot kreće. Ovo uključuje identifikaciju prepreka, značajki i drugih karakteristika okoline.

SLAM rješava oba ova problema istovremeno tako da robot koristi svoje senzore (kao što su kamere i senzori udaljenosti) kako bi prikupio podatke o okolini. Na temelju tih podataka, robot pokušava istovremeno izračunati svoj trenutni položaj i stvarati mapu okoline. To može biti izazovan problem jer su senzori često podložni šumovima, nepreciznosti i promjenjivim uvjetima okoline. U kontekstu lokalizacije, SLAM pomaže robotima da ostanu svjesni svog položaja čak i kada se kreću kroz nepoznato ili dinamičko okruženje. Na taj način, roboti mogu autonomno navigirati bez potrebe za unaprijed poznatom mapom, a također mogu ažurirati mapu kako se kreću.

# Poglavlje 6

## Implementacija algoritma za praćenje unutar ROS-a

### 6.1 Uvod

Programski kod za ROS se piše u jezicima C++, Python ili Lisp, uz korištenje biblioteka roscpp, rospy i roslibsp. Ove biblioteke omogućuju programerima da razvijaju robotske aplikacije i sisteme koristeći njima prikladne jezike. C++ se često koristi za optimizaciju performansi, dok je Python popularan zbog svoje jednostavnosti i brze implementacije. Zbog zahtjevnih resursa i složene strukture, preporučuje se pokretanje ROS-a na sistemima s dovoljno radne memorije i procesorske snage. Ubuntu Linux je prepoznat kao glavni operativni sistem za ROS zbog svoje stabilnosti i velike podrške zajednice. Pored standardnih biblioteka, postoji i Java ROS biblioteka nazvana rosjava, koja omogućava izvođenje ROS-a na Android platformi. Također, ROS se može integrirati i u Matlab toolbox, što olakšava analizu podataka i implementaciju algoritama za kontrolu robota.

Za potrebe pisanja koda za algoritam za praćenje zida i lokalizaciju korišten je C++ programska jezik. Algoritam je moguće podijeliti na nekoliko dijelova radi lakše realizacije i praćenja. Sastoji se iz pronalaska najbliže prepreke odnosno zida, kretanja ka zidu, rotaciji, te praćenju zida do ponovnog povratka u istu tačku. U nastavku su opisani osnovi ROS paketi korišteni prilikom pisanja koda.

### 6.2 Paket laser\_scan\_matcher

Paket laser\_scan\_matcher je alat za usklađivanje laserskih skeniranja. Ovaj paket omogućava skeniranje podudaranja između uzastopnih senzor\_msgs/LaserScan poruka i objavljivanje procijenjenog položaja lasera kao geometr\_msgs/Pose2D ili tf transformacije. Također, pomaže u određivanju trenutne pozicije robota u odnosu na globalni koordinatni sistem. Na temelju skeniranja laserskim senzorom i podataka iz mape, algoritam za usklađivanje skeniranja određuje gdje se robot nalazi. Algoritmi usklađivanja skeniranja često ovise o raznim parametrima, kao što su brzina, rezolucija i ugao skeniranja laserskog senzora. Paket omogućava postavljanje tih parametara kako bi se prilagodio konkretnim senzorima i okolini.

Paket je moguće koristit samostalno, bez procjene odometrije koju daju drugi senzori, ili sa odometrijom radi poboljšanja brzine i tačnosti. Unosi koje laser\_scan\_matcher prihvata su IMU (procjena za promjenu ugla orijentacije robota u obliku senzor\_msgs/IMU poruke), odometrija kotača (procjena promjene x, y i ugla orijentacije robota iz odometrijskog senzora), model kons-

tantne brzine (pretpostavlja da se robot pomaknuo na temelju procjene robotove brzine) i model nulte brzine (pretpostavlja da je robot ostao na istom mjestu).

Laser\_scan\_matcher koristi senzor\_msgs/LaserScan poruke ili senzor\_msgs/PointCloud2 poruke. Preplaćene teme su: scan (sensor\_msgs/LaserScan), cloud (sensor\_msgs/PointCloud2), imu/data (sensor\_msgs/Imu), odom (nav\_msgs/Odometry) i pose2D (geometry\_msgs/Pose2D). Ovaj paket se oslanja na transformacijski okvir koji pruža ROS za transformaciju laserskih skeniranja između različitih koordinatnih okvira. Pružene su dvije transformacije: base\_link → laser i world → base\_link.

U okviru napisanog koda korištene su senzor\_msgs/LaserScan i nav\_msgs/Odometry poruke, te scan (sensor\_msgs/LaserScan) tema. Za obradu podataka dobivenih od Velodyne 3D LiDAR senzora koristi se funkcija laserCallback. Ova funkcija se izvršava svaki put kad stigne poruka (u obliku sensor\_msgs::LaserScan) od LiDAR senzora. Ključni elementi sensor\_msgs/LaserScan poruke su angle\_min i angle\_max koji definišu opseg uglova za koje su izvršena mjerena, angle\_increment koji predstavlja korak između uzoraka, time\_increment koji označava vrijeme između uzoraka, range\_min i range\_max koji postavljaju minimalne i maksimalne vrijednosti za mjerena udaljenosti i ranges koji sadrži niz vrijednosti koje predstavljaju izmjerene udaljenosti na različitim uglovima.

```
void laserCallback(const sensor_msgs::LaserScan::ConstPtr &msg)
{
    // Pronalazak najbliže prepreke
    targetDistance = 9999.0;
    targetAngle = 0;
    for (size_t i = 0; i < msg->ranges.size(); ++i) {
        float range = msg->ranges[i];
        if (range < targetDistance) {
            targetDistance = range;
            targetAngle = msg->angle_min +
                          msg->angle_increment * i;
        }
    }
    // Postavljanje zeljenog ugla
    targetAngle = targetAngle + 1.57;
}
```

Unutar funkcije prvo se postavljaju početne vrijednosti za targetDistance (ciljna udaljenost) na veliku vrijednost (9999.0) i targetAngle (ciljni ugao) na 0. Ove varijable će se koristiti za praćenje najbliže prepreke detektirane od strane LiDAR senzora. Zatim slijedi petlja koja prolazi kroz raspoložive podatke o udaljenosti koje su dobivene od LiDAR senzora. Svaka udaljenost se uspoređuje s trenutnom vrijednošću targetDistance, a ako je nova udaljenost manja od trenutne targetDistance, targetDistance se ažurira na novu minimalnu udaljenost. Isto tako se ažurira i targetAngle kako bi odražavao ugao prema najbližoj prepreći. Konačno, targetAngle se povećava za 1.57 radijana ili 90° radi daljnje obrade u drugom dijelu koda.

Poruka nav\_msgs/Odometry je iskorištena u okviru funkcije updateRobotPosition za ažuriranje pozicije robota.

```
void updateRobotPosition(const nav_msgs::Odometry msg) {
    robotX = msg.pose.pose.position.x;
    robotY = msg.pose.pose.position.y;
}
```

Ova funkcija uzima nav\_msgs::Odometry poruku kao svoj argument, iz nje izvlači X i Y pozicije robota, te ažurira globalne varijable robotX i robotY s X i Y pozicijama, redom. Ovo omogućava drugim dijelovima koda da koriste informacije o poziciji robota.

## 6.3 Paket tf

Paket tf (skraćenica od "Transform") u ROS-u pruža snažan okvir za upravljanje transformacijama koordinata i korisniku omogućava praćenje više koordinatnih okvira tokom vremena. Ovaj paket održava odnos između koordinatnih okvira u strukturi stabla, te omogućava praćenje položaja i orijentacije različitih koordinatnih okvira u okruženju robota. Robotski sistem obično ima više 3D koordinatnih okvira koji se mijenjaju tokom vremena, poput svjetskog okvira (world frame), osnovnog okvira (base frame), okvira hvataljke (gripper frame), okvira glave (head frame) itd. Paket tf prati sve ove okvire tokom vremena i omogućava uspostavljanje veza između njih. tf može raditi u distribuiranom sistemu što znači da su sve informacije o koordinatnim okvirima robota dostupne svim ROS komponentama na bilo kojem računaru u sistemu.

Što se tiče koordinatnih okvira postoje fiksni okviri i okviri robota. Fiksni okviri su statički koordinatni okviri u okolini robota, poput okvira mape (map frame) i okvira base\_link. Ovi okviri služe kao referentni za druge okvire i ostaju nepromijenjeni tokom rada robota. Okviri robota su okviri koji se kreću s robotom i obično su povezani s različitim komponentama robota, poput senzora ili baznog okvira robota (npr. okvira base\_link). Transformacije se dijele na statičke i dinamičke. Statičke transformacije su konstantne transformacije između okvira koje se ne mijenjaju tokom vremena. Obično se koriste za fiksne odnose između okvira. Dinamičke transformacije se mogu mijenjati tokom vremena kako se robot kreće ili kako senzori pružaju nove podatke. Dinamičke transformacije kontinuirano se ažuriraju kako bi se održali tačni odnosi između okvira. Paket tf je dizajniran da besprijeckorno sarađuje sa drugim ROS paketima i bibliotekama. Obično se koristi u kombinaciji s upravljačima senzora, upravljačima robota, algoritmima lokalizacije i sistemima mapiranja.

Korištenje tf paketa demonstrirano je unutar funkcije rememberRobotPosition(), iako ona nije iskorištena u konačnom kodu. Ova funkcija prima obrađene podatke sa senzora iz funkcije laserCallback, te obavlja odgovarajuće transformacije koordinata radi ažuriranja pozicije robota.

```
void rememberRobotPosition() {
    tf::TransformListener listener;
    ros::Time currentTime = ros::Time::now();

    geometry_msgs::PointStamped laserRobotPosition;
    laserRobotPosition.header.stamp = currentTime;
    laserRobotPosition.header.frame_id = "laser";

    laserRobotPosition.point.x = robotX;
    laserRobotPosition.point.y = robotY;

    listener.waitForTransform("base_link", "laser",
        currentTime, ros::Duration(1.0));
    listener.transformPoint("base_link", laserRobotPosition,
        laserRobotPosition);
```

```
geometry_msgs::PointStamped RobotPosition;
RobotPosition.header.stamp = currentTime;
RobotPosition.header.frame_id = "base_link";

RobotPosition.point.x = laserRobotPosition.point.x;
RobotPosition.point.y = laserRobotPosition.point.y;

listener.waitForTransform("world", "base_link",
currentTime, ros::Duration(1.0));
listener.transformPoint ("world", RobotPosition,
RobotPosition);

if (!coordinatesSet) {
    rememberedX = RobotPosition.point.x;
    rememberedY = RobotPosition.point.y;
    coordinatesSet = true;
} else {
    double distanceToRemembered =
        std::hypot(RobotPosition.point.x - rememberedX,
                    RobotPosition.point.y - rememberedY);
    double distanceThreshold = 0.1;
    if (distanceToRemembered < distanceThreshold) {
        currentState = State::TERMINATE;
    }
}
}
```

Na početku je potrebno stvoriti objekat `tf::TransformListener`, koji se koristi za slušanje transformacija između koordinatnih okvira. Poruka `geometry_msgs::PointStamped` nazvana `laserRobotPosition` definirana je za pohranu položaja robota. Ova poruka sadrži vremensku oznaku i ID okvira, u ovom slučaju "laser". Zatim se postavljaju x i y koordinate položaja robota u odnosu na okvir lasera i pozicija robota se transformira u okvir "base\_link". Rezultat se pohranjuje nazad u `laserRobotPosition`. Stvara se nova `geometry_msgs::PointStamped` poruka nazvana `RobotPosition` za pohranu položaja robota. Ova poruka ima vremensku oznaku i okvir "base\_link". Postavljaju se x i y koordinate položaja robota u okviru "base\_link" kako bi se koristile u kasnijim transformacijama. Pozicija robota u okviru "base\_link" se transformira u okvir "world", a rezultat se pohranjuje nazad u `RobotPosition`. Ažuriranje položaja robota se obavlja provjerom varijable `coordinatesSet`. Ukoliko je ova varijabla postavljena na false, koordinate robota se upisuju u `rememberedX` i `rememberedY`. Ove dvije varijable pamte početnu poziciju robota i nakon što je ona zapamćena `coordinatesSet` se postavlja na true. Ako je `coordinatesSet` postavljen na true, računa se udaljenost između trenutnog položaja robota i zapisane početne pozicije pomoću Euklidove udaljenosti. Ako je ta udaljenost manja od zadanih praga (`distanceThreshold`), to znači da se robot vratio na početnu poziciju, pa se stanje postavlja na TERMINATE, što označava prekid svih radnji robota.

Funkcija je preimenovana kao `checkRobotPosition()` i modificirana tako da koristi podatke dobivene pomoću funkcije `updateRobotPosition` umjesto `tf` paketa. Princip je ostao isti, s tim da se za proračun udaljenosti robota od početne pozicije sada direktno koriste globalne varijable `robotX` i `robotY`. Također, ubačen je uslov koji provjerava vrijeme koje je proteklo od trenutka pamćenja početne pozicije, kako bi se izbjeglo gašenje robota odmah na početku kretanja.

```
void checkRobotPosition() {
    ros::Time currentTime = ros::Time::now();
    // Azuriranje pozicije robota
    if (!coordinatesSet) {
        rememberedX = robotX;
        rememberedY = robotY;
        begginTime = currentTime;
        coordinatesSet = true;
    } else if ((begginTime.toSec() -
                currentTime.toSec()) < -15) {
        // Provjera da li se robot vratio u pocetnu poziciju
        double distanceToRemembered = std::hypot(robotX -
                                                   rememberedX, robotY - rememberedY);
        double distanceThreshold = 0.4;
        if (distanceToRemembered < distanceThreshold) {
            ROS_INFO("Stigao u pocetnu poziciju");
            // Robot se vratio u pocetnu poziciju
            // Prekid svih radnji
            currentState = State::TERMINATE;
            geometry_msgs::Twist twist;
            twist.linear.x = 0.0;
            twist.angular.z = 0.0;
            cmdVelPublisher.publish(twist);
        }
    }
}
```

## 6.4 Paket geometry\_msgs

Paket geometry\_msgs je jedan od mnogih paketa u ROS-u koji sadrže definicije poruka za razmjenu podataka između različitih čvorova u ROS sistemu. Ovaj paket specifično sadrži poruke koje se koriste za opisivanje geometrijskih informacija, kao što su trodimenzionalne tačke, vektori i drugi geometrijski entiteti. Ove poruke se često koriste u različitim robotskim aplikacijama kako bi se razmjenjivali podaci o položaju, orijentaciji i drugim geometrijskim svojstvima objekata u okruženju. Neke od često korištenih poruka su geometry\_msgs/Point (opisuje tačku u trodimenzionalnom prostoru), geometry\_msgs/Vector3 (trodimenzionalni vektor), geometry\_msgs/Quaternion (opisuje rotacijski kvaternion u trodimenzionalnom prostoru), geometry\_msgs/Pose (kombinira poziciju i orijentaciju objekta u prostoru), geometry\_msgs/PoseStamped (pored pozicije i orijentacije sadrži informacije o vremenu i okviru), geometry\_msgs/Twist (opisuje brzinu i ubrzanje objekta).

Unutar koda koriste se dva tipa poruka iz geometry\_msgs paketa, geometry\_msgs/PointStamped i geometry\_msgs/Twist. Poruka geometry\_msgs/PointStamped već je pojašnjena u okviru funkcije rememberRobotPosition(). Parametri geometry\_msgs/Twist poruke su:

1. linear:

- x predstavlja linearnu brzinu duž x-osi. Pozitivne vrijednosti označavaju kretanje naprijed, dok negativne vrijednosti označavaju kretanje unazad.

- y predstavlja linearu brzinu duž y-osi i može se koristiti za lateralno kretanje.
- z predstavlja linearu brzinu duž z-osi.

2. angular:

- x predstavlja ugaonu brzinu oko x-osi. Pozitivne vrijednosti označavaju obrnuti smjer kazaljke na satu, dok negativne vrijednosti označavaju smjer kazaljke na satu.
- y predstavlja ugaonu brzinu oko y-osi.
- z predstavlja ugaonu brzinu oko z-osi.

Ovi parametri omogućavaju specificiranje načina na koji se objekat kreće u linearom i ugaonom smjeru unutar trodimenzionalnog prostora. Unutar koda `geometry_msgs/Twist` se koristi za kontrolu kretanja robota tako da se postavlja linearna i ugaona brzinu kako bi se robot kretao naprijed i obavljao rotaciju radi poravnjanja sa zidom. Detaljnije objašnjenje koda navedeno je u nastavku rada.

## 6.5 Algoritam za upravljanje robotom

U nastavku je navedena funkcija `run()` koja predstavlja glavnu petlju programa unutar koje se obavlja upravljanje robotom. U ovoj funkciji se izvršavaju različiti zadaci ovisno o trenutnom stanju robota i osigurava se kontinuirano izvršavanje programa. Najprije se postavlja frekvencija izvršavanja petlje na 10 Hz, a zatim se ulazi u while petlju koja se izvršava sve dok je ROS sistem u stanju "ok", odnosno dok nije zatražen prekid programa. Unutar petlje provjerava se uslov da je li trenutno stanje robota `TERMINATE`, što znači da je program završio i treba se prekinuti izvršavanje petlje. U switch case skretnici provjerava se trenutno stanje robota. Ovisno o trenutnom stanju, program će izvršiti odgovarajuću sekvencu akcija za to stanje. Na primjer, ako je robot u stanju `SEARCH`, izvršit će se funkcija `search()`. Ako je u stanju `APPROACH`, izvršit će se `approach()`, i tako dalje. Posljednje dvije naredbe omogućuju ROS-u da obradi sve pozive na čekanju čime je osigurano da se program redovito ažurira s novim informacijama iz okoline, a zatim se uspavljuje petlja na određeno vrijeme radi održavanja konstantnog vremenskog intervala između iteracija petlje. Ovako organizirana `run()` funkcija omogućava robotu da redovno obrađuje podatke sa senzora, donosi odluke na temelju trenutnog stanja i izvršava odgovarajuće akcije. Ako stanje robota postane `TERMINATE`, petlja će se prekinuti, a program će završiti.

```
void run() {
    ros::Rate rate(10);
    while (ros::ok()) {
        if (currentState == State::TERMINATE) {
            break;
        }
        switch (currentState) {
            case State::SEARCH:
                search();
                break;
            case State::APPROACH:
                approach();
                break;
        }
    }
}
```

```
        case State:::ROTATE:  
            rotate();  
            break;  
        case State:::LINEAR:  
            linear();  
            break;  
    }  
    ros::spinOnce();  
    rate.sleep();  
}  
}
```

Funkcija search() koja obavlja traženje prepreke se, radi svoje jednostavnosti, neće navoditi. Funkcija approach() služi za prilazak prepreci, odnosno zidu, nakon što je on detektovan. Najprije se provjerava poravnanje robota sa zidom, te ako on nije u ispravnom položaju, robot se rotira. Nakon što je postignuta željena orijentacija, robot započinje prilazak zidu. Robot nastavlja da se kreće dok se ne postigne željena udaljenost od zida, koja iznosi 2 metra, a zatim se stanje robota prebacuje u State:::ROTATE. Upravljanje robotom obavlja se pomoću geometry\_msgs::Twist poruke gdje se postavljaju linearna i ugaona brzina robota.

```
void approach() {  
    float angularTolerance = 0.2;  
    float desiredAngle = 1.57;  
    if (std::abs(targetAngle - desiredAngle)  
        > angularTolerance) {  
        float direction = (targetAngle > 1.57) ? 1.0 : -1.0;  
        geometry_msgs::Twist twist;  
        twist.linear.x = 0.0;  
        twist.angular.z = direction * 1.8;  
        cmdVelPublisher.publish(twist);  
    } else {  
        geometry_msgs::Twist twist;  
        twist.linear.x = 0.85;  
        twist.angular.z = 0.0;  
        cmdVelPublisher.publish(twist);  
    }  
    if (targetDistance < 2) {  
        currentState = State:::ROTATE;  
    }  
}
```

Kako je potrebno da se robot kreće tako da mu je zid uvijek sa desne strane, napisana je funkcija rotate() koja obavlja ovu rotaciju, odnosno poravnanje robota sa zidom. Nakon postavljanja ugaone tolerancije i željenog ugla, provjerava se orijentacija robota i u skladu s tim obavlja se upravljanje robotom ponovo korištenjem geometry\_msgs::Twist poruke. Ukoliko je postignut željeni ugao, s obzirom na postavljenu ugaonu toleranciju, stanje robota se postavlja na State:::LINEAR, čime robot prelazi na fazu linearног kretanja. Također, pozivom funkcije checkRobotPosition() postavlja se početna pozicija robota.

```
void rotate() {
    float angularTolerance = 0.2;
    float desiredAngle = 0;
    if (std::abs(targetAngle - desiredAngle)
        > angularTolerance) {
        float direction = (targetAngle > 0) ? 1.0 : -1.0;
        geometry_msgs::Twist twist;
        twist.linear.x = 0.0;
        twist.angular.z = direction * 1.8;
        cmdVelPublisher.publish(twist);
    } else {
        checkRobotPosition();
        currentState = State::LINEAR;
    }
}
```

Funkcija linear() definira ponašanje robota tokom linearног kretanja i omogуčava robotu da se kreće naprijed dok je usmјeren prema željenom uglu, a prelazi u fazu rotacije ako nije pravilno usmјeren. Najprije se poziva funkcija checkRobotPosition() radi ažuriranja trenutne pozicije robota i provjere da li se robot vratio u početnu poziciju. Robot se nastavlja kretati pravo sve dok mu je zid sa desne strane. Ukoliko ovo nije slučaj, vraća se u stanje State::ROTATE.

```
void linear() {
    checkRobotPosition();
    float angularTolerance = 0.2;
    float desiredAngle = 0;
    geometry_msgs::Twist twist;
    if (std::abs(targetAngle - desiredAngle) <
        angularTolerance) {
        if (targetDistance < 1.95) {
            ROS_INFO("Skreni lijevo");
            twist.linear.x = 0.5;
            twist.angular.z = 0.2;
            cmdVelPublisher.publish(twist);
        } else if (targetDistance > 2.05) {
            ROS_INFO("Skreni desno");
            twist.linear.x = 0.5;
            twist.angular.z = (-1) * 0.2;
            cmdVelPublisher.publish(twist);
        } else {
            ROS_INFO("Idem pravo");
            twist.linear.x = 0.6;
            twist.angular.z = 0.0;
            cmdVelPublisher.publish(twist);
        }
    } else {
        currentState = State::ROTATE;
    }
}
```

# Poglavlje 7

## Testiranje algoritma

### 7.1 Uvod

U potrazi za razvojem robusnih i učinkovitih algoritama, simulacijski okviri igraju ključnu ulogu u testiranju i provjeri njihove ispravnosti. Ovo poglavlje fokusira se na upotrebu Gazebo platforme za simulaciju kako bi se ocijenila izvedba algoritma dizajniranog za praćenje zida i lokalizaciju robota. Gazebo pruža svestranu i realističnu okolinu za provedbu ovih eksperimenata. Koristeći ovu simulacijsku platformu, moguće je temeljito ocijeniti sposobnosti algoritma u kontroliranom i dinamičkom okruženju. Kroz ovu evaluaciju temeljenu na simulaciji, cilj je demonstrirati potencijal algoritma za primjenu u stvarnom svijetu i istaknuti njegov doprinos širem području robotike i autonomnih sistema.

### 7.2 Gazebo simulator

Gazebo je open-source simulacijsko okruženje koje se često koristi u istraživanju i razvoju robotskih sistema. Ovo simulacijsko okruženje omogućava stvaranje i testiranje robotskih modela, algoritama i kontrolera u virtualnom svijetu prije nego što se oni implementiraju na stvarnim fizičkim robotima. Gazebo daje pristup simulaciji s kompletним paketom alata, razvojnih biblioteka i usluga koje olakšavaju simulaciju. Koristi se u mnogim područjima, uključujući robotiku, autonomna vozila, dronove i više.

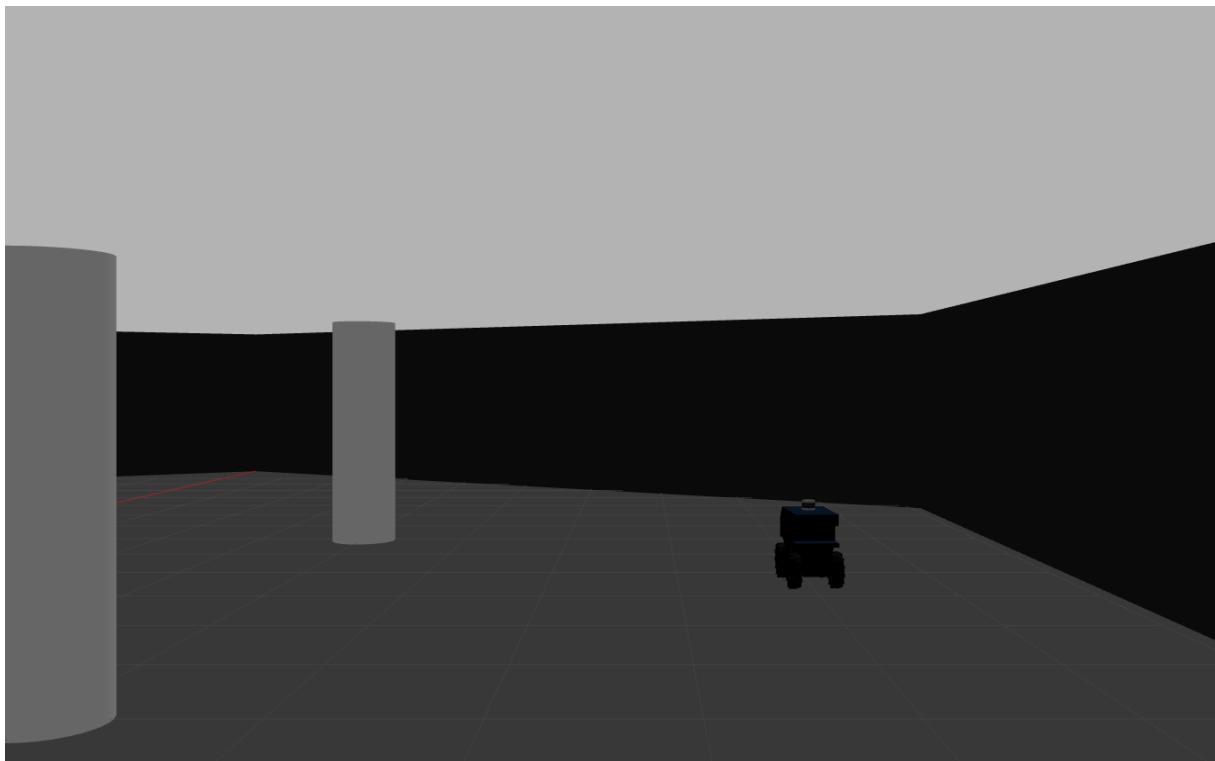
Gazebo pruža mogućnost izrade složenih 3D simulacija okoline u kojima se roboti mogu kretati, interagirati s objektima i izvršavati različite zadatke. Također, podržava različite fizičke modele, uključujući modele za dinamiku tijela, senzore, kolizije i mnoge druge aspekte koji su važni za simulaciju robotskih sistema. Vrlo se jednostavno integrira sa ROS-om i omogućava lako povezivanje ROS čvorova s robotskim modelima u Gazebo okruženju. Pored toga, podržava dodatne module koji omogućuju korisnicima prilagodbu i proširenje funkcionalnosti simulacije prema njihovim specifičnim potrebama. Jedna od osnovnih prednosti Gazebo okruženja jeste realističnost. Simulacije mogu biti prilično realistične, uključujući efekte poput gravitacije, trenja, senzorskih šumova i drugih faktora koji utječu na ponašanje robotskog sistema. S obzirom na svoju široku primjenu i zajednicu koja podržava njegov razvoj i proširenje, Gazebo simulacijsko okruženje ostaje ključni alat u robotici i autonomnoj vožnji, olakšavajući testiranje i razvoj robotskih sistema prije nego što se kreće na teren ili u proizvodnju.

### 7.3 Testiranje algoritma u okviru Gazebo okruženja

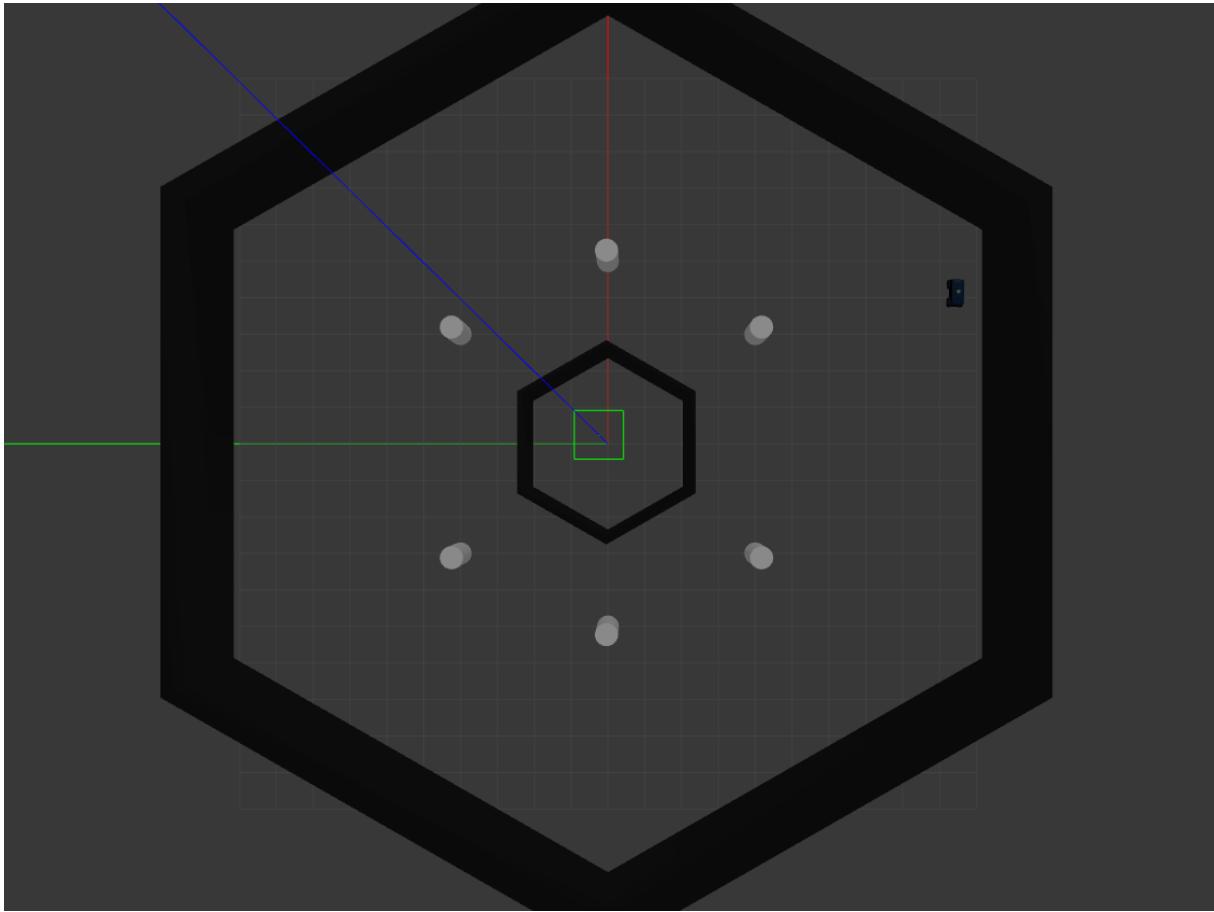
Testiranje razvijenog algoritma upravljanja obavljeno je unutar simulacijskog okvira Gazebo koji se pokreće naredbom

```
roslaunch turtlebot3_gazebo x100_world.launch
```

Prilikom otvaranja Gazebo prozora, prikazuje se simulirano okruženje u obliku heksagona, a unutar tog okruženja smješten je robot X100, slike 7.1 i 7.2. Osnovni zadatak robota X100 jeste da pronađe i priđe najbližem zidu unutar simulacije, nakon čega se mora rotirati tako da mu je zid s desne strane, te potom slijediti taj zid sve dok se ne vrati u početnu poziciju, kada će zaustaviti svoje kretanje. Nakon što je kod algoritma pokrenut, robot X100 počinje izvršavati zadane korake. Prvo se usmjerava prema najbližem zidu i postepeno se približava njemu. Nakon što je postigao određenu blizinu prema zidu, robot izvršava rotaciju kako bi postavio zid s desne strane, u skladu s postavkom zadatka. Nakon toga, započinje praćenje zida, slijedeći ga uz desnu stranu. Važno je napomenuti da se kretanje robota unutar simulacijskog okvira čini sporijim nego što bi to bilo u stvarnom okruženju. Ova prilagodba brzine bila je neophodna kako bi se osiguralo precizno izvršavanje algoritma i kako bi se omogućilo da robot pouzdano detektira povratak u početnu poziciju.

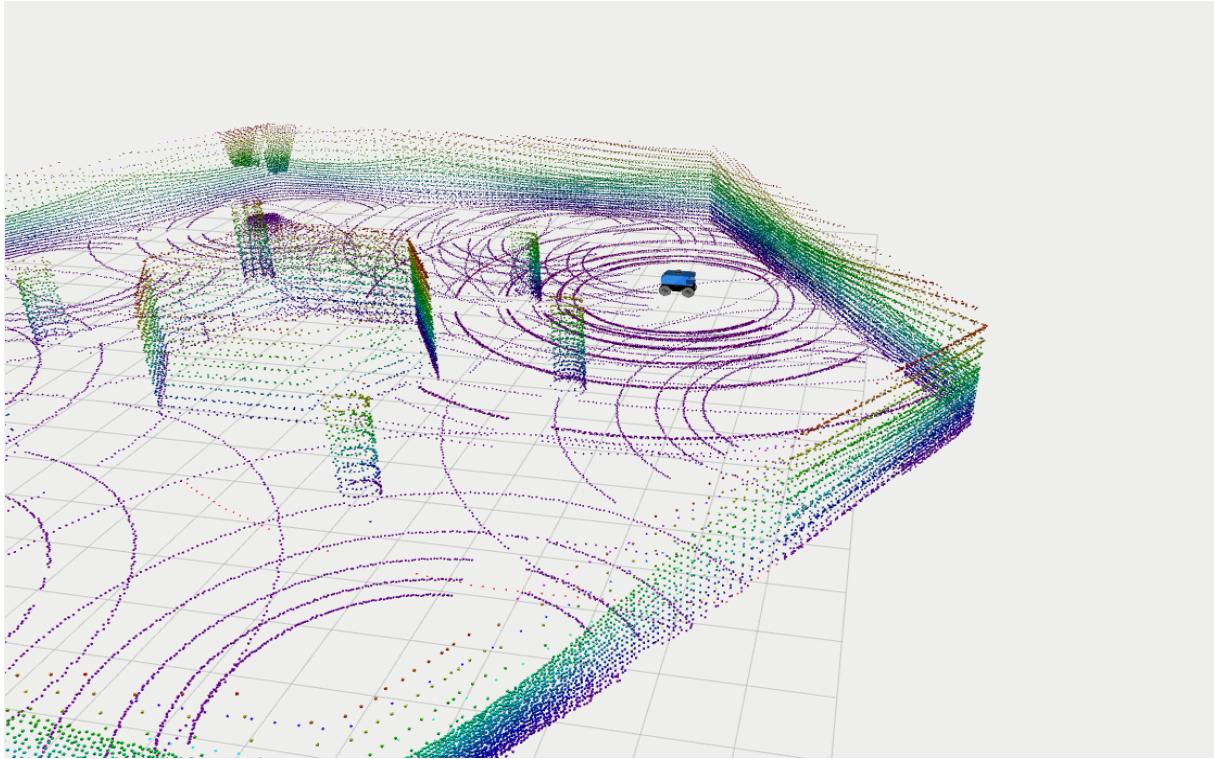


Slika 7.1: X100 unutar Gazebo okruženja



**Slika 7.2:** X100 unutar Gazebo okruženja

Ovaj testni scenarij omogućava procjenu kako se razvijeni algoritam ponaša unutar simuliranog okruženja te provjeru njegove sposobnosti izvršavanja složenih zadataka upravljanja robota. Na temelju rezultata ovih testova moguće je donijeti zaključke o efikasnosti i pouzdanosti algoritma, što će biti od ključnog značaja za dalji razvoj i primjenu robota X100 u stvarnom svijetu. Uz provođenje testiranja algoritma upravljanja, uključeno je i mapiranje okoline. Mapiranje omogućava robotu da stvori sliku okoline koju istražuje tokom svog kretanja i da je zadrži kako bi je mogao analizirati nakon povratka u početnu poziciju. Nakon što je robot izvršio sve korake zadatka i vratio se u početnu poziciju, dobivena je mapa okoline generirane unutar Rviz-a, slika 7.3. Rviz je čest alat za vizualizaciju i analizu podataka u robotskim aplikacijama te omogućava pregled generiranih mapa i drugih relevantnih podataka u stvarnom vremenu. Dobivena mapa sadrži informacije o zidovima, preprekama i drugim relevantnim karakteristikama okoline koju je robot istraživao tokom svog kretanja. Korištenje mapiranja omogućava nadogradnju algoritma jer pruža dodatne informacije o okolini. Ove informacije mogu biti korisne za buduće zadatke ili za poboljšanje samog algoritma. Na primjer, mapa okoline može se koristiti za planiranje optimalnih putanja, izbjegavanje prepreka ili za razumijevanje strukture okoline. Ovo dodatno mapiranje okruženja predstavlja važan aspekt unapređenja algoritma upravljanja i povećava korisnost robota X100 u različitim scenarijima primjene gdje je potrebna autonomna navigacija i razumijevanje okoline.



Slika 7.3: Generirana mapa unutar Rviz-a

## 7.4 Testiranje na stvarnom sistemu

Kod je moguće testirati i na stvarnom sistemu, što je u ovom slučaju mobilni robot X100. Prije samog pokretanja X100 mobilnog robota potrebno je postaviti ROS master na računaru. Komanda

```
sudo gedit ~/.bashrc
```

otvara skriptu koja se izvršava prije svake sesije i prilikom otvaranja novog terminala. U bashrc datoteci postavljaju se komande za definisanje ROS mastera i ROS slavea putem njihovih IP adresa.

```
export ROS_MASTER_URI=http://IpAdresaRacunara:11311
export ROS_IP=IpAdresaRacunara
```

Na robotu X100 obavlja se isti postupak s tim da se sada unosi IP adresa robota na sljedeći način

```
export ROS_MASTER_URI=http://IpAdresaRacunara:11311
export ROS_IP=IpAdresaX100
```

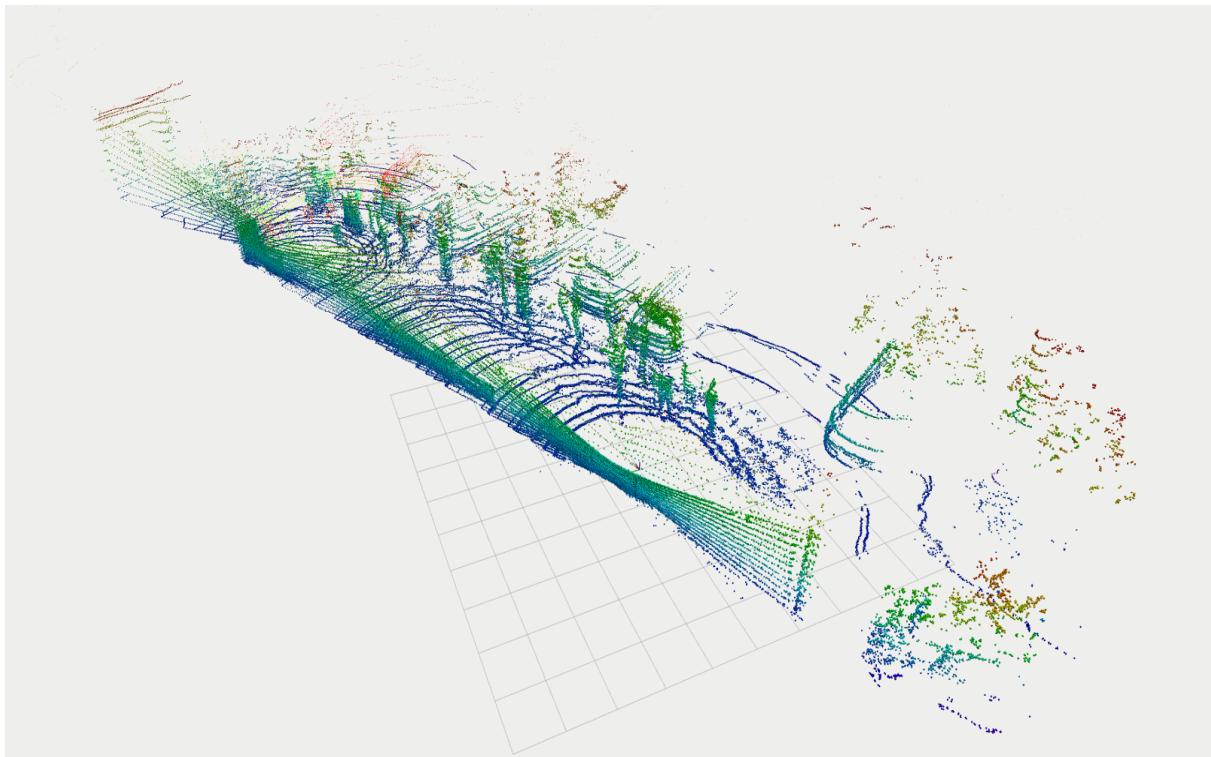
Ukoliko IP adresa nije već poznata može se iskoristiti naredba

```
hostname -I
```

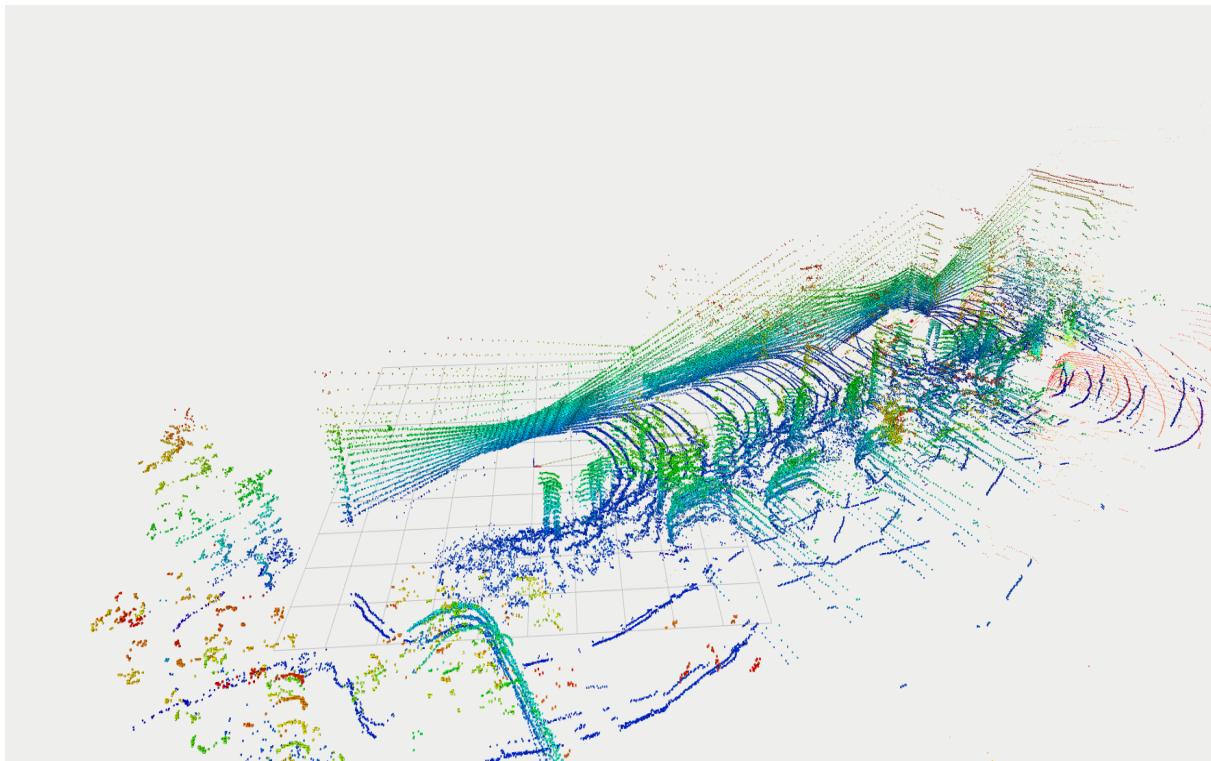
koja prikazuje popis IP adresa dodijeljenih računaru. Nakon postavljanja IP adresa potrebno je pokrenuti naredbu roscore za inicijalizaciju ROS mastera i ispravnu komunikaciju između čvorova. Unutar zasebnih terminala na X100 mobilnom robotu potrebno je otkucati naredbe:

```
roslaunch x100_base x100_base.launch  
roslaunch velodyne_pointcloud VLP16_points.launch  
roslaunch laser_scan_matcher demo.launch
```

za pokretanje robota, pokretanje lidara i konfiguraciju mogućnosti usklađivanja laserskog skeniranja. Nakon što je robot X100 konfiguriran prema postavljenim specifikacijama, kod za njegovo upravljanje je pokrenut u stvarnom okruženju ispred zgrade fakulteta. Važno je napomenuti da su se pojavili problemi sa ograničenim resursima, uključujući nedostatak vremena i adekvatnog prostora za potpunu validaciju zadatka, posebno u vezi sa provjerom da li se robot X100 zaista zaustavlja nakon povratka u početnu poziciju. Unatoč ovom ograničenju, uspješno je izvršena validacija drugih dijelova koda. Testiranje je potvrdilo da robot X100 ispravno izvršava osnovne zadatke, uključujući poravnanje sa zidom i prilagođavanje brzine kako bi se osiguralo precizno praćenje zida. Ovo je ključno za sigurno i precizno kretanje robota u stvarnom okruženju, što je od suštinskog značaja za mnoge primjene u kojima bi se robot mogao koristiti. Tokom testiranja, također je pokrenut proces mapiranja, čiji rezultati su prikazani na slikama 7.4 i 7.5. Na prikazanim mapama se mogu vidjeti registrovane površine, uključujući zid zgrade i okolnu vegetaciju. Ovo mapiranje pruža dodatne informacije o okolini i omogućava robotu da stvari sliku okoline u kojoj se nalazi. Također, unutar Rviz-a, generirane mape omogućavaju korisnicima da detaljno pregledaju informacije o okolini. Ova vizualizacija može biti korisna za analizu i dijagnostiku ponašanja robota tokom testiranja. Iako nije bilo moguće potvrditi potpuno zaustavljanje robota nakon povratka u početnu poziciju, uspješno testiranje drugih aspekata koda i uspješno mapiranje predstavljaju važan korak prema razvoju i primjeni algoritma upravljanja robota X100 u stvarnom svijetu.



**Slika 7.4:** Generirana mapa okruženja unutar Rviz-a



Slika 7.5: Generirana mapa okruženja unutar Rviz-a

# Poglavlje 8

## Zaključak

U radu je uspješno implementiran algoritam za praćenje zidova i lokalizaciju mobilnog robota X100, koristeći ROS kao temeljnu platformu. Implementacija je obuhvatila teorijski pregled ROS-a, mobilnih robota s posebnim naglaskom na X100 model, LiDAR tehnologije i lokalizacije. Kroz pažljivu analizu i kodiranje, razvijen je algoritam koji omogućava robotu X100 da precizno prati zidove u simuliranom okruženju unutar Gazebo simulacijskog okvira. Algoritam se sastoji od nekoliko ključnih koraka: pronađazak najbližeg zida, kretanje prema tom zidu, rotacija kako bi se postavio zid s desne strane robota, te praćenje zida sve do povratka u početnu poziciju, gdje se robot zaustavlja. Tokom simulacije u Gazebo okruženju, bilo je potrebno nekoliko iteracija kako bi se modificirao kod, radi postizanja optimalnog kretanja robota. Poseban izazov predstavlja problem registriranja povratka u početnu poziciju, što je uspješno riješeno prilagodbom brzine kretanja robota, omogućavajući mu da prikupi precizne podatke o svojoj poziciji. Osim simulacije, kod je testiran i na stvarnom robotu X100, gdje je potvrđeno da se robot dobro kreće uz zid. Ipak, nije bilo moguće potvrditi njegovo potpuno zaustavljanje po povratku u početnu poziciju zbog ograničenja u vremenu i dostupnom prostoru za testiranje. Kao nadogradnja algoritma, uključeno je mapiranje okoline unutar Rviz-a, što omogućava robotu da stvara sliku okoline i pohranjuje je radi buduće analize. Ovo mapiranje pruža dodatne informacije o okolini i potencijalno može poslužiti za planiranje optimalnih putanja, izbjegavanje prepreka ili bolje razumijevanje okoline za buduće zadatke. U budućim istraživanjima i primjenama, algoritam za praćenje zidova i lokalizaciju može se nadograditi i optimizirati kako bi se postigla veća preciznost i pouzdanost u stvarnom okruženju. Ovaj algoritam može imati široku primjenu u industriji, istraživanju okoliša i mnogim drugim područjima gdje se zahtjeva autonomno kretanje mobilnih robota. Kako tehnologija napreduje, za očekivati je da će ovakvi roboti igrati sve važniju ulogu u različitim sektorima, poboljšavajući učinkovitost i pridonoseći rješavanju različitih izazova.

# Poglavlje 9

## Dodatak

```
1 #include <ros/ros.h>
2 #include <sensor_msgs/LaserScan.h>
3 #include <geometry_msgs/Twist.h>
4 #include <nav_msgs/Odometry.h>
5 #include <geometry_msgs/PointStamped.h>
6 #include <cmath>
7
8 class ObstacleAvoidance {
9 public:
10     ObstacleAvoidance() {
11         // Postavljanje pocetne udaljenosti na veliku vrijednost
12         targetDistance = 9999.0;
13         // Postavljanje pocetnog stanja na trazenje prepreke
14         currentState = State::SEARCH;
15
16         laserSubscriber = nh.subscribe("/scan", 10, &ObstacleAvoidance::
17                                         laserCallback, this);
18         cmdVelPublisher = nh.advertise<geometry_msgs::Twist>("/cmd_vel",
19                                         10);
20         odomSubscriber = n.subscribe("/x100/odom", 10, &ObstacleAvoidance::
21                                         updateRobotPosition, this);
22     }
23
24     // Funkcija za obradu podataka sa senzora
25     void laserCallback(const sensor_msgs::LaserScan::ConstPtr &msg) {
26         // Pronalazak najblize prepreke
27         targetDistance = 9999.0;
28         targetAngle = 0;
29         for (size_t i = 0; i < msg->ranges.size(); ++i) {
30             float range = msg->ranges[i];
31             if (range < targetDistance) {
32                 targetDistance = range;
33                 targetAngle = msg->angle_min + msg->angle_increment * i;
34             }
35         }
36         // Postavljanje zeljenog ugla
37         targetAngle = targetAngle + 1.57;
38     }
39     void updateRobotPosition(const nav_msgs::Odometry msg) {
40         robotX= msg.pose.pose.position.x;
41         robotY= msg.pose.pose.position.y;
42     }
43 }
```

```
41 void run() {
42     ros::Rate rate(10); // Frekvencija od 10 Hz
43
44     while (ros::ok()) {
45         // Provjera zahtjeva za prekidom
46         if (currentState == State::TERMINATE) {
47             // Izlazak iz petlje i obustava svih radnji
48             break;
49         }
50         // Opcije za razlicite radnje u zavisnosti od trenutnog stanja
51         switch (currentState) {
52             case State::SEARCH:
53                 search();
54                 break;
55             case State::APPROACH:
56                 approach();
57                 break;
58             case State::ROTATE:
59                 rotate();
60                 break;
61             case State::LINEAR:
62                 linear();
63                 break;
64         }
65         ros::spinOnce();
66         rate.sleep();
67     }
68 }
69
70 private:
71     enum class State {
72         SEARCH, APPROACH, ROTATE, LINEAR, TERMINATE
73     };
74
75     ros::NodeHandle nh;
76     ros::NodeHandle n;
77     ros::Subscriber laserSubscriber;
78     ros::Publisher cmdVelPublisher;
79     ros::Subscriber odomSubscriber;
80     ros::Time begginTime;
81     float targetDistance;
82     float targetAngle;
83     float robotX;
84     float robotY;
85     State currentState;
86     float rememberedX;
87     float rememberedY;
88     bool coordinatesSet;
89
90     // Funkcija koja obavlja trazanje prepreke
91     void search() {
92         // Ako je prepreka pronadjena, prelazak na stanje prilazak
93         if (targetDistance < 9999.0) {
94             currentState = State::APPROACH;
95             coordinatesSet = false;
96         }
97     }
98 }
```

```
99 // Funkcija koja obavlja prilazak prepreci
100 void approach() {
101     float angularTolerance = 0.2;
102     float desiredAngle = 1.57;
103
104     // Provjera poravnanja sa preprekom
105     if (std::abs(targetAngle - desiredAngle) > angularTolerance) {
106         // Smjer kretanja
107         ROS_INFO("Poravnavam se");
108         float direction = (targetAngle > 1.57) ? 1.0 : -1.0;
109         geometry_msgs::Twist twist;
110         twist.linear.x = 0.0;
111         twist.angular.z = direction * 1.8;
112         cmdVelPublisher.publish(twist);
113     } else {
114         // Linearno kretanje prema prepreci
115         ROS_INFO("prilazim prepreci");
116         geometry_msgs::Twist twist;
117         twist.linear.x = 0.85;
118         twist.angular.z = 0.0;
119         cmdVelPublisher.publish(twist);
120     }
121
122     if (targetDistance < 2) {
123         // Robot se priblazio prepreci
124         ROS_INFO("prisao sam prepreci");
125         currentState = State::ROTATE;
126     }
127 }
128
129 // Funkcija koja obavlja rotaciju nakon prilaska
130 void rotate() {
131     float angularTolerance = 0.2;
132     float desiredAngle = 0;
133
134     // Provjera poravnanja sa preprekom
135     if (std::abs(targetAngle - desiredAngle) > angularTolerance) {
136         // Rotacija
137         ROS_INFO("rotiram se");
138         float direction = (targetAngle > 0) ? 1.0 : -1.0;
139         geometry_msgs::Twist twist;
140         twist.linear.x = 0.0;
141         twist.angular.z = direction * 1.8;
142         cmdVelPublisher.publish(twist);
143     } else {
144         // Prepreka je s desne strane, prebacivanje na stanje linearnog
145         // kretanja
146         rememberRobotPosition();
147         currentState = State::LINEAR;
148     }
149
150 // Funkcija koja obavlja linearno kretanje
151 void linear() {
152     rememberRobotPosition();
153     float angularTolerance = 0.2;
154     float desiredAngle = 0;
155     // float direction = (targetAngle > 0) ? 1.0 : -1.0;
```

```
156     geometry_msgs::Twist twist;
157     // Provjera poravnanja sa preprekom
158     if ( std::abs(targetAngle - desiredAngle) < angularTolerance ) {
159         // Move forward
160
161         if (targetDistance < 1.95){
162             ROS_INFO("skreni lijevo");
163             twist.linear.x = 0.4;
164             twist.angular.z = 0.2;
165             cmdVelPublisher.publish(twist);
166         }
167         else if (targetDistance > 2.05){
168             ROS_INFO("skreni desno");
169             twist.linear.x = 0.4;
170             twist.angular.z = (-1) * 0.2;
171             cmdVelPublisher.publish(twist);
172         } else {
173
174             ROS_INFO("idem pravo");
175             //geometry_msgs::Twist twist;
176             twist.linear.x = 0.5;
177             twist.angular.z = 0.0;
178             cmdVelPublisher.publish(twist);
179         } else {
180             // Prepreka nije s desne strane , prebacivanje na stanje rotacije
181             currentState = State::ROTATE;
182             /* */ else {
183                 // Prepreka nije s desne strane , prebacivanje na stanje
184                 rotacije
185                 currentState = State::ROTATE; */
186             }
187         }
188
189
190         // Funkcija za pamcenje pocetne pozicije robota i provjeru trenutne
191         // pozicije
192         void rememberRobotPosition() {
193
194             ros::Time currentTime = ros::Time::now();
195             // Azuriranje pozicije robota
196             if (!coordinatesSet) {
197                 rememberedX = robotX;
198                 rememberedY = robotY;
199                 begginTime = currentTime;
200                 coordinatesSet = true;
201             } else if ((begginTime.toSec() - currentTime.toSec()) < -15){
202                 // Provjera da li se robot vratio u pocetnu poziciju
203                 double distanceToRemembered = std::hypot(robotX - rememberedX,
204                     robotY - rememberedY);
205                 double distanceThreshold = 0.4; // Adjust this threshold as
206                 needed
207                 if (distanceToRemembered < distanceThreshold) {
208                     ROS_INFO("STIGAOOOOOOO");
209                     // Robot se vratio u pocetnu poziciju , prekid svih radnji
210                     currentState = State::TERMINATE;
211                     geometry_msgs::Twist twist;
212                     twist.linear.x = 0.0;
```

```
210         twist.angular.z = 0.0;
211         cmdVelPublisher.publish(twist);
212     }
213 }
214 };
215 };
216
217 int main(int argc, char** argv) {
218     ros::init(argc, argv, "obstacle_avoidance");
219     ObstacleAvoidance obstacleAvoidance;
220     obstacleAvoidance.run();
221 }
```

# Literatura

- [1] ROS - Robot Operating System, ROS, posljednji pristup 1.9.2023., dostupno na: <https://www.ros.org/>
- [2] Ricardo Tellez, A History of ROS, 2019.
- [3] Lentin Joseph, Learning Robotics Using Python, Packt Publishing Ltd, 2015.
- [4] rqt graph – Visualize and Debug Your ROS Graph, The Robotics Back-End, posljednji pristup 28.8.2023., dostupno na: <https://roboticsbackend.com/rqt-graph-visualize-and-debug-your-ros-graph/>
- [5] ROS - Documentation, ROS, posljednji pristup 13.9.2023., dostupno na: <http://wiki.ros.org/Documentation>
- [6] Jasmin Velagić, Mobilna robotika, Univerzitet u Sarajevu, Elektrotehnički fakultet, 2012.
- [7] Karl Bayer, Wall Following for Autonomous Navigation, Columbia University, 2012.
- [8] X100 UGV, Outdoor Unmanned Ground Vehicle For Robotics Autonomy Research, XMachines, posljednji pristup 2.9.2023., dostupno na: <https://www.xmachines.ai/x100>
- [9] What is lidar?, National Ocean Service, posljednji pristup 2.9.2023., dostupno na: <https://oceanservice.noaa.gov/facts/lidar.html>
- [10] Srushti Neoge, Ninad Mehendale, Review on LiDAR technology, 2020., dostupno na: <https://ssrn.com/abstract=3604309>
- [11] Airborne LiDAR and Bathymetry, FindLight, posljednji pristup 28.8.2023., dostupno na: <https://www.findlight.net/blog/airborne-lidar-bathymetry/>
- [12] Velodyne Puck LiDAR sensor, Velodyne Lidar, posljednji pristup 2.9.2023., dostupno na: <https://velodynelidar.com/products/puck/>
- [13] Ilya Belkin, Alexander Abramenko, Dmitry Yudin, Real-Time Lidar-based Localization of Mobile Ground Robot, Procedia Computer Science, vol. 186, pp. 440-448, 2021.