Primjena Robotskog operativnog sustava 2 (ROS2) na mobilnom robotu u Gazebo virtualnom okruženju

Prekrit, Teo

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Teo Prekrit

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

Doc. dr. sc. Marko Švaco, mag. ing. mech. Teo Prekrit

Zagreb, 2024.

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

Zahvaljujem se obitelji na stalnoj podršci kroz studiranje, kao i prijateljima na podršci, zajedničkom učenju i zabavi u slobodno vrijeme.

Zahvaljujem se i mentoru doc. dr. sc. Marku Švaci i asistentu Branimiru Ćaranu na temi, smjernicama i pomoći tijekom pisanja ovog rada.

Teo Prekrit

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 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Teo Prekrit** JMBAG: **0035220480**

Naslov rada na hrvatskom jeziku: **Primjena Robotskog operativnog sustava 2 (ROS2) na mobilnom robotu u Gazebo virtualnom okruženju**

Naslov rada na engleskom jeziku: **Application of the Robot operating system 2 (ROS2) on a mobile robot in the Gazebo virtual environment**

Opis zadatka:

Robotski operativni sustav (ROS) je okruženje koje je iznimno zastupljeno u akademskoj zajednici robotičara. U posljednjih nekoliko godina sve je veća prisutnost ROS okruženja u industriji te se stoga javila potreba za njegovim unapređenjem. Kako je ROS razvijen kao okruženje za znanstvenu zajednicu i nije dovoljno robustan, zajednica je počela razvijati ROS2. ROS2 je druga inačica ROS-a koja je razvijana s ciljem bolje i robusnije implementacije u industriji mobilnih i industrijskih robota. Glavni naglasak u ROS2 okruženju je izvedba u realnom vremenu, robusnost i sigurnost sustava.

U sklopu ovog rada potrebno je:

- Detaljno istražiti arhitekturu i mogućnosti ROS2 te ga usporediti s ROS-om.
- Istražiti i implementirati algoritme razvijene za ROS u ROS2 okruženju.
- U Gazebo simulatoru izgraditi što detaljnije virtualno okruženje Laboratorija za računalnu inteligenciju u Regionalnom centru izvrsnosti za robotske tehnologije.
- Unutar kreiranog svijeta potrebno je implementirati mobilnog robota te ostvariti komunikaciju prema robotu iz ROS2 okruženja.
- Implementirati algoritme mapiranja i lokalizacije unutar ROS2 okruženja.

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

Zadatak zadan:
30. 11. 2023.

Zadatak zadao:
Doc. dr. sc. Marko Švaco

Datum predaje rada:
1. rok: 22. i 23. 2. 2024.
2. rok (izvanredni): 11. 7. 2024.
3. rok: 19. i 20. 9. 2024.

Predviđeni datumi obrane:
1. rok: 26. 2. – 1. 3. 2024.
2. rok (izvanredni): 15. 7. 2024.
3. rok: 23. 9. – 27. 9. 2024.

Predsjednik Povjerenstva:
Prof. dr. sc. Damir Godec

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

Zadatak ovog rada je istražiti Robotski operativni sustav 2 (ROS 2) i primijeniti spoznaje na primjeru mapiranja Regionalnog centra izvrsnosti za robotske tehnologije (CRTA). Na početku rada opisana je arhitektura ROS-a 2 i najvažniji koncepti ove platforme, nakon čega slijedi usporedba s izvornom verzijom, ROS-om 1, gdje su opisane novosti koje omogućuju primjenu ROS-a 2 u realnom sektoru. Opisan je i način na koji je moguće implementirati rješenja razvijena za ROS 1 u ROS-u 2 kada ta rješenja nisu još prilagođena novom izdanju. Nakon toga slijedi pregled implementacije simultane lokalizacije i mapiranja (SLAM) na primjeru virtualne verzije CRTA-e u simulacijskom softveru Gazebo pomoću robota TurtleBot3.

Ključne riječi: Robotski operativni sustav (ROS), ROS 2, simultana lokalizacija i mapiranje, SLAM, Gazebo, TurtleBot3

SUMMARY

The goal of this paper is to research the Robot Operating System 2 (ROS 2) and apply the gained knowledge by mapping the Regional Center of Excellence for Robotic Technology (CRTA). The first part of the paper describes the architecture and prime concepts of ROS 2. This is followed by an outline of the differences between ROS 2 and its predecessor, ROS 1, that made ROS 2 a viable choice in industrial settings, as well as an overview of implementing ROS 1-compatible solutions in ROS 2. The final part of this paper deals with implementing a simultaneous localization and mapping (SLAM) solution in a virtual representation of CRTA in the Gazebo simulator using an edition of the TurtleBot3 robot.

Key words: Robot Operating System (ROS), ROS 2, Simultaneous Localization and Mapping, SLAM, Gazebo, TurtleBot3

1. UVOD

Automatizacija i robotika posljednjih su desetljeća jedne od najbrže rastućih tehnoloških grana sa širokim spektrom primjena, od automobilske, proizvodne i prehrambene industrije, svemirskih istraživanja, pa sve do medicine i transportne i skladišne logistike.

S obzirom na široku mogućnost primjene robotskih tehnologija, vjerojatno je da će se mnogi inženjeri strojarstva barem jednom susresti s barem nekim aspektom automatizacije i robotike.

No širina robotike ne očituje se samo u njenim primjenama, nego i potrebnim znanjima. Za razvoj jednog kompletnog robotskog sustava potrebna su znanja iz strojarstva i mehatronike, elektrotehnike, računarstva, umjetne inteligencije i strojnog učenja, matematike, itd. To predstavlja znatan izazov početnicima, ako ne i malu dozu straha na početku.

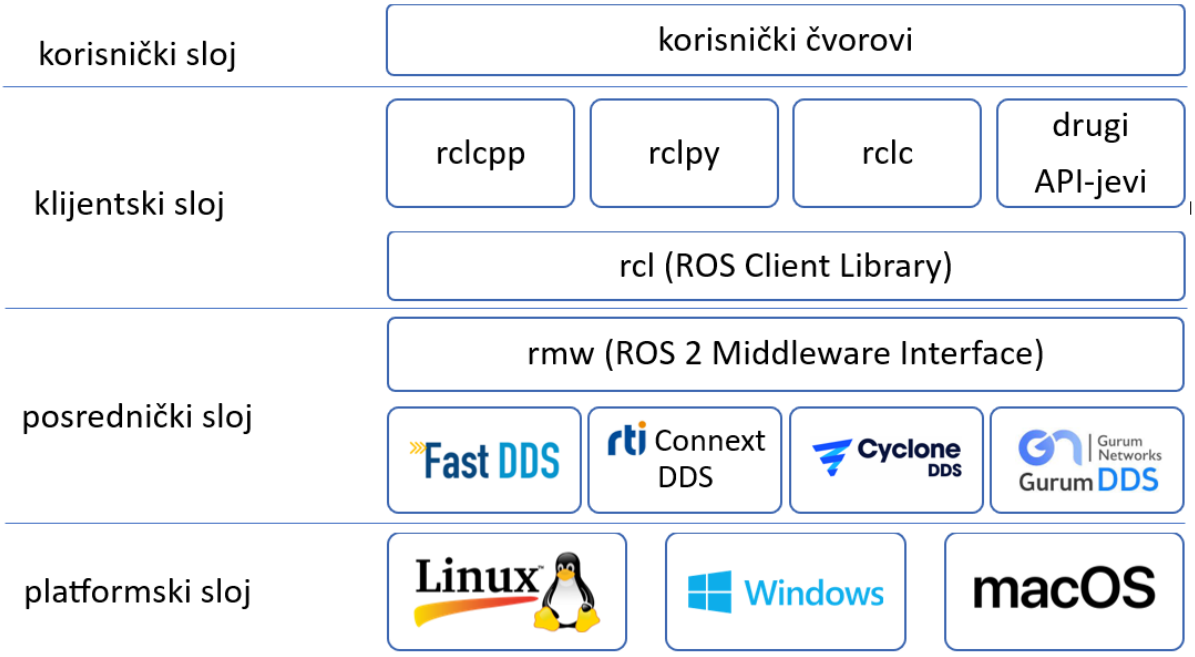
Taj su problem primijetili i Eric Berger i Keenan Wyrobek, doktoranti na Sveučilištu Stanford. Također su bili frustrirani stalnim „izmišljanjem tople vode“, tj. ponovnim kompletnim implementiranjem funkcija robotskih sustava koje je već netko izmislio za svoj sustav. U svrhu rješavanja tih problema 2007. godine započeo je razvoj Robotskog operativnog sustava (ROS). Cilj njegovog razvoja bio je stvoriti otvorenu robotsku platformu čiji će korisnici dijeliti svoja programska i druga rješenja s ostatkom ROS-ove zajednice, čime će se značajno smanjiti vrijeme razvoja novih robotskih sustava.

Prvo javno izdanje ROS-a izdano je 2010. godine pod nazivom *Box Turtle*. Kroz godine je popularnost ROS-a brzo rasla i započela je primjena ROS-a izvan istraživačkih polja, tj. u industriji. S ciljem sazrijevanja ROS-a kao stabilnog sustava prihvatljivog za korištenje u industriji, 2014. godine započeo je razvoj ROS-a 2 s podrškom za rad u stvarnom vremenu, boljom sigurnošću i robusnijom arhitekturom. Prvo izdanje ROS-a 2 objavljeno je 2018. godine pod nazivom *Ardent Apalone*. Nova izdanja ROS-a 2 izlaze svakih šest mjeseci i podržana su godinu dana. Svako drugo izdanje ROS-a 2 je verzija s dugoročnom podrškom (engl. *long-term support*, LTS) koja prima ažuriranja narednih pet godina od izdavanja.

U svrhu ovog završnog rada koristit će se najnovije izdanje s dugoročnom podrškom u trenutku pisanja, ROS 2 *Humble Hawksbill*, koje je izdano 2022. godine i Ubuntu 22.04 LTS.

2. ARHITEKTURA I OSNOVNI KONCEPTI ROS-A 2

Jedan od najvažniji aspekata ROS-a 2 je njegova modularnost, otvorenost i konfigurabilnost. Filozofija samog projekta je da ga svatko može prilagoditi svojim potrebama, bilo to u vidu robota i drugog hardvera na kojem će se pogoniti ili raznih primjena u praksi. Time je arhitekturu ROS-a najlakše dočarati kao slojeve i komponente unutar tih slojeva, kao na sljedećoj slici. [1]



Slika 1. Slojeviti prikaz arhitekture ROS-a 2 (izvor: autor)

U daljnjem će tekstu biti opisani pojedini slojevi i njihove komponente, te popratne informacije potrebne za lakše razumijevanje.

2.1. Korisnički sloj

Korisnički sloj sadrži čvorove i pomoćni programski kod (konfiguracijske datoteke, biblioteke, podatke i sl.) kojeg su pisali korisnici u određenom programskom jeziku, a koji su grupirani u pakete (engl. *packages*). Čvorovi (engl. *nodes*) su „programi“ u ROS-u koji obavljaju neke funkcije i komuniciraju s drugim čvorovima na tri načina – pomoću tema (engl. *topics*), servisa (engl. *services*) i akcija (engl. *actions*). [2]

2.1.1. Komunikacija među čvorovima

Teme su asinkroni načini komunikacije između čvorova koji radi na principu *objavljivač-pretplatnik* (engl. *publisher-subscriber*). Jedan čvor objavljuje poruke na neku temu, a drugi čvor se pretplaćuje na tu temu i prima te iste poruke. Tema može imati nula i više objavljivača i isto toliko pretplatnika. Primjer ovakve komunikacije može biti da čvor koji pogoni neki temperaturni senzor objavljuje informacije o temperaturi na neku temu, a drugi se čvorovi za čiji je rad bitna informacija o temperaturi pretplaćuju na tu istu temu. [1], [3]

Servisi su sinkroni (određeni zadatak se mora završiti prije nego se krene na sljedeći) način komunikacije između čvorova gdje jedan čvor pošalje zahtjev drugom čvoru i čeka njegov odgovor. Kod servisa se očekuje da se taj odgovor primi što prije kako bi čvor koji je poslao zahtjev nastavio s radom, zbog čega servisi nisu pogodni za dugotrajne radnje. Čvor koji je poslao zahtjev naziva se *servisni klijent* (engl. *service client*), dok se čvor koji je primio zahtjev i mora izvršiti obradu podataka naziva *servisni poslužitelj* (engl. *service server*). [1], [4]

Akcije su asinkroni način komunikacije između čvorova koje rade na principu sličnom servisima, no pogodni su za dugotrajne postupke jer čvor koji pošalje zahtjev (neki željeni cilj) ne mora čekati na *krajnji* odgovor primatelja zahtjeva. Analogno servisima, kod akcija se pošiljatelj zahtjeva naziva *akcijski klijent* (engl. *action client*), a primatelj zahtjeva naziva se *akcijski poslužitelj* (engl. *action server*). Akcijski klijent može otkazati ili promijeniti cilj akcije. Primjer primjene gdje je ovaj način komunikacije pogodan je navigacija jer ona često može trajati nekoliko minuta, a akcijski klijent može u međuvremenu obavljati druge potrebne radnje. Neki akcijski klijent može poslati akcijskom poslužitelju (u ovom slučaju navigacijskom sustavu robota) točku u kojoj se robot mora nalaziti. Navigacijski sustav počinje pomicati robota prema cilju i po putu šalje akcijskom klijentu povratne informacije o trenutnom napretku. Kada robot stigne do cilja šalje informaciju o završetku. [1], [5]

2.1.2. Struktura paketa

Paketi su glavna organizacijska jedinica bilo kakvog korisničkog programskog koda u ROS-u i omogućuju jednostavno dijeljenje vlastitog koda.

U ROS-u 2 paketi se kompiliraju pomoću alata *colcon* i sustava za kompiliranje *ament*. Mogu se pisati Python i CMake paketi, a svaki od njih ima točno određenu minimalnu strukturu. [6], [7], [8]

Za CMake pakete minimalna struktura se sastoji od sljedećih datoteka:

- *CMakeLists.txt* – opisuje kako se kompilira kod u paketu,
- *include/<naziv_paketa>* – mapa koja sadrži C++ *header* datoteke u kojima su definirane neke osnovne funkcije koje su nam potrebne,
- *package.xml* – sadrži osnovne informacije o paketu (naziv, opis, licencija, drugi potrebni paketi (engl. d*ependencies*), autor, itd.) i
- *src* – mapa koja sadrži programski kod paketa.

Za Python pakete minimalnu strukturu čine:

- *package.xml* – s istom svrhom kao i gore,
- *resource/<naziv_paketa>* – marker datoteka paketa,
- *setup.cfg* – potrebno kada paket sadrži izvršne datoteke,
- *setup.py* – opisuje instalaciju paketa i
- *<naziv_paketa>* – mapu s istim imenom kao paket u kojoj se nalazi programski kod.

2.1.3. *Launch* datoteke

Launch datoteke bitan su element ROS-a 2 jer omogućuju lakše pokretanje kompleksnih sustava s mnoštvom čvorova i parametara koje korisnik želi definirati. Mogu se pisati u Pythonu, XML-u ili YAML-u, a odabir formata ovisi o korisnikovim potrebama i znanjima. Najfleksibilniji format od ta tri je Python jer je skriptni programski jezik. Python dopušta i veću kontrolu nad pokretanjem ROS 2 paketa jer su i same ROS-ove *launch* i *launch_ros* značajke pisane u tom jeziku. [9]

2.1.4. ROS zajednica

ROS ima vrlo razvijenu zajednicu koja stvara pakete za širok spektar radnji, podršku za senzore, motore, gotova robotska rješenja i drugi hardver, dokumentaciju i drugi sadržaj od iznimne važnosti za razvoj vlastitih sustava. Time bismo ovaj sloj mogli nazvati i slojem zajednice.

2.2. Klijentski sloj

Klijentski sloj sadrži klijentske biblioteke (engl. *client libraries*) koje povezuju programski kod pisan nekim programskim jezikom sa samim ROS-om kroz aplikacijska programska sučelja (engl. *application programming interface*, API). Baza svih klijentskih biblioteka je ROS klijentska biblioteka (engl. *ROS Client Library*, RCL) koji je pisan u programskom jeziku C. RCL implementira logiku i koncepte ROS-a koji moraju funkcionirati identično za sve programske jezike i omogućuje adaptaciju drugim jezicima bez da je potrebno za svaki jezik ispočetka implementirati svu funkcionalnost. Službene biblioteke su „*rclcpp"* za programiranje u C++ i „*rclpy*" za Python, a zajednica održava klijentske biblioteke i za druge jezike – npr. C, Javu, .NET Core, C# i druge. Korisnici ne rade direktno s RCL-om, nego s višim klijentskim bibliotekama koje ga adaptiraju. Ne koristi se ni kada korisnik želi raditi u C-u, nego za to koristi biblioteku „*rclc*". [10]

2.3. Posrednički sloj

Posrednički sloj (engl. *middleware layer*) sadrži jednu od tržišno dostupnih implementacija DDS standarda i *ROS 2 posredničko sučelje* (engl. *ROS 2 Middleware Interface*, RMW). Standardna instalacija ROS-a koristi *Fast DDS*, osim *ROS-a 2 Galactic* u kojem je to *Cyclone DDS*. Korisnik može odabrati i neku drugu od četiri službeno podržane implementacije.

DDS pruža ROS-u 2 decentraliziranu komunikacijsku infrastrukturu temeljenu na modelu *objavljivač-pretplatnik* koja se izvršava u stvarnom vremenu, pogodnu za ROS-ovu distribuiranu arhitekturu. Komunikacijska infrastruktura kod ROS-a 2 je decentralizirana u smislu da se ne vodi kroz jedan središnji sustav, nego je temeljena na *peer-to-peer* principu, a međusobno otkrivanje (engl. *discovery*) čvorova vrši se kroz višeodredišno emitiranje (engl.

multicast). Time se uklanja rizik pada komunikacije između čvorova kada bi taj hipotetski središnji sustav zakazao. Svaki čvor će kroz *multicast* najavljivati svoju pristupnost drugim čvorovima na istoj ROS domeni, a koja se postavlja pomoću varijable okruženja (engl. *environment variable*) „ROS_DOMAIN_ID“. Čvorovi u istoj domeni mogu slobodno međusobno komunicirati i razmjenjivati informacije, ali to ne mogu činiti s čvorovima u drugim domenama. [1], [11], [12] Standardni protokol za *multicast* otkrivanje čvorova kojeg koristi DDS je *Simple Discovery Protocol*.

Pod distribuiranom arhitekturom podrazumijeva se da se ne moraju svi čvorovi jednog ROS sustava izvršavati na istom računalu, tj. hardveru. Na primjer, upravljački čvorovi i čvorovi za obradu prikupljenih podataka mogu se izvršavati na udaljenom računalu, dok se čvorovi za kretanje, mapiranje, lokalizaciju, navigaciju i prikupljanje informacija pomoću raznih senzore izvršavaju na samom robotu.

Uz osnovnu komunikacijsku infrastrukturu, DDS pruža i sigurnosne značajke i značajke pouzdanosti. Sigurnosne značajke uključuju šifriranje komunikacije, kontrolu pristupa itd. Šifriranje se provodi pomoći metode simetričnog šifriranja algoritmom AES-GCM. Da bi čvorovi mogli čitati šifrirane podatke, autenticiraju se digitalnim potpisima, tzv. kriptografijom javim ključem. Pomoću kontrole pristupa, autenticirani čvorovi mogu otkriti samo čvorove za koje imaju dopuštenje. [13]

Za pouzdanost su podržane politike kvalitete usluge (engl. *Quality of Service policies*, QoS). Primjeri nekih politika kvalitete usluge su vrijeme između objavljivanja i primanja neke poruke nakon kojeg će se poruka smatrati zastarjelom ili broj uzoraka nekog senzora koji će se čuvati. [14]

Službeno su podržane četiri različite implementacije DDS-a, a to *su eProsima Fast DDS, RTI Connext DDS, Eclipse Cyclone DDS* i *GurumNetworks GurumDDS*. Iako su njihove implementacije standarda DDS prilično slične, aplikacijska programska sučelja tih implementacija međusobno se razlikuju radi čega se između RCL-a i DDS-a nalazi *ROS 2 posredničko sučelje* (RMW). RMW omogućuje programerima klijentskih biblioteka pristup

različitim DDS implementacija kroz ujedinjeno, standardizirano aplikacijsko programsko sučelje, čime je moguće koristiti te klijentske biblioteke s bilo kojim DDS-om bez potrebe za prilagođavanjem. [15] DDS-ovi su vrlo složeni sustavi pa RMW pojednostavljuje API za većinu korisnika, ali i dalje dopušta pristup specifičnim API-jevima pojedinih implementacija za korisnike koji u nekim krajnjim slučajevima trebaju takav pristup. [16]

2.4. Platformski sloj

Platformski sloj predstavlja operacijski sustav na kojem se izvodi ROS 2, a to može biti neka distribucija Linuxa, Windows ili macOS. S obzirom da tim za razvoj ROS-a ima ograničene resurse i ne može podržavati svaki operacijski sustav, oni se moraju kategorizirati u tri ranga – *Tier 1, Tier 2* i *Tier 3.*

U *Tier 1* spadaju operacijski sustavi za koje tim provodi učestalo testiranje, kontinuiranu integraciju programskog koda i testiranje performansi. Za te se operacijske sustave prioriziraju programske greške u ROS-u 2, što znači da može doći do značajne odgode izdavanja nove verzije ROS-a 2, ako su kod tih operacijskih sustava u ROS-u i dalje prisutne greške koje treba ispraviti. U ovaj rang spadaju najnovija verzija *Ubuntua* s dugoročnom podrškom (engl. *long-term support,* LTS) i *Microsoft Windows 10*.

Kod *Tier* 2 operacijskih sustava periodično se provodi automatsko testiranje, ali u manjoj mjeri nego kod *Tier 1* operacijskih sustava. Programske greške se ispravljaju ovisno o slobodnim resursima u razvojnom timu jer je prioritet ispraviti greške kod *Tier 1* operacijskih sustava. To znači da do izdavanja nove verzije ROS-a 2 neće biti ispravljene sve greške kod njegovog pokretanja na tim operacijskim sustavima. U ovu kategoriju spada *Red Hat Enterprise Linux*.

Razvojni tim neće ni u kakvoj mjeri testirati ROS 2 na *Tier 3* operacijskim sustavima, nego ROS-ova zajednica prijavljuje da on funkcionira na tim platformama. Da bi sustav ušao bar u ovaj rang, za njega moraju biti dostupne i ažurne službene upute za instalaciju. Korisničku podršku za ove sustave pruža zajednica. U ovu kategoriju spada *macOS* i druge distribucije Linuxa, kao što je *Debian*. [17]

U nastavku se nalazi tablica podržanih platformi za ROS 2 Iron Irwini, preuzeta iz službene dokumentacije. ROS 2 Iron Irwini najnovija je verzija ROS-a 2 u trenutku pisanja ovog rada, no nije LTS verzija.

Tablica 1. Platforme koje podržava ROS 2 Iron Irwini [17]

Iron Irwini (May 2023 - November 2024)

Targeted platforms:

Architecture	Ubuntu Jammy (22.04)	Windows 10 (VS2022)	RHEL 9	macOS	Debian Bullseye (11)	OpenEmbedded / webOS OSE
amd64	Tier 1 [d][a][s]	Tier 1 [a][s]	Tier 2 [d][a][s]	Tier 3 [s]	Tier 3 [s]	
arm64	Tier 1 [d][a][s]				Tier 3 [s]	Tier 3 [s]
arm32	Tier 3 [s]				Tier 3 [s]	Tier 3 [s]

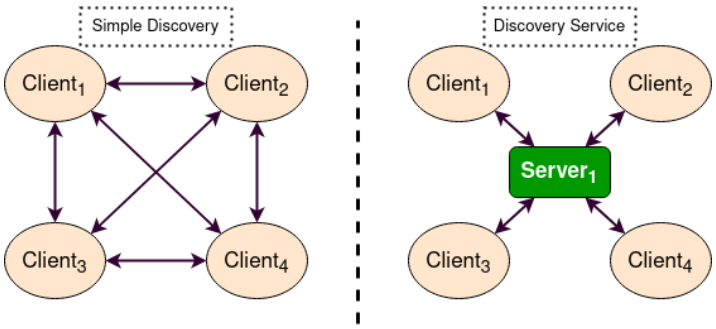
3. RAZLIKE IZMEĐU ROS-A 1 I ROS-A 2

3.1. Komunikacija između čvorova

Komunikacijski protokoli koje koristi ROS 1 su *User Datagram Protocol* (*UDP*) i *Transmission Control Protocol* (*TCP*), dok ROS 2 kao komunikacijski sloj koristi *Data Distribution Service* (*DDS*). [12]

Prijelazom na DDS omogućen je prijenos podataka u stvarnom vremenu (engl. *real-time data transmission*) i korisnici mogu prioritizirati čvorove pomoću *životnih ciklusa* (engl. *life cycles*). [18], [19] S obzirom da DDS radi na principu *objavljivač-pretplatnik*, za otkrivanje više nije potreban *ROS Master*, nego se otkrivanje vrši pomoću višeodredišnog emitiranja, dok se nakon uspostavljanja veze koristi jednoodredišno emitiranje (engl. *unicast*) čime se smanjuju hardverski zahtjevi u vidu dostupnih resursa. U osnovi, otkrivanje i povezivanje čvorova u ROS-u 1 vrši se pomoću središnjeg poslužitelja, dok se kod ROS-a 2 u istu svrhu koristi *peer-to-peer* arhitektura. [1], [12]

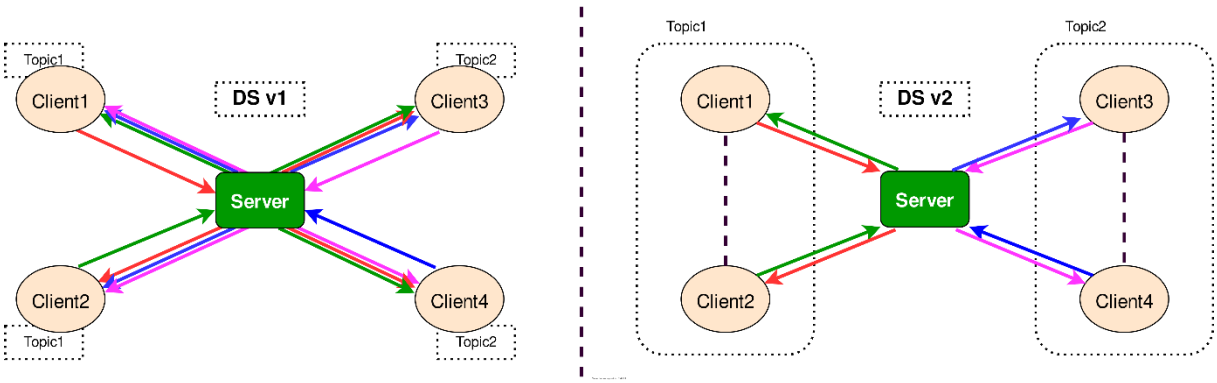
Bitno je napomenuti i da prilikom korištenja *Fast DDS-a* u ROS-u 2 također možemo umjesto *multicast* otkrivanja čvorova koristiti i središnji poslužitelj nalik ROS Masteru u ROS-u 1. Ta se funkcija zove *Fast DDS Discovery Server* koji pruža arhitekturu na principu klijent-poslužitelj. U tom slučaju svaki čvor je takozvani *klijent otkrivanja* (engl. *discovery client*) i dijeli informacije o sebi s jednim ili više *poslužitelja otkrivanja* (engl. *discovery server*) od kojih ujedno i prima informacije o drugim čvorovima. Razlika u odnosu na ROS 1 je da istovremeno može postojati više poslužitelja otkrivanja čime se izbjegava prekid komunikacije ako jedan poslužitelj prestane funkcionirati. [20]



Slika 2. *Multicast* otkrivanje i poslužitelj za otkrivanje čvorova [20]

Fast DDS Discovery Server u nekim je situacijama pogodna alternativa *multicast* otkrivanju jer *multicasting* ponekad neće biti potpuno pouzdan (npr. preko WiFi-ja) i jer se broj razmijenjenih mrežnih paketa značajno povećava s povećanjem broja čvorova u *multicastingu*.

Broj razmijenjenih mrežnih paketa dodatno je smanjen u drugoj verziji *Fast DDS Discovery Servera*, kada je uvedena funkcija filtriranja. Filtriranje se provodi na temelju tema svakog čvora, u smislu da će poslužitelj na temelju njih zaključiti hoće li dva čvora uopće htjeti komunicirati. Ako zaključi da ta dva čvora sigurno neće htjeti komunicirati, neće im ni slati informacije o njihovom međusobnom postojanju. Ta je funkcija prikazana na sljedećoj slici. [20]



Slika 3. Razlika u verzijama *Fast DDS Discovery Servera* - filtriranje [20]

Sa sigurnosne strane, omogućeno je šifriranje tema kako bi im mogli pristupiti samo autorizirani čvorovi, dok se sa strane pouzdanosti mogu definirati vlastite *politike kvalitete usluge*, a time se može postići balans između pouzdanosti poput TCP-a i brzine poput UDP-a koji su prethodno bili korišteni. Primjeri nekih politika kvalitete usluge su vrijeme između objavljivanja i primanja neke poruke nakon kojeg će se poruka smatrati zastarjelom ili broj uzoraka nekog senzora koji će se čuvati. DDS omogućuje ROS-u 2 i zamjenu čvorova tijekom rada sustava bez njegovog prethodnog gašenja (engl. *hot-swapping*). U slučaju da neki čvor zakaže tijekom rada sustava, njegovu ulogu može preuzeti zamjenski čvor bez zaustavljana rada samog sustava. [12], [14]

3.2. Naredbe i programiranje čvorova i paketa

Naredbe ROS-a 1 tipično su u obliku „ros*[naredba] [parametri]*“, dok su kod ROS-a 2 u obliku „ros2 *[naredba] [parametri]*“. U tablici u nastavku nalaze se primjeri naredbi u ROS-u 1 i ROS-u 2.

Tablica 2. Usporedba nekoliko primjera naredbi u ROS-u 1 i 2

ROS 1	ROS 2
rosnode list	ros2 node list
rostopic echo *[tema]*	ros2 topic echo *[tema]*
rosservice call *[servis] [argumenti]*	ros2 service call *[servis] [vrsta_servisa] [argumenti]*

Kod programiranja čvorova u ROS-u 2 koriste se *klijentske biblioteke* (engl. *client libraries*) *rclcpp* za C++ i *rclpy* za Python. Biblioteke „*rclcpp*“ i „*rclpy*“ analogne su verzije klijentskih biblioteka „*roscpp*“ i „*rospy*“ iz ROS-a 1 pisane za DDS-u u ROS-u 2. Biblioteke „*roscpp“* i „*rospy“* pisane su neovisno jedna o drugoj što je dovelo do razlika u njihovom ponašanju i konvencijama za imenske prostore. U ROS-u 2 to je ispravljeno uvođenjem ROS Client Libraryja (RCL-a), osnovne klijentske biblioteke pisane u C-u na koju se nadograđuju klijentske biblioteke za specifične programske jezike. Korisnik ne radi izravno s RCL-om, nego preko specifičnih biblioteka kao što su *rclcpp* i *rclpy*. [10]

Kod programiranja čvorova u Pythonu u ROS-u 2 postoje još dvije razlike u odnosu na ROS 1:

1. pretplatnici u ROS-u 2 *moraju* definirati svoj profil kvalitete usluge (engl. *Quality of Service (QoS) profile*) i
2. funkcija *spin()* izvodi se beskonačno mnogo puta umjesto jednom.

Profili kvalitete usluge su skupovi više politika kvalitete usluge. Određivanje profila kvalitete usluge je fleksibilno, no to može postati vrlo kompleksno pa ROS 2 pruža korisnicima nekoliko predefiniranih profila, npr. za podatke koje sustav prima od nekog senzora. [14]

Što se tiče alata za stvaranje i kompiliranje paketa, ROS 1 koristi *catkin*, a ROS 2 koristi *colcon*. Dok *catkin* ima naredbu za stvaranje novog paketa – „*catkin_create_package [naziv]*“, *colcon* je nema pa se za to koristi naredba *„ros2 pkg create [naziv]*“. Tom se naredbom stvara osnovna struktura paketa, tj. sve osnovne mape i datoteke. [12]

ROS 2, za razliku od ROS-a 1, podržava i izvođenje više čvorova u jednom procesu. Prednost toga može biti ubrzavanje komunikacije među čvorovima i njihovo jednostavnije pokretanje, no nedostatak je da bi greška kod jednog čvora mogla srušiti i ostale čvorove unutar istog procesa. [1], [13]

3.3. *Launch* datoteke

U ROS-u 2 *launch* datoteke mogu se pisati u Pythonu, YAML-u ili XML-u, dok je prije bilo moguće koristiti samo XML. Odabir formata ovisi o potrebama i znanju korisnika – osobe koje su prethodno koristile ROS 1 upoznate su s XML-om, dok će osobe koje trebaju veću fleksibilnost koristiti Python. [1], [9]

U tablici u nastavku možemo vidjeti primjer *launch* datoteka u obje verzije ROS-a koje su pisane za simulaciju istog virtualnog okruženja.

Tablica 3. Usporedba *launch* datoteka

empty_world.launch **(ROS 1)**	***empty_world.launch.py*** **(ROS 2)**

`<launch>`
`  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>`
`  <arg name="x_pos" default="0.0"/>`
`  <arg name="y_pos" default="0.0"/>`
`  <arg name="z_pos" default="0.0"/>`
 	
`  <include file="$(find gazebo_ros)/launch/empty_world.launch">`
`  <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/empt`
	`import os`
	
	`from ament_index_python.packages import get_package_share_directory`
	`from launch import LaunchDescription`
	`from launch.actions import ExecuteProcess`
	`from launch.actions import IncludeLaunchDescription`
	`from launch.launch_description_sources import PythonLaunchDescriptionSource`
	`from launch.substitutions import LaunchConfiguration`
	
	`TURTLEBOT3_MODEL = os.environ['TURTLEBOT3_MODEL']`
	
	`def generate_launch_description():`
	`  use_sim_time = LaunchConfiguration('use_sim_time',`

```
y.world"/>
    <arg name="paused"
value="false"/>
    <arg name="use_sim_time"
value="true"/>
    <arg name="gui"
value="true"/>
    <arg name="headless"
value="false"/>
    <arg name="debug"
value="false"/>
  </include>

  <param
name="robot_description"
command="$(find xacro)/xacro -
-inorder $(find
turtlebot3_description)/urdf/t
urtlebot3_$(arg
model).urdf.xacro" />

  <node pkg="gazebo_ros"
type="spawn_model"
name="spawn_urdf" args="-urdf
-model turtlebot3_$(arg model)
-x $(arg x_pos) -y $(arg
y_pos) -z $(arg z_pos) -param
robot_description" />
</launch>
```

```
default='True')
    world_file_name = 'empty_worlds/' +
TURTLEBOT3_MODEL + '.model'
    world =
os.path.join(get_package_share_directory(
'turtlebot3_gazebo'),
                            'worlds',
world_file_name)
    launch_file_dir =
os.path.join(get_package_share_directory(
'turtlebot3_gazebo'), 'launch')
    pkg_gazebo_ros =
get_package_share_directory('gazebo_ros')

    return LaunchDescription([
        IncludeLaunchDescription(

PythonLaunchDescriptionSource(

os.path.join(pkg_gazebo_ros, 'launch',
'gzserver.launch.py')
            ),
            launch_arguments={'world':
world}.items(),
        ),

        IncludeLaunchDescription(

PythonLaunchDescriptionSource(

os.path.join(pkg_gazebo_ros, 'launch',
'gzclient.launch.py')
            ),
        ),

        ExecuteProcess(
            cmd=['ros2', 'param', 'set',
'/gazebo', 'use_sim_time', use_sim_time],
            output='screen'),

        IncludeLaunchDescription(

PythonLaunchDescriptionSource([launch_fil
e_dir,
'/robot_state_publisher.launch.py']),

launch_arguments={'use_sim_time':
use_sim_time}.items(),
        ),
    ])
```

Vidimo da je *launch* datoteka ROS-a 1 pisana u XML-u u jednu ruku jednostavnija od *launch* datoteke za ROS 2 koja je pisana u Pythonu. No zbog te iste jednostavnosti XML *launch*

datoteka, one su manje pogodne za kompleksnije sustave, kakvi su često robotski sustavi. U odnosu na to su Python *launch* datoteke fleksibilnije što im omogućuje detaljnije definiranje parametara za pokretanje nekog sustava. Kompleksnost *launch* datoteke možemo smanjiti pisanjem više *launch* datoteka za različite podskupove sustava koje zatim možemo pozivati iz skupne, glavne *launch* datoteke.

3.4. Podržane platforme

Dok je ROS 1 dostupan samo za operacijske sustave temeljene na Linuxu, ROS 2 je dostupan za Linux, Windows i macOS. [13]

3.5. Sažetak razlika

Značajka	ROS 1	ROS 2
Mrežni prijenos podataka	TCP/UDP	DDS
Arhitektura mreže	Otkrivanje preko središnjeg poslužitelja – ROS Master	Peer-to-peer otkrivanje
Životni ciklusi	Nisu podržani	Podržani
Sigurnost	Nema	Podržano šifriranje, kontrola pristupa, itd.
Mogući broj čvorova po procesu	Jedan čvor po procesu	Više čvorova po procesu
Klijentske biblioteke za C++ i Python	roscpp i rospy	rclcpp i rclpy
Arhitektura klijentskih biblioteka	Svaka pisana neovisno o drugoj	Temeljene na zajedničkoj osnovnoj biblioteci (RCL)
Alat za kompiliranje	catkin	colcon
Tipični oblik naredbi	ros[naredba] [parametri]	ros2 [naredba] [parametri]
Formati *launch* datoteka	XML	XML, YAML, Python
Podržane platforme	Linux	Linux, Windows, macOS

4. IMPLEMENTACIJA ALGORITAMA ZA ROS 1 U ROS-U 2

Prijelazom s ROS-a 1 na ROS 2 javila se potreba za korištenjem već postojećih paketa koji još nemaju (ili u početku nisu imali) verziju prilagođenu za korištenje s ROS-om 2. U tu je svrhu razvijen paket *ros1_bridge* koji omogućuje komunikaciju između čvorova ROS-a 1 i ROS-a 2.

Paket *ros1_bridge* radi tako da pokreće čvor u obje verzije ROS-a i time prima poruke iz jedne verzije i objavljuje ih u drugoj. Teme na koje će se pretplatiti i na koje će objavljivati, tj. kako će točno spajati verzije ROS-a, može se odrediti automatski – *dynamic bridge* – ili ručno – *static bridge*.

Automatski način spajanja verzija ROS-a provodi se u nekoliko koraka. Prvo se paketi iz ROS-a 1 koji završavaju sa *_msgs* asociraju s ROS 2 paketima koji završavaju sa *_msgs* ili *_interfaces*. Zatim se spajaju poruke s istim imenom i na kraju se spajaju istovjetna polja unutar poruka. Ako jedna od poruka sadrži polja koja nisu prisutna u drugoj poruci, ta se polja zanemaruju. No kada obje poruke sadrže polja koja nisu prisutna u drugoj poruci, onda se te poruke ne spajaju. *Dynamic bridge* pogodan je za jednostavnije sustave, no kod kompleksnih sustava može degradirati njegove performanse.

Kada određeni spojevi koji bi trebali biti kompatibilni iz nekog razloga ne zadovoljavaju neki od gornjih uvjeta ili želimo uvesti neka dodatna pravila, korisnik može ručno definirati mapiranje poruka. Vlastita pravila definiraju se u YAML datoteci unutar ROS 2 paketa. YAML datoteka s vlastitim pravilima mora biti definirana u *package.xml* i *CMakeLists.txt* datotekama ROS 2 paketa koji će definirati pravila mapiranja. [21]

U nastavku će biti prikazan postupak povezivanja ROS-a 1 i ROS-a 2 na primjerima jednostavnog pošiljatelja i primatelja poruka, te mapiranju prostora u Gazebo simulatoru. Gazebo simulator i simultana lokalizacija i mapiranje detaljnije će biti opisani u sljedećim poglavljima ovog rada.

Iako je za većinu završnog rada korišten ROS 2 *Humble Hawksbill*, u ovom će dijelu biti korišten ROS 2 *Foxy Fitzroy* i ROS 1 *Noetic Ninjemys* na operacijskom sustavu Ubuntu 20.04 LTS. Razlog tome je što su to najnovija izdanja ROS-a s dugoročnom podrškom koja su službeno podržana na istoj verziji operacijskog sustava Ubuntu. Za operacijski sustav Ubuntu 22.04 LTS koji je korišten u ostatku rada ne postoji službeno kompilirano izdanje ROS-a 1.

4.1. *ros1_bridge* za sustav sa čvorovima *talker* i *listener*

Nakon instalacije oba izdanja ROS-a na prije spomenutom operacijskom sustavu, prvo ćemo definirati putanje do instalacijskih direktorija tih izdanja naredbama:

```
$ export ROS1_INSTALL_PATH=/opt/ros/noetic
$ export ROS2_INSTALL_PATH=/opt/ros/foxy
```

Zatim ćemo u ovom prvom prozoru Terminala (nazovimo ga terminal A) pripremiti ROS 1 okruženje i pokrenuti *roscore* koji čini temelj ROS-a 1, tj. pokreće ROS Master, ROS Parameter Server i čvor *rosout* za *logging* poruka, naredbama:

```
$ source /opt/ros/noetic/setup.bash
$ roscore
```

U novom prozoru *Terminala* (terminal B) ponovno pripremamo okruženje za oba izdanja ROS-a i pokrećemo *ros1_bridge* naredbama:

```
$ source /opt/ros/noetic/setup.bash
$ source /opt/ros/foxy/setup.bash
$ export ROS_MASTER_URI=http://localhost:11311
$ ros2 run ros1_bridge dynamic_bridge
```

gdje varijabla okruženja „ROS_MASTER_URI“ govori čvorovima pokrenutim u ROS-u 1 gdje se nalazi ROS Master. Za spajanje ROS-a 1 i ROS-a 2 koristit ćemo *dynamic bridge* koji će automatski spajati kompatibilne čvorove.

U trećem prozoru (terminal C) pokrećemo čvor u ROS-u 1 koji će slati poruke na neku temu. Za to ćemo se poslužiti čvorom *talker* iz pokaznog paketa *rospy_tutorials*:

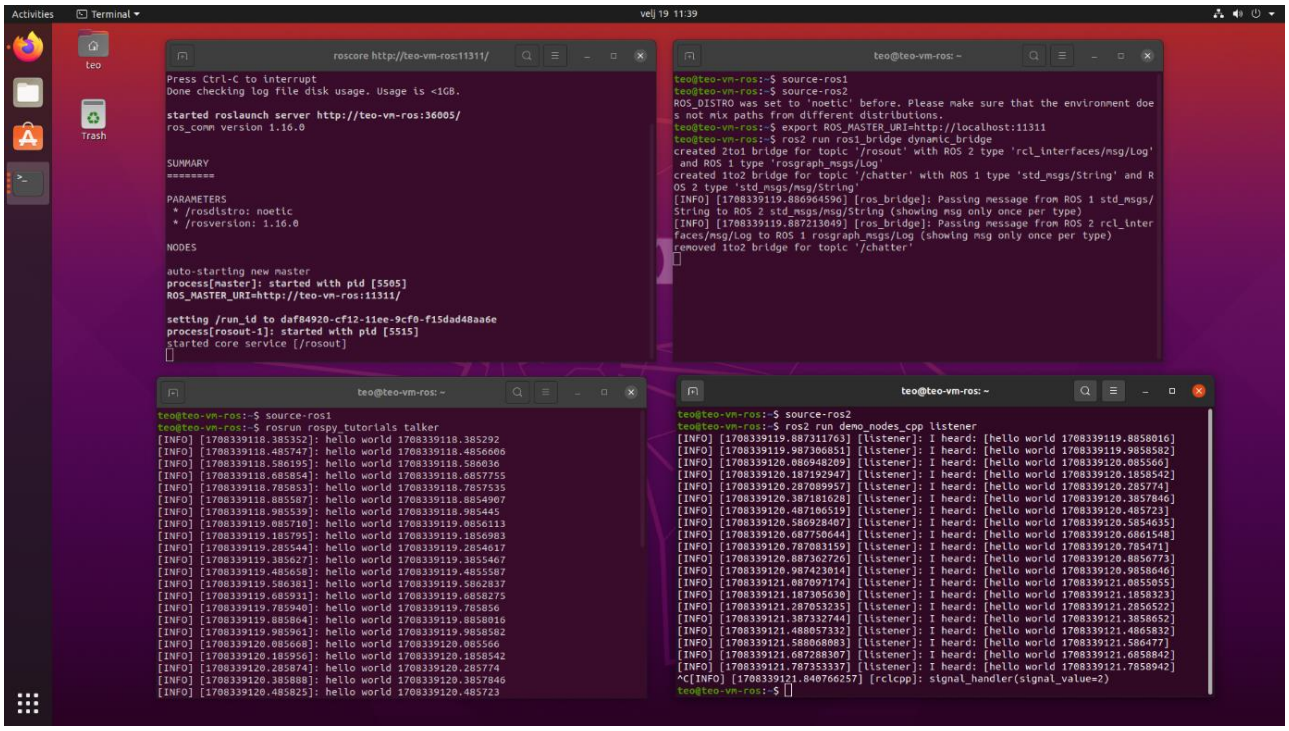
```
$ source /opt/ros/noetic/setup.bash
$ rosrun rospy_tutorials talker
```

čije poruke će preko *ros1_bridgea* primati čvor u ROS-u 2.

Taj čvor koji će primati poruke pokrećemo u četvrtom prozoru (terminal D). Također ćemo se poslužiti pokaznim primjerom *listener* iz paketa *demo_nodes_cpp* koji pokrećemo naredbama:

```
$ source /opt/ros/foxy/setup.bash
$ ros2 run demo_nodes_cpp listener
```

Na sljedećoj je slici prikazan rezultat dosadašnjeg postupka.

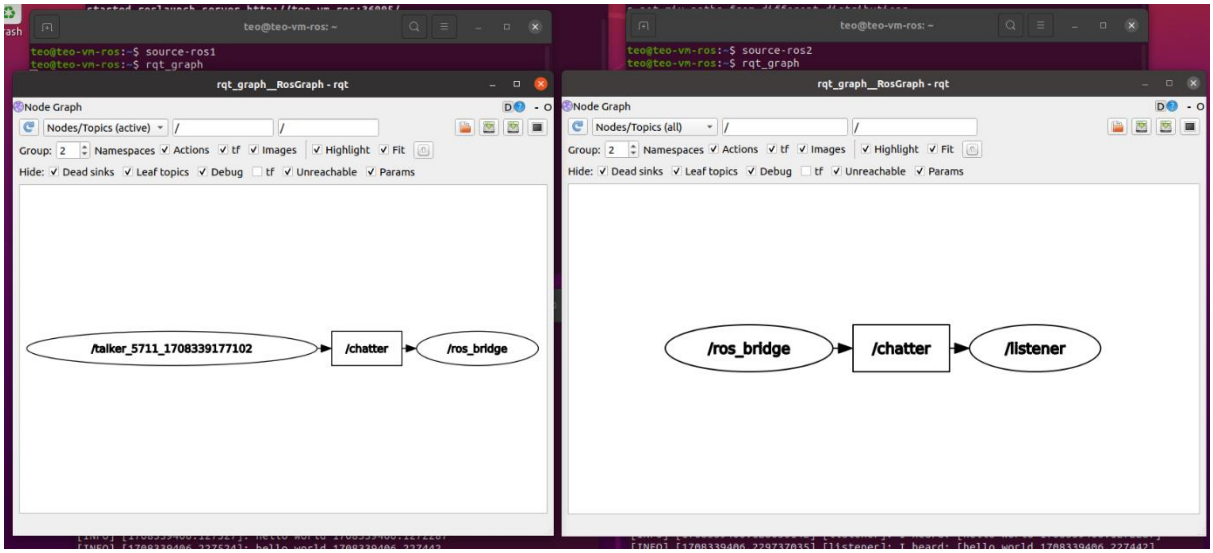


Slika 4. *ros1_bridge* s *talker* i *listener* čvorovima (izvor: autor)

Kako pokrećemo čvorove, tako će se u prozoru u kojem smo pokrenuli *ros1_bridge* ispisivati obavijesti o spajanju čvorova i poruka, npr.

```
created 1to2 bridge for topic '/chatter' with ROS 1 type
'std_msgs/String' and ROS 2 type 'std_msgs/msg/String'
```

što je vidljivo i na računalnom grafikonu prikazanom na sljedećoj slici.



Slika 5. *Computational graph* za prvi *ros1_bridge* primjer (izvor: autor)

Vidimo da čvor */talker* iz ROS-a 1 objavljuje poruke na temu */chatter* na koju je pretplaćen čvor */ros_bridge*, Čvor */ros_bridge* zatim te poruke objavljuje na temu */chatter* u ROS-u 2 na koju je pretplaćen čvor */listener*.

4.2. *ros1_bridge* na primjeru mapiranja prostora

U ovom ćemo primjeru koristiti *ros1_bridge* za mapiranje prostora u Gazebo simulatoru pomoću robota TurtleBot3. Za to će nam bit potrebni paketi stvoreni za korištenje robota TurtleBot3 u ROS-u koji su navedeni i opisani u poglavljima o mapiranju CRTA-e, a detalji o kojima nam trenutno nisu potrebni u ovom poglavlju. Kao prostor koji ćemo mapirati koristit ćemo primjer *turtlebot3_house* koji dolazi uz standardnu instalaciju paketa za TurtleBot3.

U ROS-u 1 pokrenut ćemo paket za simultanu lokalizaciju i mapiranje *turtlebot3_slam*, dok ćemo u ROS-u 2 pokrenuti paket za simulaciju *turtlebot3_gazebo* i paket za upravljanje robotom pomoću tipkovnice *turtlebot3_teleop*.

Prva dva koraka ista su kao i kod prvog primjera, gdje smo ih nazvali terminal A i terminal B. U prvom prozoru smo pokrenuli *roscore* za ROS 1, dok smo u drugom pokrenuli *ros1_bridge* u ROS-u 2.

U trećem ćemo prozoru *Terminala* u ROS-u 2 pokrenuti čvor za simulaciju naredbama:

```
$ source /opt/ros/foxy/setup.bash
$ ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```

gdje je *turtlebot3_house.launch.py* predefinirana *launch* datoteka simulacijskog okruženja koje ćemo koristiti.

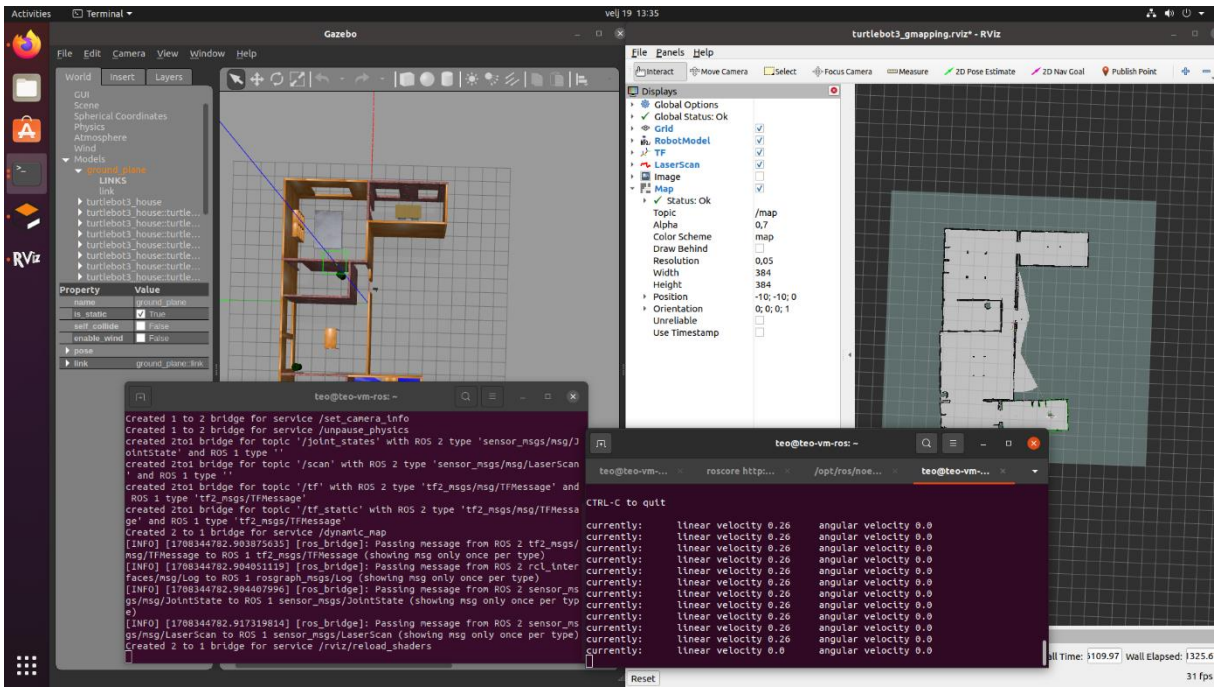
U četvrtom prozoru pokrećemo u ROS-u 1 čvor za mapiranje naredbama:

```
$ source /opt/ros/noetic/setup.bash
$ roslaunch turtlebot3_slam turtlebot3_slam.launch
```

Posljednje nam je potreban čvor za upravljanje robotom pomoću tipkovnice koji ćemo u ROS-u 2 pokrenuti u novom prozoru naredbama:

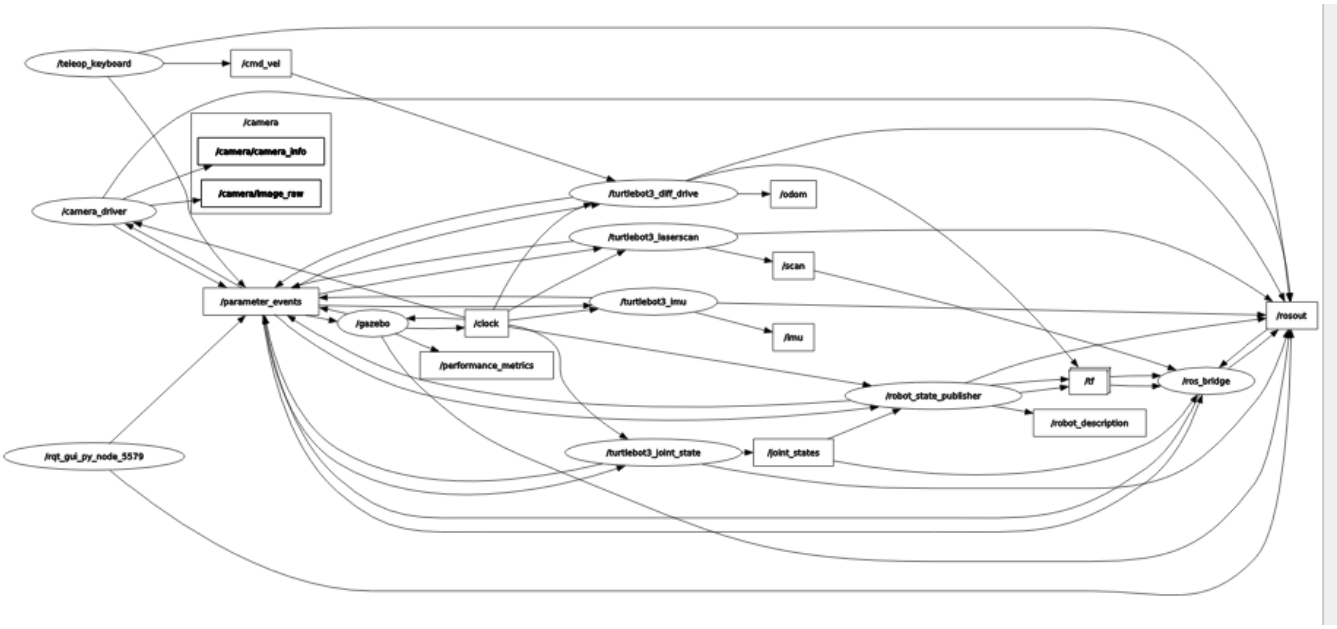
```
$ source /opt/ros/foxy/setup.bash
$ ros2 run turtlebot3_teleop teleop_keyboard
```

U nastavku je prikazan rezultat dosadašnjeg postupka gdje u lijevom prozoru *Terminala* možemo vidjeti obavijesti o spajanju poruka pomoću paketa *ros1_bridge*.



Slika 6. *ros1_bridge* za primjer mapiranja prostora (izvor: autor)

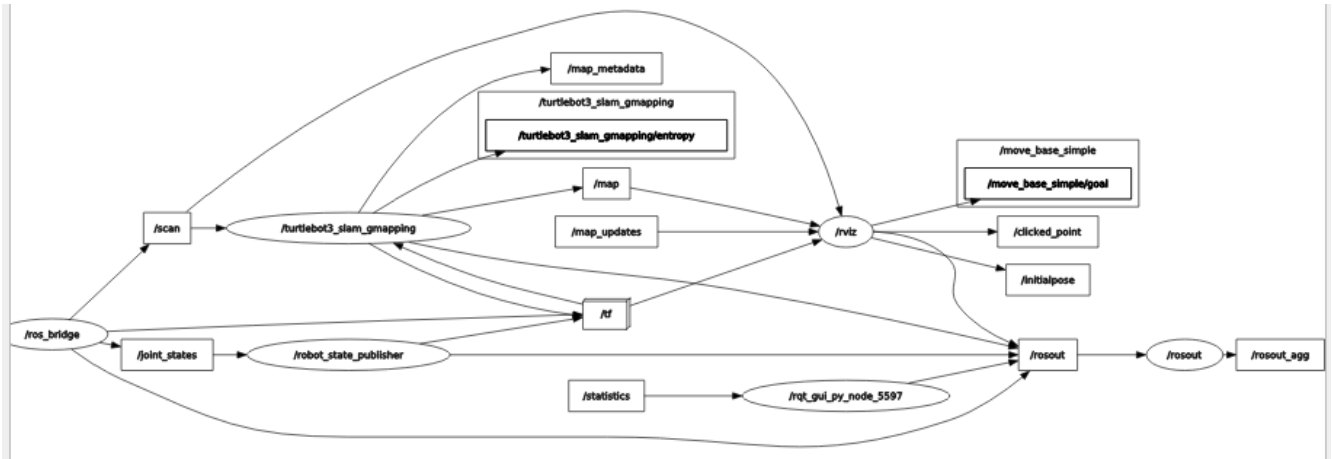
Na sljedećoj je slici prikazan *computational graph* čvorova pokrenutih u ROS-u 2 za ovaj primjer.



Slika 7. *Computational graph* ROS-a 2 za primjer mapiranja pomoću *ros1_bridge (izvor: autor)*

Vidimo da čvor */turtlebot3_laserscan* objavljuje informacije primljene od laserskog senzora na temu */scan* na koju je pretplaćen čvor */ros_bridge*. Uz to, */ros_bridge* pretplaćen je i na temu */tf* od koje prima informacije o transformacijama među *odom* i *base* okvirima (engl. *frame*) robota u simulaciji i druge teme potrebne za izračun pomaka robota u prostoru.

Na slici u nastavku prikazan je *computational graph* čvorova pokrenutih u ROS-u 1 za ovaj primjer.



Slika 8. *Computational graph* ROS-a 1 za primjer mapiranja pomoću *ros1_bridge* (izvor: autor)

Vidimo da čvor */ros_bridge* objavljuje poruke primljene od čvorova pokrenutih u ROS-u 2 na ekvivalentne teme u ROS-u 1. Tako npr. poruke koje je primio od teme */scan* u dijelu sustava pokrenutom u ROS-u 2 objavljuje na ekvivalentnu temu */scan* pokrenutu u ROS-u 1. Na tu je temu onda pretplaćen čvor */turtlebot3_slam_gmapping* koji pomoću primljenih informacija ažurira mapu prostora objavom poruka na temu */map* koju će zatim prikazati RViz. Čvorovi */turtlebot3_slam_gmapping* i */rviz* također primaju i informacije o transformacijama i pomacima koje su preko čvora */ros_bridge* primljene od odgovarajućih tema u ROS-u 2.

5. GAZEBO SIMULATOR

Za potrebe završnog rada korišten je robotski simulator Gazebo. Robotski simulatori su softverska rješenja za virtualno rekreiranje fizičkih svojstava, okruženja i kinematike robota. Simulatori omogućuju brže i jeftinije (ako ne i besplatno) programiranje i testiranje robotskih sustava, a kreirani se sustavi mogu većinom uz malo napora prenijeti na fizičke sustave, tj. u stvarni svijet.

Gazebo je besplatni robotski simulator otvorenog koda koji podržava 2D i 3D okruženja. Standardni je simulator korišten za ROS 1 i ROS 2, no mogu se koristiti i drugi simulatori, kao što je Nvidia Isaac Sim.

Slika 9. Logo Gazebo simulatora [22]

Razvoj Gazeba započeo je 2002. godine. Od 2017. godine razvoj je podijeljen na dvije aplikacije – dosadašnju verziju koja je preimenovana u Gazebo Classic i novi Ignition Gazebo. Gazebo Classic ima monolitnu arhitekturu, što znači da je neovisan o drugim aplikacijama i svi su njegovi sastavni dijelovi ujedinjeni u jednu cjelinu. Ignition Gazebo ima moderniziranu modularnu arhitekturu koja omogućuje jednostavniji razvoj simulacijskih aplikacija s bržim performansama.

S obzirom na stabilnost i proširenu funkcionalnost originalnog izdanja Gazeba, za potrebe završnog rada koristio sam Gazebo Classic, verziju 11. Podržava više pogona za fiziku (engl. *physics engine*), a zadani je ODE. Uz ODE, podržani su i Bullet, Simbody i DART. Pogon za prikaz (engl. *rendering engine*) za sučelje je OpenGL, a za 3D koristi OGRE. [22]

5.1. SDFormat

Za simulaciju objekata i robota u virtualnom okruženju, Gazebo koristi SDFormat datoteke, skraćeno SDF. Puni naziv formata je *Simulation Description Format*. To je vrsta XML datoteke koja opisuje objekte i okruženja za robotska simulacijska rješenja, kao što je i Gazebo u sklopu kojeg je format i razvijen.

U nastavku se nalazi skraćena SDF datoteka modela kocke dimenzija jedan metar.

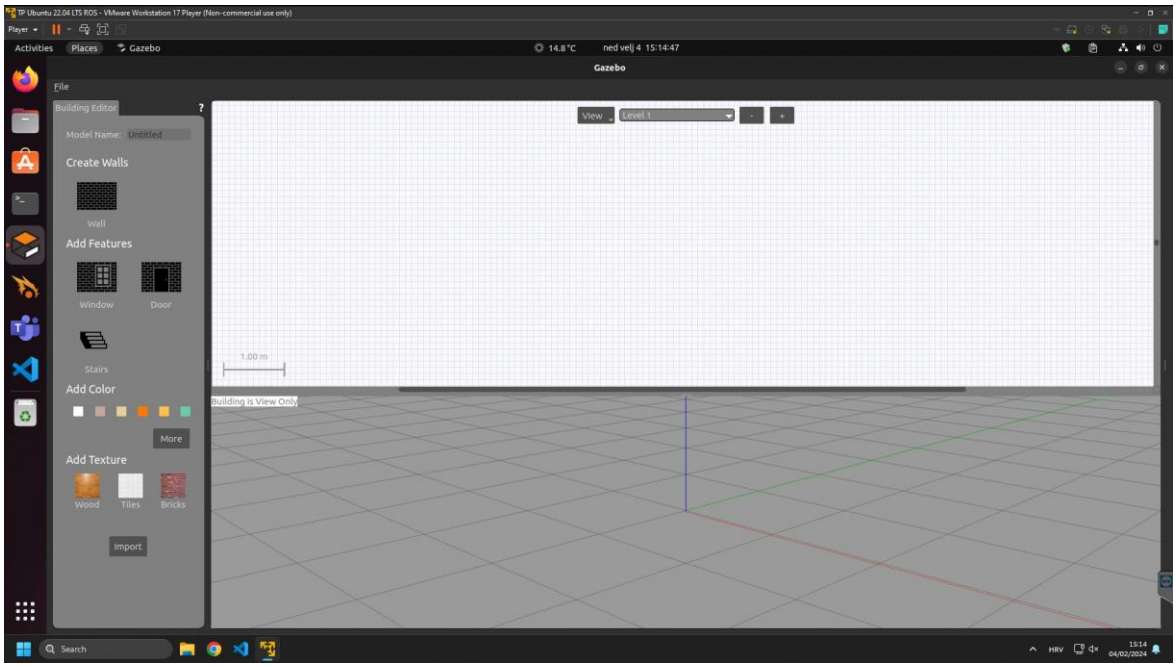
```
<?xml version='1.0'?>
<sdf version='1.7'>
  <model name='kocka'>
    <link name='link_0'>
      <inertial>
        <mass>1</mass>
        <inertia>
...
        </inertia>
      </inertial>
      <pose>0 0 0 0 -0 0</pose>
      <visual name='visual'>
        <pose>0 0 0 0 -0 0</pose>
        <geometry>
          <box>
            <size>1 1 1</size>
          </box>
        </geometry>
        <material>
          <lighting>1</lighting>
          <script>
           <uri>file://media/materials/scripts/gazebo.material</uri>
            <name>Gazebo/Grey</name>
          </script>
          <shader type='pixel'/>
        </material>
        <transparency>0</transparency>
        <cast_shadows>1</cast_shadows>
      </visual>
      <collision name='collision'>
...
      </collision>
    </link>
    <static>1</static>
    <allow_auto_disable>1</allow_auto_disable>
  </model>
</sdf>
```

U ovoj je datoteci definirana geometrija kocke, njene dimenzije, masa, materijal, momenti inercije, itd. Oznakom <*static*> definira se hoće li model biti pokretan ili statičan. Preporuča se sve statične modele označiti kao takve čime se mogu poboljšati performanse kod kompleksnih modela. Bitno je napomenuti da kod generiranja SDF datoteka, Gazebo ponekad definira vlastiti standardni materijal sive boje čime nadjača teksture definirane u Collada modelima. Brisanjem oznake <*material*> Gazebo će ponovno poštivati teksture definirane u modelu.

5.2. Model Editor i Building Editor

Modeli se mogu nabaviti u Gazebovoj vlastitoj knjižnici, no moguće je i koristiti vlastite modele nakon generiranja SDF datoteka za njih. SDF datoteke u Gazebu generiraju se iz tri podržana formata 3D modela – Collada, STL i OBJ. Za generiranje se koristi ugrađeni *Model Editor*.

Uz *Model Editor* postoji i rudimentarni *Building Editor* s osnovnim funkcijama crtanja zidova, umetanja vrata i prozora. S obzirom na kompleksniju strukturu Regionalnog centra izvrsnosti za robotske tehnologije (CRTA), *Building Editor* nije bio pogodan za modeliranje centra.



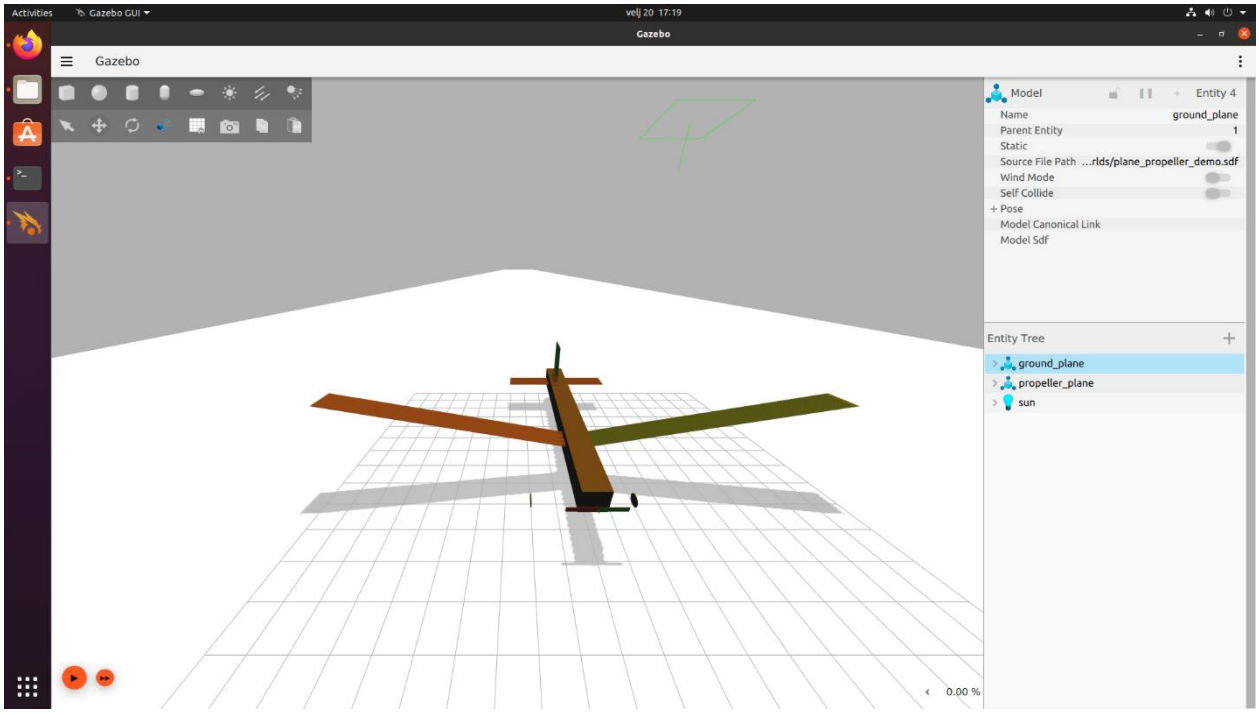
Slika 10. Gazebov *Building Editor* (izvor: autor)

Na vrhu prozora *Building Editora* nalazi se prostor za crtanje tlocrta zgrade, a na dnu prozora prikazat će se 3D model u stvarnom vremenu. Bočni izbornik sadrži opcije za umetanje zidova, prozora, vrata i stepenica, kao i opcije za umetanje tekstura i materijala.

5.3. Usporedba s Ignition Gazebom

Ignition Gazebo, koji je preimenovan u Gazebo nakon migracije s prvog izdanja koje je preimenovano u Gazebo Classic, ima modularnu arhitekturu što omogućuje lakše prilagođavanje simulacijskog sustava potrebama korisnika u odnosu na monolitni Gazebo Classic. Za usporedbu koristit će se najnovije LTS izdanje Ignition Gazeba – *Harmonic*.

Sučelje Ignition Gazeba u potpunosti je promijenjeno i omogućuje uključivanje i isključivanje panela, tj. dijelova sučelja. Na slici u nastavku možemo vidjeti novo sučelje.



Slika 11. Sučelje *Ignition Gazeba* (izvor: autor)

U desnom području prozora pokazivat će se uključeni paneli čiji izbor ovisi o potrebama korisnika.

Pogoni za fiziku javljaju se u obliku dodataka (engl. *plugin*) i sadržani su u modulu Gazebo Physics. Zadani pogon za fiziku u Ignition Gazebu je DART, a može se zamijeniti drugim službeno podržanim pogonima, kao i pogonima za koje korisnici mogu sami napisati dodatke. Neki pogoni, kao što je ODE koji je zadani pogon za fiziku u Gazebu Classic, imaju greške zbog čega još nisu službeno implementirani u novom izdanju.

Pogoni za prikaz također se javljaju u obliku dodataka i sadržani su u modulu Gazebo Rendering. Uz OGRE, sada su podržani i OGRE2 i parcijalno Optix, te bilo koji drugi pogoni za koje korisnici mogu napisati dodatke.

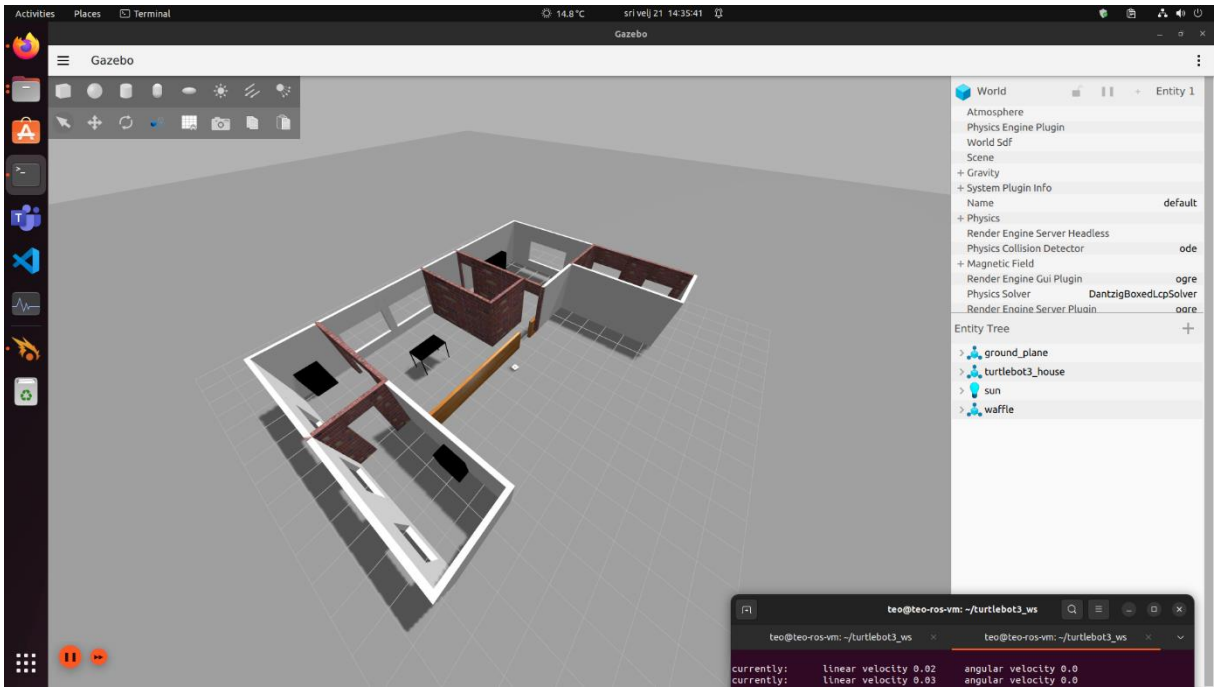
Paketi za spajanje Gazeba i ROS-a sadržani su kod Ignition Gazeba u metapaketu *ros_gz* umjesto paketa *gazebo_ros_pkgs* kod Gazeba Classic, što kod migracije simulacija iz stare u novu verziju zahtijeva promjenu zavisnih paketa (engl. *dependencies*) u datoteci *package.xml* i *launch* datotekama simulacije. Taj metapaket instalira pakete za prijenos poruka između modula *Gazebo Transport* i ROS-a, osnovne *launch* datoteke i izvršne programe (tj. čvorove), primjere i drugo. Komunikaciju kod Gazeba Classic direktno su pružali dodaci u paketu *gazebo_ros_pkgs*, dok Ignition Gazebov paket sadrži općeniti čvor za premošćivanje komunikacije. S toga je kod novog Gazeba potrebno stvoriti novu YAML datoteku u mapi *params* u kojoj će biti definirano koje će se ROS-ove i Gazebove teme spajati. Primjer jednog unosa u toj datoteci dan je u nastavku.

```
- ros_topic_name: "scan"
  gz_topic_name: "scan"
  ros_type_name: "sensor_msgs/msg/LaserScan"
  gz_type_name: "gz.msgs.LaserScan"
  direction: GZ_TO_ROS
```

Svaki unos u toj datoteci sadrži naziv teme i tip poruka u ROS-u, naziv teme i tip poruka u Gazebu i smjer u kojem će se poruke kretati.

SDF datoteke robota i virtualnog svijeta također zahtijevaju promjene kod migracije. Većina virtualnih svjetova napravljenih u Gazebu Classic referenciraju modele *sun* i *ground_plane* koji dolaze s tom verzijom Gazeba. Ignition Gazebo ne sadrži te modele nego ih je potrebno ručno dodati ili uvesti iz nekog repozitorija poput *Gazebo Fuel* pomoću oznaka <*include*>. Kod robota je potrebno zamijeniti stare dodatke za senzore ekvivalentnim dodacima iz nove verzije. [23]

U nastavku je slika simulacije *turtlebot3_house* s robotom TurtleBot3 Waffle migrirane iz Gazeba Classic u Ignition Gazebo. Ta je ista simulacija korištena u poglavlju o paketu *ros1_bridge* u Gazebu Classic.



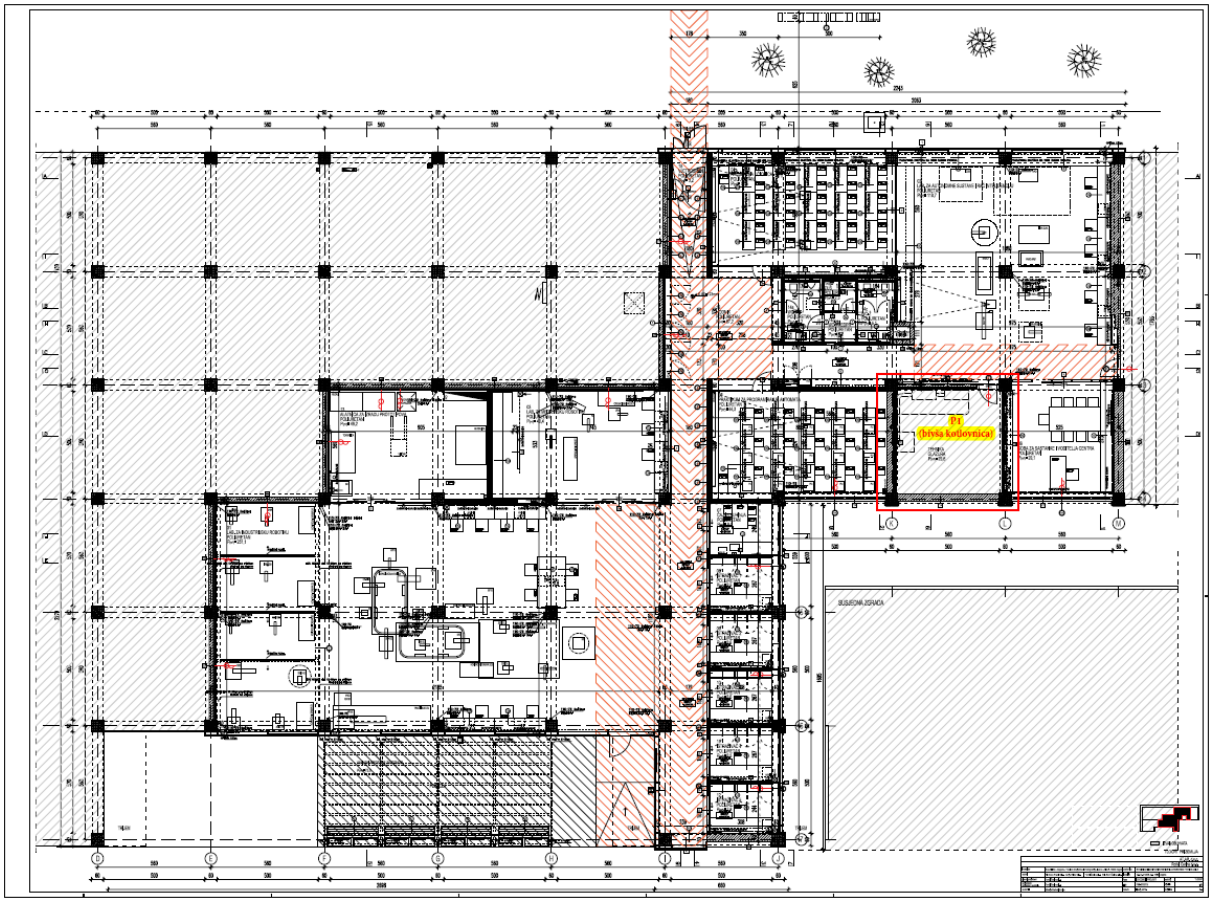
Slika 12. *turtlebot3_house* u Ignition Gazebu

Nedostatak Ignition Gazeba je što kao platformu za pokretanje službeno podržava samo Ubuntu, dok Gazebo Classic podržava i Windows i macOS. Podrška za ostale platforme trenutačno je u aktivnom razvoju u trenutku pisanja ovog rada. Aktivno se radi i na migraciji mnoštva dodataka za senzore, vizualizaciju, *Model Editora* i drugog, a uspješnost ovisi od dodatka do dodatka. [24]

6. SIMULACIJA REGIONALNOG CENTRA IZVRSNOSTI ZA ROBOTSKE TEHNOLOGIJE U GAZEBO SIMULATORU

Dio završnog zadatka je izrada modela Regionalnog centra izvrsnosti za robotske tehnologije (CRTA-e) koji se nalazi u sklopu Fakulteta strojarstva i brodogradnje. Taj će se model koristiti u simulacijskom softveru Gazebo u čijem će se okruženju kretati mobilni robot koji će mapirati prostor centra.

U svrhu izrade modela pružen mi je tlocrt centra u PDF formatu koji se u obliku slike nalazi u nastavku.



Slika 13. Tlocrt CRTA-e (izvor: CRTA)

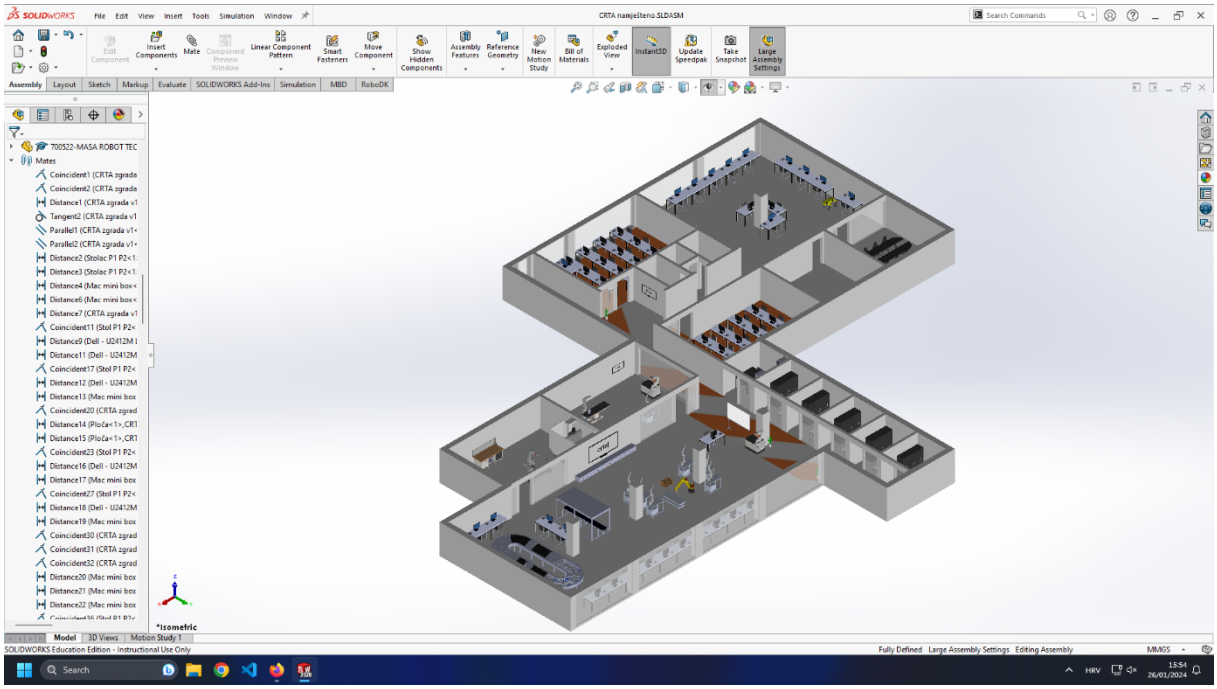
Glavni dio CRTA-e sastoji se od tri laboratorija – Autonomni sustavi (L1), Računalna inteligencija (L2) i Medicinska robotika. Uz to postoje praktikumi korišteni za održavanje nastave i studentski rad, uredi i kuhinja za zaposlenike, alatnica i popratne pomoćne prostorije.

6.1. Model Regionalnog centra izvrsnosti za robotske tehnologije (CRTA)

Model Regionalnog centra izvrsnosti za robotske tehnologije napravljen je u softveru SolidWorks francuske tvrtke Dassault Systèmes, verziji 2020 SP3.

Prvo je modeliran osnovni model CRTA-e koji sadrži zidove, pod i otvore. Zatim su modelirani neki od modela namještaja u istom softveru, dok su kompleksniji modeli (npr. roboti) preuzeti sa stranica proizvođača i servisa *GrabCAD Library* kako bi se uvelike smanjilo vrijeme modeliranja CRTA-e. *GrabCAD Library* je servis na koji korisnici mogu postavljati svoje 3D modele napravljene u bilo kojem softveru za modeliranje i omogućiti drugim korisnicima njihovo besplatno preuzimanje i korištenje. Ovisno o potrebi, nekim su modelima dodane boje i slikovne teksture (engl. *decals*) u SolidWorksu.

Nakon prikupljanja svih potrebnih modela, u SolidWorksu je napravljen sklopni model (engl. *assembly*), prikazan na slici u nastavku.



Slika 14. Model CRTA-e u SolidWorksu (izvor: autor)

Tijekom modeliranja CRTA-e bilo je potrebno voditi računa o limitiranim hardverskim resursima računala na kojem će biti pokrenuta simulacija. Model nije smio biti previše detaljan jer bi došlo do prevelikog usporavanja simulacijskog softvera i samog računala na kojem bi se

on pokretao, što bi dovelo do nemogućnosti mapiranja modela. U obzir također nisu uzeti sanitarni čvorovi i kotlovnica.

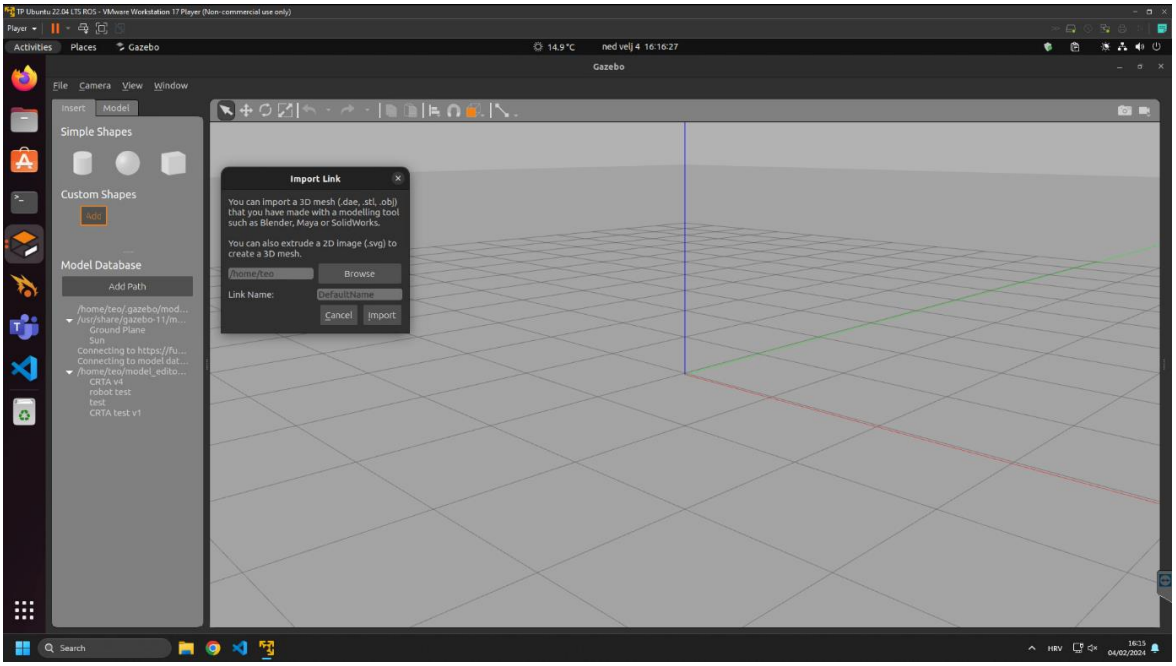
Nakon dovršetka modela, on je iz SolidWorksa iz *Assembly* datoteke (.sldasm) izvezen u STEP formatu (.step), verziji AP214. Ta je verzija STEP-a odabrana jer podržava boje. Nedostatak je što STEP ne podržava slikovne teksture iz SolidWorksa, pa neke ih površine neće zadržati.

STEP model je pomoću softvera CAD Exchanger Lab pretvoren u Collada format (.dae) koji je jedan od podržanih formata u Gazebo simulatoru. Collada format izabran je umjesto STL i OBJ formata jer podržava teksture bez potrebe za povezivanjem s eksternim datotekama za teksture.

6.2. Stvaranje okruženja CRTA-e u Gazebo simulatoru

Kako bismo model CRTA-e mogli umetnuti u novi virtualni svijet (*world*) u Gazebo simulatoru, prvo je potrebno generirati SDF datoteku za njega.

Za stvaranje SDF datoteke iskoristit ćemo funkciju *Model Editor* u Gazebo Simulatoru. Ta nam funkcija omogućuje umetanje vlastitih 3D modela u jednom od tri podržana formata (Collada, STL, OBJ). *Model Editor* unutar svog sučelja pruža i neke osnovne modele koje možemo umetnuti – valjak, kuglu i kocku.

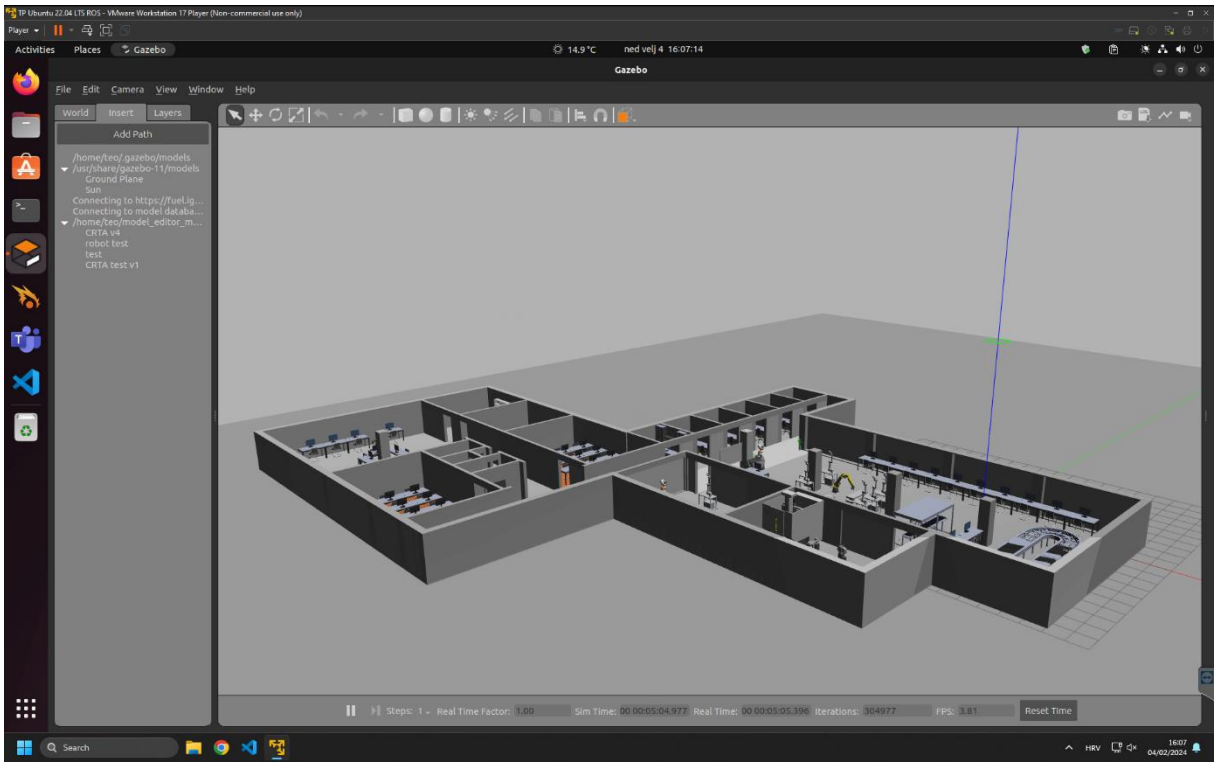


Slika 15. Gazebov *Model Editor* sa sučeljem za umetanje vlastitih modela (izvor: autor)

Umetanjem i spremanjem modela Gazebo generira dvije datoteke – *model.sdf* i *model.config*.

Datoteka *model.config* sadrži metapodatke o modelu, kao što su naziv, verzija i opis modela, naziv SDF datoteke modela, ime i adresa e-pošte autora i slično. Datoteka *model.sdf* opisuje sama svojstva modela – npr. njegovu kinematiku, kolizije, materijal, pozicije, izgled, itd. Direktno referencira Collada, STL ili OBJ model koji je korišten za generiranje SDF-a. [25], [26]

Model se zatim umetne u novi virtualni svijet (*world*). *World* je u Gazebu skup predmeta i robota u određenom okruženju, a uz to opisuje i fiziku, izvore svjetla i druge parametre simuliranog svijeta. [27]



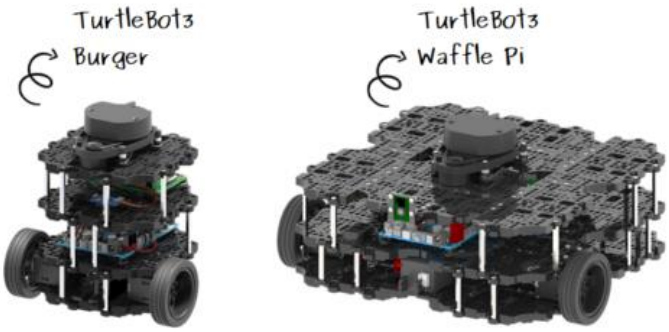
Slika 16. CRTA u novom Gazebo *worldu* (izvor: autor)

Virtualni svijet sprema se s datotečnim nastavkom *.world* što nije novi format datoteke, nego zadržava strukturu SDF datoteke. Razlika između SDF datoteke modela i *worlda* je u tome da prva smije sadržavati samo jedan model, dok druga najčešće sadrži više modela.

7. IMPLEMENTACIJA MOBILNOG ROBOTA I SLAM ALGORITMA

7.1. Općenito o robotu TurtleBot3

Za lokalizaciju i mapiranje korišten je TurtleBot3 Waffle u virtualnom okruženju. TurtleBot3 je edukativni i pristupačni mobilni robot primarno namijenjen korištenju uz ROS, a dostupan je u dva modela – *Burger* i *Waffle Pi*.



Slika 17. TurtleBot3 modeli [28]

U nastavku se nalazi skraćena tablica hardverskih specifikacija tih modela.

Tablica 4. Hardverske specifikacije TurtleBot3 modela

	Burger	Waffle Pi
Maks. translacijska brzina	0,22 m/s	0,26 m/s
Maks. rotacijska brzina	2,84 rad/s	1,82 rad/s
Maksimalni teret	15 kg	30 kg
Dimenzije	138mm x 178mm x 192mm	281mm x 306mm x 141mm
Masa	1 kg	1,8 kg
Računalo	Raspberry Pi	Raspberry Pi
Senzori	LDS-01 ili LDS-02 laserski senzor udaljenosti, 360°	LDS-01 ili LDS-02 laserski senzor udaljenosti, 360°
Kamera	Nema	Raspberry Pi Camera Module v2.1
Ekspanzijski pinovi	GPIO 18-pinski Arduino 32-pinski	GPIO 18-pinski Arduino 32-pinski
Aktuator	XL430-W250	XM430-W210

U ROS-ovom repozitoriju softvera dostupni su paketi i primjeri koji su prilagođeni i za potrebe ovog završnog rada.

7.2. Simultana lokalizacija i mapiranje (SLAM)

Simultana lokalizacija i mapiranje (engl. *Simultaneous Localization and Mapping*, SLAM) je radnja istovremenog stvaranja mape nekog nepoznatog prostora i određivanja vlastitog položaja u tom prostoru. Za rješavanje tog problema postoji više algoritama, npr. filter čestica (engl. *particle filter*) i prošireni Kalman filter (engl. *extended Kalman filter*).

Za simultanu lokalizaciju i mapiranje CRTA-e pomoću robota TurtleBot3 Waffle bit će nam potrebno nekoliko paketa:

- *navigation2,*
- *nav2_bringup,*
- *turtlebot3* i *turtlebot3_simulations* metapaketi i
- *slam_toolbox.*

7.2.1. *navigation2* i *nav2_bringup*

Navigation2 je programski okvir (engl. *framework*) koji omogućuje robotima pogonjenim ROS-om 2 navigaciju kroz kompleksna okruženja.

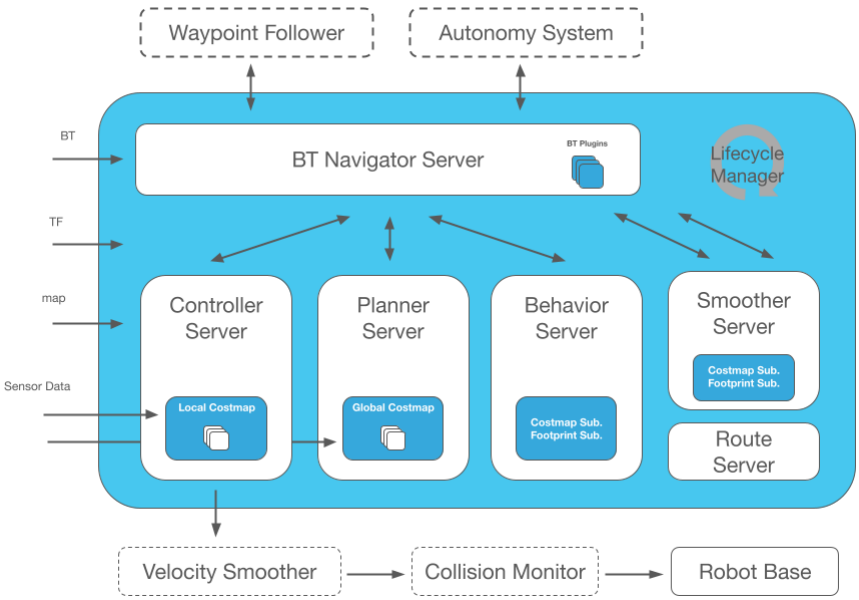
Ovaj paket pruža funkcije percepcije okruženja pomoću raznih vrsta senzora, upravljanje, dinamično planiranje putanje, lokalizaciju u okruženju, vizualizaciju, izbjegavanje kolizija s preprekama i druge značajke važne za razvoj pouzdanih mobilnih autonomnih sustava.

Uz uobičajenu navigaciju od početne do krajnje točke podržava i popratne pozicije (engl. *intermediary pose*, pozicije u kojima se mora nalaziti tijekom navigacije od početne do krajnje pozicije), praćenje nekog pokretnog predmeta, navigaciju s potpunom pokrivenošću (engl. *complete coverage navigation*, navigaciju gdje robot mora pokriti barem jednom svaku točku područja kroz koje navigira) i drugo.

Paket funkcionira pomoću stabala ponašanja (engl. *behavior trees*, BT) koja se sastoje od zadataka koje je potrebno obaviti. Prednost stabala ponašanja je njihova modularnost gdje su

kompleksni zadaci podijeljeni na jednostavnije zadatke koje je lakše implementirati i između kojih je lakše prebacivati se ovisno o situaciji u kojoj se sustav trenutno nalazi. Prednost stabala ponašanja najbolje se vidi u usporedbi sa strojem konačnih stanja (engl. *finite-state machine*, FSM) gdje, kao što i sam naziv kaže, imamo konačni broj točno određenih stanja u kojima se sustav može nalaziti u nekom trenutku. S obzirom na kompleksnost i broj situacija u kojima se mobilni robot može nalaziti, izrada FSM sustava nije vremenski isplativa jer bismo morali opisati velik broj stanja između kojih bi se sustav morao prebacivati.

Ovaj se paket sastoji od glavnog *BT Navigator* poslužitelja i četiri akcijska poslužitelja koje poziva ovisno o trenutnom zadatku – *Controller Server, Planner Server, Behavior Server* i *Smoother Server*.



Slika 18. Arhitektura paketa *navigation2* [29]

Zadatak *Controller Servera* je lokalno planiranje kretnji robota kako bi se slijedila globalno određena putanja. Uz slijeđenje globalne putanje, *Controller Server* može sadržavati i algoritme za druge lokalne zadatke, kao što je upravljanje nekim alatom ili priključivanje na baznu stanicu.

Planner Server računa globalnu putanju potrebnu da bi se došlo do nekog krajnjeg cilja, tj. da bi se obavio neki glavni zadatak. Neki od najčešćih zadataka su računanje najkraćeg puta i računanje putanje kojom će se pokriti sve točke unutar područja (korišteno npr. kod robotskih usisavača).

Behavior Server najčešće se bavi rješavanjem grešaka kod provođenja plana. Na primjer, kod greške percepcijskog sustava može doći do percepcije nepostojećih prepreka zbog čega robot može zapeti u nekom području. *Behavior Server* će očistiti prepreke u *costmapu* i time omogućiti robotu da se pomakne u čisti prostor.

Optimalna globalna putanja koju proračuna *Planner* često sadržava oštra skretanja i neravnu putanju. Zadatak *Smoother Servera* je poboljšati globalnu putanju tako da ublaži nagla skretanja i poveća razmak od prepreka. [30], [29]

Paket *nav2_bringup* je primjer sustava za pokretanje paketa *navigation2*. Korisnici ga slobodno mogu prilagoditi svojim potrebama, ovisno o primjeni. [31]

7.2.2. *turtlebot3* i *turtlebot3_simulations* metapaketi

Metapaket *turtlebot3* instalira pakete:

- *turtlebot3_bringup* – skripte za pokretanje,
- *turtlebot3_description* – 3D modeli robota,
- *turtlebot3_example* – primjeri primjene robota,
- *turtlebot3_navigation* – skripte za pokretanje navigacije,
- *turtlebot3_slam* – skripte za pokretanje SLAM-a i
- *turtlebot3_teleop* – paket za upravljanje robotom pomoću tipkovnice. [32]

Metapaket *turtlebot3_simulations* instalira pakete:

- *turtlebot3_fake* – pokreće lažni čvor za testiranje u RViz-u bez pravog robota i
- *turtlebot3_gazebo* – paket za simulaciju robota u Gazebu. [33]

7.2.3. *slam_toolbox*

Paket *slam_toolbox* sadrži alate za 2D simultano lokaliziranje i mapiranje. Uz uobičajeno stvaranje i spremanje mape prostora, podržava i remapiranje, prekidanje i nastavljanje mapiranja iz spremljenog grafa poza, sinkrono i asinkrono mapiranje, itd. Koristit će se Karto algoritam.

Slam_toolbox pretplaćuje se na teme */scan* i */tf*, a objavljuje na teme */map* i */pose*. Cijeli računalni grafikon (engl. *computational graph*) bit će prikazan u daljnjem tekstu. [34]

7.3. Implementacija robota i SLAM-a u modelu CRTA-e

Nakon što su instalirani svi potrebni paketi može se započeti s implementacijom robota i SLAM-a za mapiranje virtualnog modela CRTA-e u Gazebu.

Za svaki novi prozor *Terminala* kroz koji ćemo pokrenuti sve potrebne pakete moramo pomoću naredbe *source* i ROS-ove konfiguracijske skripte *setup.bash* postaviti varijable okruženja potrebne za funkcioniranje ROS-a. Uz to, moramo i postaviti varijablu okruženja *TURTLEBOT3_MODEL* koja određuje koji ćemo od prethodno navedenih modela robota koristiti.

Ovaj postupak možemo i automatski pokrenuti tako da obje potrebne naredbe dodamo u datoteku *.bashrc* koja je konfiguracijska datoteka *Terminala* u operacijskom sustavu Ubuntu. Za to se možemo poslužiti naredbama:

```
$ echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
```

i

```
$ echo "export TURTLEBOT3_MODEL=waffle" >> ~/.bashrc
```

Naredba *echo* će naredbe u navodnicima dodati na kraj skrivene datoteke *.bashrc* koja se nalazi u *Home* direktoriju trenutno prijavljenog korisnika. Jednom kada unesemo gornje dvije naredbe, više ih ne trebamo unositi kod otvaranja novih prozora *Terminala*. U prvoj naredbi „*humble*“, ako se razlikuju, trebamo zamijeniti odgovarajućom verzijom ROS-a 2 koju smo instalirali, a „*waffle*“ modelom robota TurtleBot3 koji želimo koristiti.

Prva naredba koju ćemo pokrenuti je:

```
$ ros2 launch turtlebot3_gazebo turtlebot3_crta_v4.launch.py
```

koja pokreće simulaciju CRTA-e i Turtlebot3 Waffle robota u Gazebo simulatoru. Parametri su definirani u *launch* datoteci „*turtlebot3_crta_v4.launch.py*“, a dio nje istaknut je ispod.

```
…
    use_sim_time = LaunchConfiguration('use_sim_time',
default='true')
    x_pose = LaunchConfiguration('x_pose', default='0')
    y_pose = LaunchConfiguration('y_pose', default='0')

    world = os.path.join(
        get_package_share_directory('turtlebot3_gazebo'),
        'worlds',
        'CRTA_world_v4.world'
    )
…
```

U *launch* datoteci *„turtlebot3_crta_v4.launch.py“* definirano je da se za simulaciju koristi virtualno okruženje definirano u datoteci *CRTA_world_v4.world* i da se u tom okruženju postavlja TurtleBot3 Waffle u točki (0,0) koordinatnog sustava svijeta, tj. u njegovom ishodištu koje je definirano tijekom generiranja *world* datoteke.

Zatim se naredbom:

```
$ ros2 launch nav2_bringup navigation_launch.py use_sim_time:=True
```

pokreće navigacijski stog (engl. *navigation stack*) i definiraju se parametri akcijskih poslužitelja opisanih u poglavlju *7.2.1 navigation2 i nav2_bringup*. Parametrom *use_sim_time* definira se hoće li se koristiti interni sat simulacije, tj. Gazeba. Kada bismo pokretali na stvarnom robotu, taj parametar bismo postavili na *false*.

Sljedeće pokrećemo *slam_toolbox* naredbom:

```
$ ros2 launch slam_toolbox online_async_launch.py use_sim_time:=True
```

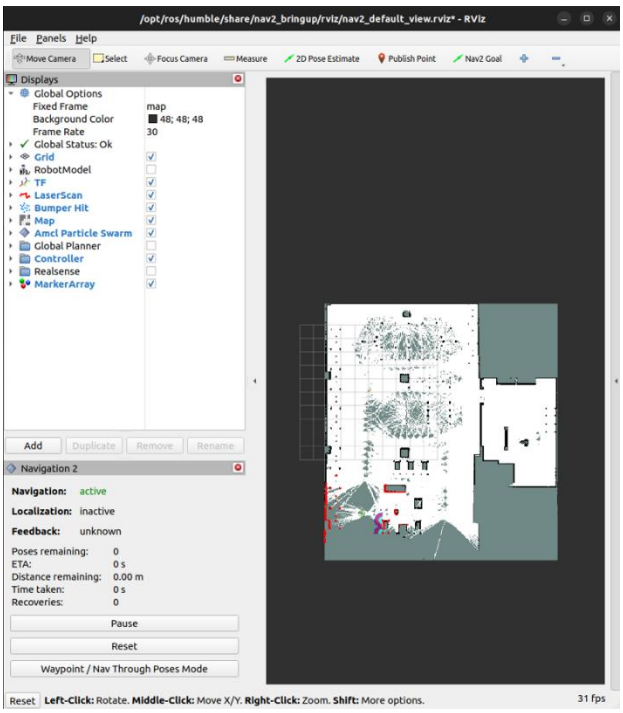
gdje smo odabirom predefinirane *launch* datoteke odredili način na koji će se provoditi simultana lokalizacija i mapiranje. *Online* znači da će se SLAM provoditi tijekom rada robota čime ćemo odmah vidjeti mapu koja se generira, ali će kvaliteta te mape biti nešto niža jer istovremeno izbjegavamo kolizije i upravljamo robotom. Za razliku od toga, kod *offline* SLAM-

a kvaliteta mape će biti nešto bolja jer će se ona generirati nakon što se robot više ne kreće i ne primamo nove informacije od senzora. *Async* označuje asinkroni SLAM, što znači da će se u trenutku ažuriranja mape koristiti najnovije informacije primljene od senzora, a informacije primljene između dva ažuriranja zanemariti. Time možemo „uštedjeti“ na hardverskim resursima kada su oni ograničeni, ali može doći do nedosljednosti između stvarnog stanja i prikaza na mapi, npr. pomak neke prepreke u odnosu na njenu stvarnu lokaciju. S druge strane, sinkroni SLAM će obraditi sve primljene informacije čime će mapa biti točnija, ali koristi više hardverskih resursa i obrada podatka neizbježno će zaostati za brzinom pristizanja novih informacija. *Slam_toolbox* sadrži i dodatak za *RViz* u kojem se tijekom *online* SLAM-a može pauzirati obrada novih informacija sa senzora kako bi se stigle obraditi dosad pristigle informacije.

Kako bismo vidjeli mapu koja se stvara pokrenut ćemo RViz naredbom:

```
$ ros2 run rviz2 rviz2 -d
/opt/ros/humble/share/nav2_bringup/rviz/nav2_default_view.rviz
```

sa standardnom konfiguracijom definiranom u datoteci *nav2_default_view.rviz*. Uređivanjem te datoteke ili stvaranjem vlastite može prilagoditi prikaz svoj potrebama, npr. isključivanjem prikaza *Global Plannera* kao što je to učinjeno na donjoj slici.

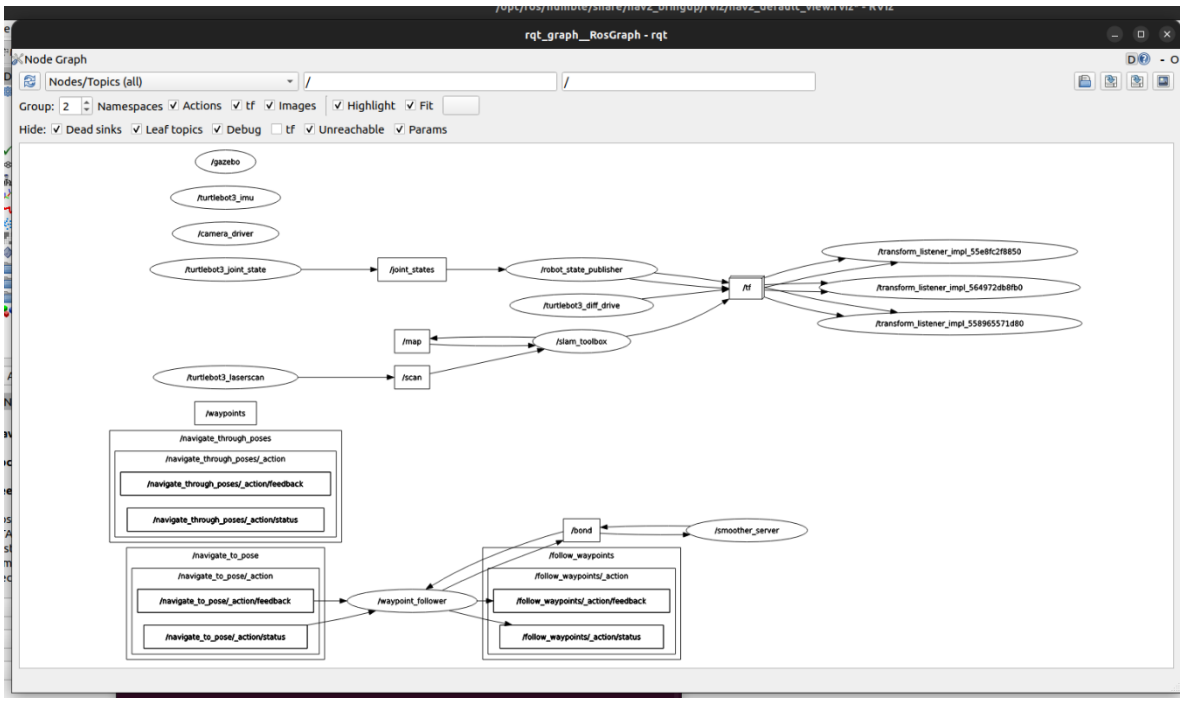


Slika 19. Mapa u RVizu (izvor: autor)

Posljednje ćemo pokrenuti čvor za upravljanje robotom pomoću tipkovnice iz prethodno instaliranog paketa *turtlebot3_teleop* naredbom:

```
$ ros2 run turtlebot3_teleop teleop_keyboard
```

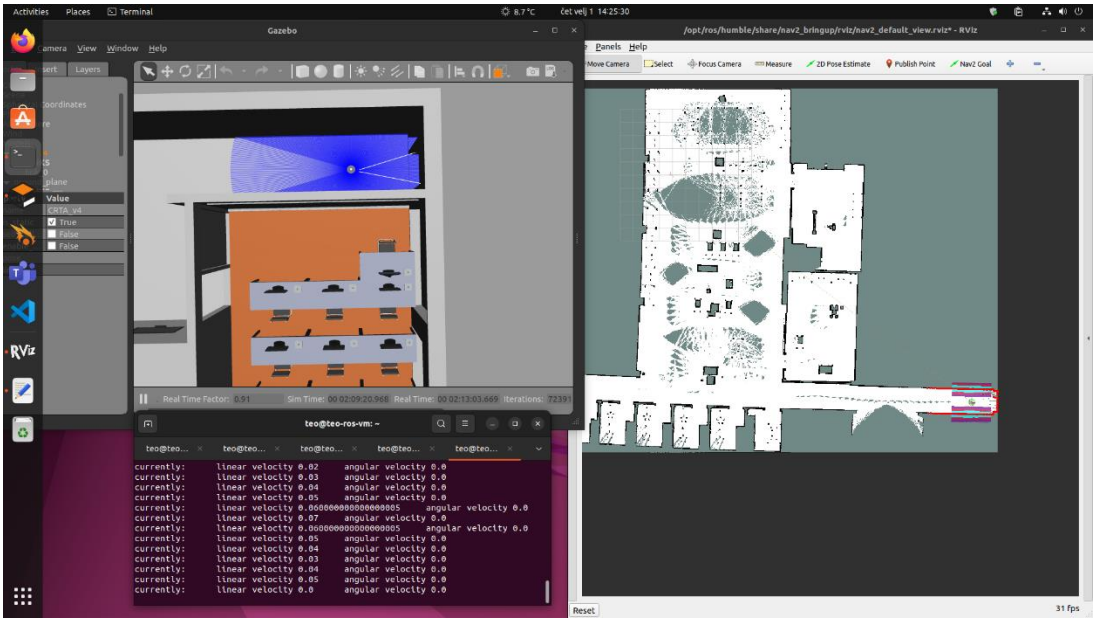
Pokretanjem svih potrebnih čvorova možemo započeti lokalizaciju i mapiranje. Naredbom *rqt_graph* možemo vizualizirati *computational graph* pokrenute simulacije.



Slika 20. *Computational graph* SLAM-a CRTA-e (izvor: autor)

Iz *computational grapha* vidimo da čvor */turtlebot_laserscan* objavljuje informacije primljene od senzora na temu */scan* na koju se pretplaćuje čvor */slam_toolbox*. Čvor */slam_toolbox* obradom primljenih informacija ažurira mapu objavom poruka na temu */map*.

Kretanjem robota kroz virtualno okruženje laserski senzor primat će nove informacije čime će se postepeno generirati mapa prostora, tj. mreža zauzetosti (engl. *occupancy grid*). Na sljedećoj slici prikazan je jedan trenutak u postupku SLAM-a.



Slika 21. Postupak mapiranja CRTA-e (izvor: autor)

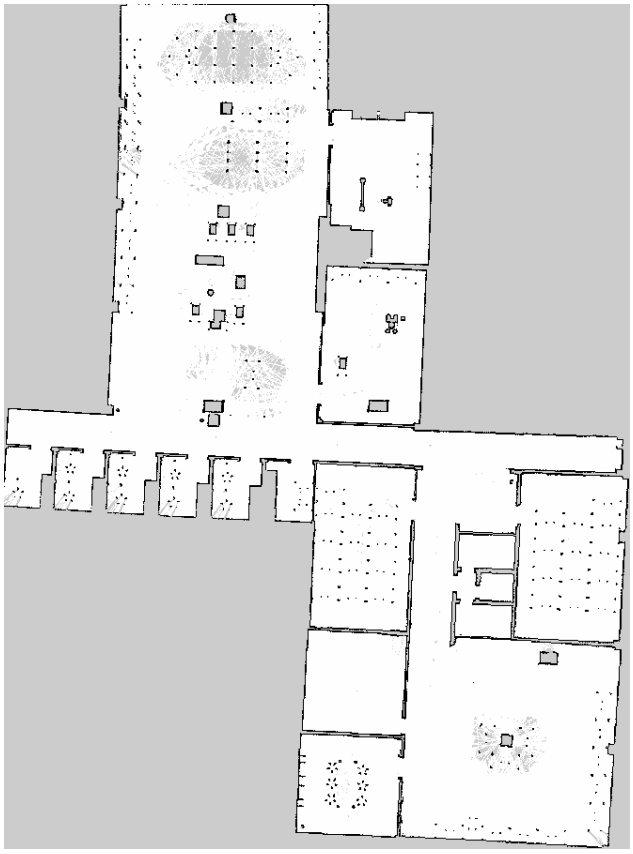
Prolazom kroz cijelo virtualno okruženje dobivamo konačnu mapu. Kroz neka područja potrebno je proći više puta kako bi se popunile sve rupe u mapi koje nastaju kada laserski senzor s ograničenim dometom ne naiđe na prepreku.

Mapu spremamo naredbom:

```
$ ros2 run nav2_map_server map_saver_cli -f crta_map_v2
```

gdje je posljednji parametar naziv, tj. putanja gdje želimo mapu spremiti. Ako navedemo samo naziv mape bez pune putanje ona će se spremiti u korisnikov *Home* direktorij.

Konačno dobivena mapa prikazana je u nastavku.



Slika 22. Konačna mapa CRTA-e (izvor: autor)

Spremanjem mape stvaraju se dvije datoteke, jedna s datotečnim nastavkom *.pgm* i druga s nastavkom *.yaml.* Prva datoteka je sama mapa u kojoj je bijelom bojom prikazan slobodan prostor, crnom bojom prepreka, a sivom bojom nepoznati prostor. Nepoznati prostor većinom se nalazi unutar zatvorenih crnih kontura prepreka. PGM je kratica za *Portable Gray Map*, slikovnu datoteku u kojoj svaki piksel može predstavljati sivu boju od 8 ili 16 bita. Druga datoteka sadrži metapodatke o mapi, kao što su naziv *.pgm* datoteke gdje su podaci mape, njezino ishodište, rezoluciju, itd. U nastavku se nalazi *.yaml* datoteka generirane mape.

```
image: crta_map_v2.pgm
mode: trinary
resolution: 0.05
origin: [-37.6, -25.1, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

Konačna mapa zatim se može koristiti za navigiranje virtualnim okruženjem pomoću *navigation2* i RViza u kojem se može odabrati ciljana poza robota.

8. ZAKLJUČAK

U prvom dijelu ovog rada opisana je arhitektura ROS-a 2 koja je modularna i robusna što omogućuje primjenu ove platforme u proizvodnim okruženjima i industriji. Jedna od najvećih prednosti ROS-a je njegova zajednica koja razvija i daje u otvoreni pristup mnoštvo rješenja za različite probleme s kojima se susreću i početnici i iskusni robotičari. ROS-ova zajednica predstavlja neprocjenjivu vrijednost ovoj platformi i bez nje ona ne bi zaživjela.

Nakon toga opisane su razlike između ROS-a 1 i ROS-a 2 u kojem smo zaključili da je ROS 1 više fokusiran na istraživačko polje i razvoj same platforme. ROS 1 mogli bismo nazvati i dokazom koncepta (engl. *proof-of-concept*) koji pokazuje da postoji potražnja za takvim otvorenim sustavom s bujnom zajednicom. ROS 2 s druge strane poboljšava temelje uspostavljene ROS-om 1 kako bi se mogao koristiti u realnom sektoru, tj. industriji, čime se u zajednicu dovode i mnoge tvrtke s velikim resursima koje također počinju pridonositi platformi.

U posljednjem dijelu rada pokazano je kako se ROS 2 može iskoristiti za rješavanje jedne od najčešćih primjena robotike – mapiranje nekog prostora u svrhu korištenja prikupljenih podataka za autonomno kretanje i obavljanje nego zadatka. To je urađeno u virtualnom okruženju pomoću simulacijskog softvera Gazebo čija je primjena u današnje vrijeme usko vezana uz ROS, iako to nije jedina njegova primjena.

Zaključak rada je da ROS 2 uvelike olakšava razvoj vlastitog robotskog sustava i da je ta platforma spremna za primjenu izvan okvira akademske zajednice.

LITERATURA

[1] F. M. Rico, *A concise introduction to robot programming in ROS2*, First edition. Boca Raton: CRC Press, 2023.

[2] „Nodes — ROS 2 Documentation: Iron documentation“. Pristupljeno: 02. kolovoza 2023. [Na internetu]. Dostupno na: https://docs.ros.org/en/iron/Concepts/Basic/About-Nodes.html

[3] „Topics — ROS 2 Documentation: Iron documentation“. Pristupljeno: 02. kolovoza 2023. [Na internetu]. Dostupno na: https://docs.ros.org/en/iron/Concepts/Basic/About-Topics.html

[4] „Services — ROS 2 Documentation: Iron documentation“. Pristupljeno: 02. kolovoza 2023. [Na internetu]. Dostupno na: https://docs.ros.org/en/iron/Concepts/Basic/About-Services.html

[5] „Actions — ROS 2 Documentation: Iron documentation“. Pristupljeno: 02. kolovoza 2023. [Na internetu]. Dostupno na: https://docs.ros.org/en/iron/Concepts/Basic/About-Actions.html

[6] „Creating a package — ROS 2 Documentation: Humble documentation“. Pristupljeno: 05. veljače 2024. [Na internetu]. Dostupno na: https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html#customize-package-xml

[7] „Organizing Files and Folders Inside a ROS 2 Package – Automatic Addison“. Pristupljeno: 05. veljače 2024. [Na internetu]. Dostupno na: https://automaticaddison.com/organizing-files-and-folders-inside-a-ros-2-package/

[8] „ROS 2 developer guide — ROS 2 Documentation: Humble documentation“. Pristupljeno: 05. veljače 2024. [Na internetu]. Dostupno na: https://docs.ros.org/en/humble/The-ROS2-Project/Contributing/Developer-Guide.html#filesystem-layout

[9] „Using Python, XML, and YAML for ROS 2 Launch Files — ROS 2 Documentation: Foxy documentation“, ROS2 Documentation. Pristupljeno: 04. travnja 2023. [Na internetu]. Dostupno na: https://docs.ros.org/en/foxy/How-To-Guides/Launch-file-different-formats.html

[10] „Client libraries — ROS 2 Documentation: Iron documentation“. Pristupljeno: 31. srpnja 2023. [Na internetu]. Dostupno na: https://docs.ros.org/en/iron/Concepts/Basic/About-Client-Libraries.html

[11] „Discovery — ROS 2 Documentation: Iron documentation“. Pristupljeno: 02. kolovoza 2023. [Na internetu]. Dostupno na: https://docs.ros.org/en/iron/Concepts/Basic/About-Discovery.html

[12] A. Koubâa, Ur., *Robot Operating System (ROS): The Complete Reference (Volume 5)*. u Studies in computational intelligence, no. 895. Cham: Springer, 2021.

[13] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, i W. Woodall, „Robot Operating System 2: Design, architecture, and uses in the wild“, *Sci. Robot.*, sv. 7, izd. 66, svi. 2022, doi: 10.1126/scirobotics.abm6074.

[14] „Quality of Service settings — ROS 2 Documentation: Rolling documentation“. Pristupljeno: 28. srpnja 2023. [Na internetu]. Dostupno na: https://docs.ros.org/en/rolling/Concepts/Intermediate/About-Quality-of-Service-Settings.html

[15] „Different ROS 2 middleware vendors — ROS 2 Documentation: Iron documentation“. Pristupljeno: 28. srpnja 2023. [Na internetu]. Dostupno na: https://docs.ros.org/en/iron/Concepts/Intermediate/About-Different-Middleware-Vendors.html

[16] W. Woodall, „ROS on DDS“. Pristupljeno: 28. srpnja 2023. [Na internetu]. Dostupno na: https://design.ros2.org/articles/ros_on_dds.html

[17] „REP 2000 -- ROS 2 Releases and Target Platforms (ROS.org)“. Pristupljeno: 31. srpnja 2023. [Na internetu]. Dostupno na: https://www.ros.org/reps/rep-2000.html#abstract

[18] „demos/lifecycle/README.rst at iron · ros2/demos“, GitHub. Pristupljeno: 28. srpnja 2023. [Na internetu]. Dostupno na: https://github.com/ros2/demos/blob/iron/lifecycle/README.rst

[19] „Managed nodes“. Pristupljeno: 28. srpnja 2023. [Na internetu]. Dostupno na: http://design.ros2.org/articles/node_lifecycle.html

[20] „Using Fast DDS Discovery Server as discovery protocol [community-contributed] — ROS 2 Documentation: Humble documentation“. Pristupljeno: 03. veljače 2024. [Na internetu]. Dostupno na: https://docs.ros.org/en/humble/Tutorials/Advanced/Discovery-Server/Discovery-Server.html

[21] „ros1_bridge/doc/index.rst at master · ros2/ros1_bridge“, GitHub. Pristupljeno: 06. prosinca 2023. [Na internetu]. Dostupno na: https://github.com/ros2/ros1_bridge/blob/master/doc/index.rst

[22] „Gazebo“. Pristupljeno: 04. veljače 2024. [Na internetu]. Dostupno na: https://classic.gazebosim.org/

[23] „Gazebo - Docs: Migration from ROS 2 Gazebo Classic“. Pristupljeno: 21. veljače 2024. [Na internetu]. Dostupno na: https://gazebosim.org/docs/harmonic/migrating_gazebo_classic_ros2_packages

[24] „Gazebo - Docs: Feature Comparison“. Pristupljeno: 20. veljače 2024. [Na internetu]. Dostupno na: https://gazebosim.org/docs/harmonic/comparison#physics

[25] „Gazebo : Tutorial : Model structure and requirements“. Pristupljeno: 04. veljače 2024. [Na internetu]. Dostupno na: https://classic.gazebosim.org/tutorials?tut=model_structure&cat=build_robot#ModelConfig

[26] „SDFormat Home“. Pristupljeno: 04. veljače 2024. [Na internetu]. Dostupno na: http://sdformat.org/

[27] „Gazebo : Tutorial : Building a world“. Pristupljeno: 04. veljače 2024. [Na internetu]. Dostupno na: https://classic.gazebosim.org/tutorials?tut=build_world

[28] „ROBOTIS e-Manual“, ROBOTIS e-Manual. Pristupljeno: 05. veljače 2024. [Na internetu]. Dostupno na: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/

[29] „Nav2 — Nav2 1.0.0 documentation“. Pristupljeno: 06. veljače 2024. [Na internetu]. Dostupno na: https://navigation.ros.org/

[30] „Navigation Concepts — Nav2 1.0.0 documentation“. Pristupljeno: 06. veljače 2024. [Na internetu]. Dostupno na: https://navigation.ros.org/concepts/index.html#behavior-trees

[31] „navigation2/nav2_bringup/README.md at main · ros-planning/navigation2 · GitHub“. Pristupljeno: 07. veljače 2024. [Na internetu]. Dostupno na: https://github.com/ros-planning/navigation2/blob/main/nav2_bringup/README.md

[32] „GitHub - ROBOTIS-GIT/turtlebot3: ROS packages for Turtlebot3“. Pristupljeno: 07. veljače 2024. [Na internetu]. Dostupno na: https://github.com/ROBOTIS-GIT/turtlebot3/

[33] „ROBOTIS-GIT/turtlebot3_simulations“. ROBOTIS, 24. siječnja 2024. Pristupljeno: 07. veljače 2024. [Na internetu]. Dostupno na: https://github.com/ROBOTIS-GIT/turtlebot3_simulations

[34] S. Macenski, „SteveMacenski/slam_toolbox“. 08. veljače 2024. Pristupljeno: 08. veljače 2024. [Na internetu]. Dostupno na: https://github.com/SteveMacenski/slam_toolbox

PRILOZI

GitHub: https://github.com/teoopr/turtlebot3_simulations