Integracija robotskog operativnog sustava u IsaacSim virtualnom okruženju

Maček, Marija

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Marija Maček

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

Doc. dr. sc. Marko Švaco, mag.ing. Marija Maček

Zagreb, 2023.

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

Zahvaljujem se mentoru doc.dr.sc. Marku Švaci na zadanoj temi i asistentu Branimiru Ćaranu na usmjeravanju i savjetima tijekom pisanja rada.

Zahvaljujem se obitelji i prijateljima na podršci tijekom studiranja.

Marija Maček

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: **Marija Maček** JMBAG: **0035219430**

Naslov rada na hrvatskom jeziku: **Integracija robotskog operativnog sustava u IsaacSim virtualnom okruženju**

Naslov rada na engleskom jeziku: **Integration of the Robot Operating System in the IsaacSim virtual environment**

Opis zadatka:

NVIDIA Isaac Sim, koju pokreće Omniverse, skalabilna je aplikacija za simulaciju robotskih sustava i alat za generiranje umjetnih podataka koji pokreće fotorealistična, fizički precizna virtualna okruženja za razvoj, testiranje i upravljanje robotima temeljenima na umjetnoj inteligenciji. Isaac Sim sadrži mogućnost integracije robotskog operativnog sustava (ROS) za upravljanje virtualnim robotima.

U sklopu ovog rada potrebno je:

- Napraviti usporedbu simulatora Gazebo i Isaac Sim.
- Ostvariti komunikaciju iz robotskog operativnog sustava (ROS) prema Isaac Sim simulatoru.
- U Isaac Sim simulatoru što realnije rekreirati Regionalni centar izvrsnosti za robotske tehnologije (CRTA).
- Implementirati model mobilnog robota diferencijalne kinematike unutar simulatora te ga upravljati pomoću ROS-a.
- Implementirati algoritam mapiranja te ga testirati u Isaac Sim simulatoru na kreiranom svijetu Regionalnog centra izvrsnosti za robotske tehnologije.

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

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
30. 11. 2022.	**1. rok:** 20. 2. 2023. **2. rok (izvanredni):** 10. 7. 2023. **3. rok:** 18. 9. 2023.	**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 OZNAKA

Oznaka	Jedinica	Opis
$\dot{x}$	m/s	Brzina robota u smjeru osi x
$\dot{y}$	m/s	Brzina robota u smjeru osi y
$\dot{\theta}$	rad/s	Rotacijska brzina robota u x-y
v	m/s	Linearna brzina robota
ω	rad/s	Kutna brzina robota
θ	rad	Orijentacija robota u x-y
r	m	Polumjer kotača
L	m	Udaljenost između kotača

SAŽETAK

Zadatak ovog rada je integracija robotskog operativnog sustava (ROS) u IsaacSim virtualnom okruženju. U prvom dijelu rada dan je pregled simulacijskih okruženja IsaacSim i Gazebo te njihova usporedba. Zatim su objašnjeni principi ROS-a i njegova integracija u virtualnim okruženjima, konkretno IsaacSim-u. Nakon toga dan je pregled mobilnih robota i njihovih primjena.

U radu je stavljen fokus na integraciju ROS-a u IsaacSim na način da se iz ROS čvorova upravlja mobilnim robotom diferencijalne kinematike. Na mobilnog robota dodani su potrebni senzori za mapiranje prostora. Implementacijom algoritma mapiranja izrađena je karta okupiranosti Regionalnog centra izvrsnosti za robotske tehnologije (CRTA) čiji model je napravljen u Autodesk Inventor-u.

Ključne riječi: sim-to-real, IsaacSim, ROS (Robotski operativni sustav), mobilni robot, mapiranje prostora

SUMMARY

The task of this paper is to integrate the Robot Operating System (ROS) into the IsaacSim virtual environment. The first part of the paper provides an overview of the simulation environments IsaacSim and Gazebo and their comparison. Then, the principles of ROS and its integration into virtual environments, specifically IsaacSim, are explained. After that, a review of mobile robots and their applications is given.

The paper focuses on integrating ROS into IsaacSim in a way that the differential drive mobile robot is controlled from ROS nodes. The necessary sensors for mapping the space are added to the mobile robot. By implementing a mapping algorithm, an occupancy map of the Regional Center of Excellence for Robotics Technologies (CRTA), whose model was made in Autodesk Inventor, was created.

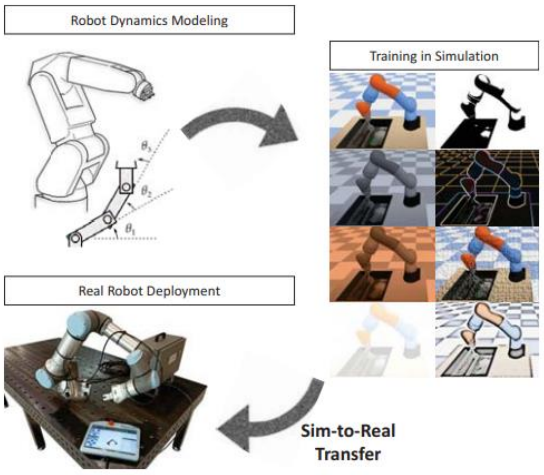
Key words: sim-to-real, IsaacSim, ROS (Robot Operating System), mobile robot, mapping

1. UVOD

Podržano učenje (eng. *reinforcement learning – RL*) je posljednjih godina doživjelo veliki uspjeh na raznim poljima robotike za kontrolu složenih ili više-robotskih sustava. Inspiriran načinom učenja kod ljudi putem procesa pokušaj-pogreška, RL algoritmi svoje znanje temelje na nagrađivanju agenta koji se ponaša na određeni način u različitim situacijama. To zahtijeva velik broj različitih eksperimenata, vremena i novca, što dovodi do nedostataka podržanog učenja u stvarnom svijetu. Također, učenje s pravim robotima dovodi do potencijalno opasnih i neočekivanih ponašanja. [1]

Jedno od rješenja ovih problema je prijenos simulacije na stvarnost, tj. *sim-to-real transfer*. Osnovna ideja je treniranje robota u simulatoru koji približno opisuje stvarno okruženje i prijenos već istreniranog modela robota u stvarnost. *Sim-to-real transfer* pruža neograničenu količinu podataka te smanjuje vrijeme, troškove i rizike koji se javljaju kod treniranja robota u stvarnosti.

Sim-to-real transfer postiže se kroz nekoliko koraka. Prvo je potrebno skupiti podatke iz pravog svijeta kojima su opisani uvjeti s kojima bi se robot mogao susresti kao što su informacije o okruženju, osvjetljenje te interakcije među objektima. Zatim se iz skupljenih podataka rekreira simulacijsko okruženje koje što vjernije opisuje stvarno uključujući i model robota. nakon toga prelazi se na treniranje robota u simulaciji. Proces treniranja koristi metode strojnog učenja kako bi se poboljšalo ponašanje robota u simulacijskom okruženju. Završni korak je prijenos podataka na stvarnog robota i usporedba ponašanja sa simulacijom. Ukoliko je ponašanje u stvarnom svijetu približno jednako onom u simulaciji, *sim-to-real transfer* smatra se uspješnim.



Slika 1. Sim-to-real transfer [1]

Kako god ta ideja zvučala primamljivo, *Sim-to-real transfer* suočava se problemom degenerirane izvedbe robota u stvarnom okruženju zbog nepodudaranja između simuliranog i stvarnog okruženja, odnosno *sim-to-real gap*.

Problem *sim-to-real gap* pokušava se riješiti razvojem snažnijih simulatora i metodom randomizacije domene (eng. *domain randomization*). Tom metodom nastoji se nasumično odrediti dinamika okoline izlaganjem robota različitim situacijama u fazi treniranja. [2]

Jedan od snažnih simulatorskih okruženja kojim se nastoji smanjiti ili potpuno ukloniti *sim-to-real gap* je NVIDIA IsaacSim koji je ujedno i tema ovog rada. Uspješan razvoj, treniranje i testiranje kompleksnih robota za primjenu u stvarnom svijetu zahtijeva vjernost simulacije i točnost fizikalnih zakona. IsaacSim je simulacijska platforma koja pruža realistična virtualna okruženja koja blisko oponašaju stvarne uvjete za treniranje i testiranje robota. Svrha IsaacSim-a je osigurati simulacijsko sučelje koje se može koristiti za točno predviđanje robotskog ponašanja u stvarnom svijetu. IsaacSim omogućava sveobuhvatno simulacijsko okruženje koje uključuje podršku za fiziku, više-robotske sustave i široki raspon senzora i aktuatora. Omogućuje i alate za skupljanje i analizu podataka što olakšava provjeru i usporedbu rezultata iz simulacije i stvarnog okruženja.

Jedna od glavnih prednosti rada na simuliranim robotima u odnosu na stvarne je sposobnost simulatora da generiraju sintetičke podatake koji su jeftiniji i dostupniji. Dio IsaacSim aplikacije je NVIDIA Isaac Replicator – alat za generiranje sintetičkih podataka. Generiranjem sintetičkih podataka okruženja u kojem se robot nalazi i korištenjem metode strukturirane randomizacije domene (eng. *structured domain randomization*) u Isaac Replicator-u, moguće je trenirati robota da bude svjestan svog okruženja, i svih promjena vezanih uz isti, u stvarnom svijetu.

NVIDIA IsaacSim razvijen je kako bi pomogao nadvladati *sim-to-real gap* u robotici. Njegovim korištenjem programeri i istraživači mogu smanjiti vrijeme potrebno za razvoj, testiranje i implementaciju algoritama na stvarne robote uz povećanje pouzdanosti i s boljim performansama u stvarnom svijetu. [3]

2. SIMULACIJSKA OKRUŽENJA

U svijetu robotskih tehnologija, simulacija ima ključnu ulogu u razvoju i testiranju autonomnih sustava. Kako roboti postaju sve složeniji i sveprisutniji u industrijskim, ali i svakodnevnim okruženjima, ključno je da mogu spoznati što točniju i kvalitetniju 3D sliku prostora u kojem se nalaze.

Robotski simulatori koriste digitalni prikaz (eng. digital twin) robota u virtualnom okruženju kako bi testirali njihovo ponašanje i različite mogućnosti bez aplikacije na stvarnog robota. To znači da će se različiti dijelovi, zajedno sa različitim algoritmima i interakcijama sa stvarnim svijetom, testirati prije proizvodnje i implementacije na stvarnog robota što dovodi do značajne uštede vremena i novca. [4]

Robotski simulatori u prošlosti bili su rađeni za određene robote u određenim okolnostima. Posljednjih godina, s razvojem robotike i tehnologije općenito, s razvojem *physics engine,* omogućen je razvoj novih simulatora koji omogućuju implementiranje svih vrsta robota uz kombinaciju fizikalnih zakona (gravitacija, trenje, itd.) i realnih grafičkih prikaza. Kako bi se osigurala prenosivost koda iz simulatora na stvarne robote, što je i konačni cilj, oni su s vremenom prestali biti klasa za sebe već su postali dio većih platformi. Na tim platformama često se odvijaju različiti među-programi stoga je i njihove performanse i funkcije potrebno uzeti u obzir kod inspekcije vjernosti simulacije. [5]

Danas postoje mnoga okruženja za simulaciju robota, a dva od popularnijih su Isaac Sim i Gazebo.

U ovom dijelu rada biti će detaljnije objašnjene mogućnosti Isaac Sim-a i Gazebo-a kao i njihove primjene u razvoju robota. Prvo će biti dan kratak pregled za svaki simulator, a zatim će slijediti njihova usporedba po određenim kriterijima.

2.1. Nvidia IsaacSim

Nvidia Omniverse IsaacSim je simulator virtualnog okruženja temeljen na NVIDIA Omniverse platformi. To je skalabilna robotska aplikacija i alat za generiranje sintetičkih podataka koji omogućuje fotorealistično i fizikalno precizno virtualno okruženje, što pruža bolji i brži način razvoja, testiranja i upravljanja AI robota.

IsaacSim razvijen je za istraživače, inženjere i robotičare koji žele razvijati i testirati algoritme strojnog učenja te simulirati kompleksne scenarije prije korištenja u stvarnom svijetu. Omogućuje visoko realistično virtualno okruženje koje koristi napredni *physics engine* NVIDIA PhysX za prikazivanje ponašanja objekata i robota. Podržava široki raspon senzora,

uključujući kamere, lidar, ultrazvučni senzor i senzor dubine koji se koriste za sakupljanje podataka za strojno učenje. IsaacSim također omogućuje skup alata za vizualizaciju i analizu simuliranih podataka kao što su IsaacGEM, vizualni programski alat za kreiranje kompleksnih algoritama korištenjem *drag-and-drop* sučelja, zatim TensorBoard, alat koji se koristi za vizualizaciju i analizu podataka za strojno učenje, čime korisnicima olakšava razumijevanje i poboljšavanje algoritama.

Jedna od glavnih značajki IsaacSim-a je njegova integracija s NVIDIA SDK (*Software Development kit*). Ta integracija omogućuje korisnicima nesmetan prijenos algoritama iz simulacije na stvarnog robota, što olakšava testiranje i validaciju programa. U to je uključena integracija s NVIDIA Jetson platformom koja je skup hardverskih i softverskih alata za razvijanje i prenošenje robotskih aplikacija. Time je korisnicima omogućeno treniranje modela s podacima prikupljenim iz simulacijskog okruženja i prenošenje istih na Jetson uređaje za korištenje u stvarnom okruženju. Također, IsaacSim je razvijen za nesmetan rad NVIDIA GPU što korisnicima omogućuje visoke performanse i preciznu simulaciju. Temelji se na *Unreal Engine 4* za renderiranje čime se osigurava vjerno 3D okruženje koherentno sa stvarnošću u stvarnom vremenu.

IsaacSim podržava programske jezike Python i C++ što omogućuje korisnicima pisanje koda u jeziku s kojim su upoznati. Pomoću Isaac Bridge ekstenzije povezuje se s ROS1 i ROS2 sustavom čime se omogućuje korištenje *open-source* knjižnica i alata što dodatno olakšava razvoj robotskih aplikacija. Iako platforma uključuje široki raspon simulacijskih alata za manipulaciju objektima, kontrolu robota i generiranje podataka sa senzora, korisnicima je omogućeno pisanje prilagođenih skripti za programiranje robotskog ponašanja i automatizaciju zadataka u simulaciji.

IsaacSim koristi se za robotska istraživanja, razvoj autonomnih vozila i industrijsku automatizaciju. Uključuje knjižnicu prethodno izrađenih modela robota i okruženja od jednostavnih robota sa nekoliko zglobova do kompleksnih sistema kao što su humanoidni roboti i autonomna vozila. Omogućuje i podršku za razvoj i testiranje algoritama za autonomnu vožnju, uključujući prethodno izrađeni virtualni svijet koji simulira vožnju u stvarnim scenarijima kao što su raskrižja, autoceste i gradske ceste.

IsaacSim omogućuje alate za kolaboracijski razvoj i testiranje što znači da više korisnika istovremeno može raditi u istom simulacijskom okruženju i dijeliti podatke i algoritme. To je osobito značajno za razvojne timove s puno istraživača koji rade na istom projektu.

Glavni nedostatak IsaacSim-a su velika početna ulaganja jer za njegovu instalaciju i pokretanje treba imati snažno računalo kako bi se omogućilo korištenje performansi. [6]

2.2. Gazebo

Gazebo je 3D simulator virtualnog okruženja otvorenog izvora koji omogućuje korisnicima simulaciju i vizualizaciju robota u virtualnom okruženju. Originalno je bio razvijen na US National Institute of Standards and Technology (NIST) kao dio njihovog robotičarskog programa, a sada ga održava Open Source Robotics Foundation. Gazebo je korišten u različitim razvojnim projektima uključujući DARPA Robotics Challenge natjecanje za razvoj humanoidnog robota koji može obavljati kompleksne zadatke u katastrofičnim scenarijima.

Gazebo koristi DART fizički motor za prikaz fizikalno realističnog okruženja i simulaciju ponašanja robota u istom, a kompatibilan je i sa Bullet, ODE i Simbody motorima. Narpredna 3D grafika omogućena je korištenjem Ogre 2.1 motora za renderiranje koji nudi pristup najnovijim tehnikama za renderiranje uključujući poboljšanu kartu sjena i PBR materijale.

Sljedeća karakteristika Gazebo-a je što pruža podršku za simulaciju senzora poput kamere, lidara, IMU, kontaktni i senzor magnetskog polja te omogućuje validaciju algoritama temeljenih na senzorima u virtualnom okruženju.

Jedna od glavnih značajki Gazebo-a je njegova modularnost i integracija s ROS1/ROS2 sustavom. ROS pruža set priključaka za Gazebo koji omogućuju komunikaciju robota i ROS čvorova. Korisnici također mogu izraditi prilagođene priključke za dodavanje novih značajki i modificiranje postojećih ili mogu koristiti postojeće priključke iz zajedničke knjižnice s velikim opsegom podataka i raznih rješenja.

Gazebo podržava široki raspon robotskih platformi od najjednostavnijih robota na kotačima do kompleksnih humanoidnih robota te može simulirati različite vrste okruženja uključujući vanjsko, unutarnje i svemirsko okruženje. Gazebo omogućuje jednostavno korisničko sučelje za stvaranje i uređivanje robotskih modela koji se mogu učitati iz različitih CAD alata. Robotski modeli mogu sadržavati detalje poput senzora, aktuatora i mehaničkih dijelova i mogu se prilagoditi da odgovaraju zahtjevima pojedinog robota.

Gazebo pruža mogućnost simulacije više različitih robota istovremeno u istom okruženju i time omogućuje korisnicima testiranje i validaciju algoritama koji uključuju kolaboraciju među robotima. Gazebo pruža podršku za C++ i XML programske jezike, no moguće ga je povezati s Phyton-om, Java-om i MATLAB-om kroz priključke.

Gazebo uglavnom dolazi zajedno s instalacijom ROS-a i za njega nije potrebno snažno računalo, a za prikaz grafike pokazalo se da najbolje rade NVIDIA GPU. Mali inicijalni troškovi, odlična povezanost s ROS knjižnicama i algoritmima čine ga popularnim u svijetu robotike, no mana mu je što nema realistično renderiranje. [7]

2.3. Ostali simulatori

Osim IsaacSim-a i Gazebo-a, postoji još mnogo različitih robotskih simulatora, a neki od njih opisani su u nastavku.

RoboDK je simulator za izvanmrežno programiranje i simulaciju industrijskih robota. Moguće ga je instalirati na Winows, Linux, Mac OS ili Android operativni sustav i dostupan je u obliku besplatne probne verzije i za kupovinu na RoboDK stranici. RoboDK pruža opsežnu knjižnicu CAD modela s više od 500 različitih profesionalnih industrijskih robota i alata. Nudi intuitivno korisničko sučelje te nisu potrebne napredne programerske vještine, stoga je jednostavno vizualizirati pokretanje robota. Jedna od glavnih prednosti programa je jednostavnost prenošenja podataka direktno na robota, kojeg je moguće kalibrirati putem modula čime se povećava točnost simulacije, koristeći RoboDK Post Processor.

Webots simulator razvijen je 1996. godine na Tehničkom institutu u Švicarskoj. Webots pruža potpuno razvojno okruženje za programiranje, modeliranje i simulaciju robota te se može povezati s aplikacijama drugih proizvođača putem TCP/IP protokola. Nudi jednostavno sučelje za dodavanje postojećih robota i komponenti poput aktuatora, senzora, objekata, namještaja i drugo. Webots je aplikacija otvorenog koda s opsežnom dokumentacijom i aktivnom zajednicom, koristi ODE fizički motor te uključuje C++ kompajler za testiranje i kontrolu algoritama, no moguće je koristiti i druge jezike.

CoppeliaSim ili *V-REP* je simulator zatvorenog izvora s besplatnom edukacijskom licencom i moguće ga je koristiti na Windows-u i Linux-u. Svaki objekt u programu može biti zasebno kontroliran putem skripti, priključaka i ROS čvorova. CoppeliaSim omogućuje različite robote, objekte i senzore te je svaki model fleksibilan, prenosiv i skalabilan. Također je moguće napraviti vlastitog robota. Glavni elementi CoppeliaSim-a su objekti poput zglobova, oblika senzora, inverzna kinematika, prepoznavanje sudara i mehanizmi kontrole. Simulator koristi četiri različita fizička motora ODE, Bullet, Vortex Dynamic i Newton te je u usporedbi s Gazebo simulatorom stabilniji i jednostavniji za postavljenje i pokretanje. [8]

2.4. Usporedba IsaacSim i Gazebo simulatora

Tablica 1. Usporedba Gazebo i IsaacSim simulatora

	IsaacSim 	**Gazebo** 
Generalne informacije	NVIDIA	Open Source Robotics Foundation
Podržani OS	Linux (Ubuntu), Windows 10/11	Linux (Ubuntu), MAC OS
RAM	32 GB	4 GB
VRAM	8 GB	1 GB
CPU	Intel Core i7, AMD Ryzen 5	Intel Core i5
GPU	Nvidia GeForce RTX 2070	Nvidia GPU se pokazao da radi najbolje
Pohrana	50 GB SSD	500 MB
Pohrana na Cloud	Da	Da
Jezik za programiranje	Python	C++, XML
Physical engine	NVIDIA PhysX	ODE, Bullet, Simbody, DART
Generiranje sintetičkih podataka	Da (Isaac Replicator)	Da
Graphics rendering engine	Unreal Engine 4.20.3	Ogre 2.1.
Realistično renderiranje	Da	Ne
Podržane CAD datoteke	URDF, STEP, MJCF, USD, FBX	SDF/URDF, OBJ, STL
ROS podrška	ROS1 i ROS2	ROS1 i ROS2
Licence	Besplatan za preuzimanje (BSD-3 licenca)	Besplatan i otvorenog izvora
Multithreading	Da	Da
Teleoperacija	Tipkovnica, gamepad	Tipkovnica, gamepad

Podržani roboti	Mobilni, humanoidni, industrijski, autonomna vozila, više robota istovremeno	Mobilni, humanoidni, industrijski, više robota istovremeno
Podržani aktuatori	Fiksirani, rotacijski, prizmatični, sferni zglob, zupčanici, udaljeni zglob, D6 zglob, zupčasta letva sa zupčanikom	Rotacijski, prizmatični, vijčani i sferni zglob
Podržani alati	hvataljke	hvataljke
Podržani senzori	Senzor kamere (sa zvukom), senzor dubine kamere, fisheye kamera, lidar, senzor sile i momenta, ultrazvuk, IMU senzor, kontaktni senzor	Senzor kamere, senzor dubine kamere, senzor udaljenosti i blizine, lidar, senzor sile, senzor magnetskog polja, IMU senzor
Primjene	Manipulacija, navigacija	Manipulacija, navigacija
OSTALO	Strojni vid, planiranje trajektorije, inverzna kinematika, simulacija usisa, simulacija deformacije, GPS, VR podrška, simulacija tečenja, simulacija ljudskog kretanja	Planiranje trajektorije, inverzna dinamika, inverzna kinematika, GPS, VR podrška, simulacija tečenja, DEM simulacija

Tablica 1. prikazuje usporedbu Gazebo i IsaacSim simulatora.

Glavna razlika između njih je što IsaacSim omogućuje realistično renderiranje što pruža osjećaj stvarnog svijeta te je sukladno s tim lakše naučiti robota kako da se ponaša u simulaciji i prenesti to na stvarnog robota. Prednost Gazebo-a nad IsaacSim-om je što nisu potrebna velika inicijalna ulaganja jer nije potrebno snažno računalo za pokretanje i instalaciju, što kod IsaacSim-a nije slučaj. Gazebo je platforma otvorenog izvora te postoji veliki raspon već napravljenih algoritama i priključaka što olakšava korisnicama upoznavanje s programiranjem robotskih aplikacija te je usko povezan s ROS-om. IsaacSim je besplatna aplikacija i omogućuje korištenje i povezivanje s NVIDIA ekosustavom koji pruža razne priključke i algoritme te ga je moguće povezati s ROS-om putem Isaac Bridge ekstenzije. Što se tiče hardvera, IsaacSim podržava prijenos programa na uređaje s NVIDIA Jetson hardverom, dok se program iz Gazebo-a može prenesti na široki raspon uređaja.

Sve u svemu, oba simulatora imaju svoje prednosti i mane te odabir između njih ovisi o specifičnim zahtjevima i potrebama projekta te mogućnostima i preferencijama korisnika.

3. ROBOTSKI OPERATIVNI SUSTAV (ROS)

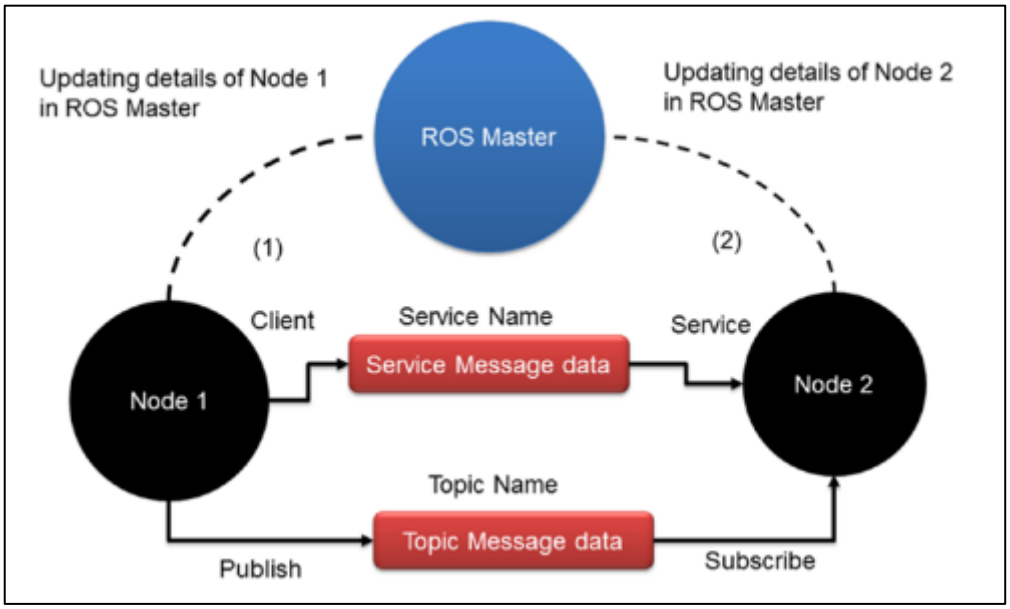
Robotski operativni sustav (ROS) je besplatna platforma za programiranje robota otvorenog izvora (eng. *open-source*) te se koristi u komercijalne i istraživačke svrhe. ROS projekt započeo je 2007. godine na Sveučilištu u Stanfordu. Prije razvoja ROS-a nije postojala zajednička platforma za razvoj robotskih aplikacija, nego je svaki programer trebao ispočetka napisati program za svog robota koji se u većini slučajeva nije mogao ponovno koristiti. Nakon razvoja ROS-a, programiranje robota postalo je jednostavnije i brže. ROS je platforma otvorenog izvora što znači da su svi alati, kodovi i programi javni i dostupni svima na korištenje i prilagođavanje.

3.1. Arhitektura i osnovni koncepti ROS-a

Arhitektura ROS-a bitna je za jednostavniji razvoj robotskih programa. ROS se dijeli na razvojnu (eng. *computation*) razinu, razinu organizacije datoteka (eng. *filesystem*) i razina zajednice (eng. *community*).

3.1.1. Razvojna (eng. computation) razina

Glavna karakteristika ROS-a kod programiranja robota je mogućnost ostvarivanja komunikacije između procesa (eng. *interprocess comunication*), tj. osigurava sučelje za razmjenu informacija između dva programa ili procesa na način prikazan slikom ispod.



Slika 2. ROS komunikacija [10]

Slika 2. prikazuje dva programa označena kao čvor 1 (eng. *node 1*) i čvor 2 (eng. *node 2*). Čvorovi su osnovne jedinice ROS sustava i u njima su zapisani procesi. Prednost korištenja

čvorova je u tome što, ukoliko neki dio programa ne radi, nije potrebno provesti analizu i prepravak cijelog programa, nego samo jednog dijela.

Kada se bilo koji od čvorova pokrene, on ostvaruje komunikaciju s ROS Masterom i šalje mu sve svoje informacije, uključujući i vrstu podataka koje šalje ili prima. Čvorovi koji šalju podatke zovu se objavljivači (eng. *publisher nodes*), a oni koji primaju podatke zovu se pretplatnici (eng. *subscriber nodes*). Dakle, ROS Master je središnji dio ROS sustava koji nadzire rad čvorova i povezuje ih. Na računalu može biti pokrenut samo jedan ROS Master.

Djeluje na način da, ako čvor 1 objavljuje neki podatak „X“ i taj isti podatak je zahtjevan od strane čvora 2, ROS Master šalje informaciju čvorovima da ostvare komunikaciju.

ROS čvorovi mogu međusobno slati različite vrste podataka koje se nazivaju ROS poruke (eng. *ROS messages*). ROS poruke su osnovne jedinice za komunikaciju u ROS sustavu i šalju se između čvorova putem ROS tema (eng. *ROS topics*). Jedan čvor može se pretplatiti ili objavljivati na više tema.

ROS komunikacija između čvora 1 i čvora 2 može se objasniti na primjeru robota s kamerom i zaslonom za prikaz. Čvor 1 zaslužan je za čitanje i obradu slika s kamere i objavljivanje istih na ROS temu. Čvor 2 pretplaćuje se na tu temu te prima sliku s kamere i objavljuje ju na zaslonu. Te slike šalju se između čvorova u obliku ROS poruka putem ROS teme.

Osim komunikacije putem ROS tema, čvorovi mogu komunicirati i putem ROS servisa (eng. *ROS service*). Komunikacija putem ROS servisa radi na principu server/klijent (eng. *server/client*) na način da jedan čvor šalje zahtjev drugom čvoru i čeka odgovor. Kada čvor želi koristiti neki servis, stvara klijenta i šalje zahtjev server čvoru koji omogućuje taj servis. Kada server zaprimi zahtjev, obavlja zahtijevanu radnju i vraća odgovor.

ROS servisi mogu se također objasniti na primjeru robota s kamerom i zaslonom za prikaz. Čvor 1 ponaša se kao upravljački program za kameru i služi za čitanje i obradu slika s kamere te objavu istih na ROS teme. Čvor 2 zaslužan je za primanje slika s kamere i prikaz na zaslonu. Umjesto da se čvor 2 pretplaćuje na temu, on putem ROS servisa zahtijeva sliku s kamere na način da stvara klijenta za taj servis i šalje zahtjev čvoru 1 da mu pošalje zadnju učitanu sliku. Čvor 1 prima taj zahtjev, određuje koja je zadnja učitana slika i šalje je kao odgovor na zahtjev čvora 2 koji tu sliku objavljuje na zaslonu. Korištenjem ROS servisa čvor 2 je siguran da je na zaslonu objavio posljednju učitanu sliku s kamere, stoga su ROS servisi pogodni za zadatke koji zahtijevaju specifičan odgovor.

3.1.2. Organizacija datoteka (eng. filesystem) u ROS-u

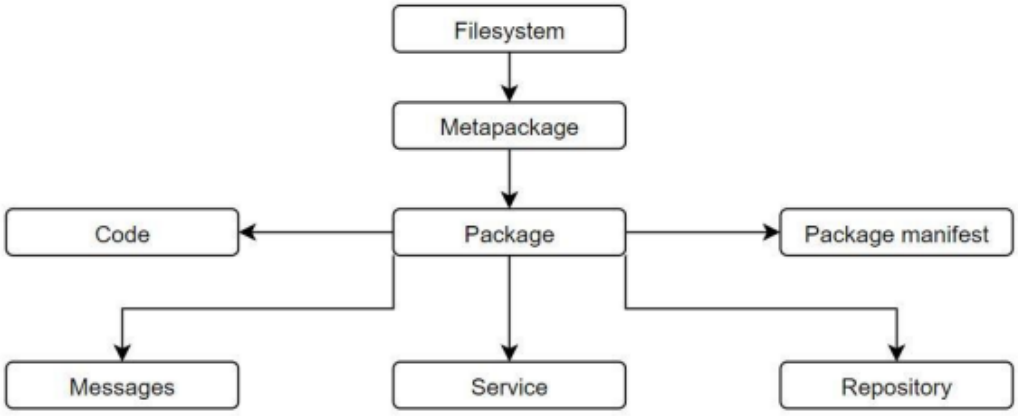
Kao što mu ime kaže, ROS nije pravi operativni sustav. To je meta operativni sustav koji posjeduje neke funkcionalnosti pravih operativnih sustava kao što je višenitnost (eng. *multithreading*), upravljanje paketima i apstrakciju *hardware*-a. *Multithreading* u ROS-u odnosi se na to da čvorovi mogu sadržavati više niti (eng. *thread*) za obavljanje više zadataka istovremeno. Na primjer, robot ima dva motora za pokretanje i lidar za percepciju prostora i moguće je napraviti jedan čvor koji je zaslužan za kontrolu motore i obradu podataka s lidara. Apstrakcija *hardware*-a omogućuje programiranje uređaja. Prednost je ta što je kod za određeni uređaj, recimo senzor, potrebno napisati samo jednom i on će biti primjenjiv u različitim situacijama.

Upravljanje paketima omogućuje organizaciju *software*-a u zasebne jedinice koje se nazivaju ROS paketi (eng. *ROS packages*). Slika 2. prikazuje strukturu i organizaciju datoteka u ROS-u. ROS meta paketi (eng. *ROS metapackages*) grupiraju slične pakete za određene namjene i ne sadrže izvorne i programske datoteke. Svi izvorni kodovi, programske datoteke, međuovisnosti i ostalo, zapisano je u ROS paketima.

Manifest paketa (eng. *package manifest*) je XML unutar ROS paketa koja sadrži osnovne informacije o paketu uključujući ime, opis, autora, međuovisnosti, i dr.

ROS repozitorij (eng. *ROS repository*) je skup ROS paketa i služi za osvježavanje novih verzija u ROS-u.

ROS poruke (eng. *messages*) definiraju se i spremaju u *msg* datoteku, a ROS servisi u *srv* datoteku.



Slika 3. Organizacija datoteka u ROS-u [11]

3.1.3. ROS zajednica (eng. community)

Glavni razlog tako brzog rasta popularnosti i razvoja ROS-a je podrška zajednice (eng. *community support*). ROS programeri iz cijelog svijeta aktivno razvijaju i održavaju ROS pakete, odgovaraju na pitanja korisnika te dijele različite ROS pakete. Svi postojeći ROS alati, kao i odgovori stručnjaka na moguće nedoumice i probleme mogu se pronaći na službenim ROS stranicama [9].

Prednost ROS-a je što se ROS paketi mogu koristiti više puta na različitim robotima i time se skraćuje vrijeme razvoja programa. ROS također ima implementirane često korištene algoritme u robotici kao što su PID i SLAM algoritmi.

Još jedna od prednosti ROS-a je što podržava programske jezike korištene u razvoju robotskih aplikacija kao što su Python, C++ i Lisp. ROS osigurava biblioteke za te jezike, što znači da programeri mogu koristiti funkcionalnosti koje pruža ROS u tom programskom jeziku.

Za više informacija o ROS-u, osnovne koncepte, instalaciju, korištenje te kako ga implementirati na robote vidjeti [10] i [11].

3.2. Integracija ROS-a sa simulatorima

Robotski operativni sustav (ROS) može se integrirati sa raznim simulatorima čime se stvara snažan alat za razvoj, testiranje i evaluaciju robotskih sustava. Korištenjem ROS-a u simulatorima omogućeno je sigurno i kontrolirano testiranje algoritama u virtualnom okruženju prije implementacije na pravog robota. Time se ubrzava proces razvoja robotskih aplikacija i smanjuje mogućnost grešaka koje mogu dovesti do oštećenja robota. Integracijom ROS-a u simulator omogućuje se korištenje ROS paketa i alata poput ROS vizualizacije i navigacije.

Postoji nekoliko načina na koje ROS može biti integriran sa simulatorima. Simulatori poput Gazebo-a i V-REP-a imaju ugrađenu podršku za ROS što im omogućuje korištenje ROS čvorova i tema. Neki simulatori, kao što su MORSE i Stage, razvijeni su specifično za rad s ROS-om i pružaju korisnicima sveobuhvatan set ROS alata za testiranje algoritama. Postoje i simulatori koji izvorno ne podržavaju ROS, no moguće ih je primijeniti s ROS-om primjenom *ROS Wrapper*-a. *ROS-wrapper* je čvor koji se napravi na dijelu programskog koda kako bi se za njega napravilo ROS sučelje. [12]

3.2.1. Integracija ROS-a u IsaacSim-u

Integracija IsaacSim-a s ROS-om ostvaruje se putem ROS Bridge ekstenzije. ROS Bridge omogućuje ROS čvorovima komunikaciju s IsaacSim okruženjem. Time se omogućuje

korištenje ROS paketa i alata za kontrolu robota, čitanje podataka sa senzora i interakciju s virtualnim okruženjem.

ROS Bridge ekstenzija omogućena je prema zadanim postavkama IsaacSim-a te ju je moguće isključiti u *Extension Menager*-u. IsaacSim može ostvariti komunikaciju i s ROS 2 sustavom pomoću ROS 2 Bridge ekstenzije. Samo jedna od ekstenzija može biti omogućena u danom trenutku.

Slika 4. ROS Bridge ekstenzija. Izvor: Autor

IsaacSim kompatibilan je sa ROS distribucijama ROS Noetic i ROS Melodic, a za ROS 2 sa ROS 2 Foxy. ROS Bridge ne pokreće sam ROS Master-a, stoga ga je, prije početka korištenja ROS paketa, potrebno pokrenuti naredbom `roscore`.

Više informacija o instalaciji i postavljanju radnog okruženja vidjeti na [13].

S instalacijom IsaacSim-a dolazi lista ROS paketa napravljenih za IsaacSim, a neke od njih su:

- *carter_2dnav*: sadrži datoteke za pokretanje i parametre za NVIDIA Carter robota kod primjera ROS navigacije
- *carter_description:* opis modela NVIDIA Carter robota
- *isaac_tutorials:* sadrži datoteke za pokretanje, RViz *.config* datoteke i skripte za ROS tutoriale
- *cortex_control:* alati za omogućavanje komunikacije između Cortex-a i kontrolera
- *cortex_control_franka:* sadrži pokretačke datoteke i python čvorove za kontrolu fizičkog robota Franka putem Cortex-a
- *isaac_moveit:* sadrži pokretačku i konfiguracijsku datoteku za pokretanje ROS MoveIt

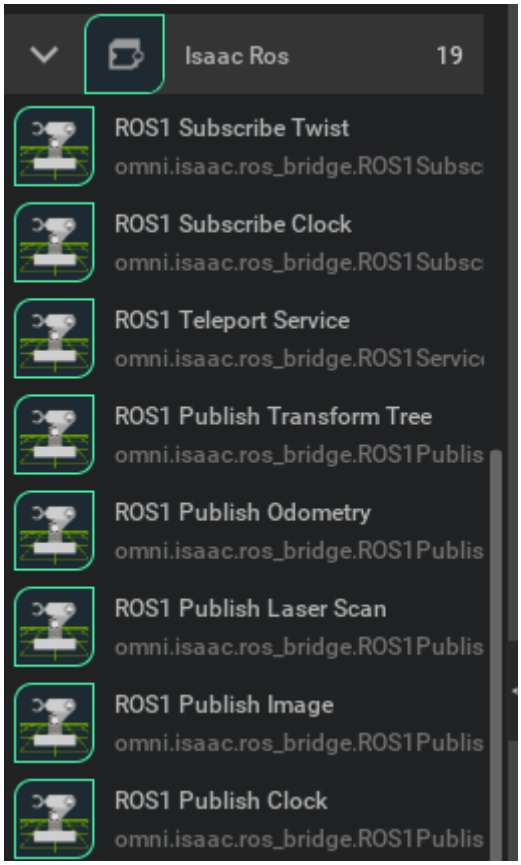
Ostali uključeni ROS paketi mogu se pronaći na [13].

Sve funkcionalnosti u IsaacSim-u povezane s ROS-om su u obliku *Omnigraph* čvorova.

Omnigraph je sučelje za grafičko programiranje. *Omnigraph* u IsaacSim-u je glavni pokretač Replicator-a, ROS Bridge-a i omogućava pristup mnogim senzorima, kontrolerima, vanjskim ulazno/izlaznim uređajima i drugo.

OmniGraph čvorovi u IsaacSim-u označavaju poseban tip grafičkih čvorova kojima je moguće povezati različite sustave unutar Omniverse-a, a prikazuju se pomoću *Action Graph*-a. U ROS *OmniGraph* čvorovima postavljaju se parametri poput ROS teme i vrste ROS poruka koje čvor može očekivati te se povezuju s ostalim *OmniGraph* čvorovima.

Neki ROS *Omnigraph* čvorovi koji se mogu pronaći kod stvaranja *Action Graph*-a su:



Slika 5. ROS Omnigraph čvorovi. Izvor: Autor

Korišteni *Omnigraph* čvorovi biti će detaljnije opisani u nastavku.

4. MOBILNA ROBOTIKA

Mobilna robotika je grana robotike koja se bavi problematikom pokretanja autonomnih i automatskih vozila. Razlika mobilne i ostalih grana robotike je ta što se ona bavi problematikom poveznom uz razumijevanje prostora velikih razmjera što uključuje probleme mobilnosti, spoznaje i percepcije. Mobilna robotika je interdisciplinarno područje koje obuhvaća strojarstvo, programiranje, elektrotehničko inženjerstvo, kognitivnu psihologiju, percepciju, neuroznanosti i mehatroniku.

Mobilan robot je programski upravljan stroj koji koristi senzore, kamere i druge tehnologije kako bi spoznao svoje okruženje i istim se kretao. Za rad koriste kombinaciju umjetne inteligencije (eng. *artificial intelligence* – AI) i fizičkih robotskih elemenata kao što su kotači, gusjenice ili noge. Mobilni roboti postaju sve popularniji u različitim sektorima rada zato što mogu obavljati zadatke koji su opasni za ljude.

Mobilne robote moguće je podjeliti u grupe s obzirom na dva osnovna kriterija. Prva podjela je po kriteriju okruženja u kojem rade, npr. bespilotne letjelice (eng. *unmanned aerial vehicle* – UAV) ili dronovi koji lete zrakom, vozila bez vozača (eng. *unmanned ground vehicles* – UGV) koji se kreću zemljom, podvodna autonomna vozila (eng. *autonomous underwater vehicles* – AUV) koja mogu samostalno pronaći kurs i kretati se kroz vodu, i dr.

Slika 6. Autonomna podmornica [19]

Drugi kriterij podjele je način kretanja robota koji može biti autonomni i teleoperiran ili vođeni. Teleoperiran mobilni roboti zahtijevaju određene informacije i neku vrstu vođenja kako bi se pokretali, dok se autonomni mobilni roboti (eng. *autonomous mobile robot* – AMR) mogu kretati i istraživati svoje okruženje bez vanjskih uputa.

Osnovne funkcije mobilnih robota su mogućnost pokretanja i istraživanja, a većinom se koriste u industriji za prijevoz tereta u skladištima i distribucijskim centrima. Na mobilnog robota moguće je ugraditi robotsku ruku pa oni zajedno mogu obavljati kompleksne zadatke u pokretu kao što je asistencija medicinskom osoblju kod operacije. Mobilni roboti također se često

koriste za istraživanje oceana, svemira, nuklearnih postrojenja i ostalih okruženja koja su opasna za ljude.

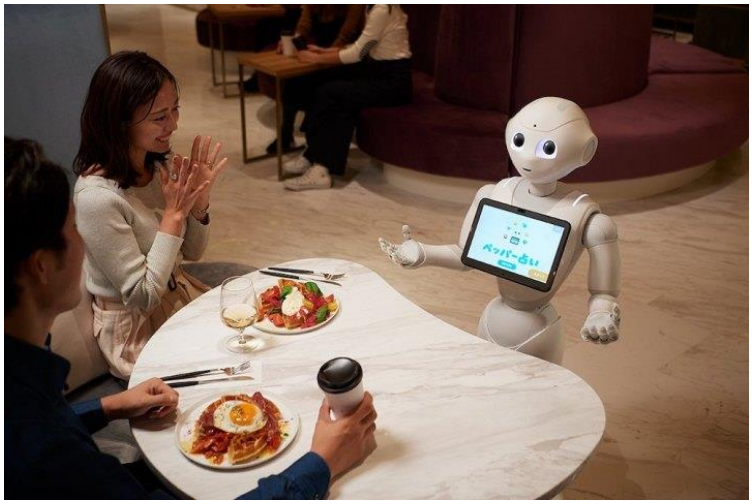
Slika 7. Fraunhofer IML – autonomni robot za skladišta [20]

Glavna prednost korištenja mobilnih robota je njihova mogućnost strojnog vida. Korištenje kompleksnih senzora omogućuje im spoznavanje i prilagodbu okruženju koji se stalno mijenja u stvarnom vremenu. [14]

Iako se mobilni roboti većinom koriste u industriji, postoji sve više primjera korištenja u svakodnevnom životu. Najbolji primjer toga je robotski usisavač prikazan slikom ispod koji se sve češće može vidjeti u kućanstvima.

Slika 8. Robotski usisavač iRobot Roomba 680 [21]

Još jedan od primjera gdje je moguće vidjeti mobilne robote u stvarnom životu je Pepper PARLOR – kafić u Tokyo-u u kojem roboti rade zajedno s ljudima kao konobari.

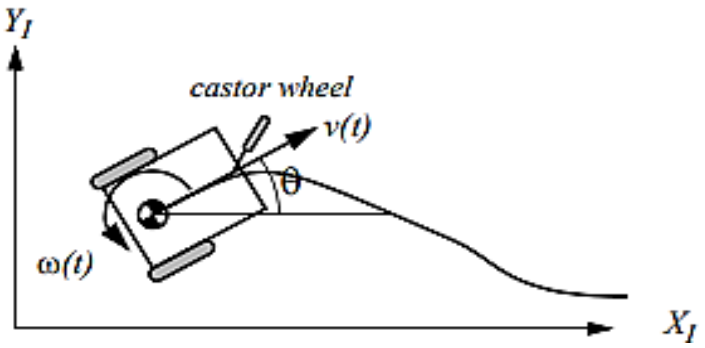


Slika 9. Pepper PARLOR. [22]

4.1. Kinematika mobilnog robota

Kinematika je grana mehanike koja opisuje kretanje objekata bez da uzima u obzir sile koje to kretanje uzrokuju. U slučaju mobilne robotike, kinematika se odnosi na proučavanje kretanja robota i vezu sa kretanjem kotača. Dvije vrste kinematike korištene u robotici su inverzna i direktna kinematika. Na primjeru mobilnog robota, direktna kinematika određuje krajnji položaj i krajnju brzinu robota na temelju brzine pogonskih kotača, a inverzna kinematika računa brzine pogonskih kotača da bi se određenom brzinom došlo u određeni položaj.

Kinematika mobilnog robota može se opisati setom jednadžbi koje povezuju brzinu i poziciju robota sa kretanjem kotača ili nogu. Za mobilnog robota s kotačima, kinematika se može opisati korištenjem diferencijalnog modela koji pretpostavlja da se robot kreće na način da mu se okreću dva kotača na istoj osi pokretana različitim motorima. Kinematičke jednadžbe diferencijalnog modela povezuju linearnu i kutnu brzinu robota s kutnom brzinom kotača.



Slika 10. Diferencijalni model mobilnog robota u globalnom koordinatnom sustavu [15]

Kinematički model diferencijalnog mobilnog robota dan je slijedećim setom jednadžbi:

$$\dot{x} = v \cdot \cos\theta \tag{1}$$

$$\dot{y} = v \cdot \sin\theta \tag{2}$$

$$\dot{\theta} = \omega \tag{3}$$

Gdje je $\dot{x}$ promjena pozicije robota u vremenu u smjeru osi x, $\dot{y}$ promjena pozicije robota u vremenu u smjeru osi y, $\dot{\theta}$ promjena orijentacije robota u vremenu uzrokovana rotacijom oko osi z, θ je trenutna orijentacija robota, v je linearna brzina robota, a ω kutna brzina robota. Isto se može zapisati u matričnom obliku:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Linearna i kutna brzina robota mogu se izraziti sljedećim jednadžbama koje povezuju brzine kotača i brzinu robota:

$$v = \frac{r \cdot (\omega_r + \omega_l)}{2} \tag{4}$$

$$\omega = \frac{r \cdot (\omega_r - \omega_l)}{L} \tag{5}$$

Gdje je r radijus kotača, L udaljenost između kotača, ω_r kutna brzina desnog kotača, a ω_l kutna brzina lijevog kotača.

Te jednadžbe mogu se koristiti za kontrolu kretanja mobilnog robota diferencijalne kinematike zadavanjem linearne i kutne brzine mobilnog robota i računanjem odgovarajućih brzina kotača robota. U ovom radu to će biti primijenjeno na način da će se putem tipkovnice zadavati linearna i kutna brzina robota, a simulator će izračunavati odgovarajuće zakrete i brzinu okretanja kotača. Isto se može napraviti na način da simulator daje vrijednost enkodera pa *diff_drive_controller* računa brzinu koju integrira da bi dobio poziciju. [15]

4.2. NVIDIA Carter mobilni robot

Mobilan robot izabran za zadatak rada je NVIDIA Carter autonomni mobilni robot koji je razvijen kao platforma kako bi demonstrirao mogućnosti Isaac SDK-a. Carter je robot diferencijalne kinematike te koristi lidar i kameru za percepciju svijeta. [16]

Slika 11. NVIDIA Carter mobilni robot [16]

4.2.1. Sustav mobilnog robota NVIDIA Carter

Mobilan robot je kombinacija različitih fizičkih i programibilnih komponenti. U pogledu fizičkih komponenti, mobilni robot je kombinacija podsustava koji zajedno komuniciraju i omogućuju kretanje robota.

Slika 12. Komponente sustava mobilnog robota Carter. [16]

Tablica 2. Sustavi mobilnog robota

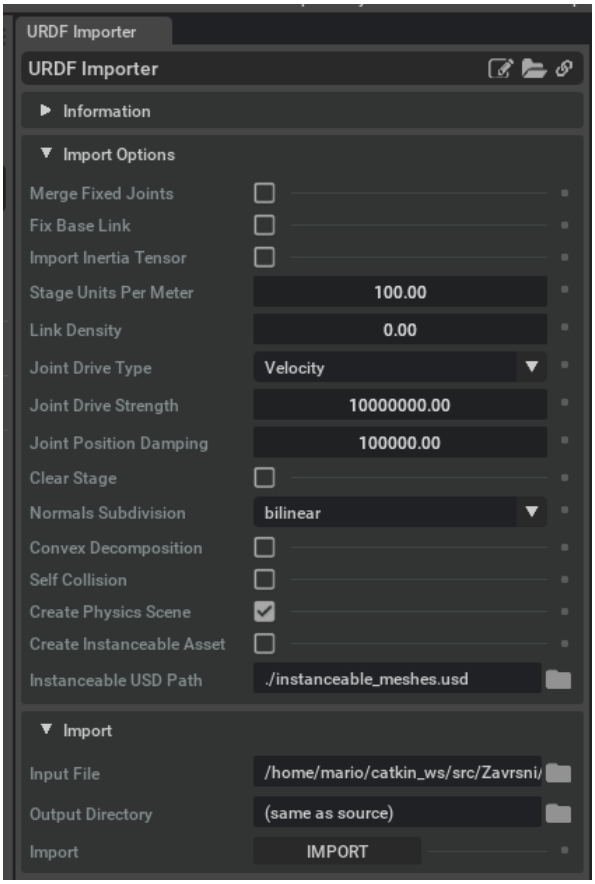
Sustav mobilnog robota	Funkcija	Carter
Upravljački sustav	upravljanje radom ostalih sustava te uspostavljnje i održavanje komunikaciju između njih	NVIDIA Jetson TX2 računalo
Senzorski sustav	prikupljanje podataka iz robotskog okruženja i omogućuje robotu da stvori sliku prostora u kojem se nalazi	Velodyne P16 lidar
		ZED Camera
		Bosch BMI160 IMU
Pogonski sustav	sastoji se od aktuatora i kotača koji zajedno robotu omogućuju kretanje	Segway RMP 210

4.2.2. Mobilni robot Carter u IsaacSim-u

Carter-ova digitalna kopija (eng. *Digital twin*) dolazi u paketu s instalacijom IsaacSim-a u *.urdf* (eng. *Unified Robotic Description Format*) formatu za prikaz primjera ROS navigacije, stoga nije bilo potrebno konstruirati robot.

Robota je potrebno u IsaacSim učitati u *.urdf* formatu zato što je to standardizirani XML format za prikazivanje robota ROS-u, a sadrži kinematiku i dinamiku robota. Učitavanje robota vrši se preko IsaacSim ekstenzije *URDF Importer*, na način prikazan slikom ispod. Za pokretanje ekstenzije potrebno je ići na *IsaacUtils > Workflows > URDF Importer*.

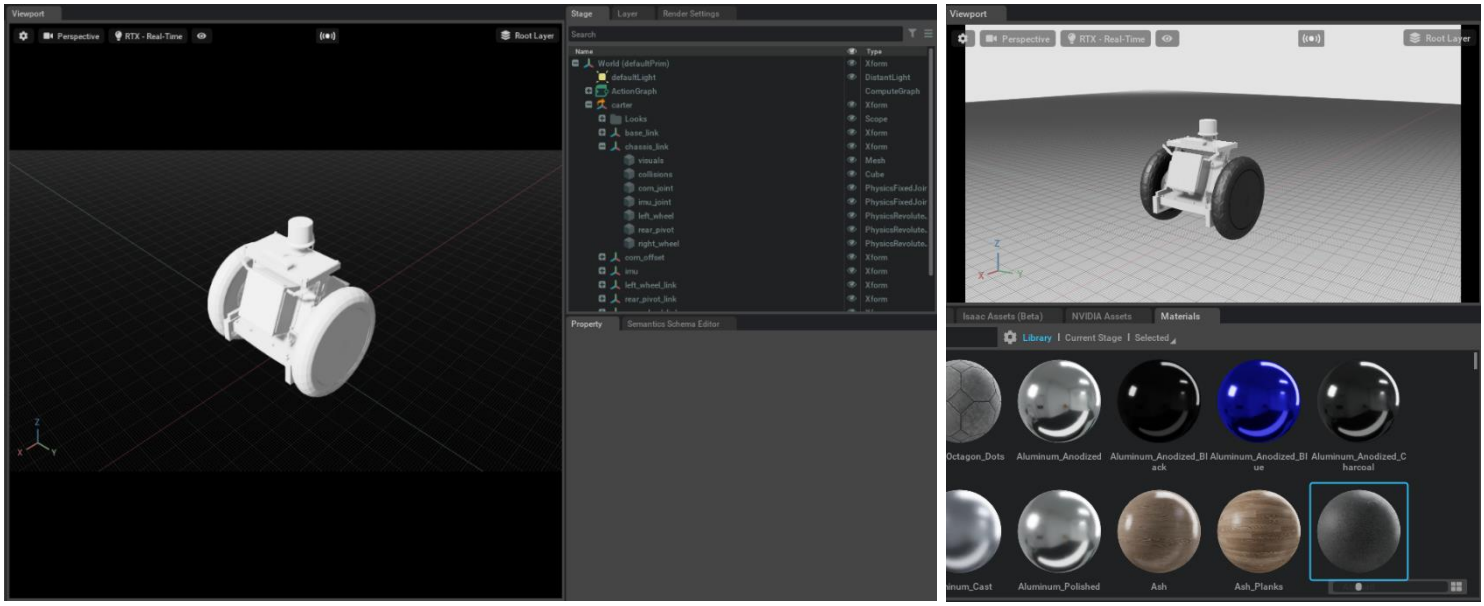
URDF Importer pretvara *.urdf* datoteke u *.usd* (eng. *Universal Scene Description*) datoteke. *Universal Scene Description* je 3D format otvorenog izvora koji koristi IsaacSim i sadrži opis prostora, a razvijen je od strane Pixar Animation Studios.



Slika 13. URDF Importer. Izvor: Autor

Slika prikazuje prozor koji se otvori kod pokretanja URDF Importer-a. U prozoru je potrebno odznačiti *Fix base link* zato što radimo s mobilnim robotom i *Clear Stage* kako bi zadržali postojeće okruženje. Također je potrebno *Joint Drive Type* postaviti na *Velocity* zato što iz ROS-a šaljemo poruku na *cmd_vel* u kojoj su sadržane translacijska i rotacijska brzina te se one preračunavaju u brzine lijevog i desnog kotača. Važno je napomenuti da su zadane mjerne jedinice URDF Importer-a centimetri (cm), stoga je potrebno *Stage Units Per Meter* staviti 100.00 kako dimenzije robota bile u metrima. Potrebno je da mjerne jedinice budu metri zato što je to zadana jedinica u ROS-u.

Pod *Input File* potrebno je učitati put do Carter-ove *.urdf* datoteke, a *Output Directory* označava mapu u koju će se spremiti novonastala *.usd* datoteka te može biti ista kao i mapa u kojoj se nalazi *.urdf* datoteka, ali može biti i različita.



Slika 14. Carter u IsaacSim-u. Izvor: Autor

Učitavanjem *.urdf* datoteke stvara se *.usd* datoteka robota u mapi i robot se prikazuje na zaslonu kao što je prikazano slikom 13. lijevo.

Kako *.urdf* datoteka sadrži kinematiku i dinamiku, učitani robot ima već sve zglobove povezane te zglobove u kotačima postavljene kao *Revolute Joint* kako bi se mogli okretati.

Kod pretvaranja *.urdf* u *.usd* datoteku u IsaacSim-u često dolazi do ne učitavanja dodijeljenih materijala, stoga ih je potrebno dodati naknadno. IsaacSim ima knjižnicu često korištenih materijala, a neki su prikazani slikom 13. desno i dodijeljeni na mobilnog robota.

4.2.3. Senzori u IsaacSim-u

U IsaacSim-u postoji mogućnost dodavanja različitih senzora za percepciju okruženja i stanja robota, a prikazani su tablicom ispod.

Tablica 3. Senzori u IsaacSim-u

SENZOR
Camera
PhysX Range Sensors (Lidar, Ultrasonic, Generic Range)
RTX Lidar Sensor
Force Sensor
Contact Sensor
IMU Sensor

Camera senzor stvara se dodavanjem koordinatnog sustava kamere na željenu poziciju na robotu na način: *Create* > *Camera*. Slika s kamere vidljiva je u *viewport*-u te je moguće istovremeno pratiti prikaz s više kamera. Podatci s kamere mogu se dobiti korištenjem programa za renderiranje koji su dio Omniverse-a iz sučelja ili putem Python skripte.

PhysX Range (Lidar, Ultrasonic, Generic Range) senzori omogućuju simulaciju senzora temeljenih na rasponu. U IsaacSim-u moguće je prikazati ponašanje lidar senzora, ultrazvučnih senzora i senzora generičkog raspona. Lidar (*light detection and ranging*) senzor koristi laserske zrake kako bi „vidio" svijet u 3D i tako omogućio robotu točnu sliku okruženja. U IsaacSim-u moguće je prikazati rotirajući lidar ili lidar generičkog raspona na način: *Create* > *Isaac* > *Sensors* > *Lidar* > *Rotating/Generic*. Lidar generičkog raspona omogućuje korisnicima potpunu kontrolu uzorka skeniranja i frekvencije senzora što je pogodno za senzore koji imaju asimetrični, neponavljajući ili nerotirajući uzorak skeniranja. Također je moguće prikazati i ultrazvučni senzor na način: *Create* > *Isaac* > *Sensors* > *Ultrasonic*. Ultrazvučni senzori uglavnom se koriste za mjerenje udaljenosti objekata emitiranjem ultrazvučnih signala.

RTX Lidar senzor zahtijeva RTX platformu, tj. RTX GPU i služi kao alternativa *PhysX* lidaru. Podržava statički i rotirajući lidar te je njegovo ponašanje opisano JSON datoteci koja dolazi zajedno s instalacijom IsaacSim-a. RTX ili *Real-Time Ray Tracing* je tehnologija za renderiranje koju je razvila Nvidia i omogućuje fizikalno točnu simulaciju ponašanja svijetla u stvarnom vremenu što rezultira visokorealističnom grafikom.

Force Sensors, tj. senzori sile pretvaraju mehaničku silu u signale koji se potom mogu mjeriti. U IsaacSim-u mogu se dodati na kruta tijela, a podatci se mogu iščitati kroz *ArticulationView*. Dodaju se na način da se označi kruto tijelo na koji se senzor želi dodati i zatim: *Add* > *Physics* > *Articulation Force Sensor*.

Contact Sensors, tj. kontaktni senzori detektiraju promjenu položaja, ubrzanja, sile, momenta i sl. U IsaacSim-u, kontaktni senzor omogućuje očitavanje na bilo kojoj brzini, daje skalarne vrijednosti te pomoćne programe za očitavanje neobrađenih kontaktnih podataka iz simulacije. Kontaktni senzor u IsaacSim-u dodaje se na način: *Create* > *Isaac* > *Sensors* > *Contact Sensor,* a njegov izvorni kod moguće je vidjeti na primjeru: *IsaacExamples* > *Sensors* > *Contact*.

IMU Sensor (Inertial measurment unit) u IsaacSim-u prati kretanje robota i ispisuje očitanja sa simuliranog akcelerometra i giroskopa. IMU senzor dodaje se na način: *Create* > *Isaac* > *Sensors* > *IMU Sensor,* a njegov izvorni kod moguće je vidjeti na *IsaacExamples* > *Sensors* > *IMU*.

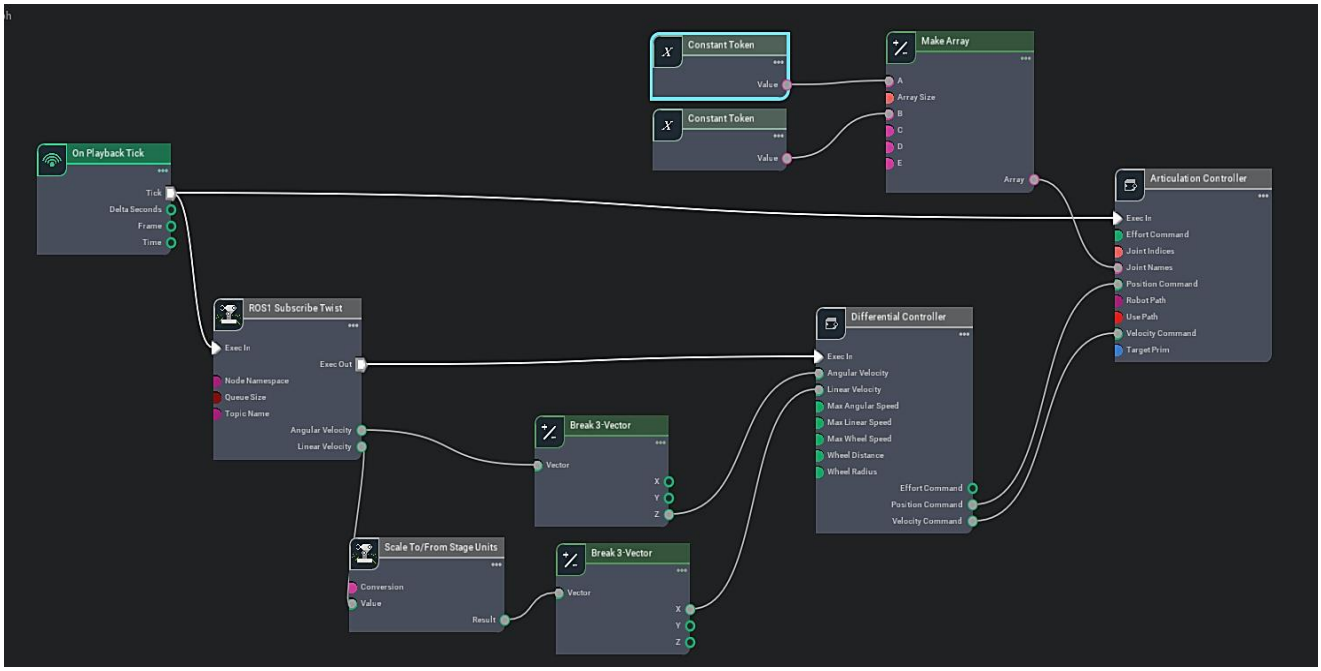
4.3. Upravljanje robotom u IsaacSim-u

U sklopu zadatka potrebno je pomoću ROS čvorova u IsaacSim-u omogućiti kretanje robota i postaviti mu senzore potrebne za mapiranje prostora.

Kao što je ranije rečeno, ROS čvorovi u IsaacSim-u prikazuju se preko *Action Graph*-a. Prvo ga je potrebno stvoriti na način: *Window* > *Visual Scripting* > *Action Graph.*

4.3.1. Pokretanje robota

Graf napravljen za pokretanje robota pomoću tipkovnice prikazan je slikom 14.



Slika 15. Action Graph za pokretanje. Izvor: Autor

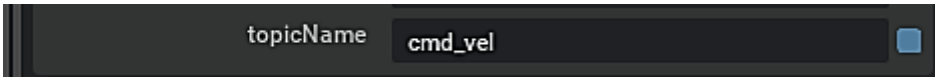
On Playback Tick

čvor koji daje naredbu ostalim čvorovima da izvršavaju svoje funkcije kada je simulacija pokrenuta, tj. pritiskom na *'Play'*.

ROS1 Subscribe Twist

ROS čvor koji se pretplaćuje na temu `/cmd_vel` koja prenosi `geometry_msgs/Twist` poruku. Ta poruka sadrži komponente brzine i zakreta robota u obliku vektora.

U ROS čvoru potrebno je definirati ime teme u polju *topicName* što je vidljivo na slici 15.



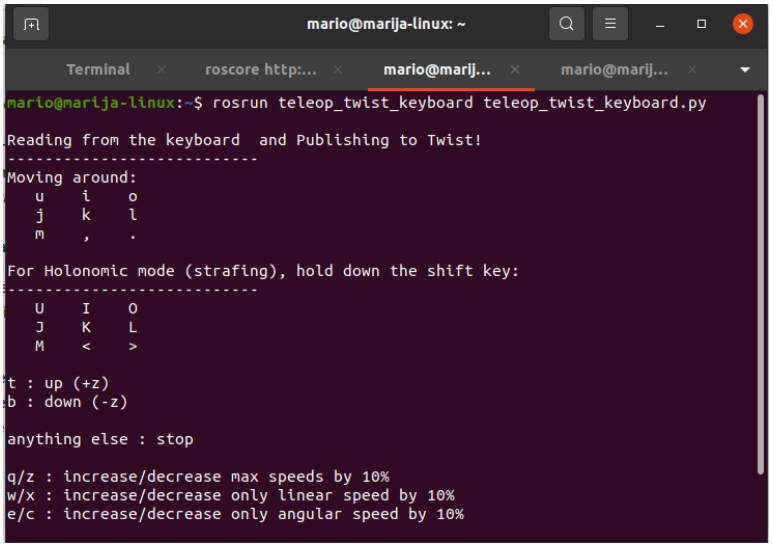
Slika 16. ROS1 Subscribe Twist – topicName. Izvor: Autor

Kako bi se robot pokretao potrebno je objaviti Twist poruke na temu `/cmd_vel`.

ROS paket koji je korišten za pokretanje robota i objavljivanje na temu je *teleop_twist_keyboard* koji se pokreće u Terminalu naredbom:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

te se otvara tipkovnica kojom je moguće upravljati robotom i njegovom linearnom i kutnom brzinom:



Slika 17. teleop_twist_keyboard. Izvor: Autor

Dakle, pokretanjem simulacije u IsaacSim-u, `roscore` i *`teleop_twist_keyboard`* u Terminalu, *ROS1 Subscribe Twist* pretplaćuje se na temu `/cmd_vel`, a `teleop_twist_keyboard` objavljuje na tu temu.

ROS1 Subscribe Twist spojen je sa *Differential Controller* na izlazu i šalje mu signal da je zaprimio poruku o linearnoj i kutnoj brzini.

Scale To/From Stage Units

Pretvara mjerne jedinice zaprimljenih poruka u mjerne jedinice IsaacSim-a. Zadane mjerne jedinice u IsaacSim-u za dužinu su centimetri (cm), a za kut radijani (rad) stoga kutnu brzinu nije potrebno pretvarati.

Break 3-Vector Node

Čvor koji rastavlja trodimenzionalni vektor na komponente. Linearna i kutna brzina u `geometry_msgs/Twist` dolaze u obliku trodimenzionalnih vektora, ali čvor *Differential Controller* prihvaća samo linearnu brzinu u smjeru osi x i zakret oko osi z, stoga je prvo potrebno te vektore rastaviti na komponente.

Differential Controller

Čvor koji prima zahtijevanu brzinu od *ROS1 Subscribe Twist* čvora i računa brzinu okretanja kotača robota. U postavkama čvora potrebno je odrediti radijus kotača i udaljenost između kotača što je prikazano slikom 17.

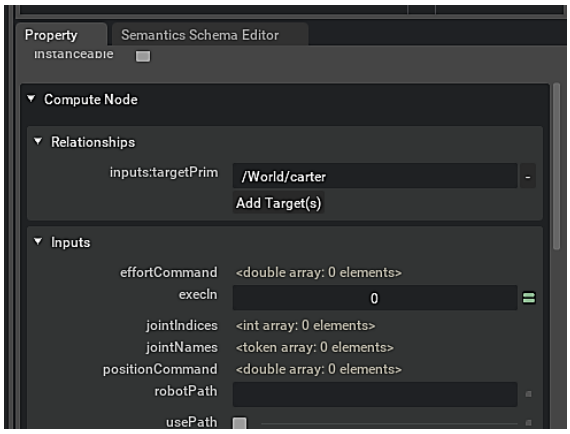
Slika 18. Differential Controller – postavke. Izvor: Autor

Articulation Controller

Čvor koji je zadužen za robota. On uzima imena zglobova definirane u čvoru *Make Array* koje treba pokretati te ih pokreće na način koji mu je zadao *Differential Controller*, u ovom slučaju *Velocity Command* i *Position Command* – zahtjev za brzinu i poziciju zglobova.

Articulation Controller spojen je na čvor *On Playback Tick* što znači da, ako ne dobije nove poruke (ako se putem tipkovnice ne promjeni smjer kretanja robota ili brzina), nastaviti će izvršavati zadnju poruku koju je dobio. To znači da kod pokretanja robota putem tipkovnice nije potrebno cijelo vrijeme držati određenu tipku za pokretanje u određenom smjeru, nego je potrebno pritisnuti samo jednom i robot će se kretati u tom smjeru sve do kad se ne zada drugi smjer.

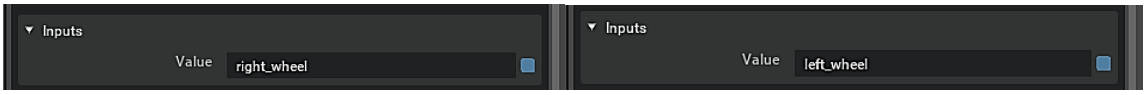
U postavkama čvora potrebno je definirati robota na koji se odnosi pokretanje klikom na *AddTarget* u polju *inputs:targetPrim* i odabirom Carter robota iz stabla što je prikazano slikom 18.



Slika 19. Articulation Controller – postavke. Izvor: Autor

Constant token

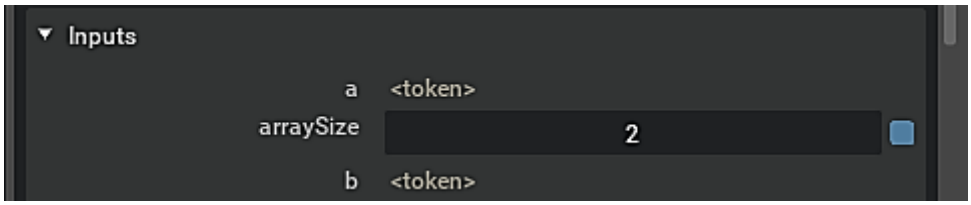
Čvor koji sadrži vrijednost tokena, tj. 64-bitnu vrijednost koja prikazuje internirani niz. U ovom slučaju, *Constant Token* sadrži imena zglobova robota koje pokreće *Articulation Controller*. Imena Carter-ovih zglobova su *right_wheel* i *left_wheel*.



Slika 20. Constant token. Izvor: Autor

Make Array

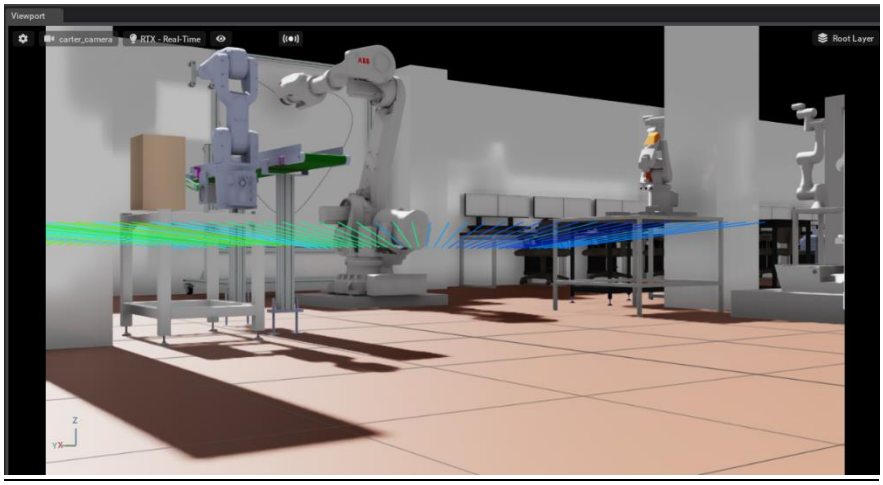
Čvor koji stvara niz vrijednosti. U ovom slučaju, uzima vrijednosti iz *Constant Token-a*, tj. imena zglobova, pretvara ih u niz i taj niz dolazi do *Articulation Controler*-a kako bi on znao koje zglobove pokretati. U postavkama čvora potrebno je definirati veličinu niza u polju *arraySize* koji je u ovom slučaju 2.



Slika 21. Make Array. Izvor: Autor

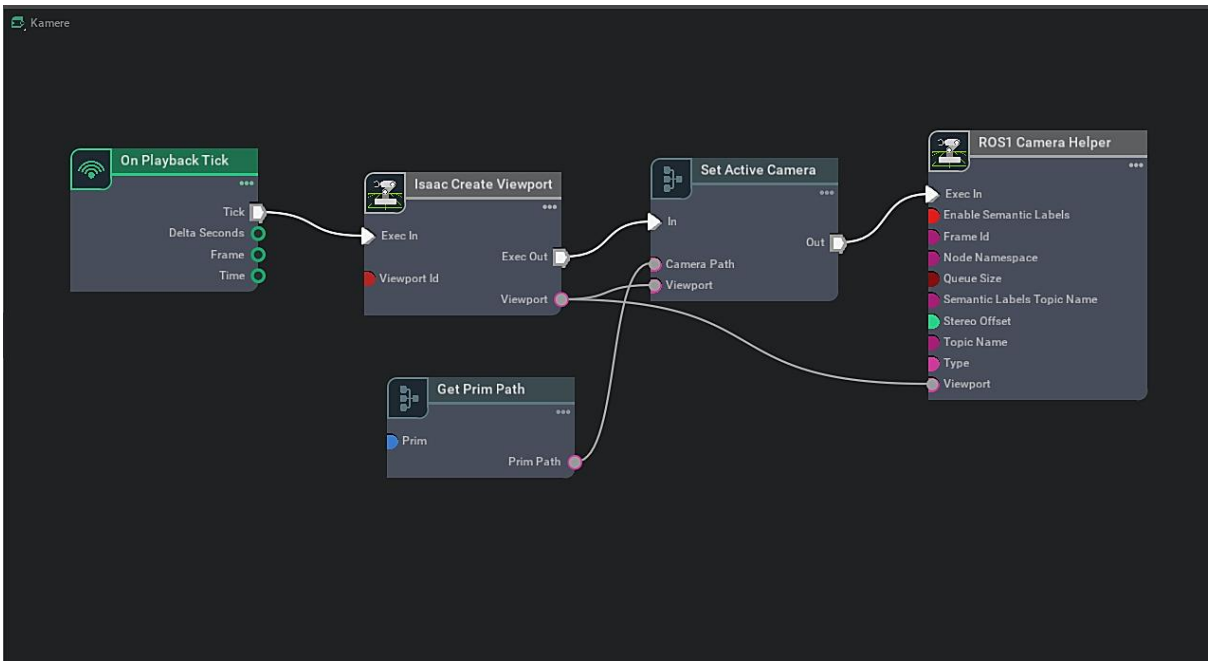
4.3.2. Kamera

Na Carter-a je potrebno dodati kameru za vizualni prikaz. Kamera se dodaje na način *Create > Camera* i potrebno ju je u stablu povući pod *chassis_link*, tj. tijelo robota tako da se kamera kreće zajedno s robotom i namjestiti njen koordinatni sustav na željenu poziciju. Slika 21. prikazuje pogled iz kamere na robotu.



Slika 22. carter_camera. Izvor: Autor

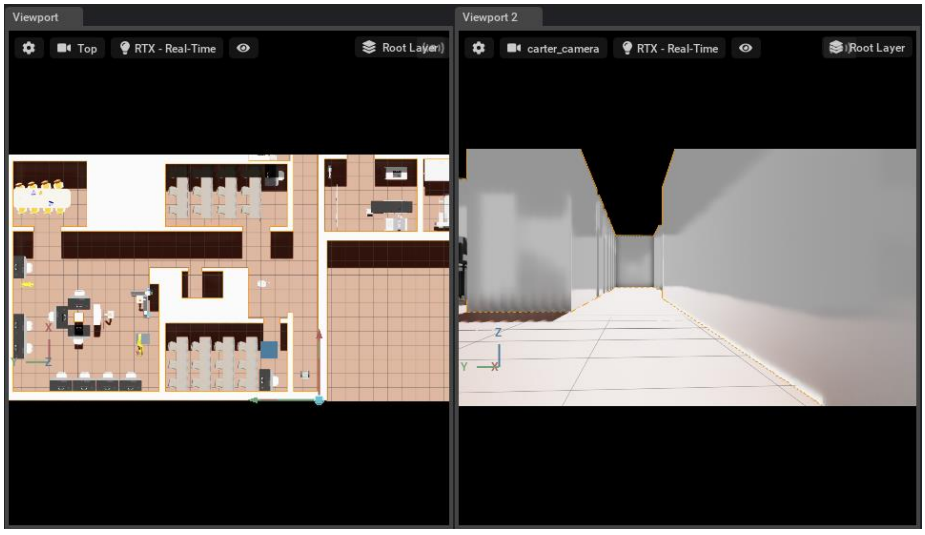
Za povezivanje podataka s kamere i ROS-a potrebno je napraviti *Action Graph* za kameru prikazan slikom 22. ROS prenosi slike putem `/rgb` teme u obliku `sensor_msgs/Image` poruka, stoga je potrebno sliku s kamere iz IsaacSim-a objaviti na `/rgb` temu.



Slika 23. Action Graph za kameru. Izvor: Autor

Isaac Create Viewport

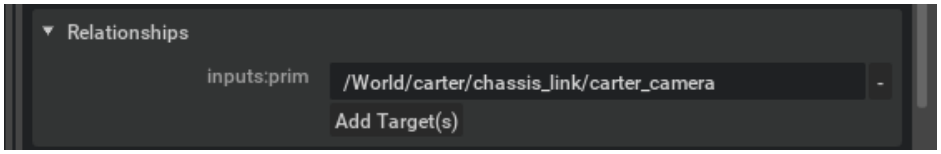
Čvor koji stvara novi prozor. U ovom slučaju, kod pokretanja simulacije stvara se novi prozor s pogledom iz kamere na robotu kako bi se lakše pratilo kretanje i kako bi se iz tog pogleda uzeli renderirani podaci za ROS. U postavkama čvora potrebno je definirati ime tog novog prozora u polju *viewport,* u ovom slučaju *Viewport 2.*



Slika 24. Isaac Create Viewport. Izvor: Autor

Get Prim Path

Čvor koji stvara putanju iz neke veze. U ovom slučaju stvara putanju do kamere robota. u postavkama čvora potrebno je u polju *inputs:prim* klikom na *Add Target* odabrati kameru robota.



Slika 25. Get Prim Path. Izvor: Autor

Set Active Camera

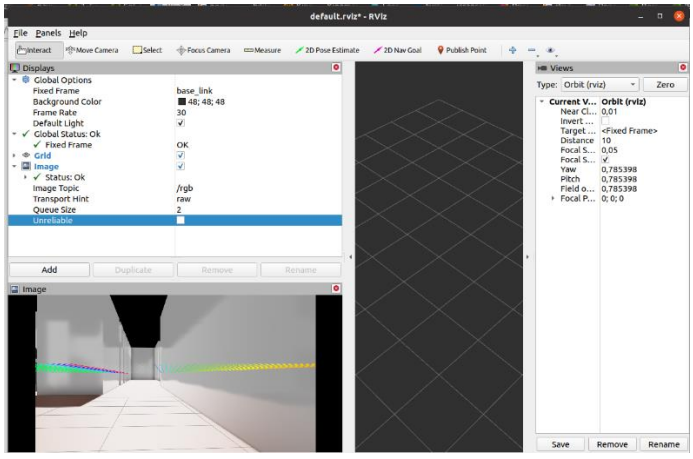
Čvor koji spaja pogled iz kamere, tj. *Viewport 2* s kamerom robota. taj čvor zapravo predstavlja fizičku kameru s koje se uzimaju podaci potrebni za ROS komunikaciju.

ROS1 Camera Helper

ROS čvor koji objavljuje podatke s kamere na temu `/rgb` i aktivira se kad dobije informaciju da je kamera postavljena iz čvora *Set Active Camera*. U postavkama čvora potrebno je postaviti vrstu kamere iz koje se dobivaju podatci i temu na koju se pretplaćuje što je prikazano slikom 25.

Slika 26. ROS1 Camera Helper. Izvor: Autor

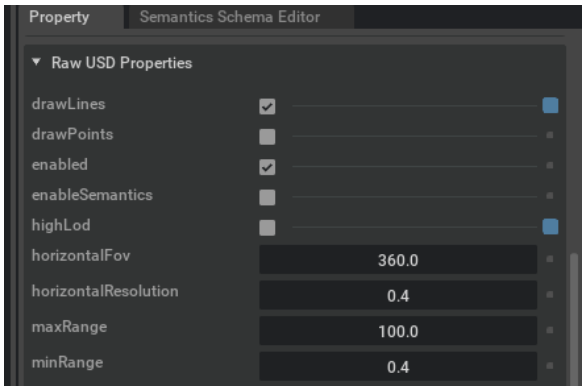
Kako bismo vizualizirali što robot vidi, potrebno je pokrenuti ROS Visualization u Terminalu naredbom `rviz`. Pokretanjem `rviz`-a, on se pretplaćuje na `/rgb` temu i prima `sensor_msgs/Image` poruke od *ROS1 Camera Helper* čvora.



Slika 27. `rviz` – pretplata na `/rgb` temu. Izvor: Autor

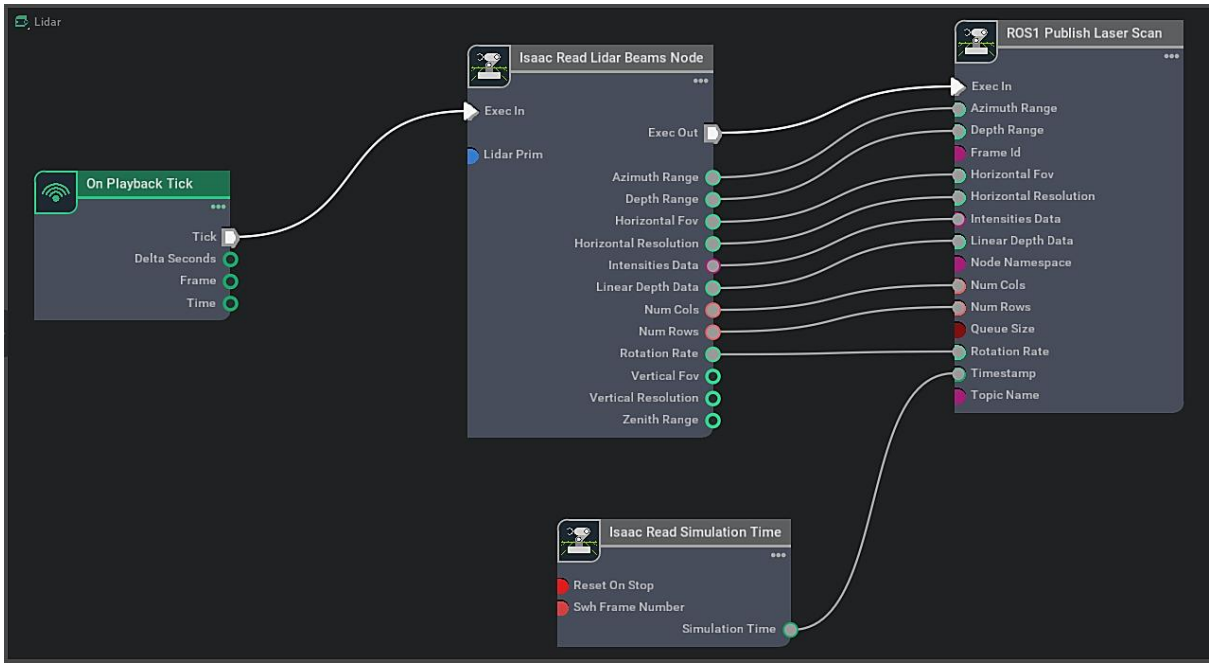
4.3.3. Lidar

Na Carter-a je potrebno dodati i lidar za percepciju okruženja. Lidar se dodaje klikom na *Create > Isaac > Sensors > Lidar > Rotating.* Dodani lidar potrebno je u stablu staviti pod *chassis_link*, tj. tijelo robota i postaviti njegov koordinatni sustav na mjesto gdje se lidar nalazi na robotu. U postavkama lidara potrebno je postaviti *maxRange* na 100, zato što je to maksimalni doseg Velodyne P16 lidara, što znači da će lidar ignorirati sve što je izvan radijusa od 100 metara. Također je poželjno označiti *drawLines* kako bi se vizualizirao sken s lidara. Ostali podatci ostavljeni su kako su zadani u IsaacSim-u.



Slika 28. Postavke lidar-a. Izvor: Autor

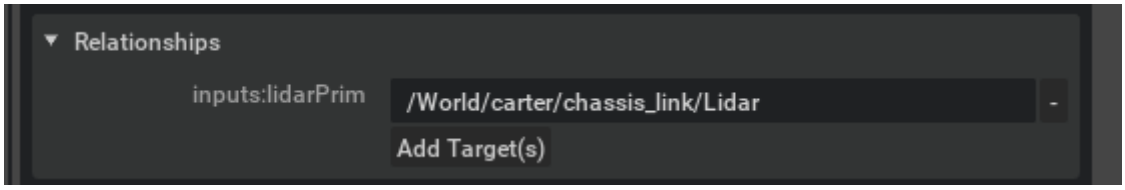
Za povezivanje ROS teme i podataka s lidara, potrebno je napraviti *Action Graf* prikazan slikom 28.



Slika 29. Action graf za lidar. Izvor: Autor

Isaac Read Lidar Beams Node

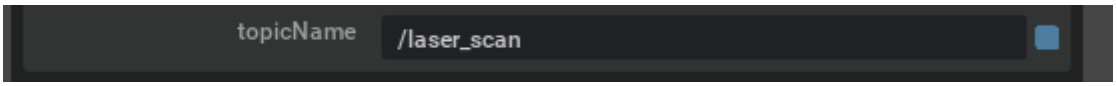
Čvor koji prikuplja informacije o lidaru i podatke s lidara kada se pokrene simulacija. U postavkama čvora potrebno je dodati lidar klikom na *Add Target* u polju *inputs:lidarPrim* kao što je prikazano slikom 29.



Slika 30. Isaac Read Lidar Beams. Izvor: Autor

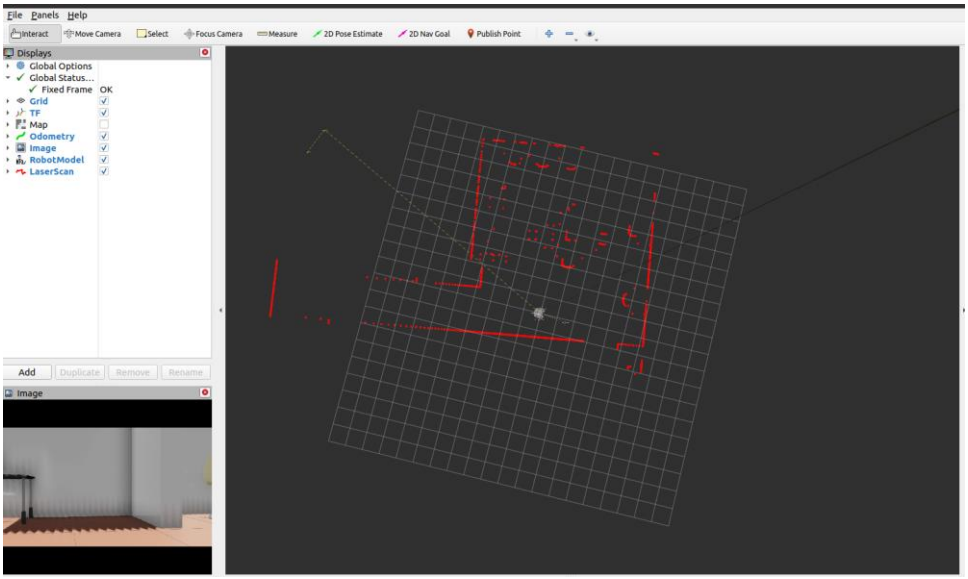
ROS1 Publish Laser Scan

ROS čvor zaslužan za objavljivanje podataka s lidara na ROS temu `/laser_scan`. U postavkama čvora potrebno je definirati temu na koju čvor objavljuje u polju *topicName*.



Slika 31. ROS1 Publish Laser Scan. Izvor: Autor

Za vizualiziranje podataka s lasera, potrebno je otvoriti `rviz` koji se pretplaćuje na temu `/laser_scan` i prima `sensor_msgs/LaserScan` poruke od *ROS1 Publish Laser Scan* čvora.



Slika 32. `rviz` – pretplata na `/laser_scan` temu

Isaac Read Simulation Time

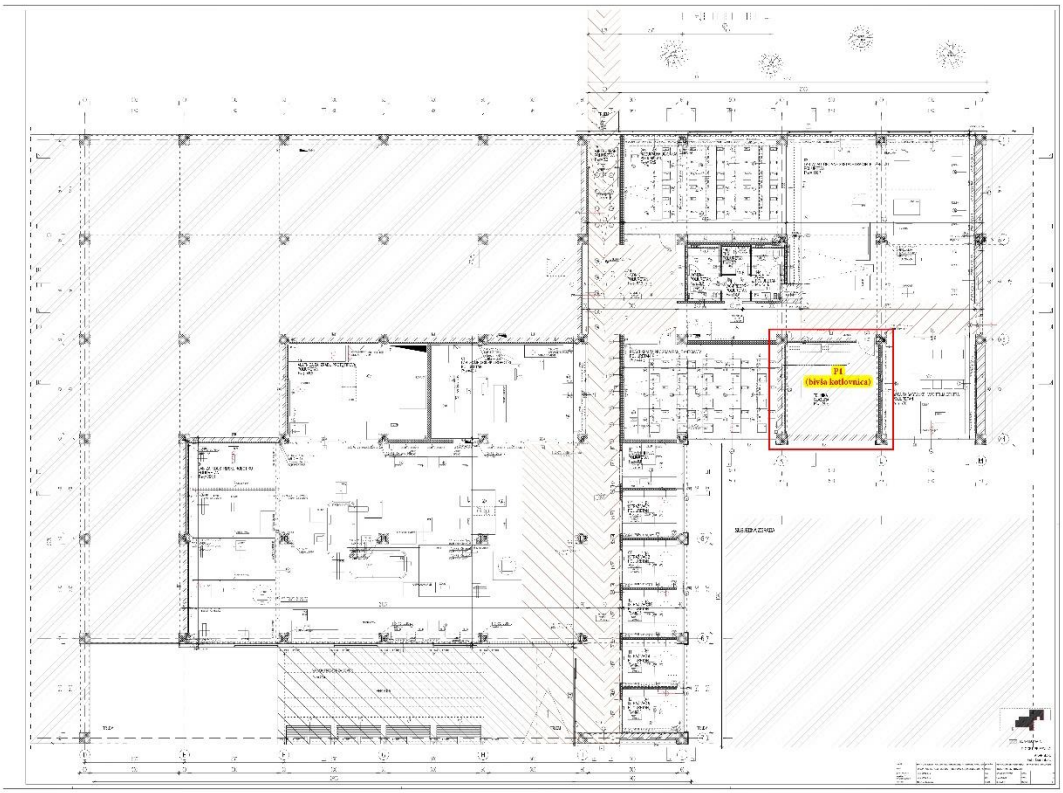
Čvor koji koristi vrijeme iz simulacije kako bi bilježio `sensor_msgs/LaserScan` poruke.

5. VIRTUALNI MODEL REGIONALNOG CENTRA IZVRSNOSTI ZA ROBOTSKE TEHNOLOGIJE

U sklopu zadatka potrebno je napraviti što vjerniji model Regionalnog centra izvrsnosti za robotske tehnologije (CRTA) gdje će se robot kretati i mapirati prostor.

5.1. 2D tlocrt CRTA-e

Model CRTA-e napravljen je prema dobivenom tlocrtu prikazanom slikom ispod.



Slika 33. Tlocrt CRTA-a

CRTA se sastoji od tri laboratorija – laboratorij L1 za računalne inteligencije, laboratorij L2 za autonomne sustave i laboratorij medicinske robotike. U CRTA-i također postoje dva praktikuma za održavanje nastave, alatnica te uredi, kuhinja i konferencijska soba.

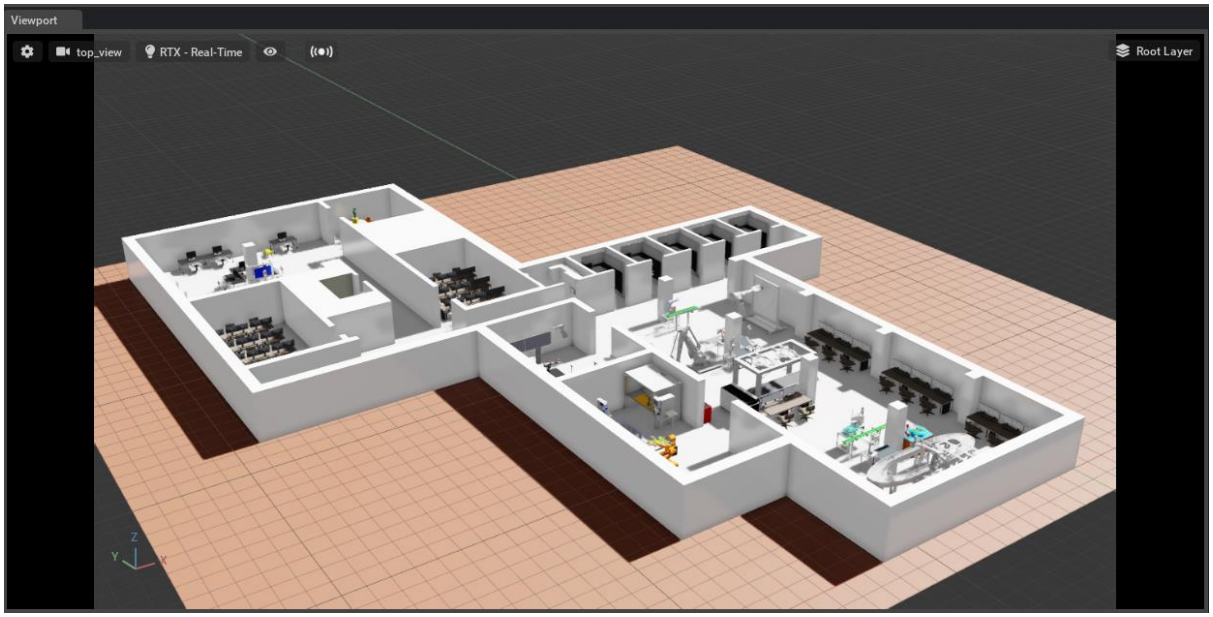
Kako je CRTA centar za robotske tehnologije, u istoj se nalazi velik broj raznih robota koje je potrebno barem približno točno prikazati u 3D modelu kako bi robot mogao generirati što vjerniju mapu prostora u simulatoru koju kasnije može koristiti u stvarnosti.

5.2. 3D model CRTA-e

3D model CRTA-e modeliran je u CAD alatu Autodesk Inventor Professional i prikazan slikom ispod.

Slika 34. Model CRTA-e u Inventor-u

Budući da Inventor pruža mogućnost spremanja modela kao *.usd* datoteke koje koristi IsaacSim, model je spremljen kao *.usd* datoteka i učitan u IsaacSim.
Model CRTA-e u IsaacSim-u prikazan je slikom 35.



Slika 35. Model CRTA-e u IsaacSim-u

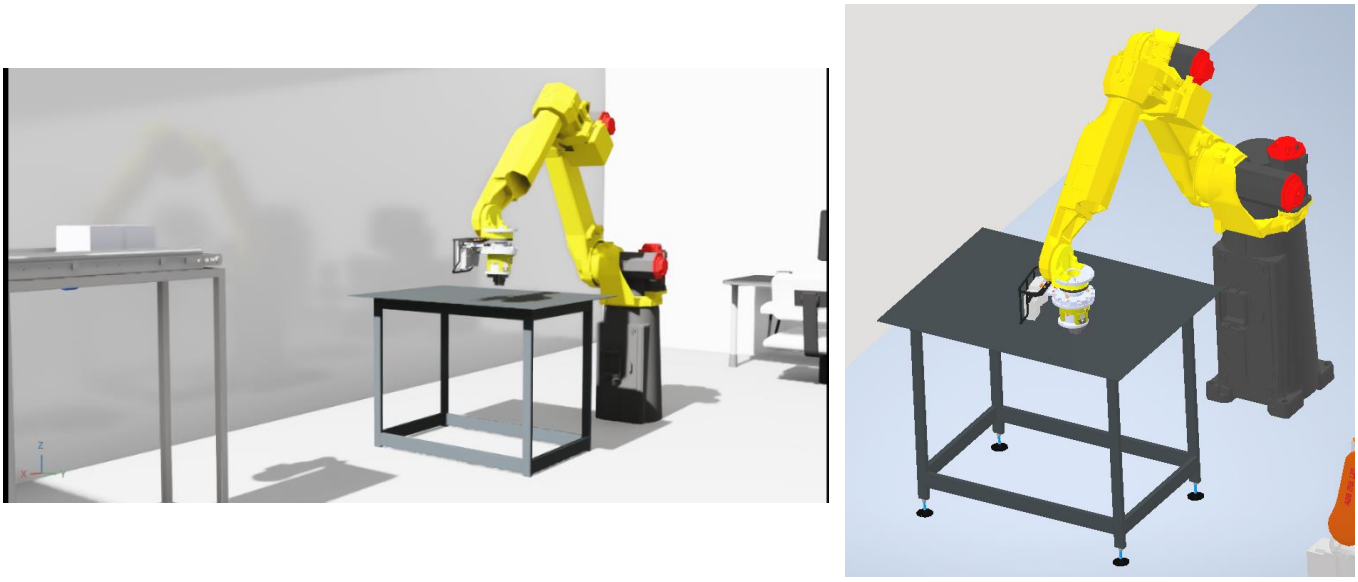
Na prvi pogled vidljiva je razlika u kvaliteti prikaza modela u Inventor-u i IsaacSim-u. Usporedbom modela vidljivo je kako se u IsaacSim-u virtualno okruženje prikazuje puno realnije, što je i glavni cilj i fokus IsaacSim-a.

Virtualno kruženje u IsaacSim-u sastoji se od nekoliko glavnih komponenti. Prva je naravno 3D model okruženja izrađen u nekom od softvera za modeliranje, u ovom slučaju Autodesk Inventor. Virtualno okruženje u IsaacSim-u uključuje i *physical engine* koji simulira fizikalne zakonitosti poput gravitacije, kolizije i trenja. U IsaacSim-u potrebno je dodati kolidere klikom na model CRTA-e u stablu i zatim *Add* > *Physics* > *Colliders Present* čime se model označava kao kruto tijelo. Taj korak potreban je kako bi robot znao gdje su mu prepreke i kako ne bi prolazio kroz njih.

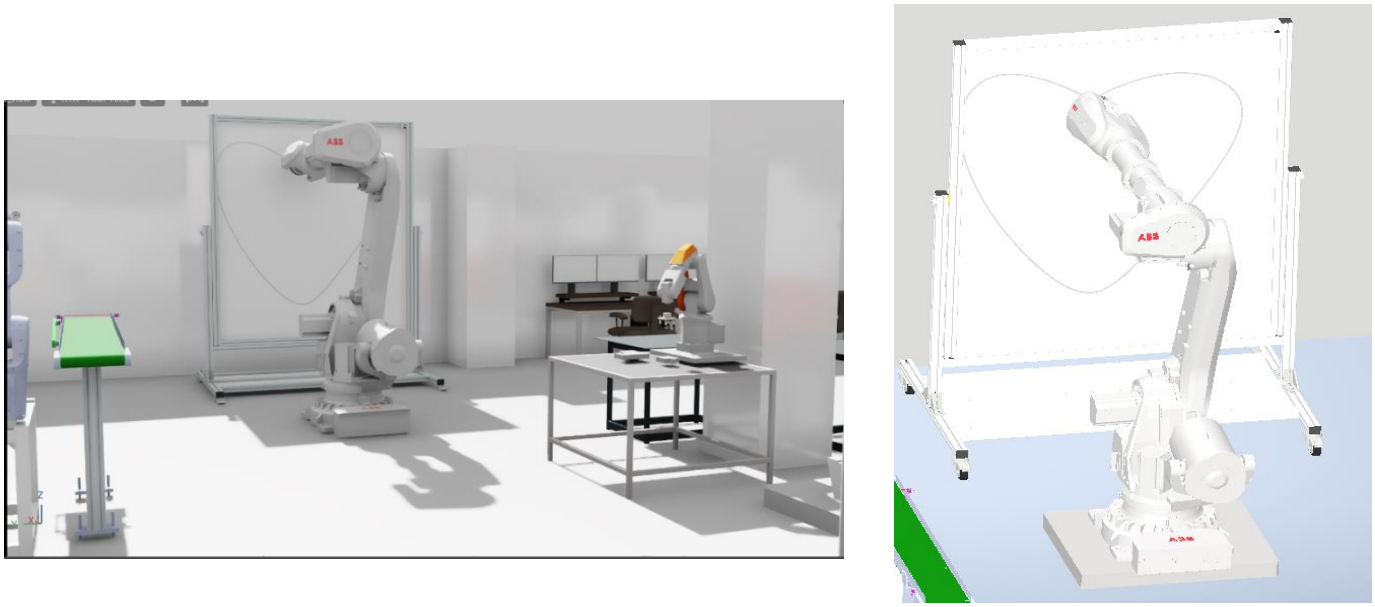
Virtualno okruženje također uključuje simulirano osvjetljenje i renderiranje čime se osigurava vizualni izgled i karakteristike stvarnog svijeta. Kod učitavanja modela automatski se postavlja *defaultLight*, no moguće je dodati različite vrste osvjetljenja poput kupolastog ili udaljenog osvjetljenja koje je također moguće prilagoditi prema vlastitim preferencijama. U ovom slučaju korišteno je kupolasto osvjetljenje ili *Dome Light*.

Slika 36. Osvjetljenje u IsaacSim-u

Slikama 37. i 38. prikazani su detalji iz IsaacSim virtualnog okruženja i Invnetor-a. Na slikama su prikazani isti modeli robota u IsaacSim-u i Inventor-u te je vidljivo koliko realističan prikaz pruža IsaacSim.



Slika 37. Model robota u IsaacSim-u i Inevntor-u



Slika 38. Model robota u IsaacSim-u i Inventor-u

6. MAPIRANJE PROSTORA

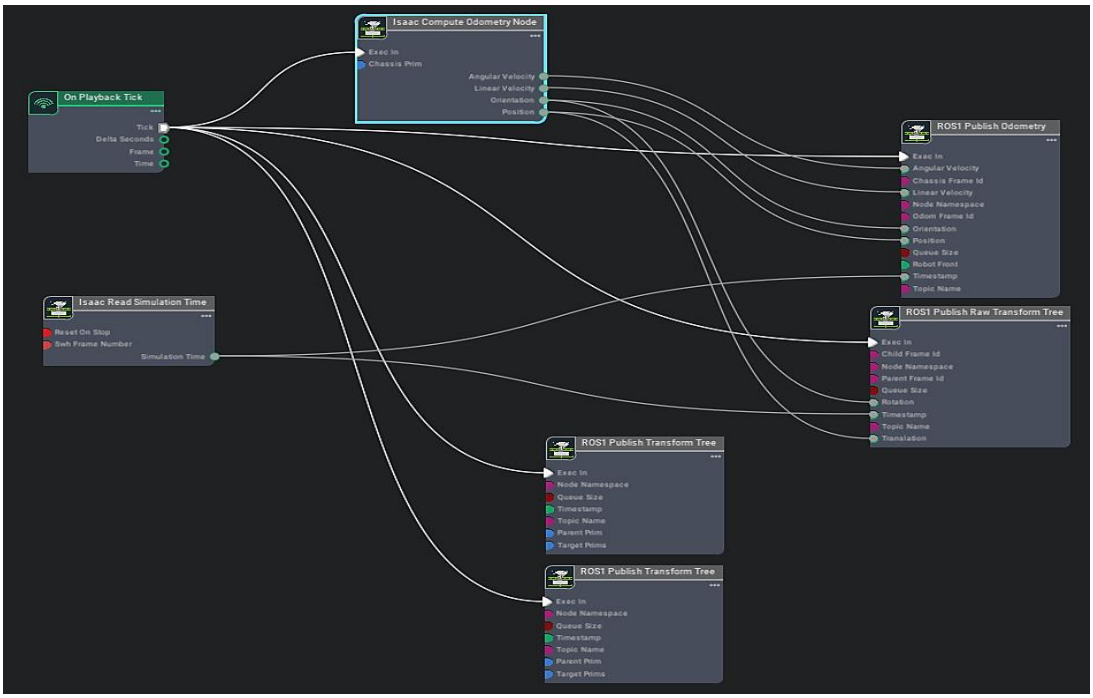
Mapiranje je proces stvaranja mape koju mogu čitati i čovjek i mobilni robot. Mobilni robot izrađuje mapu prostora koristeći odgovarajuće senzore kao što su kamere i lidar senzori, a izrađena mapa se kasnije može koristiti za lokalizaciju i navigaciju.

Jedna od najčešće korištenih paketa za mapiranje je SLAM (*Simultaneous Localization and Mapping*) metoda. SLAM metoda dostupna je za preuzimanje s ROS platforme i omogućuje konstruiranje i ažuriranje mape uz istovremeno praćenje lokacije robota. U sklopu SLAM metode postoji mnogo različitih algoritama za mapiranje prostora poput *GMapping, Core SLAM, Graph SLAM i Hector SLAM* algoritma. Za mapiranje prostora u ovom radu korišten je *gmapping* paket, točnije `slam_gmapping`. [17]

6.1. Stablo transformacija i odometrija

`slam_gmapping` paket koristi očitanja s lasera i odometriju kako bi izgradio kartu prostora. Odometrija je način na koji robot određuje svoju poziciju u prostoru u odnosu na neku poznatu poziciju. Djeluje na način da se koriste enkoderi kako bi se odredila pozicija, tj. pređena udaljenost od početne pozicije mobilnog robota.

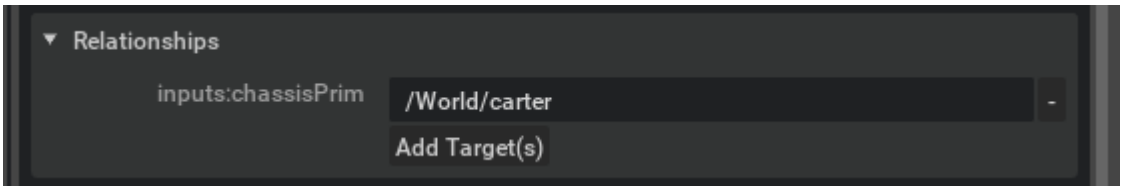
Za mapiranje prostora potrebno je odrediti transformacije između koordinatnih sustava robota. `slam_gmapping` pretplaćuje se, uz `/laser_scan`, i na `/tf` temu koja sadrži transformaciju laser → baza robota i transformaciju baza robota → odom (početni koordinatni sustav u odnosu na kojeg se enkoderima mjeri pozicija).



Slika 39. Action Graph za transformacije i odometriju. Izvor: Autor

Isaac Compute Odometry

Čvor koji je zaslužan za izračunavanje trenutne pozicije robota. U postavkama čvora potrebno je klikom na *Add Target* u polju *ChassisPrim* postaviti Carter-a.



Slika 40. Isaac Compute Odometry. Izvor: Autor

Izlazni podaci iz čvora su `nav_msgs/Odometry` poruke, tj. procijenjena kutna i linearna brzina te orijentacija i pozicija robota.

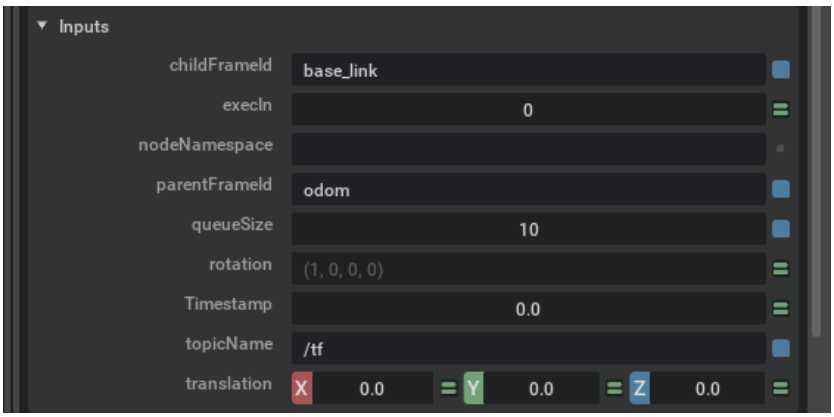
ROS1 Publish Odometry

ROS čvor koji je zaslužan za objavljivanje odometrije, tj. `nav_msgs/Odometry` poruka na ROS temu `/odom`. U postavkama čvora potrebno je u polje *chassisFrameId* staviti base_link jer je to koordinatni sustav koji je vezan za robota i s obzirom na njega je uzeta linearna i kutna brzina. U polje *odomFrameId* potrebno je staviti odom jer je to koordinatni sustav koji predstavlja svijet i s obzirom na njega se izračunava pozicija i rotacija robota. U polje *topicName* potrebno je staviti `/odom` budući da je to tema na koju se objavljuje odometrija.

Slika 41. ROS1 Publish Odometry. Izvor: Autor

ROS1 Publish Raw Transform Tree

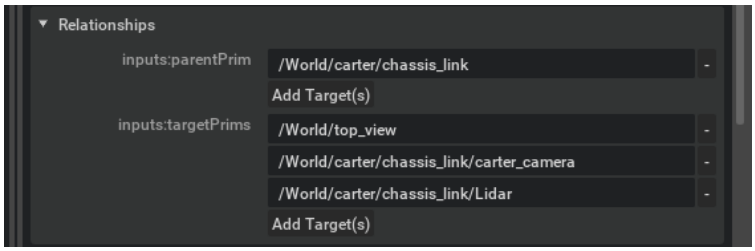
ROS čvor zaslužan za individualne transformacije, u ovom slučaju za transformaciju između base_link (polje *childFrameId*) koordinatnog sustava i odom (polje *parentFrameId*) koordinatnog sustava. Ta transformacija potrebna je kako bi se mogla znati pozicija robota i objavljuje se na ROS temu `/tf` koju je potrebno definirati u polju *topicName.*



Slika 42. ROS1 Publish Raw Transform Tree. Izvor: Autor

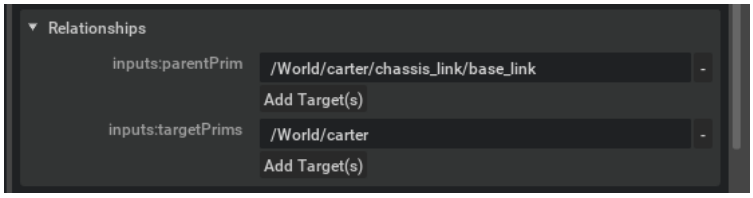
ROS1 Publish Transform Tree

ROS čvor zaslužan za ostale transformacije. Za praćenje pozicije kamera i lidara u globalnom koordinatnom sustavu, potrebno je napraviti relativnu transformaciju između koordinatnog sustava tijela robota chassis_link i njihovih koordinatnih sustava.



Slika 43. ROS1 Publish Transform Tree 1. Izvor: Autor

Nakon toga potrebno je napraviti transformaciju između svih koordinatnih sustava na robotu i base_link koordinatnog sustava kako bi se sve što je povezano s robotom moglo objaviti na `/tf` temu.



Slika 44. ROS1 Publish Transform Tree 2. Izvor: Autor

6.2. *gmapping*

ROS paket *gmapping* zaslužan je za mapiranje prostora na način da stvara mapu popunjenosti (eng. *grid occupany map*) korištenjem podataka sa senzora. Paket koristi *GMapping* algoritam koji se temelji na *Rao-Blackwellized* filteru čestica. Ovaj pristup koristi filter čestica u kojem svaka čestica nosi dio mape prostora. [18]

Gmapping čvor pokreće se u Terminalu na sljedeći način:

```
rosrun gmapping slam_gmapping scan:=laser_scan
```

Pokretanjem čvora on se pretplaćuje na `/tf` i `/laser_scan` teme od kojih dobiva podatke o transformacijama koordinatnih sustava i očitanja s lidara i njegove parametre koji su mu potrebni za kreiranje mape.

Teme koje objavljuje su:

`/map_metadata` tema koja sadrži opće podatke vezane za generiranu mapu

`/map` tema koja sadrži podatke koji predstavljaju 2D mapu prostora

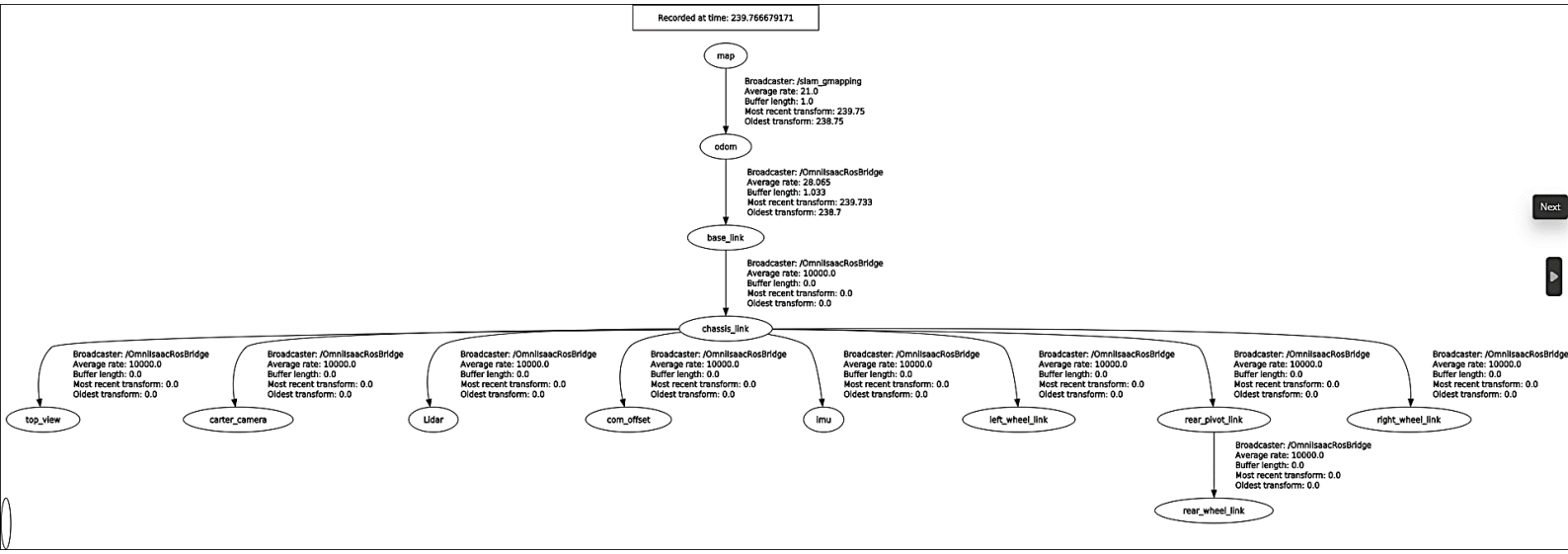
`/entropy` tema koja sadrži podatke o procijenjenoj entropiji

Više o *gmapping* čvoru i parametrima koje je potrebno postaviti vidjeti na [y].

Da bi adekvatno radio, *gmapping* zahtijeva transformacije *laser_scan* → *base_link* i *base_link* → *odom* koje su osigurane u prošlom koraku, a on osigurava transformaciju *map* → *odom* kojom se određuje pozicija robota unutar kreirane mape.

Slikom ispod prikazane su sve transformacije koje su potrebne za mapiranje prostora, a slika se dobije pokretanjem naredbe u terminalu:

```
rosrun rqt_tf_tree rqt_tf_tree
```



Slika 45. Stablo transformacija. Izvor: Autor

Tablica 4. prikazuje pregled ROS tema i ROS poruka korištenih u simulaciji.

Tablica 4. Pregled ROS tema

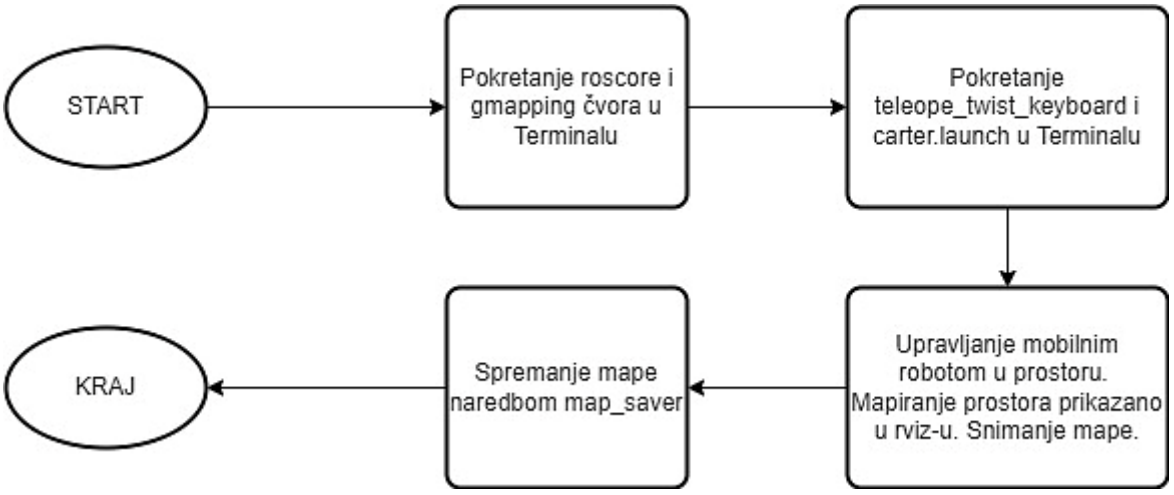
ROS tema	ROS poruka	FUNKCIJA
`/cmd_vel`	`geometry_msgs/Twist`	Sadrži podatke o kutnoj i linearnoj brzini robota
`/laser_scan`	`sensor_msgs/LaserScan`	Sadrži podatke dobivene sa senzora
`/rgb`	`sensor_msgs/Image`	Sadrži podatke o slici dobivenoj s kamere
`/odom`	`nav_msgs/Odometry`	Sadrži podatke o približnoj pozicija i brzina robota u prostoru
`/tf`	`tf2_msgs/TFMessage`	Sadrži podatke o transformacijama koordinatnih sustava
`/map`	`nav_msgs/OccupancyGrid`	Predstavlja 2D mrežnu mapu u kojoj svako polje predstavlja vjerojatnost okupiranja

Pokretanjem simulacije u IsaacSim-u, aktiviraju se sljedeći ROS čvorovi:

Tablica 5. Omnigraph ROS čvorovi

OMNIGRAPH ČVOR	FUNKCIJA
ROS1 Subscribe Twist	Pretplaćuje se na `/cmd_vel` temu te daje signal diferencijalnom i artikulacijskom kontroleru za pokretanje robota
ROS1 Publish Odometry	Objavljuje odometriju dobivenu od *Isaac Compute Odometry* čvora
ROS1 Publish Raw Transform Tree	Objavljuje transformaciju između *odom* i *base_link* koordinatnog sustava na `/tf` temu
ROS1 Publish Transform Tree 1	Objavljuje statičku transformaciju između *base_link* i *carter* koordinatnih sustava (*carter* sadrži sve ostale koordinatne sustave)
ROS1 Publish Transform Tree 2	Objavljuje statičku transformaciju između *chassis_link* i *lidar* koordinatnih sustava
ROS1 Publish Laser Scan	Objavljuje 2D sken s lasera dobiven od *Isaac Read Lidar Beam* čvora na `/laser_scan` temu

Nakon pokretanja simulacije moguće je pokrenuti proces mapiranja prostora. Prostor koji je mapiran je Regionalni centra izvrsnosti za robotske tehnologije (CRTA) čiji model je prikazan i opisan u poglavlju 5.



Slika 46. Shema postupka. Izvor: Autor

Slika 37. shematski prikazuje postupak primijenjen u ovom radu za mapiranje prostora.

Nakon pokretanja simulacije u terminalu se prvo pokreće `roscore` kao ROS master. Zatim je potrebno pokrenuti *gmapping* čvor kako bi se omogućilo mapiranje prostora.

Kako bi se moglo upravljati robotom pomoću tipkovnice, potrebno je pokrenuti `teleop_twist_keyborad`.

Na kraju se pokreće *carter.launch* datoteka prikazana u nastavku.

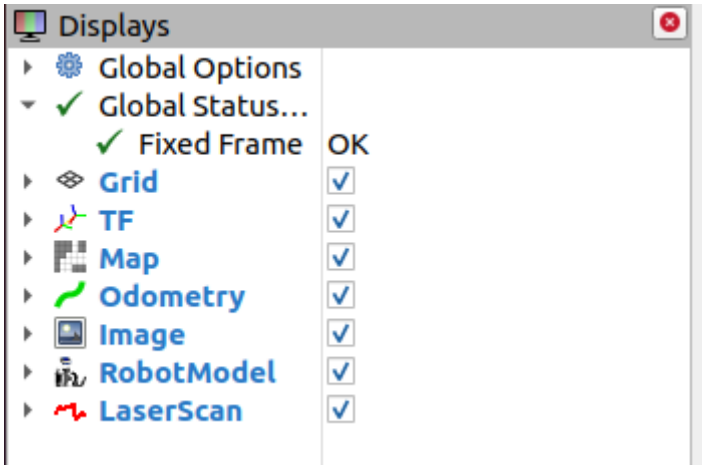
```
  <launch>
      <param name="use_sim_time" value="true" />

      <!-- Load Robot Description -->
      <arg name="model" default="$(find carter_description)/urdf/carter.urdf"/>
      <param name="robot_description" textfile="$(arg model)" />

      <node type="rviz" name="rviz" pkg="rviz" args="-d $(find
carter_2dnav)/rviz/carter_2dnav2.rviz" />

  </launch>
```

Pokretanjem datoteke *carter.launch* učitava se opis mobilnog robota iz njegovog *.urdf* modela kako bi se on mogao prikazati u `rviz`-u. Zatim se pokreće `rviz` sa svim potrebnim argumentima za mapiranje prostora definiranim u *carter_2dnav2.rviz.*

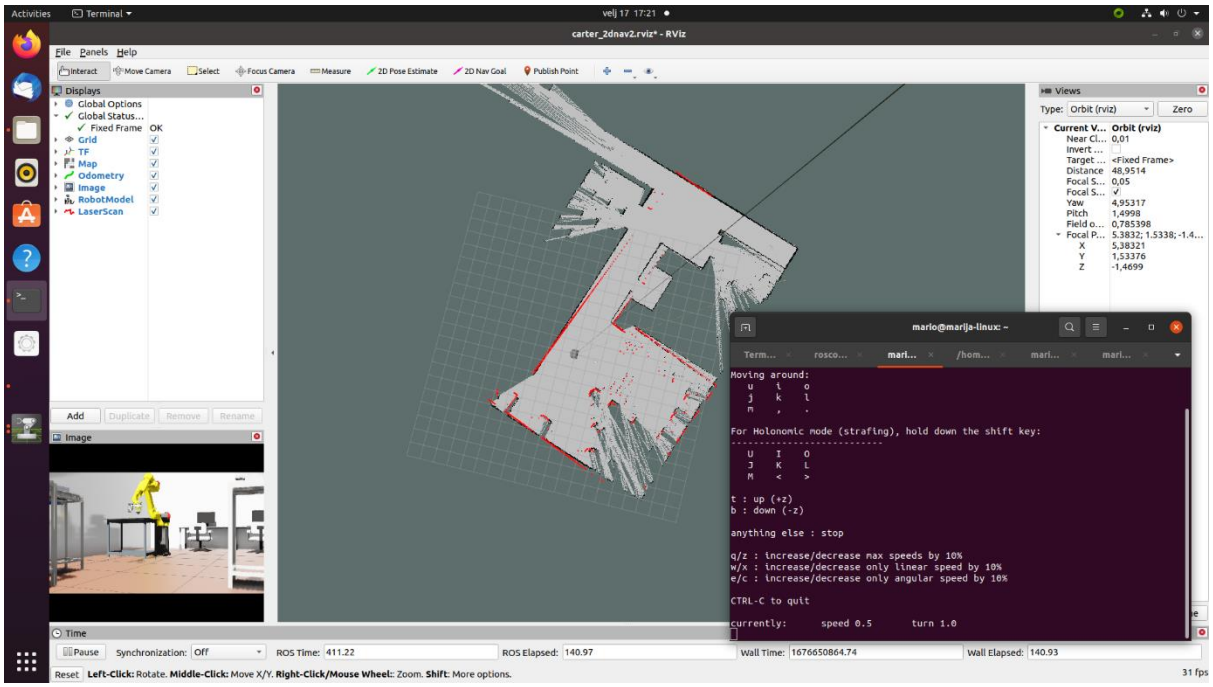


Slika 47. Argumenti koji se objavljuju u `rviz`-u. Izvor: Autor

Za snimanje mape prostora potrebno je u terminalu pokrenuti naredbu:

```
rosbag record -O mylaserdata /laser_scan /tf
```

Nakon pokretanja svih čvorova potrebno je voziti robota po okruženju i u rviz-u se postepeno generira i snima mapa prostora što je prikazano slikom ispod.



Slika 48. Proces mapiranja. Izvor: Autor

Nakon što se izgenerira cijela mapa prostora, istu je potrebno spremiti kako bi se mogla koristiti kasnije, pokretanjem čvora `map_saver` u sklopu paketa `map_server`:

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

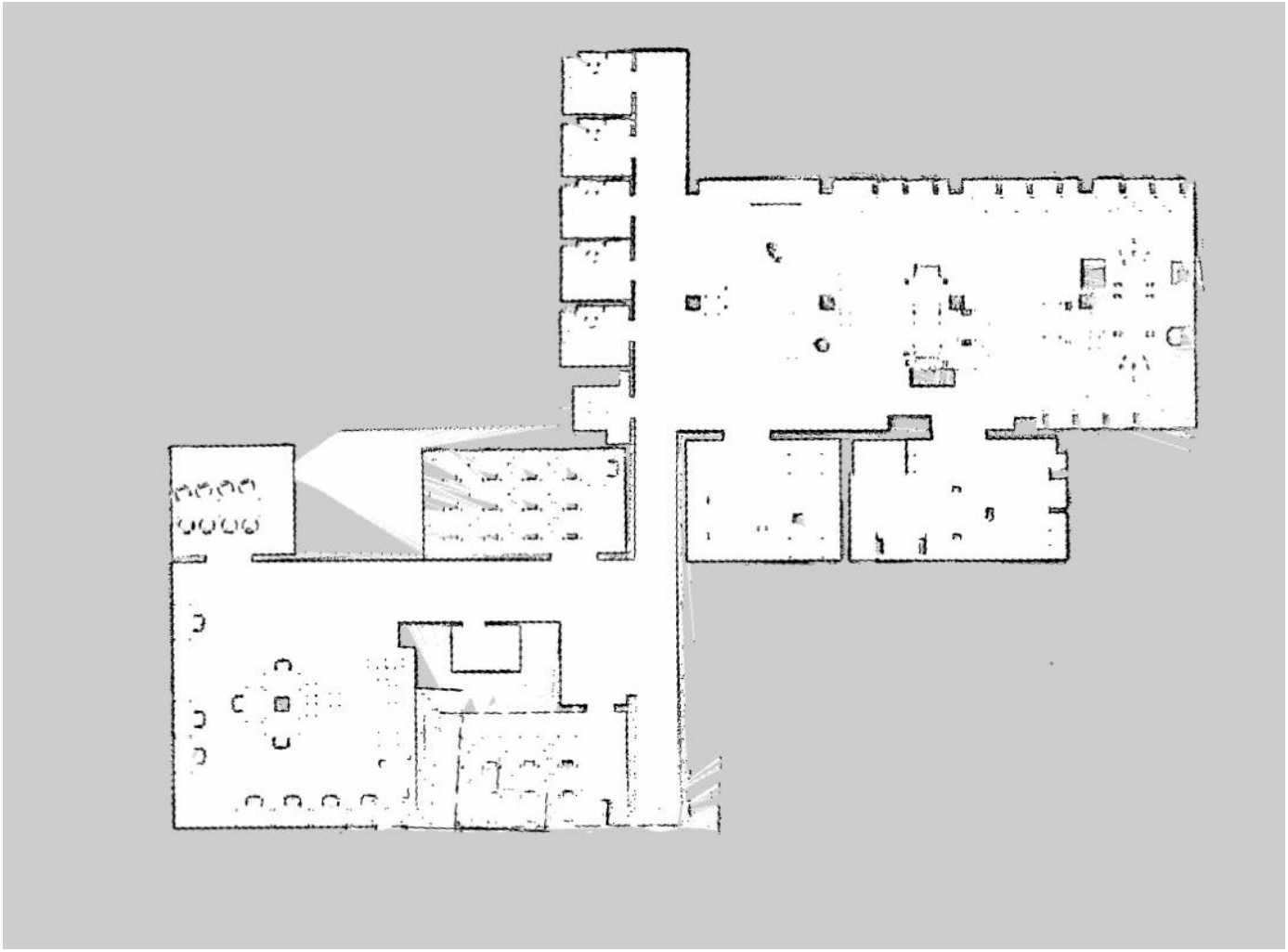
Spremanjem mape prostora stvaraju se *.yaml* datoteka i *.pgm* datoteka. *.yaml* datoteka sadrži parametre vezane uz izgeneriranu mapu, a *.pgm* sadrži izgled mape.

Parametri dobiveni spremanjem *.yaml* datoteke su:

- *image* – putanja do *.pgm* datoteke
- *resolution* – rezolucija mape
- *origin* – koordinate ishodišta mape
- *negate* – parametar za zamjenu slobodno/okupiranog prostora mape
- *occupied_tresh* – parametar koji se uzima kao donja granica za okupiranost prostora, pikseli s vjerojatnošću okupiranosti većom od te vrijednosti smatraju se okupiranima
- *free_tresh* – parametar koji se uzima kao gornja granica za određivanje slobodnog prostora, pikseli koji imaju vjerojatnost okupiranosti manju od ove vrijednosti smatraju se slobodnim prostorom

```
image: CRTA.pgm
resolution: 0.050000
origin: [-100.000000, -100.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Kako bi se dobivena mapa prostora mogla prikazati u ovom radu, iz *.pgm* datoteke pretvorena je u *.jpg* datoteku.

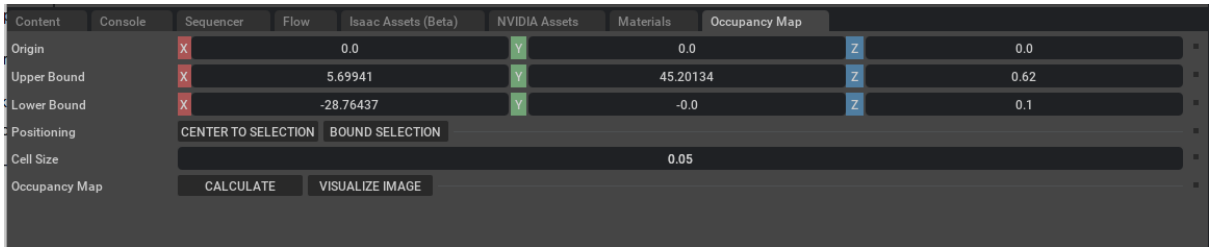


Slika 49. Tlocrt CRTA-e dobiven procesom mapiranja. Izvor: Autor

6.3. *Occupancy Map Generator* ekstenzija

U IsaacSim-u postoji ekstenzija za generiranje 2D mape okupiranosti *Occupancy Map Generator*. Ekstenzija radi na način da generira binarnu mapu okupiranosti prostora na zadanoj visini. Za određivanje ukoliko je neko područje okupirano ili nije koristi fizikalne podatke postavljene ranije (*Colliders Present*).

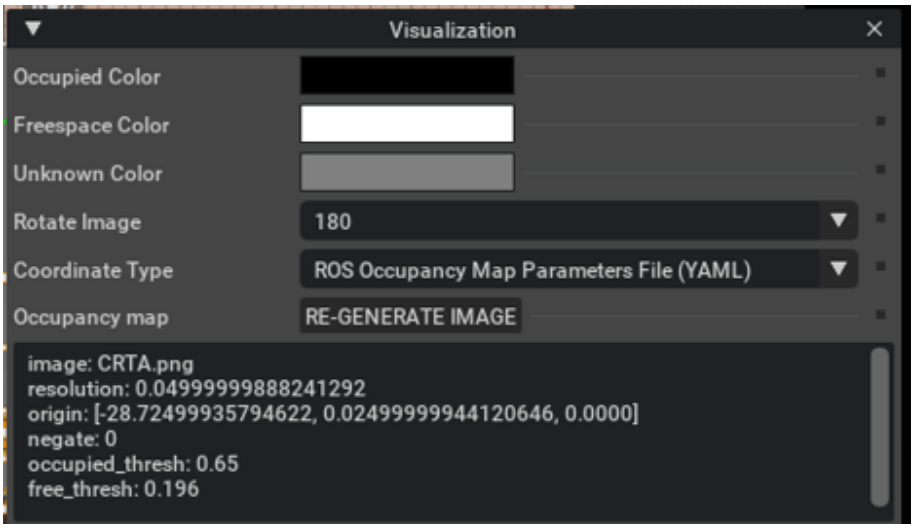
Za pristup ekstenziji potrebno je ići na *Isaac Utils* > *Occupancy Map* čime se otvara prozor prikazan slikom 50.



Slika 50. *Occupancy Map* ekstenzija

- *Origin* – neokupirana lokacija unutar prostora koji se želi mapirati
- *Lower/Upper Bound* – područje unutar kojeg se provodi mapiranje

Prvo je u stablu potrebno označiti model CRTA-e. Za donju granicu ručno je stavljeno z = 0.1, a za gornju z = 0.62 jer je to udaljenost Carter-ovog lidara od poda. Zatim je potrebno kliknuti na *Bound Selection* čime se na temelju zadanih podataka i modela crte izračunavaju ostali parametri za određivanje granica okupiranog prostora.
Nakon toga potrebno je kliknuti na *Calculate,* a nakon toga na *Visualize Image* čime se otvara prozor s mapom okupiranog prostora prikazan slikom 51.

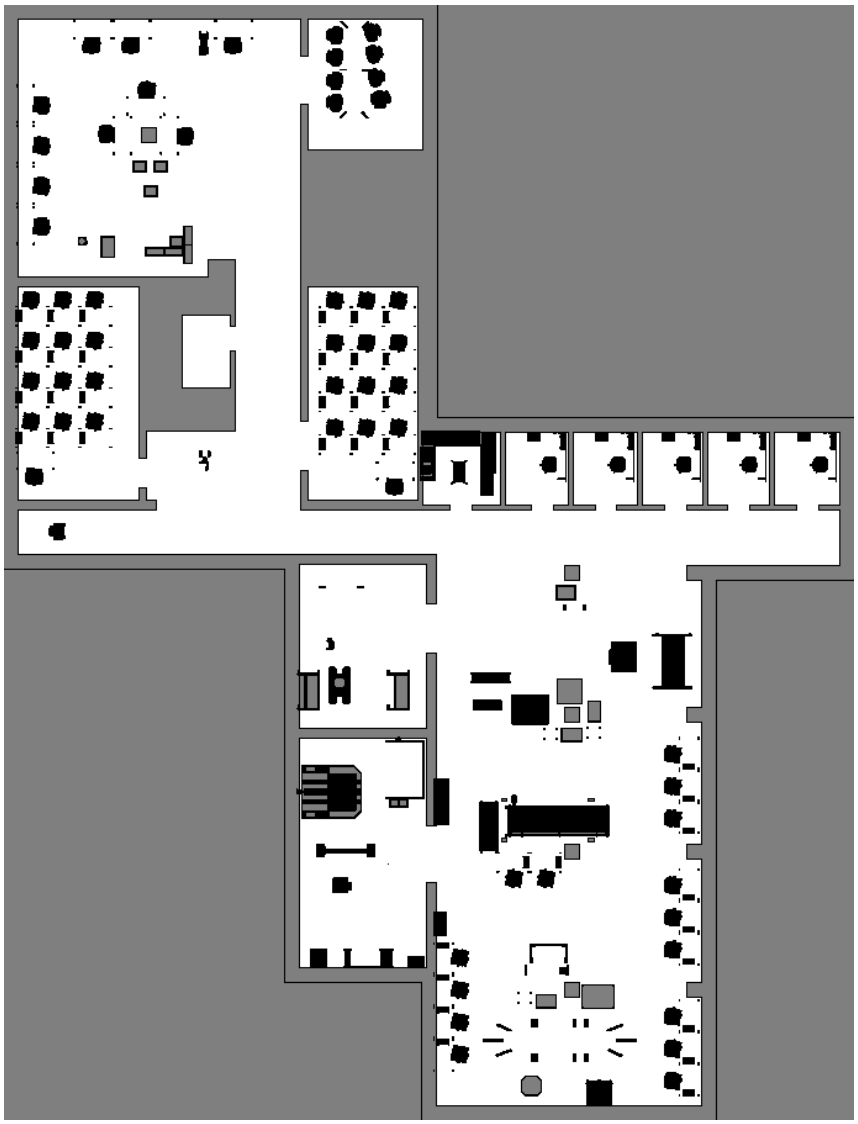


Slika 51. *Occupancy Map* ekstenzija

U novootvorenom prozoru vidljiva je mapa okupiranosti prostora. Crnom bojom označen je okupirani prostor, bijelom slobodan prostor, a sivom nepoznat prostor.

Dobivena mapa rotirana je za 180 stupnjeva i pod *Coordinate Type* postavljeno je *ROS Occupancy Map Parameters File (YAML)* kako bismo dobili parametre *.yaml* datoteke. Zatim je potrebno kliknuti na *Re-generate Image* čime se osvježi slika te se iznad nje izgeneriraju *.yaml* paramteri koje je potrebno prebaciti u ručno napravljenu datoteku.

Klikom na *Save Image* slika se sprema na računalo u *.png* obliku. Mapa dobivena *Occupancy Map* ekstenzijom u IsaacSim-u prikazana je slikom 52.



Slika 52. Mapa prostora dobivena ekstenzijom u IsaacSim-u. Izvor: Autor

Izgenerirane mape približno su jednake, samo što mapa dobivena *Occupancy Map* ekstenzijom ima čišću sliku no potrebno je vjerno modelirati okruženje, dok kod mapiranja *gmapping*-om može doći do pogreška uzrokovanih naglim okretanjima i velikim brzinama.

7. ZAKLJUČAK

U ovom radu prikazan je postupak integracije Robotskog operativnog sustava u IsaacSim virtualnom okruženju na primjeru upravljanja mobilnim robotom. U prvom dijelu rada napravljena je usporedba IsaacSim i Gazebo virtualnog okruženja čime se pokazalo da izbor okruženja za programiranje robota ovisi o specifičnim zahtjevima problema. IsaacSim pruža realističan prikaz okruženja koji daje osjećaj stvarnosti što omogućuje lakše treniranje robota u simulaciji čime se reduciraju pogreške koje se inače javljaju kod prebacivanja programa na stvarnog robota. S druge strane, Gazebo je modularniji, jednostavnija je integracija s ROS-om te postoji velika knjižnica algoritama za pomoć korisnicima oko specifičnih problema. Programe iz Gazebo-a je također moguće prebaciti na skoro sve vrste hardvera, dok IsaacSim podržava prebacivanje samo na robota s NVIDIA hardverom.

Zatim je dan pregled ROS-a i njegove integracije sa simulatorskim okruženjima. Korištenjem ROS-a u simulatorima omogućeno je sigurno i kontrolirano testiranje algoritama u virtualnom okruženju prije implementacije na pravog robota. Izrada programa korištenjem ROS-a u IsaacSim-u izvodi se pomoću grafova što omogućuje vizualizaciju procesa i jednostavnije programiranje, budući da nije potrebno pisati kod.

U sljedećem dijelu rada dan je pregled mobilnih robota i njihovih funkcija te je objašnjena kinematika diferencijalnog modela mobilnog robota. Diferencijalni model je najjednostavniji model kinematike robota i pretpostavlja da se robot kreće na način da mu se okreću dva kotača na istoj osi pokretana različitim motorima. NVIDIA Carter je mobilni robot diferencijalne kinematike koji je korišten za potrebe simulacije ovog rada.

Zadatak rada je implementacija ROS čvorova za upravljanje mobilnim robotom u IsaacSim-u te mapiranje prostora. Za potrebe mapiranja prostora, izrađen je virtualni model Regionalnog centra izvrsnosti za robotske tehnologije (CRTA) u Autodesk Inventor-u. Učitavanjem modela u IsaacSim vidljive su njegove sposobnosti realističnog prikaza okruženja korištenjem osvjetljenja i realističnim renderiranjem. Za upravljanje robotom i mapiranje prostora bilo je potrebno napraviti grafove za povezivanje s *cmd_vel*, *rgb*, *laser_scan* i *tf* ROS temama, a mapiranje prostora obavljeno je korištenjem ROS paketa *gmapping* čime se dobila karta okupiranosti CRTA-e koja se kasnije može koristiti za navigaciju i lokalizaciju.

Sve u svemu, implementacija ROS-a u IsaacSim ne zahtijeva napredno znanje programiranja i omogućuje izradu robotskih aplikacija u realističnom virtualnom okruženju gdje se robota može naučiti određenim radnjama te isto prebaciti na stvarnog robota bez straha da će doći do nepodudaranja u ponašanju.

REFERENCE

[1] Zhao, W., Queralta, J.P. and Westerlund, T. (2020) "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. Dostupno na: https://doi.org/10.1109/ssci47803.2020.9308468.

[2] Chen, X. *et al.* (2021) *Understanding domain randomization for sim-to-real transfer*, *arXiv.org*. Dostupno na: https://arxiv.org/abs/2110.03239v1 (Pristupljeno: February 13, 2023).

[3] *Closing the sim2real gap with Nvidia Isaac Sim and Nvidia Isaac Replicator* (2022) *NVIDIA Technical Blog*. Available at: https://developer.nvidia.com/blog/closing-the-sim2real-gap-with-nvidia-isaac-sim-and-nvidia-isaac-replicator/ (Pristupljeno: February 13, 2023).

[4] Ellis, J. (2022) *What is robotics simulation?, Easy Tech Junkie*. Available at: https://www.easytechjunkie.com/what-is-robotics-simulation.htm (Pristupljeno: February 3, 2023).

[5] Harris, A. and Conrad, J.M. (2011) "Survey of Popular Robotics Simulators, Frameworks, and Toolkits."

[6] *Isaac Sim highlights* (2023) *Isaac Sim Highlights - Omniverse Robotics documentation*. Dostupno na: https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim/reference_talks_videos.html (Pristupljeno: February 19, 2023).

[7] (no date) *Gazebo*. Dostupno na: https://gazebosim.org/ (Pristupljeno: February 20, 2023).

[8] *Comparison of main simulators*. Dostupno na: https://learn.e.ros4.pro/en/robotic_simulators/comparison/ (Pristupljeno: February 13, 2023).

[9] *Robot operating system ROS*. Dostupno na: https://www.ros.org/ (Pristupljeno: February 11, 2023).

[10] Joseph, L. (2018) *Robot Operating System (ROS) for absolute beginners: Robotics Programming made easy*. Berkeley, CA: Apress.

[11] Ćaran, B. (2021) "Planiranje kretanja i ispitivanje točnosti pozicioniranja mobilnog robota."

[12] Farzan M. Noori, David Portugal, Rui P. Rocha, and Micael S. Couceiro (2017) "On 3D Simulators for Multi-Robot Systems in ROS: MORSE or Gazebo?"

[13] *Ros & Ros 2 installation* (2023) Dostupno na: https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim/install_ros.html (Pristupljeno: February 11, 2023).

[14] Brush, K. (2019) *What is a mobile robot? - IoT Agenda*. TechTarget. Dostupno na: https://www.techtarget.com/iotagenda/definition/mobile-robot-mobile-robotics (Pristupljeno: February 8, 2023).

[15] Siegwart, R., Nourbakhsh, I. and Scaramuzza, D. (2004) *Autonomous mobile robots*.

[16] *NVIDIA Carter - ISAAC 2020.2 documentation*. Dostupno na: https://docs.nvidia.com/isaac/archive/2020.2/doc/tutorials/carter_hardware.html (Pristupljeno: February 8, 2023).

[17] Norzam, W.A.S. (2019) "Analysis of Mobile Robot Indoor Mapping using GMapping Based SLAM with Different Parameter." IOP Publishing Ltd.

[18] *GMapping*, *OpenSLAM.org*. Dostupno na: https://openslam-org.github.io/gmapping.html (Pristupljeno: February 18, 2023).

[19] *Unmanned underwater vehicles from Atlas Maridan*, *AZoRobotics.com*. Dostupno na: https://www.azorobotics.com/equipment-details.aspx?EquipID=506 (Pristupljeno: February 21, 2023).

[20] *Mobile Transport Robots - Fraunhofer IML* (2023) *Fraunhofer Institute for Material Flow and Logistics*. Dostupno na: https://www.iml.fraunhofer.de/en/mobile-transport-robots.html (Pristupljeno: February 21, 2023).

[21] *Roomba 680: Roomba 680, Priručnik iRobot Roomba 680 (Hrvatski - 14 stranica)*. Dostupno na: https://www.prirucnici.hr/irobot/roomba-680/priru%C4%8Dnik?p=2 (Pristupljeno: February 21, 2023).

[22] Mogg, T. (2019) *Softbank enters the cafe business with new robot-filled Pepper Parlor, Digital Trends*. Digital Trends. Dostupno na: https://www.digitaltrends.com/cool-tech/softbank-enters-the-cafe-business-with-new-robot-filled-pepper-parlor/ (Pristupljeno: February 21, 2023).