Integracija robotskog operativnog sustava (ROS) na mobilnom robotu FESTO Robotino

Cirkvenčić, Karlo

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

ZAVRŠNI RAD

Karlo Cirkvenčić

Zagreb, 2023.

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

ZAVRŠNI RAD

Integracija robotskog operativnog sustava (ROS) na mobilnom robotu FESTO Robotino

Mentor:
Doc. dr. sc. Marko Švaco

Student:
Karlo Cirkvenčić

Zagreb, 2023.

Zahvaljujem mentoru doc. dr. sc. Marku Švaci na zadavanju teme i pruženoj pomoći tijekom izrade rada te kolegi Branimiru Ćaranu na stalnoj dostupnosti i usmjeravanju.

Izjava

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

Zagreb, rujan 2023. Karlo Cirkvenčić

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

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 23 – 6 / 1**	
Ur.broj: **15 - 1703 - 23 -**	

ZAVRŠNI ZADATAK

Student: **Karlo Cirkvenčić** JMBAG: **0035220802**

Naslov rada na hrvatskom jeziku: **Integracija robotskog operativnog sustava (ROS) na mobilnom robotu FESTO Robotino**

Naslov rada na engleskom jeziku: **Integration of the Robot Operating System (ROS) on the FESTO Robotino mobile robot**

Opis zadatka:

U sklopu završnog rada potrebno je razviti upravljanje mobilnim robotom FESTO Robotino primjenom robotskog operativnog sustava (ROS). Upravljanje robotom pomoću robotskog operativnog sustava pruža korisniku okolinu za razvoj modularne upravljačke programske podrške, komunikacijsku infrastrukturu koja povezuje programske komponente te otvorenu biblioteku implementiranih algoritama. ROS omogućuje primjenu već razvijenih algoritama na bilo kojem robotu uz manje prilagodbe zato što se unutar ROS-a koristi standard za razmjenu poruka koji je strogo definiran.

U sklopu rada potrebno je:

- Detaljno proučiti dokumentaciju i API *(*eng. *Application Programming Interface*) za FESTO za Robotino robota i opisati sve njegove značajke.
- Na robota je potrebno integrirati računalo koje će komunicirati s robotom i omogućiti upravljanje putem ROS-a.
- Integrirati 2D lidar na robota i omogućiti komunikaciju s ROS-om.
- Implementirati algoritme mapiranja, lokalizacije i navigacije.
-

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

Zadatak zadan:
30. 11. 2022.

Datum predaje rada:
1. rok: 20. 2. 2023.
2. rok (izvanredni): 10. 7. 2023.
3. rok: 18. 9. 2023.

Predviđeni datumi obrane:
1. rok: 27. 2. – 3. 3. 2023.
2. rok (izvanredni): 14. 7. 2023.
3. rok: 25. 9. – 29. 9. 2023.

Zadatak zadao:
Doc. dr. sc. Marko Švaco

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

Sadržaj

Popis slika

Popis tablica

Popis kratica

- CRTA - Regionalni centar izvrsnosti za robotske tehnologije
- ROS - robotski operativni sustav
- API - aplikacijsko programsko sučelje
- USB - vrsta serijske sabirnice koja omogućuje spajanje računalnih uređaja u lanac i veću propusnost podataka
- IP - standard koji definira način na koji se podaci prenose po mreži i prepoznaju kao poruke

Sažetak

Zadatak završnog rada je na robotsku platformu Robotino 3, tvrtke FESTO, instalirati robotski operativni sustav (ROS) s ciljem implementacije algoritama mapiranja, lokalizacije i navigacije. Pomoću ROS-a izvodi se upravljanje robotom. Detaljno je razrađen proces mapiranja prostora CRTA-e, lokalizacija robota te ostvarivanje autonomne navigacije robota u spomenutom prostoru. Svi potrebni parametri definirani su i obrazloženi prilikom implementacije navedenih algoritama. Kako bi cijeli proces činio zaokruženu cjelinu, pred kraj rada iznesene su mogućnosti poboljšanja i daljnjeg unaprijeđenja korištene robotske platforme.

Ključne riječi: ROS, SLAM, Robotino, navigacija

Summary

The assingment of final paper is to install a robot operating system (ROS) on the robot platform Robotino 3, manufactured by Festo company, with the aim of implementing mapping, localization and navigation algorithms. ROS is used to control the robot. The process of mapping the space of the CRTA, the localization of the robot and the realization of autonomous navigation of the robot in the mentioned space are elaborated in detail. All the necessary parameters are defined and explained during the implementation of the mentioned algorithms. In order to make the whole process well rounded the possibilities of improvement and further advancement of the used robotic platform are presented towards the end of the paper.

Keywords: ROS, SLAM, Robotino, navigation

Poglavlje 1.

Uvod

Mobilna robotika svakim danom sve više prožima našu svakodnevicu. Od nekad skupe i teško dostupne tehnologije, koja se činila samo vizionarskim pothvatom, postala je dostupna pojedincu. Cijena potrošačke elektronike, sve je više padala napretkom tehnologije te omogućila korištenje ovako sofisticirane tehnologije u domaćinstvu. Od robota za usisavanje do dronova za rekreaciju, mnogi ne mogu više ni zamisliti neke dijelove svog života bez mobilne robotike. Upravo zbog brzorastućeg trenda korištenja mobilnih robota u mnogim područjima važno je baviti se temom mobilne robotike i svim njenim aspektima s ciljem poboljšanja i daljnjeg unaprijeđenja ove širokoprimjenjive znanosti. Mobilna robotika danas je zastupljena u mnogim sektorima, sve od medicine i nuklearne tehnologije pa do oceanografije, uz brojna druga područja. Sustavi mobilne robotike sve se češće nalaze u industrijskim pogonima, s posebnim naglaskom na transport tereta s jedne na drugu lokaciju unutar industrijskog pogona.

Kako bi mobilni roboti mogli izvršavati svoje zadaće, koriste sustave lokomocije, upravljanja, senzorike i navigacije. Navedeni sustavi omogujućuju mobilnom robotu mapiranje određenog prostora, lokalizaciju u prostoru, a potom i autonomnu navigaciju po istom tom prostoru. Cijeli postupak mapiranja, lokalizacije i navigacije mobilnog robota objašnjen je kroz ovaj rad. Mobilna platforma korištena za postizanje tih ciljeva je *Robotino* 3, tvrtke $Festo$, za čije se upravljanje koristi robotski operativni sustav (*eng. Robot Operating System - ROS*). U radu su opisani pogonski, senzorski i upravljački sustavi mobilne platforme, te njezino dostupno sučelje za programiranje aplikacija(eng. Application Programming Interface - API). Objašnjeni su osnovni principi na kojima se ROS bazira, kao i svi osnovni paketi ROS-a koji su korišteni pri izradi rada.

Poglavlje 2.

Robotska platforma Robotino

Radi lakšeg razumijevanja, opisivanje robotske platforme podijeljeno je na hardver i softver. Počinje se od hardverskog sučelja u kojem se govori o lokomociji, senzorskom sustavu pomoću kojeg platforma prima podatke iz okoline te o samom upravljačkom sustavu koji je zadužen za upravljanje i odlučivanje. Kod softverskog sučelja se govori o načinima programiranja same robotske platforme.

Slika 2.1: Robotska platforma Robotino 3

2.1. Hardversko sučelje robotske platforme

Robotino je mobilni robot tvrtke *FESTO* stvoren u svrhe učenja i istraživanja [1]. Opremljen je s tri švedska kotača, međusobno razmaknuta za 120°, što samom robotu nudi mogućnost ravninskog kretanja u svim smjerovima, tj. ima tri stupnja slobode gibanja. Također sama platforma dolazi s 9 infracrvenih senzora i branikom osjetljivim na dodir. Što se tiče propriocepcijskih senzora, svaki od tri motora zaduženih za pogonjenje švedskih kotača sadrži inkrementalni enkoder koji služi mjerenju kuta zakreta motora. Pomoću enkodera se može odrediti pozicija, brzina i akceleracija mobilnog robota. Od upravljačkog sustava, Robotino posjeduje računalo s operativnim sustavom *Linux* i mikrokontroler koji je zadužen za kontrolu napona, upravljanje motora te upravljanjem digitalnim i analognim ulazima i izlazima. U nastavku je svaki od sustava objašnjen detaljnije.

2.1.1. Pogonski sustav

Po vrsti pogona, mobilni roboti s kotačima dijele se na one koji koriste diferencijalni i one koji koriste holonomski pogon. Robotino dolazi s holonomskim pogonom. Holonomski pogon omogućuje robotu kretanje u svim smjerovima i rotaciju na mjestu.

Pogonski sustav robotina se sastoji od tri motora, tri inkrementalna enkodera, tri reduktora i tri švedska kotača. Svaki od tri kotača je pogonjen zasebno pomoću svog motora. Motor radi na principu istosmjerne struje, nazivnog napona 40 V.

Tablica 2.1. prikazuje tehničke karakteristike istosmjernog motora.

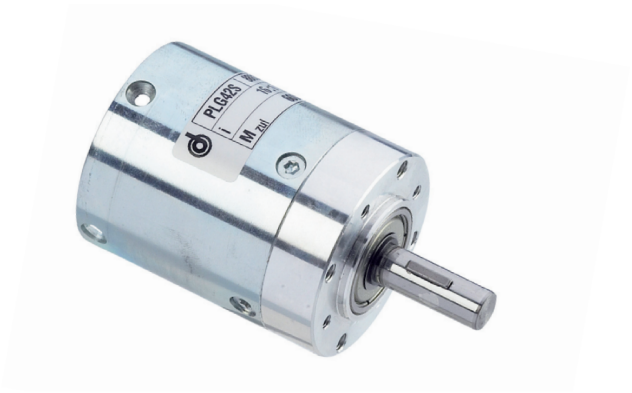
Tablica 2.1: Tehničke karakteristike motora GR 42 x 40 [2]

Nazivni napon	VDC	40
Kontinuirana nazivna brzina	rpm	3400
Kontinuirani nazivni moment	Ncm	5.7
Kontinuirana struja	A	0.8
Početni moment	Ncm	36
Početna struja	A	3.97
Brzina pri praznom hodu	rpm	3950
Struja pri praznom hodu	A	0.12
Struja demagnetizacije	A	6.3
Inercija rotora	gcm^2	110
Težina motora	g	490

Na samom motoru nalazi se inkrementalni enkoder. Na temelju vrijednosti dobivenih od enkodera, kontroler motora može njime upravljati. Između svakog motora i kotača nalazi se reduktor. Reduktor je planetarni s prijenosnim omjerom 32:1. Veliki prijenosni omjer omogućava visoku preciznost mobilnog sustava pri malim brzinama.

Tablica 2.2: Tehničke karakteristike reduktora PLG 42 S [3]

Prijenosni omjer	/	32
Učinkovitost	/	0.81
Broj faza	/	2
Kontinuirani nazivni moment	NCm	600
Težina reduktora	kg	0.37
Aksijalno opterićenje	N	150
Radijalno opterićenje	N	250



Slika 2.2: Planetarni reduktor PLG 42 S [3]

2.1.2. Senzorski sustav

Robotino 3 ima ugrađen branik osjetljiv na dodir i infracrvene senzore udaljenosti. U slučaju sudara, branik je zadužen za zaustavljanje motora te prekidanje programa koji je u izvršavanju. Branik je postavljen na donjem rubu kućišta.

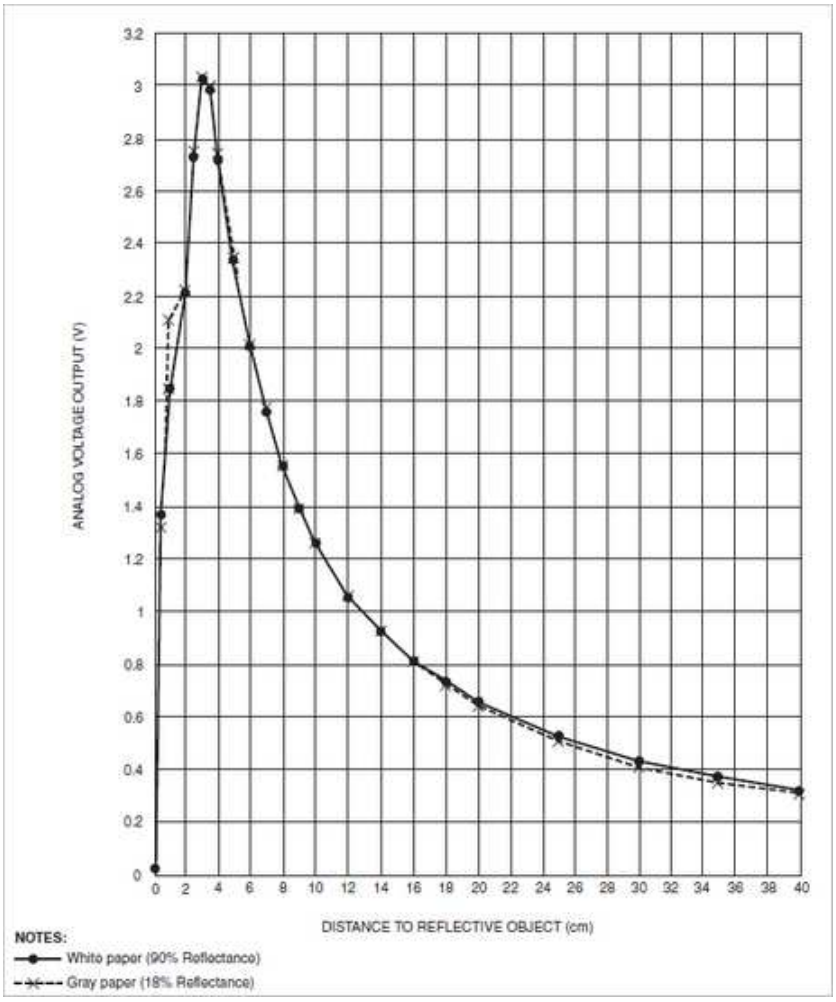
Slika 2.3: Pozicija branika osjetljivog na dodir

Infracrveni senzori omogućavaju određivanje udaljenosti stranih predmeta od robotske platforme. Robotino je opremljen s ukupno 9 infracrvenih senzora, smještenih cijelom cirkumferencijom njegovog kućišta. Senzori su međusobno razmaknuti za 40°. Svaki senzor udaljenosti očitava razinu napona čija vrijednost ovisi o udaljenosti do reflekti-

rajućeg objekta. Radni raspon očitanja udaljenosti im iznosi od 4 do 30 cm.

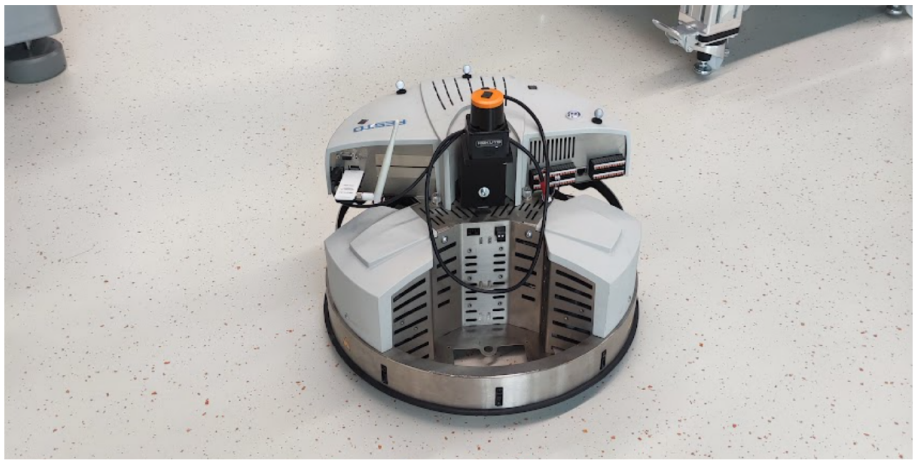
Slika 2.4: Pozicija infracrvenih senzora [4]

Slika 2.5 prikazuje karakteristiku infracrvenih senzora. Na apcisti se nalazi udaljenost od reflektirajućeg objekta, dok je na ordinati vrijednost izlaznog napona.



Slika 2.5: Karakteristika infracrvenih senzora [4]

Naknadno je postavljen lidar senzor zbog zahtjeva mapiranja i lokalizacije prostora. Lidar je vrsta senzora baziran na principu lasera, koji mjeri udaljenost [5]. Udaljenost se preračunava na temelju izmjerenog vremena koje je potrebno da se laserska zraka vrati do odašiljača.
Odabrani postavljeni senzor je senzor Hokuyo UST-10LX, 2D laserski skener. Senzor ima zakret od 270°. Montiran je na centralnom stupu kućišta, s prednje strane kako bi mogao skenirati prostor ispred sebe.



Slika 2.6: Hokuyo senzor montiran na Robotino-a

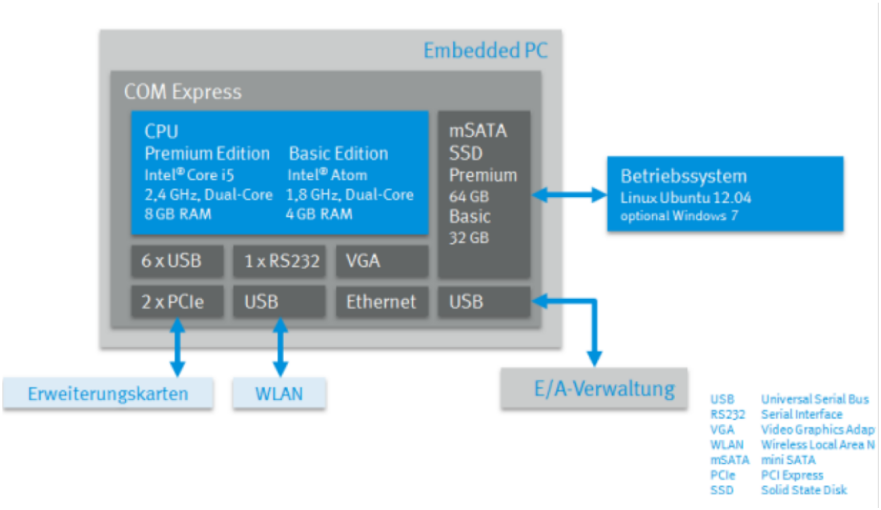
Tablica 2.3. prikazuje tehničke karakteristike odabranog lidar senzora.

Tablica 2.3: Tehničke karakteristike lidar senzora Hokuyo UST-10LX [6]

Napon napajanja	VDC	12/24
Struja napajanja	mA	150 ili manje
Valna duljina	nm	905
Minimalna udaljenost detektiranja	m	0.06
Maksimalna udaljenost detektiranja	m	30
Kut zakreta	°	270
Brzina skeniranja	ms	25
Kutna rezolucija	°	0.25

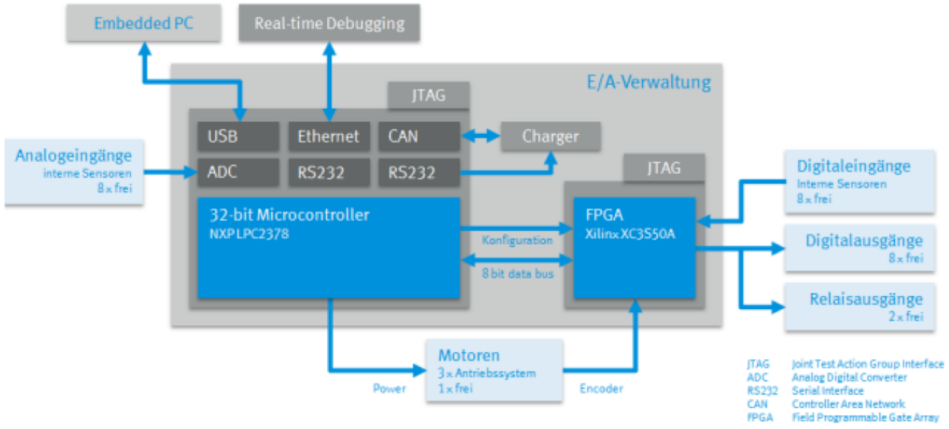
2.1.3. Upravljački sustav

Upravljački sustav robota se sastoji od 2 dijela, ugrađenog računala i mikrokontrolera. Operativni sustav ugrađenog računala je *Linux*, na kojem je osnova robotski operativni sustav (ROS). Za rad računala zadužen je procesor Intel® Core™ I5-520E 2.4GHz Dual Core s ugrađenom RAM memorijom od 8 gb [7]. Slika 2.7. prikazuje blok šemu ugrađenog računala.



Slika 2.7: Blok shema ugrađenog računala [7]

Mikrokontrolerska pločica je zadužena za upravljanje robotom na najnižoj razini. Upravljanje robota omogućuje 32-bitni mikrokontroler LPC2378 [8]. Njegova zadaća je kontrola napona u sustavu, kontrola motora te upravljanje digitalnih i analognih ulaza i izlaza. Računalo i mikrokontrolerska pločica povezani su putem USB protokola. Slika 2.8. prikazuje blok šemu ugrađenog mikrokontrolera.



Slika 2.8: Blok shema mikrokontrolera [9]

2.2. Softversko sučelje robotske platforme

Mobilnu platformu Robotinu moguće je programirati pomoću nekoliko programskih jezika kao što su C, C++, Java, .Net, Matlab, LabView i Microsoft Robotics Developer Studio [10]. Također, sama platforma nudi podršku SmartSoft-u i robotskom operativnom sustavu (ROS). U svrhu ovog rada, programira se u ROS-u. ROS paket *robotino_node* omogućuje komunikaciju između Robotino API2 i ROS-a. Više o navedenom paketu slijedi u 3. poglavlju.

2.2.1. Robotino REST-API

REST API je aplikacijsko programsko sučelje koje je u skladu s ograničenjima REST arhitektonskog stila i omogućuje interakciju s REST web uslugama [11].
API je skup definicija i protokola za izgradnju i integraciju aplikacijskog softvera. Odgovoran je za ostvarivanje komunikacije između dva računala ili sustava. Ako korisnik želi komunicirati s računalom kako bi dohvatio informacije ili izvršio funkciju, API pomaže priopćiti ono što korisnik želi reći drugom sustavu, kako bi on mogao razumijeti i ispuniti zahtjev [11].
Robotino web sučelje se temelji na REST-ful poslužitelju koji sluša na portu 80. Poslužitelj nudi podatke na Robotino-vom web sučelju te se može koristiti za pristup njegovim senzorima i aktuatorima [12].

Koristeći metodu GET s Robotina se mogu zatražiti sljedeći podaci:

- /cam0 - za dobivanje slike s kamere
- /sensorimage - za dobivanje slike senzora udaljenosti i senzora osjetljivog na dodir
- /data/festoolcharger - podaci s Festool Li-Ion baterija
- /data/powermanagment - nudi informacije o naponu i struji sustava
- /data/charget0 - podaci za ugrađeni punjač 0
- /data/charget0 - podaci za ugrađeni punjač 1
- /data/services - podaci o specifičnim servisima Robotino-a
- /data/servicestatus/xxx - daje status od servisa xxx
- /data/analoginputarray - za dobivanje podataka svih analognih ulaza u obliku vektora
- /datadigitalinputarray - za dobivanje podataka svih digitalnih ulaza u obliku vektora; vrijednosti su istinite ili ne istinite
- /data/digitaloutputstatus - daje status svih digitalnih izlaza; vrijednosti su istinite ili ne istinite
- /data/relaystatus - nudi informacije o statusu svih releja
- /data/bumper - daje stanje senzora osjetljivog na dodir
- /data/distancesensorarray - očitanja sa svih senzora udaljenosti; podaci su grupirani u vektoru
- /data/odometry - podaci o odometriji Robotino-a
- /data/imageversion - za zatraživanje verzije Robotino operativnog sustava

Kao što se podaci mogu zatražiti s Robotino-a, isto tako se mogu poslati na njegov server. Pomoću metode POST ili PUT moguće je objavljivati na sljedeće adrese:

- /data/omnidrive - omogućuje postavljanje linearne i rotacijske brzine

- /data/digitaloutput - za postavljanje digitalnih izlaza; brojem se definira koji će se izlaz postaviti istinito ili lažno
- /data/digitaloutputarray - za grupno postavljanje digitalnih izlaza u obliku vektora; prvi član vektora označava izlaz 0, dok zadnji član vektora označava izlaz 7
- /data/relay - omogućuje postavljanje stanja releja.
- /data/relayarray - za grupno postavljanje stanja releja u obliku vektora

Za detaljnije informacije o podacima koje Robotino API nudi upućuje se čitatelja na [12].

Poglavlje 3.

Robotski operativni sustav

Robotski operativni sustav (skraćeno ROS) nije u suštini operativni sustav, već softversko okruženje koje radi unutar *Ubuntu Linux*-a [13]. Kao takav, sadrži mnoge alate i biblioteke za robotske sustave. Iako ROS nije pravi operativni sustav, pruža usluge koje bi korisnik očekivao od operativnog sustava kao što su: apstrakcija hardvera, kontrola uređaja niske razine, imeplementacija često korištenih funkcija, prijenos poruka između procesa i upravljanje paketima [14]. Također, nudi alate i biblioteke za izgradnju, pisanje i pokretanje koda na više računala [14].

ROS koristi arhitekturu grafova što je zapravo *peer-to-peer* mreža ROS procesa koji zajedno obrađuju podatke [15]. Osnovne koncepte u toj arhitekturi predstavljaju čvorovi (*eng. nodes*), *Master*, server parametara, servisi (*eng. service*), teme (*eng.* topics) i torbe (*eng. bags*). Svaki od njih pruža podatke na jedinstven način.

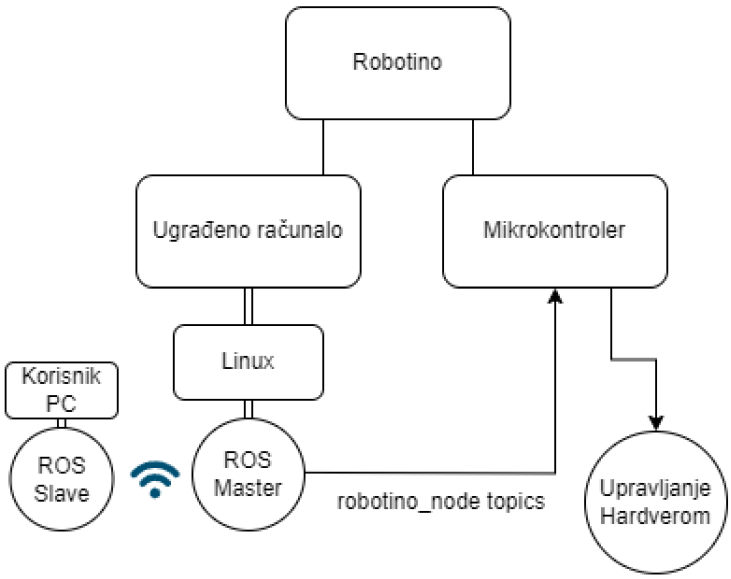
Da bi sve navedeno moglo funkcionirati kao jedna cjelina i međusobno komunicirati potreban je *Master*. *Master* je zaslužan za međusobne interakcije između čvorova te prati tko objavljuje na teme i tko se na njih pretplaćuje.

Čvorovi su izvršne datoteke koje koriste ROS za komunikaciju s drugim čvorovima. U suštini to su dijelovi koda koji je zaslužan za neku izvršnu operaciju te može objavljivati podatke na temu ili dobivati podatke s teme tako da se na nju pretplati. Zaslužni su za procesiranje podataka sa senzora, aktuatora, kontrolera i drugih sustava unutar robota koji komuniciraju preko ROS-a.

Teme su imenovane sabirnice preko kojih čvorovi razmjenjuju poruke*(eng. messages)*. Svaka tema ima jasno definirani tip poruke koji može primati i slati. Tijekom razmjenjivanja poruka preko tema, čvorovi ne znaju s kim su u interakciji. Umjesto

tema, čvorovi mogu komunicirati preko servisa. Servisi su funkcije koje čvorovi pozivaju da bi se izvršile nad nekim drugim čvorom.

Za spremanje i pristupanje konfiguracijskim parametrima čvorovi mogu koristiti server unutar ROS-a zvan *parameter server* [16]. Funkcionira na principu rječnika *(eng. dictionary)* koji sadrži globalne varijable koje su dostupne s bilo kojeg mjesta u trenutnom ROS okruženju. Za detaljniji opis navedenih koncepti čitatelja se upućuje na [15]. Slika 3.1. prikazuje na koji način se ostvaruje komunikacija između robota i korisnika te kako ROS ima utjecaj na sam hardver robota.



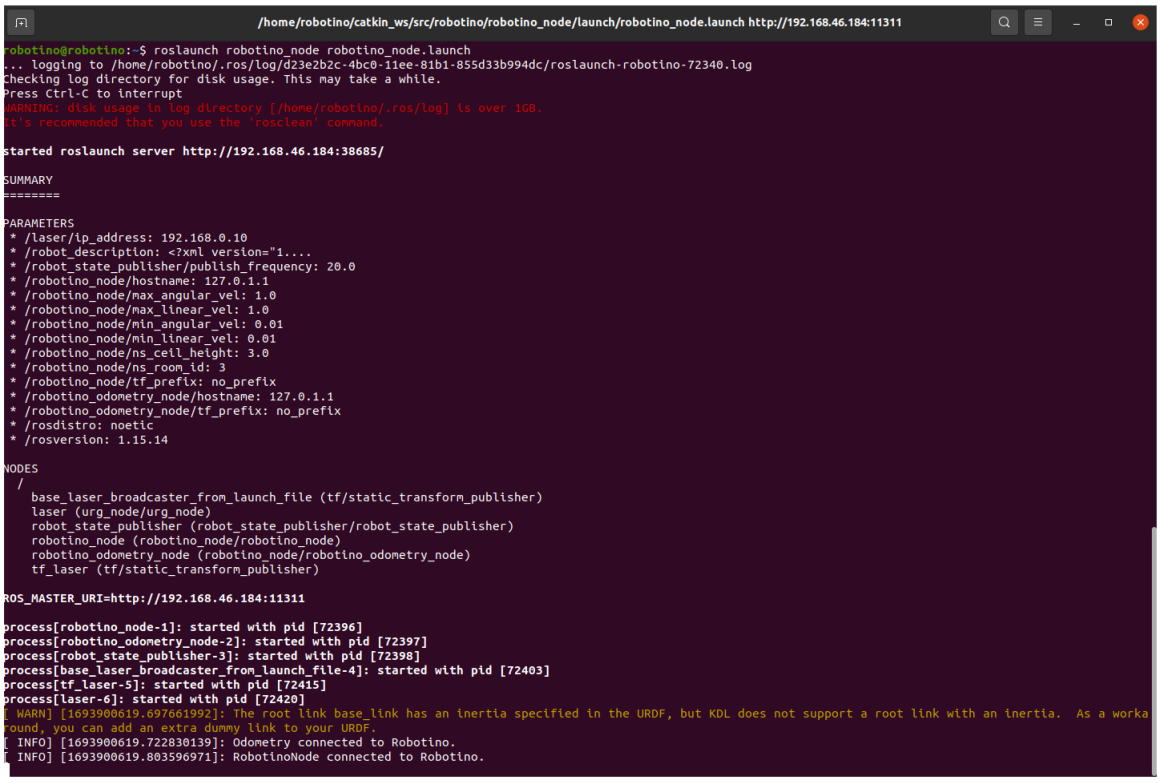
Slika 3.1: Blok shema ostvarene komunikacije putem ROS-a

3.1. robotino_node

Robotino_node je paket u kojem se nalazi čvor robotino_node koji implementira funkcije koje definiraju sučelje Robotino API2 [17]. Također sam čvor objavljuje podatke s Robotina na različite teme i implementira nekoliko servisa. Za pouzdani rad pretplaćuje se na nekoliko tema kako bi dobio tražene vrijednosti.
Cijeli paket se pokreće pomoću dane *launch* datoteke korištenjem naredbe:

```
$ roslaunch robotino_node robotino_node.launch
```

S obzirom da pokrećemo preko *launch* datoteke, nema potrebe za pokretanjem zasebnog *Master* čvora. Njega će automatski pokrenuti *launch* datoteka kako je prikazano na slici 3.2.



Slika 3.2: Pokretanje *robotino_node launch* datoteke

Nakon pokretanja, *robotino_node* se pretplaćuje na slijedeće teme:

- cmd_vel - pomoću ove teme Robotino zna brzine koje mora postići; brzine zadaje sam korisnik ili neki drugi čvor; jedan od načina zadavanja brzina je putem čvora *teleop_twist_keyboard*; prilikom navigacije čvor *move_base* objavljivat će na temu *cmd_vel* izračunate brzine koje robot mora postići kako bi došao u određenu poziciju.

Teme na koje čvor robotino_node objavljuje su:

- *analog_readings* - tema na koju se objavljuju vrijednosti analognog ulaza

- *bumper* - objavljuje stanje senzora branika osjetljivog na dodir; vrijednost *true* označava da je došlo do sudara, dok vrijednost *false* označava da robot nije u kontaktu s okolinom
- *digital_readings* - objavljuje vrijednosti digitalnog ulaza
- *distance_sensors* - objavljuje vrijednosti 9 infracrvenih senzora u obliku *point cloud*-a.
- *encoder_readings* - objavljuje vrijednosti svakog enkodera zasebno
- *motor_readings* - objavljuje vrijednosti svakog motora zasebno
- *odom* - sadrži podatke o odometriji robota.

Parametri koji se učitaju pokretanjem čvora robotino_node su:

- *hostname* - parametar koji definira Robotino IP adresu
- *max_linear_vel* - parametar koji definira maksimalnu linearnu brzinu u m/s. Zadana vrijednost iznosi 0,2
- *min_linear_vel* - definira minimalnu linearnu brzinu u m/s; zadana vrijednost iznosi 0,05
- *max_angular_vel* - definira maksimalnu kutnu brzinu u rad/s; zadana vrijednost iznosi 1,0
- *min_angular_vel* - definira minimalnu kutnu brzinu u rad/s; zadana vrijednost iznosi 0,1.

U sklopu čvora dostupni su servisi: *set_encoder_position* i *reset_odometry* [17].

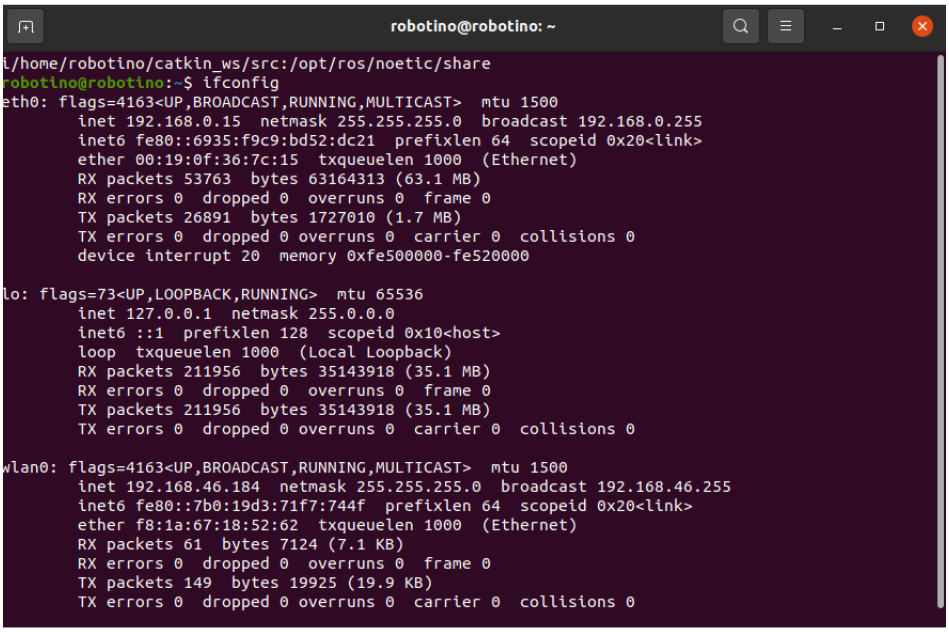
3.2. Definiranje Master/Slave odnosa

Jedna od značajki ROS-a je ostvarivanje *master/slave* komunikacije. Kada se dva ili više računala s ROS-om nalaze na istoj mreži, moguće je izvršavati naredbe pomoću jednog *Master* čvora. S obzirom da *robotino_node.launch* pokreće *Master*, moguće je povezati drugo računalo, koje se nalazi na istoj mreži, na njega.

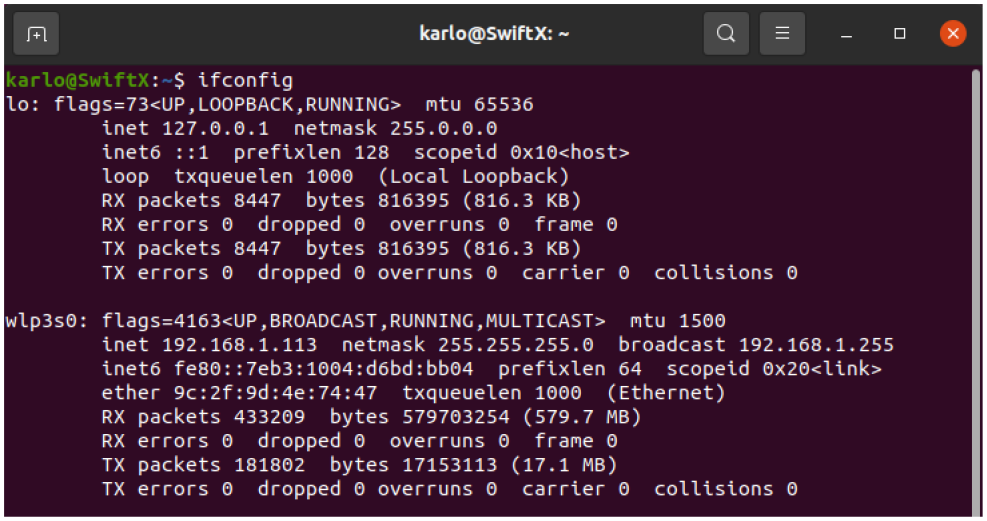
Kako bi se ostvarila komunikacija, potrebno je poznavati IP adresu robota i osobnog računala. IP adresa oba računala se može saznati upisivanjem sljedeće naredbe u terminal:

```
$ ifconfig
```

Slika 3.3 i 3.4 prikazuju tražene IP adrese robota i osobnog računala.



Slika 3.3: IP adresa Robotina



Slika 3.4: IP adresa osobnog računala

3.2.1. Master

Definiranje *master* računala ostvaruje se uređivanjem *.bashrc* datoteke na željenom računalu. U ovom slučaju *master* računalo će biti Robotino. *Bashrc* datoteka otvara se upisivanjem sljedeće naredbe u terminalu *master* računala:

```
$ gedit .bashrc
```

Definiranje *master* računala ostvaruje se upisivanjem sljedećih naredbi u *.bashrc* datoteku:

```
export ROS_MASTER_URI=http://192.168.46.184:11311/
export ROS_IP=192.168.46.184
```

gdje je 192.168.46.184. ip adresa robota.

3.2.2. Slave

Kao podređeno računalo (*eng. slave*) definira se osobno računalo. Korištenjem iste naredbe u terminalu otvara se *.bashrc* datoteka:

```
$ gedit .bashrc
```

Definiranje podređenog računala ostvaruje se upisivanjem sljedećih naredbi:

```
export ROS_MASTER_URI=http://192.168.46.184:11311/
export ROS_IP=192.168.1.113
```

gdje je 192.168.46.184 IP adresa Robotina dok je 192.168.1.113 IP adresa osobnog računala.

3.2.3. Testiranje komunikacije

Da bi se testirala komunikacija, potrebno je pokrenuti *Master* pomoću *robotino_node.launch* datoteke.
Ako se uspješno uspostavila komunikacija, na *slave* računalu bi se trebale vidjeti teme koje je *robotino_node* uspješno objavio. Teme koje čvor objavljuje trebale bi se vidjeti upisivanjem sljedeće naredbe u terminalu *slave* računala:

```
$ rostopic list
```

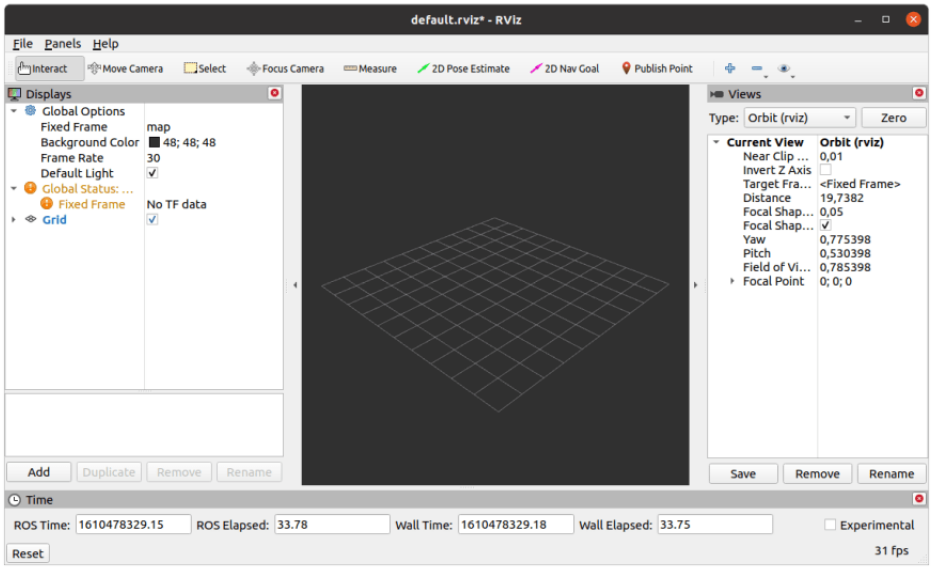
3.3. Paketi za vizualizaciju i testiranje mobilnog robota

3.3.1. rviz

Rviz je čvor koji služi za vizualizaciju robota, senzora i algoritama te podataka koji se dobivaju od njih [18]. Pomoću *rviz* čvora ovdje su vizualizirani procesi mapiranja, lokalizacije i navigacije. *Rviz* omogućuje korisniku da učita spremljenu mapu prostora, u istoj da izvrši lokalizaciju robota te da mu zada pozicija za navigaciju. Grafičko sučelje paketa *rviz* pokrećemo naredbom:

```
$ rosrun rviz
```

Slika 3.5. prikazuje izgled sučelja rviz.



Slika 3.5: Izgled sučelja *rviz*

S lijeve strane sučelja dodaju se različiti segmenti koje korisnik želi vizualizirati. U svrhe mapiranja, lokalizacije i navigacije koristit će se sljedeći segmenti [19]:

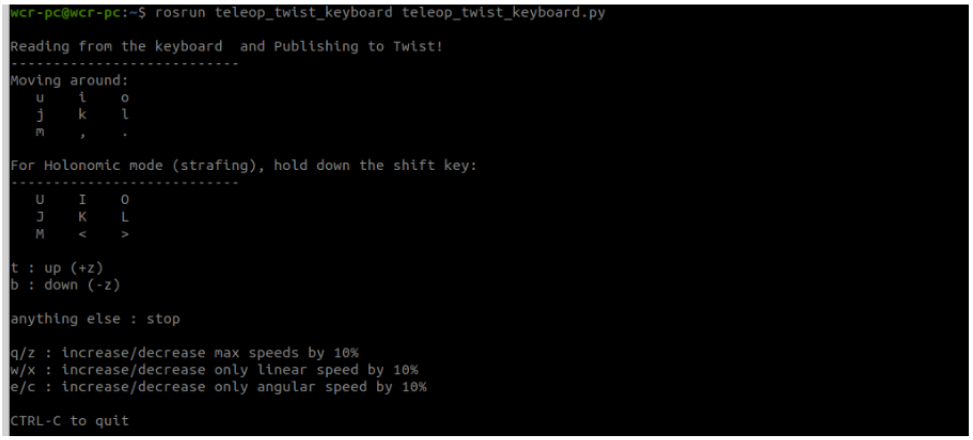
- Map - služi za vizualizaciju već snimljene mape ili za prikaz mape tokom procesa mapiranja
- PoseArray - vizualizira oblak strelica, koji se generira prikazom poruka geometry_msgs::PoseStamped služi za proces lokalizacije, tokom kojeg dolazi do konvergiranja i smanjivanja spomenutog oblaka
- GlobalPlan - prikazuje u realnom vremenu trajektoriju globalnog planera
- Polygon - služi za vizualizaciju gabaritnih mjera robota
- LaserScan - pretplaćuje se na temu *scan* te vizualizira podatke dobivene s lidar senzora
- Local Costmap - prikazuje lokalnu costmap-u
- Global Costmap - prikazuje globalnu costmap-u.

3.3.2. teleop_twist_keyboard

Teleop_twist_keyboard je čvor koji služi za upravljanje mobilnog robota pomoću tastature [20]. Sljedećom naredbom pokrećemo spomenuti čvor:

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

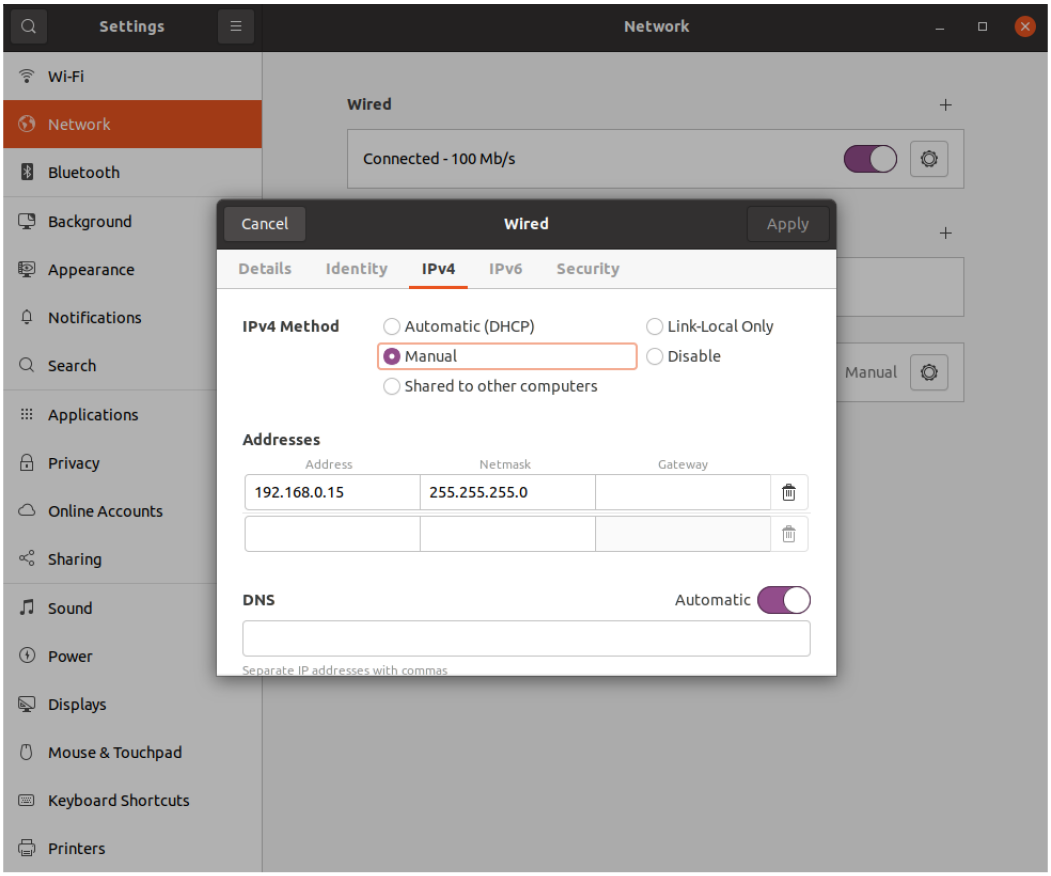
Pokretanjem naredbe, u terminalu se prikazuju instrukcije za korištenje navedenog čvora, kao na slici 3.6.



Slika 3.6: Pokrenut *teleop_twist_keyboard* čvor

3.4. Implementacija lidara Hokuyo UST-10LX

Kako bi ROS mogao primati vrijednosti koje senzor šalje, potrebno je definirati parametar adrese lidar senzora. Na *master* računalu, unutar mrežnih postavki, definira se statična IP adresa lidar senzora. Senzor dolazi s vlastitom predefiniranom IP adresom koja glasi: 192.168.0.10 [6]. Na temelju nje se odabire statična IP adresa koja glasi: 192.168.0.15/24. Navedena notacija IP adrese znači da prva 24 bita definira mrežni identifikator, a sve što dođe iza toga definira računalni identifikator [21]. Postavljanje mrežnih postavki kao sa slike 3.7 definira statičnu IP adresu.



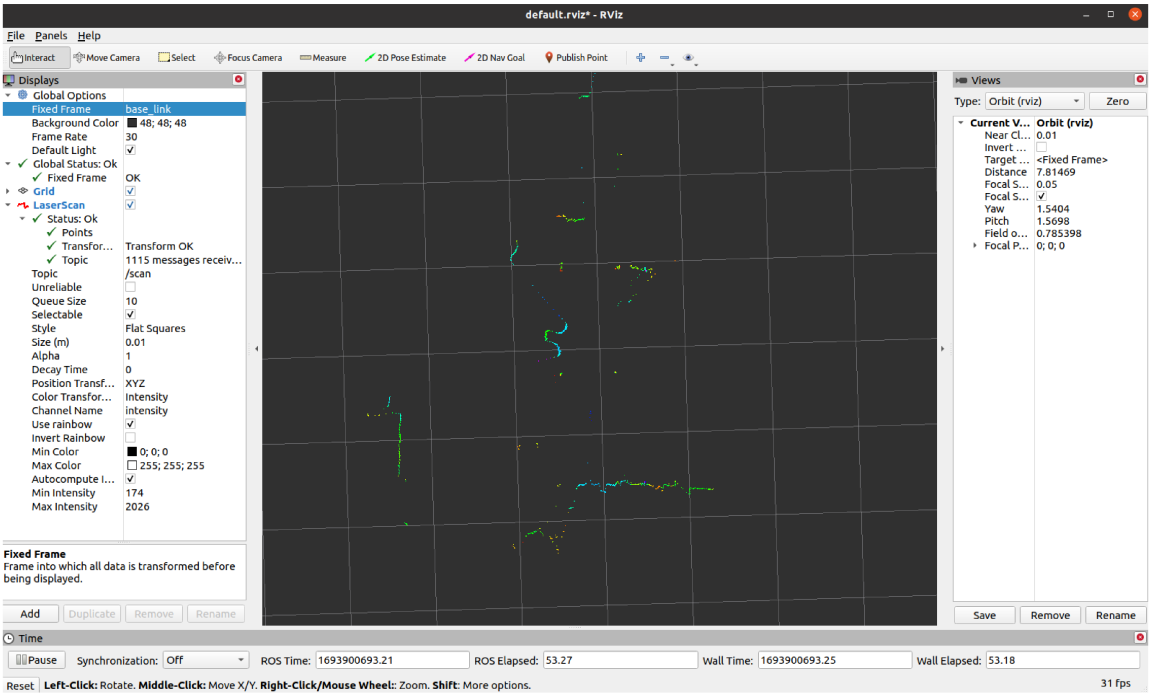
Slika 3.7: Postavljanje statične IP adrese

3.4.1. urg_node

Urg_node je paket razvijen za korištenje lidar senzora unutar ROS-a [22]. Sadrži istoimeni čvor koji omogućuje čitanje podataka s lidar senzora. Prilikom pokretanja navedenog čvora, potrebno je definirati IP adresu lidar senzora. To se izvršava sljedećom naredbom:

```
$ rosrun urg_node urg_node _ip_address:=192.168.0.10
```

Nakon pokretanja čvora *urg_node*, on počinje objavljivati na temu *scan*. Poruke koje objavljuje na tu temu vezane su za udaljenosti predmeta i kut zakreta. Tu temu koristit će drugi čvorovi prilikom mapiranja, lokalizacije i navigacije.
Odabirom segmenta *scan* unutar *rviz* sučelja, dobije se vizualizacija podataka s lidar senzora. Slika 3.8. prikazuje očitanja s lidar senzora u *rviz* sučelju.



Slika 3.8: Vizualizirana očitanja lidar senzora u alatu *rviz*

Poglavlje 4.

Navigacijski paket

Navigation stack je skup paketa koji se koriste za mapiranje prostora te lokalizaciju i navigaciju u prostoru [23].Paketi korišteni u ovom radu su: *gmapping* za mapiranje, *amcl* za lokalizaciju i *move_base* za navigaciju. Paketi iz *navigation stack-a* koriste informacije o odometriji i vrijednosti sa senzora kako bi robotu slali brzine koje treba postići. Zbog toga je jedan od zahtjeva za korištenje *navigation stack-a* taj da robotska platforma prima naredbe u obliku brzina kako bi se kretala po prostoru [23]. Navedeni paketi implementirani su pomoću podređenog računala.

Poglavlje 5.

Mapiranje prostora CRTA-e

5.1. Paket gmapping

Gmapping je paket unutar *navigation stack-a* koji sadrži *slam_gmapping* čvor [24]. Pojam gmapping označava algoritam za mapiranje prostora. Taj algoritam koristi Rao-Blackweillzed filtar čestica kako bi izgradio mapu na temelju podataka s laserskog senzora [25].
Prije korištenja navedenog čvora, moraju se osigurati sljedeće transformacije:

- *robot_state_publisher* → *base_link*
- *base_link* → *odom*.

Pokretanje Gmapping čvora izvršava se u terminalu sljedećom naredbom:

```
$ rosrun gmapping slam_gmapping scan:=scan
```

Prilikom pokretanja, čvor se pretplaćuje na sljedeće teme:

- *tf* - tema koja sadrži nužne informacije o koordinatnim sustavima lasera, baze robota i odometrije
- *scan* - tema na koju se objavljuju podaci od lidar senzora

Također, isti čvor objavljuje sljedeće teme:

- *map_metadata* - tema na koju se objavljuju podaci vezani uz novonastalu mapu; podaci su ažurirani periodički.

- *map* - sadrži podatke koji su odgovorni za vizualizaciju 2D mape. Podaci su ažurirani periodički
- *entropy* - tema koja sadrži procjenu entropijske raspodjele s obzirom na poziciju robota.

Paket Gmapping nudi jedan servis: *dynamic_map*. Pozivanjem ovog servisa dobivaju se podaci o trenutnoj mapi.

5.2. Definiranje *launch* datoteke

Slam_gmapping čvor se može pozvati izvršavanjem prethodno navedene naredbe. Pozivanjem čvora svi ponuđeni parametri bit će postavljeni na temelju predefiniranih vrijednosti. Kako bi se povećala kvaliteta mapiranja, korisno je konfigurirati ponuđene parametre. Neki od parametara koje čvor nudi za postavljanje su:

- *odom_frame* - koordinatni sustav odometrije robota
- *base_frame* - kooridnatni sustav mobilne baze
- *map_frame* - koordinatni sustav mape
- *map_update_interval* - vrijeme (u sekundama) između svakog ažuriranja mape. Smanjivanjeg ovog broja mapa se ažurira češće, što zahtjeva veću procesnu snagu
- *maxUrange* - maksimalna iskoristivost dosega laserskih zraka; iz prakse se preporučuje da se uzme malo manji broj od nominalne maksimalne udaljenosti koju laser može očitati
- *maxRange* - maksimalni doseg senzora; preporuča se da se uzme broj veći ili jednak stvarnom maksimalnom dosegu senzora
- *particles* - broj čestica koji se uzima u filter u svrhe mapiranja
- *linearUpdate* - parametar koji određuje koliko se robot mora linearno pomaknuti da bi algoritam procesirao podatke
- *angularUpdate* - parametar koji određuje koliko se robot mora kutno zakrenuti da bi algoritam procesirao podatke

- *temporalUpdate* - izvrši skeniranje ako je zadnje skeniranje starije od vremena ažuriranja u sekundama
- *delta* - parametar koji određuje rezoluciju mape
- *ogain* - pojačanje koje se koristi prilikom zaglađivanja senzorskih podataka.

Ostali parametri koji se očitaju prilikom pozivanja *slam_gmapping* čvora: *throttle_scans, sigma, kernelSize, lstep, astep, iterations, lsigma, lskip, minimumScore, srr, srt, str, stt, resampleThreshold, xmin, ymin, xmax, ymax, llsamplerange, llsamplestep, lasamplerange lasamplestep, transform_publish _period, occ_thresh* [24].

Unutar *launch* datoteke pokrećemo *slam_gmapping* čvor, učitavamo njegove parametre te pokrećemo *rviz*.

```
  <?xml version="1.0"?>
<launch>

<!-- POKRENI MAPIRANJE -->
 <arg name="scan_topic" default="/scan" />
 <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping">
  <rosparam>
   odom_frame: odom
   base_frame: base_link
   map_frame: map
   map_update_interval: 0.5
   maxUrange: 9.5
   maxRange: 10.5
   particles: 100

   # Frekvencija ažuriranja
   linearUpdate: 0.1
   angularUpdate: 0.1
   temporalUpdate: 2.0
   resampleThreshold: 0.5
```

```
    #Početna veličina mape
    xmin: -10.0
    ymin: -10.0
    xmax: 10.0
    ymax: 10.0
    delta: 0.05
    ogain: 3.0

  </rosparam>
 </node>

<!-- POKRENI RVIZ -->
<node pkg="rviz" type="rviz" name="rviz">
</node>
</launch>
```

Na početku *launch* datoteke definira se da se radi o XML prezentacijskom jeziku. Naredba $< launch >$ označuje početak *.launch* datoteke. Pokreće se *slam_gmapping* čvor sa svojim parametrima koji su definirani od strane korisnika. Ostali parametri koji nisu definirani unutar *.launch* datoteke, učitavaju se s predefiniranim vrijednostima.
Drugi čvor koji se pokreće je *rviz*. Pomoću *rviz* grafičkog sučelja prati se pokrivenost područja tokom mapiranja.
Launch datoteka završava naredbom $< /launch >$.

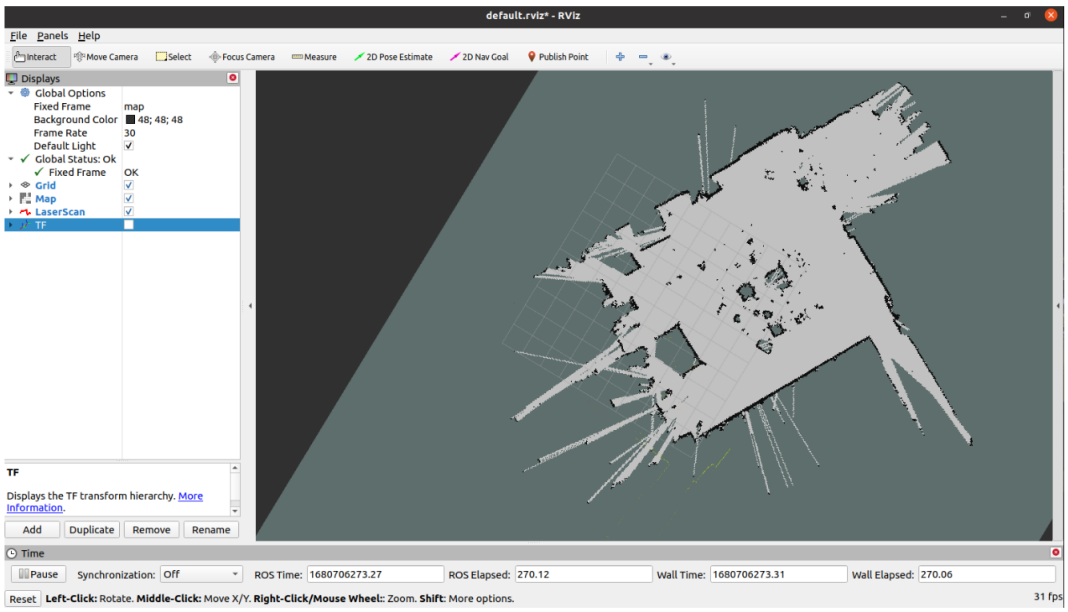
5.3. Proces mapiranja prostora

Pokretanjem *.launch* datoteke započinje proces mapiranja prostora. To se izvršava sljedećom naredbom:

```
$ roslaunch robotino mapping.launch
```

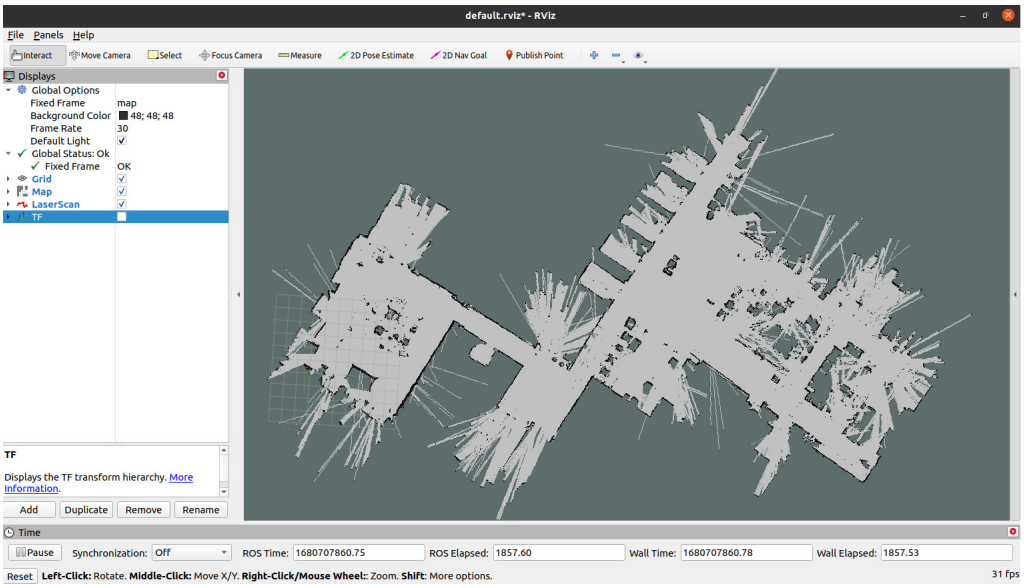
gdje *robotino* označava ime paketa stvorenog od strane korisnika. U njemu su spremljene sve *.launch* datoteke koji se koriste u ovom radu te se mogu pronaći u prilogu.
Naknadno se pokreće čvor *teleop_twist_keyboard*, kako je opisano u poglavlju 3.3.2. Kako

bi proces mapiranja bio što uspješniji, potrebno je proći kroz što više dijelova prostora koji se mapira. Sa slike 5.1. vidljivo je kako na nekim mjestima dolazi do propuštanja laserskih zraka lidar senzora.



Slika 5.1: Početak procesa mapiranja

Do toga dolazi jer su ti zidovi staklene površine. To se može naknadno obraditi u nekom od alata za grafičko uređivanje slike. Moguće je obrisati laserske zrake koje su prošle preko linije zida te povući crnu liniju koja bi označavala kraj tog zida. Nakon što je korisnik zadovoljan dobivenom mapom, koja je vidljiva u *rviz-u*, proces mapiranja može završiti.



Slika 5.2: Zadovoljavajuća pokrivenost mapiranog prostora

Kako bi se novonastala mapa kasnije mogla koristiti u svrhe lokalizacije i navigacije, potrebno ju je spremiti na računalo. Spremanje mape se izvršava pozivanjem čvora pod nazivom *map_saver*. Isti pripada paketu *map_server* [26]. Prije njegovog pozivanja, potrebno je u terminalu doći do željenog direktorija gdje će se mapa spremiti [27]. Do direktorija se dolazi pomoću naredbe:

```
$ cd catkin_ws/src/robotino/maps
```

Spremanje mape u navedenom direktoriju izvršava se sljedećom naredbom:

```
$ rosrun map_server map_saver -f CRTA
```

Spremanjem mape prostora stvaraju se dvije datoteke [26]. Jedna datoteka ima nastavak *.pgm* i predstavlja izgled mape u pikselima. Boja piksela određuje okupiranost prostora. Bijeli pikseli označavaju slobodan prostor, dok crni pikseli označavaju njegovu zauzetost. Druga datoteka završava nastavkom *.yaml* i ona sadrži meta podatke mape [26]. Podaci koji su sadržani u *.yaml* datoteci su:

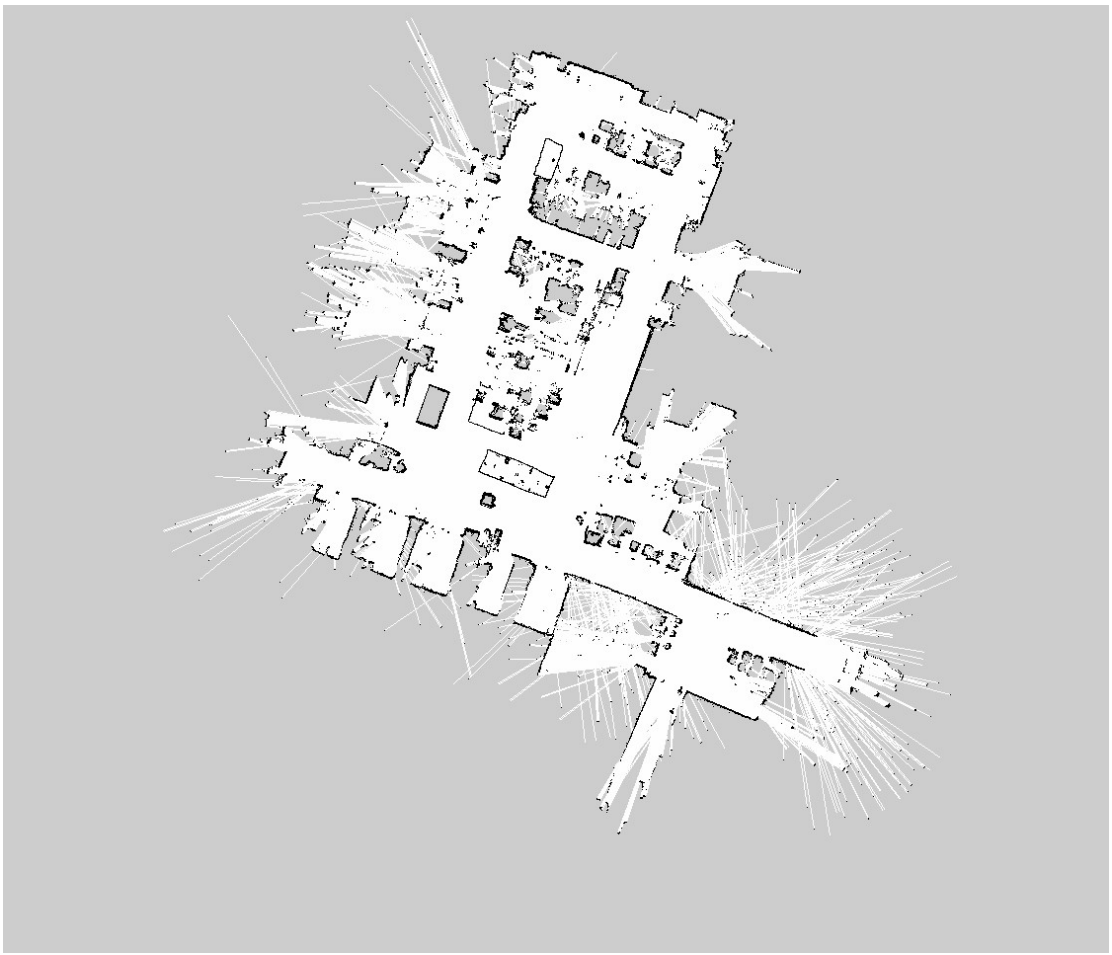
- *image* - putanja do *.pgm* datoteke, može biti apsolutna ili relativna
- *resolution* - rezolucija mape prikazana u metrima po pikselu

- *origin* - 2D pozicija ishodišta koordinatnog sustava mape
- *occupied_thresh* - pikseli s vjerojatnošću okupiranja većom od ovog parametra smatraju se kao okupirani (prikazani su crnom pikselom)
- *free_thresh* - pikseli s vjerojatnošću okupiranja većom od ovog parametra smatraju se slobodnim pikselima (prikazani su bijelim pikselom)
- *negate* - parametar koji služi za zamjenu semantike bijelo/crnih piksela i slobodnog/okupiranog prostora
- *mode* - proizvoljni parametar; nije korišten u ovom radu.

Meta podaci koji se nalaze unutar datoteke *CRTA.yaml*:

```
image:    CRTA.pgm
resolution:    0.05
origin:    [-10.0, -45.0, 0.0]
negate:    0
occupied_thresh:    0.65
free_thresh:    0.196
```

Slika 5.3. prikazuje izgled *CRTA.pgm* datoteke:



Slika 5.3: Snimljena mapa prostora CRTA

Poglavlje 6.

Lokalizacija robota u prostoru

Učitavanjem snimljene mape javlja se problem robotskog nepoznavanja vlastite lokacije na toj mapi. Taj problem se rješava lokalizacijom.
Lokalizacijom robot može pronaći sebe u učitanoj mapi, korištenjem lokalizacijskih algoritama i svojih eksteroceptivnih senzora. Ona se izvršava na način da se robota smjesti bilo gdje u prostoru te da mu se učita već snimljena mapa tog istog prostora. Na mapi se odredi približna lokacija gdje se robot nalazi. Vozeći robota po prostoru, on uspijeva odrediti svoju poziciju pomoću stacionarnih elemenata koji se nalaze na mapi i u prostoru. Za potrebe ovog rada, u tu svrhu se koristi *amcl* paket unutar ROS-a [28]. Navedeni paket sadrži istoimeni čvor koji je zadužen za proces lokalizacije.

6.1. *amcl* paket

Amcl je probabilistički sustav lokalizacije za robote koji se kreću u ravnini [28]. Implementira adaptivni Monte Carlo algoritam za lokalizaciju koji koristi filter čestica za praćenje robota u odnosu na poznatu mapu [38]. Algoritam izračunava distribuciju vjerojatnosti za sve moguće pozicije u okruženju [39]. Ako robot ne zna svoju početnu poziciju algoritam, radi ravnomjernu distribuciju mogućih pozicija po prostoru. Kako se robot kreće po prostoru, tako se ponovno raspodjeljuju čestice mogućih pozicija, s obzirom na informacije koje prima sa senzora. Uzorkovanje čestica se temelji na Bayersovoj estimaciji [29]. Uspješnom lokalizacijom dolazi do konvergiranja čestica prema stvarnom položaju robota.

6.2. Definiranje *launch* datoteke za lokalizaciju

Korištenjem *amcl* paket-a implementira se istoimeni čvor koji koristi algoritam za lokalizaciju. *Amcl* tijekom svog rada koristi mapu kreiranu pomoću laserskog senzora, vrijednosti koje dobavlja laser i poruke o tranformacijama. Kao rezultat daje na izlazu procjenu pozicije robota [28].
Pokretanje čvora se izvršava upisivanjem sljedeće naredbe u terminal:

```
$ rosrun amcl amcl scan:=scan
```

Nakon pokretanja isti se pretplaćuje na sljedeće teme:

- *scan* - tema na koju dolaze informacije s lidar senzora
- *tf* - tema koja sadrži potrebne transformacije
- *initalpose* - tema na koju dolaze informacije o početnoj poziciji
- *map* - sadrži mapu skeniranog prostora.

Izvršavanjem navedenog algoritma, AMCL svoje rezultate objavljuje na sljedeće teme:

- *amcl_pose* - tema koja sadrži estimaciju pozicije robota u prostoru
- *particlecloud* - skup estimiranih pozicija koje održava filter
- *tf* - objavljuju se informacije o transformacijama između *odom* koordinatnog sustava i koordinatnog sustava *map*.

Također čvor nudi nekoliko servisa koje je moguće pozvati:

- *global_localization* - započinje proces globalne lokalizacije, gdje su sve čestice nasumično generirane
- *request_nomotion_update* - servis za ručno izvršavanje ažuriranja i objavljivanja ažuriranih čestica
- *set_map* - servis za ručno postavljanje nove mape i pozicije
- *static_map* - servis koji se samostalno poziva od strane amcl-a kako bi se učitala mapa koja se koristi za lokalizaciju.

Kao što je bio slučaj s definiranjem parametara *gmapping* čvora, isto je potrebno napraviti s parametrima *amcl* čvora. Parametri koje *amcl* čvor nudi dijele se na tri grupe. Tri grupe parametara pokrivaju parametre vezane za filter čestica, parametre vezane za laser i parametre vezane uz odometriju samog robota. U ovom radu su objašnjeni parametri koji će se koristiti u *.launch* datoteci. Čitatelja se upućuje na [28] za dodatno objašnjenje svih dostupnih parametara.
Parametri vezani uz filter čestica koji se koriste u ovom radu su:

- *min_particles* - minimalni dozvoljeni broj čestica koje sadrže informacije o poziciji i orijentaciji robota
- *max_particles* - maksimalni dozvoljeni broj čestica koje sadrže informacije o poziciji i orijentaciji robota
- *kld_err* - maksimalno dozvoljeno odstupanje između prave distribucije i procjenjene distribucije
- *kld_z* - gornji standardni normalni kvantil za *(1-p)*, gdje je p vjerojatnost da će pogreška na estimiranom poremećaju biti manja od *kld_err*
- *update_min_d* - potrebni translacijski pomak u metrima prije ažuriranja filtera
- *update_min_a* - potrebni rotacijski pomak u radianima prije ažuriranja filtera
- *resample_interval_* - potrebni broj ažuriranja filtra prije ponovnog uzorkovanja
- *transform_tolerance* - vrijeme za koje je potrebno odgoditi transformaciju, kako bi se pokazalo da je transformacija valjana u budućnosti
- *recovery_alpha_slow* - eksponencijalna stopa propadanja za spori filter s prosječnom težinom, koji se koristi pri odlučivanju kada će se filter popraviti/obnoviti s dodavanjem nasumičnih pozicija
- *recovery_alpha_fast* - eksponencijalna stopa propadanja za brzi filter s prosječnom težinom, koji se koristi pri odlučivanju kada će se filter popraviti/obnoviti s dodavanjem nasumičnih pozicija
- *initial_pose_x* - početna pozicija u smjeru osi X, koristi se za inicilizaciju filtera sa Gaussovom distribucijom

- *initial_pose_y* - početna pozicija u smjeru osi Y, koristi se za inicilizaciju filtera sa Gaussovom distribucijom
- *initial_pose_z* - početna pozicija u smjeru osi Z, koristi se za inicilizaciju filtera sa Gaussovom distribucijom
- *selective_resampling* - kada je ovaj parametar uključen, smanji se period uzorkovanja kada ono nije potrebno, pomaže pri izbjegavanju uskraćivanja čestica.

Ostali parametri vezani uz filter čestica: *initial_cov_xx, initial_cov_yy, initial_cov_aa, gui_publish_rate, save_pose_rate, use_map_topic, first_map_only.*
Parametri vezani uz laser:

- *laser_min_range* - minimalna udaljenost laserske zrake koja će se uzeti u obzir
- *laser_max_range* - maksimalna udaljenost laserske zrake koja će se uzeti u obzir
- *laser_z_hit* - težina za z_hit varijablu modela
- *laser_z_short* - težina za z_short varijablu modela
- *laser_z_max* - težina za z_max varijablu modela
- *laser_z_rand* - težina za z_rand varijablu modela
- *laser_sigma_hit* - standardna devijacija za Gausov model korišten u z_hit dijelu modela
- *laser_lambda_short* - eksponencijalna raspodjela parametra za z_short dio modela
- *laser_model_type* - parametar za određivanje modela
- *laser_likelihood_max_dist* - maksimalna udaljenost do proširene prepreke na mapi, koristi se u likelihood_field modelu.

Parametri vezani za odometriju modela su:

- *odom_model_type* - odabire se model odometrije koji se koristi; može se izabrati između *diff, omni, diff-corrected ili omni_corrected*

- *odom_alpha_1* - parametar koji određuje očekivani šum u procjeni rotacije odometrije iz rotacijske komponente kretanja robota.
- *odom_alpha_2* - parametar koji određuje očekivani šum u procjeni rotacije odometrije iz translacijske komponente kretanja robota.
- *odom_alpha_3* - određuje očekivani šum u procjeni translacije odometrije iz rotacijske komponente kretanja robota.
- *odom_alpha_4* - određuje očekivani šum u procjeni translacije odometrije iz translacijske komponente kretanja robota.
- *odom_alpha_5* - parametar za šum koji se povezan s translacijskim gibanjem.

Ostali parametri koji se tiču odometrije su: *odom_frame_id, base_frame_id, global_frame_id, tf_broadcast* [28].
Vrijednosti objašnjenih parametara koje su korištene u ovom radu dane su niže:

```
min_particles:    500
max_particles:    3000
kld_err:    0.05
kld_z:    0.99
update_min_d:    0.2
update_min_a:    0.5
resample_interval: 1
transform_tolerance:    0.1
recovery_alpha_slow:    0.0
recovery_alpha_fast:    0.0
initial_pose_x:    0.0
initial_pose_y:    0.0
initial_pose_z:    0.0

laser_max_beams:    30
laser_z_hit:    0.5
laser_z_short:    0.05
laser_z_max:    0.05
```

```
laser_z_rand:    0.5
laser_sigma_hit:    0.2
laser_lambda_short:    0.1
laser_lambda_short:    0.1
laser_model_type: likelihood_field
laser_likelihood_max_dist:  2.0

odom_model_type:    omni
odom_alpha1:    0.2
odom_alpha2:    0.2
odom_alpha3:    0.8
odom_alpha4:    0.2
odom_alpha5:    0.1
```

Launch datoteka definira se na sličan način kao i u prethodnom poglavlju. Umjesto *gmapping* čvora, pokreće se *amcl* čvor s korisnički definiranim parametrima. Svi ostali parametri koji nisu u *.launch* datoteci učitavaju se s predefiniranim vrijednostima [28]. Uz *amcl* čvor, još se pokreće i *rviz* sučelje. U *rviz-u* se koriste isti segmente kao što su se koristili u 5 poglavlju. Cijela *.launch* datoteka dana je u prilogu.

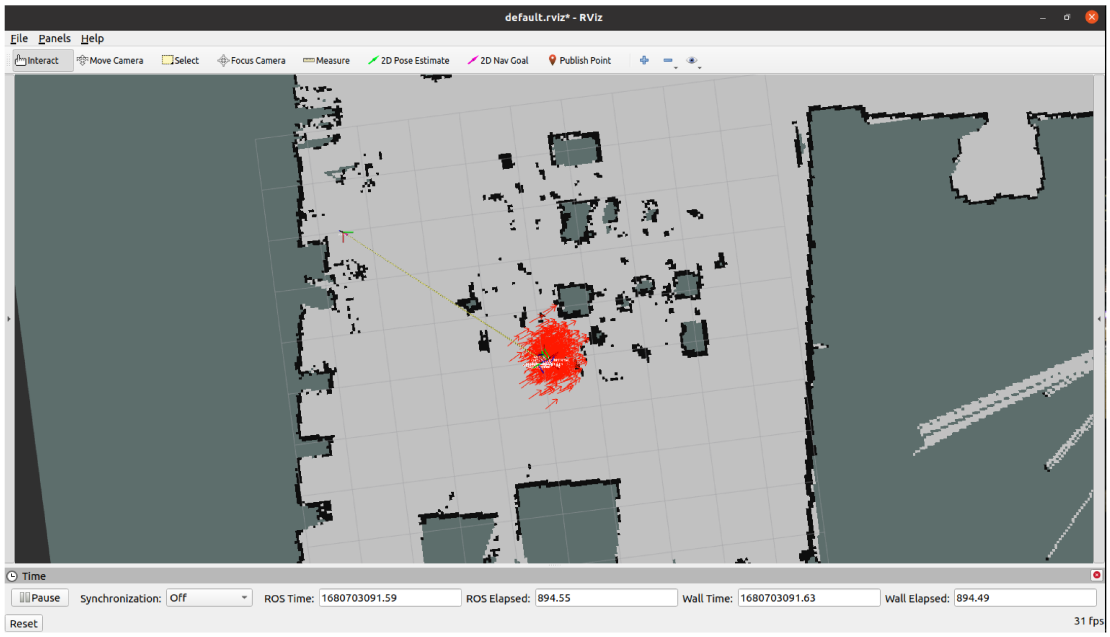
6.3. Proces lokalizacije u prostoru CRTA-e

Početak lokalizacije započinje pokretanjem *.launch* datoteke sljedećom naredbom:

```
$ roslaunch robotino localization.launch
```

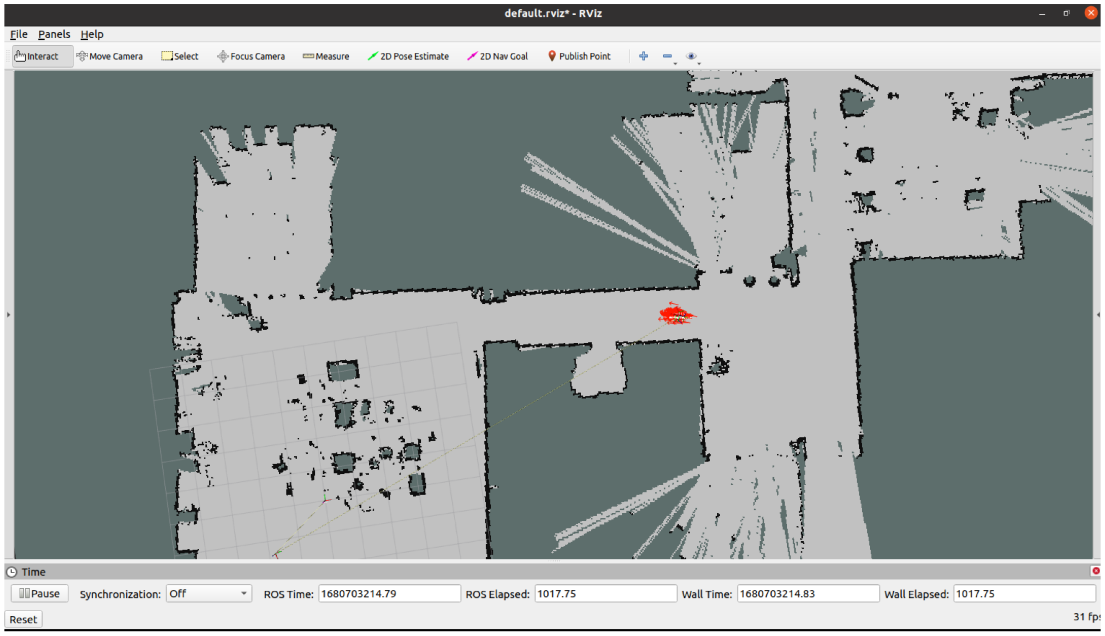
Paralelno uz to, u novom prozoru terminala, pokreće se čvor teleop_twist_keyboard. Pokretanjem *.launch* datoteke, otvara se *rviz* sučelje u koje se snimljena mapa prostora u kojem će se robot lokalizirati. U parametrima nije predefinirana početna pozicija robota, tako da se ista postavlja ručno. Odabirom na značajku koji nudi *rviz 2D Pose Estimate* postavlja se približna pozicija i orijentacija robota. Za demonstraciju objašnjenog, robota se postavlja u sredinu prostora. Odabirom na značajku *2D Pose Estimate*, odabire se približna pozicija robota na danoj mapi. Pritiskom i držanjem

tipke miša određuje se njegova orijentacija. Uočava se kako se stvorilo polje čestica koje sadrže poziciju i orijentaciju oko robota. Slika 6.1. prikazuje početni položaj robota s poljem čestica prije početka lokalizacije.



Slika 6.1: Estimirana pozicija robota u prostoru prije lokalizacije

Posljednji korak do uspješne lokalizacije zahtjeva upravljanje robotom po prostoru. Koristeći tipkovnicu robota se vozi po prostoru. Lidar senzorom, robot prepoznaje značajke prostora, što vodi do konvergiranja polja čestica orijentacije i pozicije.
Kako bi se postigla što bolja lokalizacija i konvergencija čestica pozicije i orijentacije, preporuka je da se robota vozi pored što više stacionarnih značajki prostora. Poželjno je robota voziti duž hodnika, blizu stupova i blizu zidova.
Nakon nekoliko minuta vožnje po prostoru vidljivo je da se polje čestica suzilo. Suženje polja čestica kao što je prikazano na slici 6.2, pokazuje na uspješnost lokalizacije.

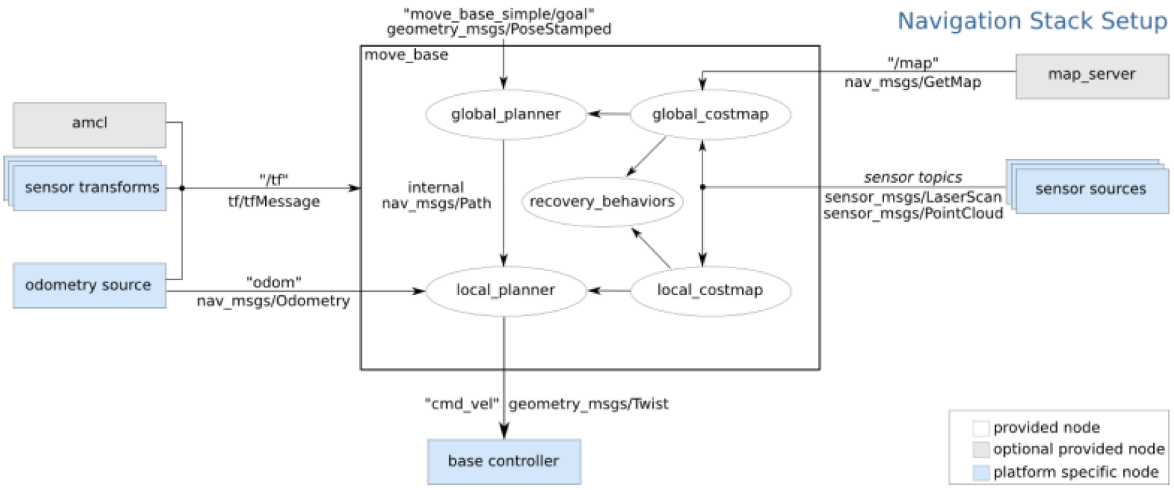


Slika 6.2: Uspješna lokalizacija robota u prostoru

Poglavlje 7.

Autonomna navigacija u prostoru

Navigacija, u kontekstu mobilne robotike, odnosi se na sposobnost robota da se autonomno kreće kroz prostor i samostalno izbjegava prepreke. Navigacija mobilnih robota zahtjeva kombinaciju senzora za percepciju okoline i algoritama za obradu podataka koji senzori pružaju. Dani algoritmi su zaslužni za izradu trajektorija i za slanje potrebnih brzina robotu kako bi izvršio svoj zadatak.
Sljedeća slika daje pregled potrebnih informacija koje je potrebno dostaviti ROS-u kako bi navigacija uspješno radila.



Slika 7.1: Blok dijagram rada *navigation stack-a* [30]

Za uspješnu navigaciju putem ROS okruženja potrebno je čvoru za navigaciju osigurati informacije kao što su informacije o mapi, odometriji, signalima senzora i krajnjoj

točci [30]. Čvor koji se koristi u ovom radu za proces navigacije zove se *move_base* [31]. Postavljanje navigacije svodi se na postavljanje opće *costmap-e*, lokalne i globalne *costmap-e* te lokalnog i globalnog planera. Navedeni segmenti navigacije postavljaju se u obliku *.yaml* datoteka [32]. *Yaml* datoteke su smještene unutar korisnički kreiranog paketa *robotino_ws*. Nalaze se u direktoriju pod nazivom *config*. Pokretanje navigacije se izvodi pozivanjem *navigation.launch* datoteke. Ona je zaslužna za pokretanje navigacijskog čvora i učitavanje svih predefiniranih parametara unutar *config* direktorija. Niže je objašnjen svaki od spomenutih segmenta, dok se sve navedene datoteke nalaze u prilogu.

7.1. *move_base*

Move_base je paket unutar ROS-a koji omogućuje vođenje robota do željene pozicije koristeći *navigation stack* [31]. Sadrži istoimeni čvor koji se poziva sljedećom naredbom:

```
$ rosrun move_base move_base
```

Nakon pozivanja isti se pretplaćuje na sljedeće teme:

- *move_base/goal* - sadrži informacije o cilju koji robot mora postići u prostoru
- *move_base/cancel* - sadrži informacije o zahtjevu za prekid izvršavanja cilja
- *move_base_simple/goal* - pruža sučelje bez akcije, za korisnike koje ne zanima praćenje procesa izvršavanja zadatka.

Teme na koje move_base objavljuje tokom svog izvođenja su:

- *move_base/feedback* - tema koja sadrži povratnu informaciju o trenutnoj poziciji robota u prostoru
- *move_base/status* - pruža informacije o statusu postizanja cilja
- *move_base/result* - ova tema je prazna za *move_base action*
- *cmd_vel* - na ovu temu šalje potrebne brzine robotu koje su potrebne za postizanje cilja.

Servisi koje je moguće pozvati u sklopu čvora su:

- *make_plan* - servis koji dozvoljava korisniku da dobije plan do cilja, bez njegovog izvršavanja
- *clear_unknown_space* - servis koji omogućuje da se očisti nepoznat prostor u neposrednoj blizini robota
- *clear_costmaps* - servis koji omogućuje da se očiste prepreke u *costmap-i*; korisniku se preporučuje da ovaj servis koristi s oprezom, jer može doći do zabijanja u prepreke koje više nisu u *costmap-i*.

Također, čvor nudi nekoliko parametara koje je potrebno definirati prije samog procesa navigacije. Parametre koji su navedeni niže, pozivaju se preko *.launch* datoteke, parametri su definirani u *.yaml* datoteci. Isto vrijedi i za parametre koji se postavljaju u vezi globalne i lokalne *costmap-e*.
Parametri koje čvor nudi za konfiguriranje glase:

- *base_global_planner* - naziv dodatka za globalni planer
- *base_local_planner* - naziv dodatka za lokalni planer
- *recovery_behaviors* - lista ophođenja u slučaju da robot zapne i ne može pronaći izlaz
- *controller_frequency* - frekvencija kojom kontroler šalje brzine kontroleru robota
- *planner_patience* - vrijeme u sekundama, potrebno planeru da pronađe važeći plan prije nego izvrši operacije za čišćenje prostora
- *controller_patience* - vrijeme koliko dugo će kontroler čekati bez primanja važeće kontrole, prije nego krene u izvođenje operacija u slučaju zastoja
- *conservative_reset_dist* - udaljenost od robota (u metrima), iznad koje će prepreke biti očišćene s costmap-e
- *recovery_behavior_enabled* - parametar koji omogućuje hoću li se koristiti operacije u slučaju zastoja

- *clearing_rotation_allowed* - ako je ovaj parametar uključen, robot će izvršavati rotacijsko gibanje kako bi očistio prostor okolo sebe
- *shutdown_costmaps* - ako je ovaj parametar aktivan, costmap će se ugasiti kada je move_base neaktivan
- *oscillation_timeout* - koliko dugo, u sekundama, su dozvoljene oscilacije prije nego što robot krene s izvršavanjem operacija u slučaju zastoja
- *oscillation_distance* - koliko daleko, u metrima, se robot mora kretati kako se ne bi smatralo da oscilira na mjestu
- *planner_frequency* - frekvencija s kojom će se izvršavati provođenje globalne trajektorije
- *max_planning_retires* - koliko puta je planeru dozvoljeno da pronađe izlaz, prije nego što se aktiviraju operacije u slučaju zastoja.

Objašnjeni parametri su korišteni prilikom definiranja značajki globalnog i lokalnog planera. Definirani su u *.yaml* datotekama koje su dane u prilogu. Također, svako podpoglavlje niže sadrži izdvojene korištene parametre s korištenim vrijednostima [31].

7.1.1. Costmap

Prilikom navigacije, algoritam koristi dvije mape težinskih faktora, *costmap*, koje sadrže informacije o preprekama u prostoru [33]. Radi se o globalnoj i lokalnoj mapi. Globalna mapa se koristi za planiranje trajekotrije duž cijelog prostora. Lokalna mapa se koristi u svrhe izbjegavanja prepreka, koje su u neposrednoj blizini robota.

Paket *costmap_2D* pruža mogućnost implementacije mapa težinskih faktora. Navedene mape uzimaju podatke s lidar senzora te stvaraju 2D ili 3D mape okupiranosti prostora. Također, paket nudi mogućnost izrade mape okupiranosti prostora na temelju već snimljene mape te izrade mape težinskih faktora s očitanjima senzora u realnom vremenu.
Korištenjem *costmap_2d* paketa, isti se pretplaćuje na temu *footprint*, s koje uzima informacije o gabaritnim dimenzijama robota.

Prilikom rada objavljuje isti objavljuje na teme:

- *costmap* - tema na koju se objavljuju vrijednosti težinskih faktora unutar mape
- *costmap_updates* - tema na koju se objavljuju ažurirane vrijednosti težinskih faktora unutar mape
- *voxel_grid* - ukoliko postoji senzorska oprema koja dobavlja informacije o 3D prostoru, te informacije budu objavljene na ovoj temi.

Obje mape sadrže zasebnu datoteku u kojoj su definirani parametri mape. Također, postoje neki parametara koji su zajednički objema mapa. Ti parametri definirani su u *.yaml* datoteci koja se zove *common_costmap*.
Parametri koji se koriste za definiranje mapa težinskih faktora objašnjeni su niže:

- *obstacle_range* - maksimalni doseg izražen u metrima u kojem je moguće postaviti prepreke u *costmap-i* koristeći podatke sa senzora
- *raytrace_range* - maksimalni doseg izražen u metrima u kojem je moguće raspoznati prepreke na mapi koristeći podatke sa senzora
- *robot_radius* - parametar koji definira gabaritnu mjeru radijusa robota
- *inflation _radius* - radius koji se stvara okolo prepreka, tumači se kao zauzeti prostor
- *footprint_clearing_enabled* - ako je postavljeno istinito, robotski otisak će označiti prostor po kojem se kreće kao slobodan
- *global_frame* - parametar koji određuje što predstavlja globalni koordinatni sustav
- *robot_base_frame* - parametar koji određuje koordinatni sustav robotske baze
- *transform_tolerance* - Određuje kašnjenje podataka transformacije koje je prihvatljivo u sekundama
- *update_frequency* - parametar koji definira frekvenciju ažuriranja mape
- *publish_frequency* - parametar koji definira frekvenciju objavljivanja mape

- *rolling_window* - parametar koji definira hoće li mapa pratiti robota kroz prostor
- *static_map* - parametar koji definira da li će mapa biti statična
- *resolution* - parametar koji definira rezoluciju mape.

Za adekvatan rad *costmap_2d* paketa potrebno je osigurati transformaciju iz globalnog koordinatnog sustava u koordinatni sustav robota (*robot_base_frame*).
Parametri koji su zajednički objema mapama su dani niže:

```
obstacle_range: 2.5
raytrace_range: 3.0
robot_base_frame: base_link
robot_radius: 0.225
inflation_radius: 0.3
footprint_clearing_enabled: true
transform_tolerance: 6.0
map_type: costmap

static:
    map_topic: map
    subscribe_to_updates: true

obstacles_laser:
    observation_sources: laser_sensor
    laser_sensor: {data_type: LaserScan, clearing: true,
    marking: true, topic: scan, inf_is_valid: true}

inflation:
    inflation_radius: 0.25
    cost_scaling_factor: 1
```

Parametri globalne mape težinskih faktora su:

```
global_costmap:
  global_frame: map
```

```
  update_frequency: 3.0
  static_map: true
  robot_base_frame: base_link

  plugins:
  - {name: static, type: "costmap_2d::StaticLayer"}
  - {name: inflation, type: "costmap_2d::InflationLayer"}
```

Parametri lokalne mape težinskih faktora su:

```
local_costmap:
  global_frame: odom
  update_frequency: 5.0
  publish_frequency: 1.0
  robot_base_frame: base_link
  static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.1
  width: 4.0
  height: 4.0

  plugins:
  - {name: obstacles_laser, type: "costmap_2d::ObstacleLayer"}
  - {name: inflation, type: "costmap_2d::InflationLayer"}
```

Za pregled svih parametara koji se učitaju s predefiniranim vrijednostima čitatelja se upućuje na [33].

7.1.2. Globalni planer

Algoritam koji je zadužen za stvaranje trajektorije i koji vodi robota do željenog cilja uz izbjegavanje prepreka zove se globalni planer. U ROS-okruženju on je implementiran

kao dodatak čvoru *move_base* te se definira unutar *.launch* datoteke [34]. Tema na koju *global_planner* objavljuje naziva se *plan*. Većina parametara koji su dostupni za definiranje globalnog planera određuju njegovu vrstu. Dva parametra imaju utjecaj na planiranje trajektorije, a to su:

- *default_tolerance* - parametar koji određuje toleranciju krajnjeg cilja planera
- *allow_unknown* - parametar koji definira hoće li se robot kretati po neistraženim dijelovima mape.

Dodatak nudi četiri dostupna planera koja se mogu koristiti, a to su: *navfn, djikstra, quadratic, grid_path i old_navfn, A* i dijkstra.* Planeri se razlikuju u načinu izračunavanja trajektorije. Za detaljnije objašnjenje pojedinog planera čitatelja se upućuje na dokumentaciju [34].
Globalni planer koji je odabran u svrhe ovog rada je *navfn.* Vrijednosti parametara koji su postavljeni za odabrani planer u obliku *.yaml* datoteke su:

```
    NavfnROS:
allow_unknown:    true
default_tolerance:    0.01
```

Ostali parametri koji se inicijaliziraju s predefiniranim vrijednostima mogu se pronaći na [34].

7.1.3. Lokalni planer

Base_local_planner je paket koji implamentira algoritme za lokalnu navigaciju robota u ravnini [35]. Zadatak lokalnog planera je da uz pomoć globalne trajektorije šalje brzine potrebne za dostizanje cilja uz izbjegavanje prepreka. Algoritam radi na principu stvaranja više loklanih trajektorija. Svaka stvorena trajektorija se vrednuje. Trajektorije na kojima robot dolazi u koliziju s preprekama budu odbačene. Najviše vrednovana trajektorija se izabire te se šalju brzine potrebne za njezino postizanje na robota. Ovaj postupak se ponavlja sve dok robot ne dostigne svoje odredište. Za detaljno objašnjenje algoritma vidi [35].

Tijekom rada lokalni planer se pretplaćuje na temu *odom.* Tema *odom* sadrži informacije o odometriji robota koje daju lokalnom planeru informacije o brzini robota. Informacije koje lokalni planer nudi objavljuje se na teme: *global_plan, local_plan i cost_cloud.*

Navedene teme se koriste isključivo u svrhe vizualizacije trajektorija.

Za definiranje ponašanja lokalnog planera postoji mnogo parametara. Mogu se grupirati u parametre vezane uz robota, toleranciju dostizanja cilja i točnost praćenja trajektorije. Za detaljan opis parametara vidi [35]. Svaki od planera sadrži parametre koji su specifični za taj planer. U svrhe ovog rada kao lokalni planer odabran je *teb_local_planner*. Parametri specifični za taj planer mogu se pronaći na [36]. Uz temu *odom*, *teb_local_planner* se pretplaćuje još na temu *obstacles* i *via_points*. Dodatne teme na koje objavljuje su:

- *teb_poses* - diskretna lista pozicije trenutnog lokalnog plana, koristi se u svrhe vizualizacije
- *teb_markers* - sadrži dodatne informacije za vizualizaciju, vezane uz prepreke
- *teb_feedback* - poruka ove teme sadrži isplaniranu trajektoriju uz profil brzina i listu prepreka. Primarno se koristi u svrhe procjene i debugging-a.

Odabrani parametri za *teb_local_planner* koji se koriste u ovom radu su dani niže:

```
    TebLocalPlannerROS:

odom_topic: odom
map_frame: /odom

# Trajectory

teb_autosize: True
dt_ref: 0.3
dt_hysteresis: 0.1
global_plan_overwrite_orientation: True
max_global_plan_lookahead_dist: 3.0
feasibility_check_no_poses: 5

# Robot
```

```
max_vel_x: 0.4
max_vel_y: 0.2
max_vel_x_backwards: 0.15
max_vel_theta: 0.3
acc_lim_x: 0.5
acc_lim_y: 0.3
acc_lim_theta: 0.5
min_turning_radius: 0.0
footprint_model: # types: "point", "circular", "two_circles",
# "line", "polygon"
  type: "point"
  radius: 0.225
  line_start: [-0.3, 0.0]
  line_end: [0.3, 0.0]
  front_offset: 0.2
  front_radius: 0.2
  rear_offset: 0.2
  rear_radius: 0.2
  vertices: [ [0.25, -0.05], [0.18, -0.05], [0.18, -0.18],
  [-0.19, -0.18], [-0.25, 0], [-0.19, 0.18], [0.18, 0.18],
  [0.18, 0.05], [0.25, 0.05] ]

# GoalTolerance

xy_goal_tolerance: 0.35
yaw_goal_tolerance: 0.2
free_goal_vel: False

# Obstacles
inflation_dist: 0.0
min_obstacle_dist: 0.35
include_costmap_obstacles: True
costmap_obstacles_behind_robot_dist: 1.0
```

```
obstacle_poses_affected: 25
costmap_converter_plugin: ""
costmap_converter_spin_thread: True
costmap_converter_rate: 5

# Optimization

no_inner_iterations: 5
no_outer_iterations: 4
optimization_activate: True
optimization_verbose: False
penalty_epsilon: 0.1
weight_max_vel_x: 2
weight_max_vel_theta: 1
weight_acc_lim_x: 1
weight_acc_lim_y: 0
weight_acc_lim_theta: 1
weight_kinematics_nh: 500
weight_kinematics_forward_drive: 1000
weight_kinematics_turning_radius: 1
weight_optimaltime: 1
weight_obstacle: 50

# Homotopy Class Planner

enable_homotopy_class_planning: True
enable_multithreading: True
simple_exploration: False
max_number_classes: 4
roadmap_graph_no_samples: 15
roadmap_graph_area_width: 5
h_signature_prescaler: 0.5
h_signature_threshold: 0.1
```

```
obstacle_keypoint_offset: 0.1
obstacle_heading_threshold: 0.45
visualize_hc_graph: False
```

7.1.4. Proces navigacije robota u prostoru CRTA-e

Kako bi pokretanje procesa navigacije bilo što lakše, kreirana je *navigation.launch* datoteka. Unutar njega definirano je pokretanje već spomenutih *.launch* datoteka za lokalizaciju i navigaciju te čvor za učitavanje mape i *rviz*-a. Ovim putem definirani su svi potrebni parametri. Pokretanje spomenute *.launch* datoteke izvršava se sljedećom naredbom:

```
$ roslaunch robotino navigation.launch
```

U posebnom terminalu pokreće se još čvor za upravljanje robotom kako je opisano u 3.3.2. poglavlju. Izgled *navigation.launch* datoteke dan je niže:

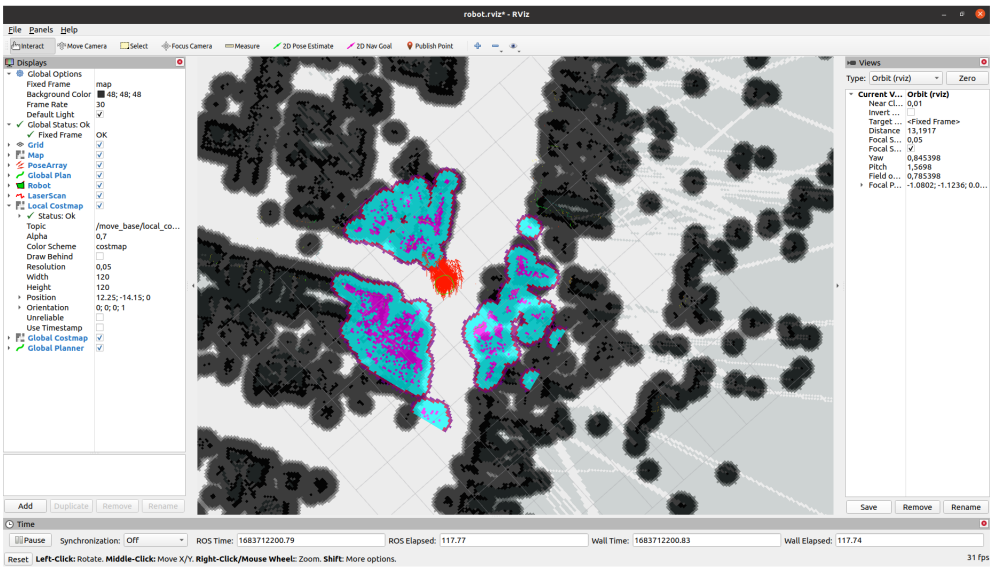
```
<?xml version="1.0"?>
<launch>

<!--  UCITAJ MAPU -->
  <arg name="map_file" default="$(find robotino)/maps/CRTA.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server"
  args="$(arg map_file)" />

<!-- POKRENI LOKALIZACIJA -->
  <include file="$(find robotino)/launch/localization.launch" />

<!-- POKRENI NAVIGACIJU -->
 <include file="$(find robotino)/launch/move_base.launch" />

<!-- POKRENI RVIZ -->
  <node pkg="rviz" type="rviz" name="rviz" args="-d
  $(find robotino)/config/robot.rviz">
```

```
  </node>

</launch>
```

Navigation.launch datoteka je zadužena za pokretanje *move_base* čvora preko svoje zasebne *.launch* datoteke pod nazivom *move_base.launch*. Navedena *.launch* datoteka izgleda ovako:

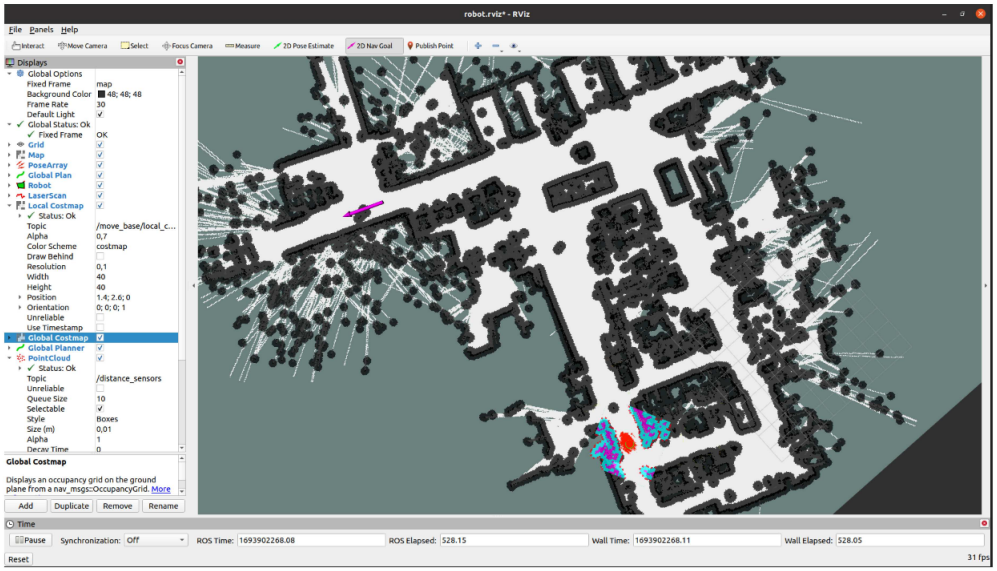
```
    <?xml version="1.0"?>
<launch>
  <arg name="no_static_map" default="true"/>
  <arg name="base_global_planner" default="navfn/NavfnROS"/>

  <node pkg="move_base" type="move_base" respawn="false"
  name="move_base" output="screen">

    <rosparam file="$(find robotino)/config
    /base_local_planner_params.yaml" command="load"/>
    <rosparam file="$(find robotino)/config
    /costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find robotino)/config
    /costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find robotino)/config
    /local_costmap_params.yaml" command="load" />
    <rosparam file="$(find robotino)/config
    /global_costmap_params.yaml" command="load" />
    <rosparam file="$(find robotino)/config
    /base_local_planner_params.yaml" command="load" />

    <param name="base_local_planner"
    value="teb_local_planner/TebLocalPlannerROS" />
    <param name="controller_frequency" value="10.0" />
  </node>
</launch>
```

Unutar nje definiran je globalni i lokalni planer. Također, *move_base.launch* je zadužen za učitavanje parametara lokalne i globalne *costmap-e*, kao što i parametara lokalnog planera.
Nakon pokretanja *.launch* datoteke, otvara se *rviz*. Unutar *rviz*-a vidi se mapa prostora, preko koje se učitava mapa težinskih faktora, odnosno *costmap*. *Costmap* dodaje oko elementi u prostoru određenu imaginarnu širinu kako bi robot mogao pratiti trajektoriju sa sigurne udaljenosti od prepreka i zidova.



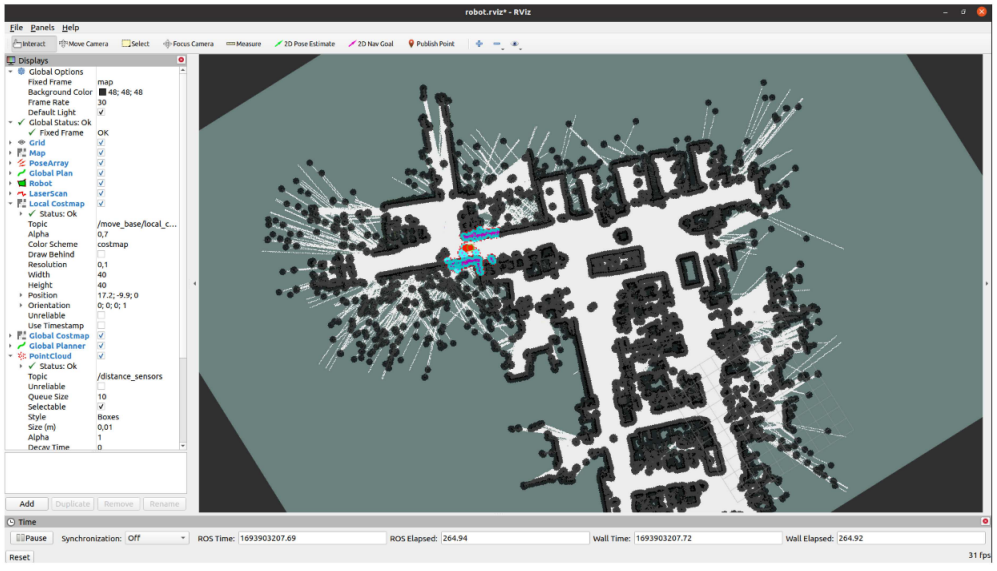
Slika 7.2: Lokalna i globalna *costmap-a*

Prije nego se robotu zada željeni cilj, potrebno je robota lokalizirati u prostoru. Lokalizacija se izvršava kako je opisano u poglavlju 6.
Kada se postigne zadovoljavajuća lokalizacija, moguće je zadati robotu cilj za autonomno navigiranje. Zadavanje željenog cilja, odnosno pozicije i orijentacije, izvršava se koristeći alat unutar *rviz-a 2D Nav Goal*. Grafički odabran cilj objavljuje se na temu *move_base_simple/goal*.



Slika 7.3: Zadavanje cilja robotu

Slika 7.4. prikazuje ostvarivanje zadanog cilja robota.



Slika 7.4: Uspješno postizanje cilja

Poglavlje 8.

Mogućnost poboljšanja sustava

Unutar ovog poglavlja navode se moguća poboljšanja robotskog sustava *Robotino*. Cilj ovog poglavlja je da inspirira čitatelje koji će se susresti sa spomenutim robotskim sustavom.
U vidu navigacije i autonomnog kretanja po prostoru predlaže se integracija već postojećeg taktilnog senzora i infracrvenih senzora.

Taktilni senzor, odnosno *bumper*, komunicira s ROS-om preko *robotino_node*-a, ali nema utjecaj na brzinu koja se šalje na robotsku platformu. Kada postoji više čvorova koji objavljuju brzine na temu *cmd_vel*, kao što su *move_base* i *teleop_twist_keyboard*, nužno je odabrati jednu. Iz tog razloga predlaže se korištenje paketa *twist_mux*, koji se ponaša kao multipleksor [37]. Navedeni paket pruža čvor koji se pretplaćuje na teme koje objavljuju *geometry_msgs::Twist* vrstu poruke te bira jednu koristeći sustav prioriteta. Potrebno bi bilo napisati čvor, koji će čitati poruke s teme *bumper* te u slučaju sudara slati brzinu s vrijednosti 0 na temu *cmd_vel*.

Integracija infracrvenih senzora je zamišljena u vidu mape težinskih faktora. Ideja je da se infracrveni senzori dodaju kao zaseban sloj u *costmap-i*. Trebalo bi prilagoditi poruke koje tema *distance_sensors* objavljuje te definirati njihov izvor unutar *costmap_common_params.yaml* datoteke. Integracijom navedenih senzora poboljšalo bi se izbjegavanje prepreka koje se nalaze u prosotru ispod visine lidar senzora.

Poglavlje 9.

Zaključak

Cilj rada bio je implementacija robotskog operativnog sustava i navigacijskih paketa na inudstrijski mobilni robot Robotino. ROS kao otvorena platforma skraćuje inžinjerima vrijeme potrebno da osposobe određeni robotski sustav, koristeći već gotove pakete i algoritme. Navigacijski paket se sastoji od paketa za mapiranje prostora, lokalizacije i navigacije. Kako bi se robot autonomno kretao po prostoru prvo mu treba mapa tog prostora. Ona može biti unaprijed učitana ili se može stvoriti uz pomoć lidar senzora. U svrhe ovog rada, prostor se mapirao te se u istom izvršavala lokalizacija robota. Naposljetku izvršena je navigacija robota po prostoru. Navigacija funkcionira uz pomoć lokalnog i globalnog planera. Globalni planer određuje trajektoriju do konačnog cilja, dok lokalni planer ažurira trajektorije s ciljem izbjegavanja prepreka.

Literatura

[1] https://ip.festo-didactic.com/InfoPortal/Robotino3/Overview/EN/index.html. Kolovoz 2023.

[2] https://ip.festo-didactic.com/InfoPortal/Robotino3/document/robotinomotorstechnicaldescription_en_de.pdf. Kolovoz 2023.

[3] https://ip.festo-didactic.com/InfoPortal/Robotino3/document/robotinoplanetarygearboxtechnicaldescription_en_de.pdf. Kolovoz 2023.

[4] https://ip.festo-didactic.com/InfoPortal/Robotino3/Hardware/Sensors/EN/DistanceSensors.html. Kolovoz 2023.

[5] https://en.wikipedia.org/wiki/Lidar. Kolovoz 2023.

[6] https://www.hokuyo-aut.jp/dl/UST-10LX_Specification.pdf. Kolovoz 2023.

[7] https://ip.festo-didactic.com/InfoPortal/Robotino3/Hardware/Controller/EN/EmbeddedPC.html. Kolovoz 2023.

[8] https://ip.festo-didactic.com/InfoPortal/Robotino3/document/robotinomicrocontrollerdatasheet.pdf. Kolovoz 2023.

[9] https://ip.festo-didactic.com/InfoPortal/Robotino3/Hardware/Controller/EN/Microcontroller.html. Kolovoz 2023.

[10] https://ip.festo-didactic.com/InfoPortal/Robotino3/Software/EN/index.html. Kolovoz 2023.

[11] https://www.redhat.com/en/topics/api/what-is-a-rest-api. Kolovoz 2023.

[12] https://wiki.openrobotino.org/index.php?title=Rest_api. Kolovoz 2023.

[13] https://www.theconstructsim.com/what-is-ros/. Kolovoz 2023.

[14] http://wiki.ros.org/ROS/Introduction. Kolovoz 2023.

[15] http://wiki.ros.org/ROS/Concepts. Kolovoz 2023.

[16] https://roboticsbackend.com/what-is-a-ros-parameter/. Kolovoz 2023.

[17] http://wiki.ros.org/robotino_node. Kolovoz 2023.

[18] http://wiki.ros.org/rviz. Kolovoz 2023.

[19] http://wiki.ros.org/rviz/UserGuide. Kolovoz 2023.

[20] http://wiki.ros.org/teleop_twist_keyboard. Kolovoz 2023.

[21] https://www.finnrietz.dev/linux/hokuyo-ros-setup/. Kolovoz 2023.

[22] http://wiki.ros.org/urg_node. Kolovoz 2023.

[23] http://wiki.ros.org/navigation. Kolovoz 2023.

[24] http://wiki.ros.org/gmapping. Kolovoz 2023.

[25] https://openslam-org.github.io/gmapping.html. Kolovoz 2023.

[26] http://wiki.ros.org/map_server. Kolovoz 2023.

[27] https://learnubuntu.com/change-directory/. Kolovoz 2023.

[28] http://wiki.ros.org/amcl. Kolovoz 2023.

[29] https://en.wikipedia.org/wiki/Bayes_estimator. Kolovoz 2023.

[30] http://wiki.ros.org/navigation/Tutorials/RobotSetup. Kolovoz 2023.

[31] http://wiki.ros.org/move_base. Kolovoz 2023.

[32] https://yaml.org/spec/1.2.2/. Kolovoz 2023.

[33] http://wiki.ros.org/costmap_2d. Kolovoz 2023.

[34] http://wiki.ros.org/global_planner. Kolovoz 2023.

[35] http://wiki.ros.org/base_local_planner. Kolovoz 2023.

[36] http://wiki.ros.org/teb_local_planner. Kolovoz 2023.

[37] http://wiki.ros.org/twist_mux. Kolovoz 2023.

[38] Frank Dellaert i dr. “Monte carlo localization for mobile robots”. *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*. Sv. 2. IEEE. 1999., str. 1322–1328.

[39] Dieter Fox i dr. “Monte carlo localization: Efficient position estimation for mobile robots”. *Aaai/iaai* 1999.343-349 (1999.), str. 2–2.

Prilog

1. Kodovi i datoteke korištene prilikom implementacije algoritama -`https://github.com/karlocirkvencic/robotino_SLAM`