

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski studij računarstva

Diplomski rad

**Upravljanje robotom i kartiranje okoline
u Unity 3D**

Rijeka, veljača 2021.

Aleks Marković
0069069268

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski studij računarstva

Diplomski rad

**Upravljanje robotom i kartiranje okoline
u Unity 3D**

Mentor: izv. prof. dr. sc. Kristijan Lenac

Rijeka, veljača 2021.

Aleks Marković
0069069268

**SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE**

Rijeka, 6. ožujka 2020.

Zavod: **Zavod za računarstvo**
Predmet: **Mobilna robotika**
Polje: **2.09 Računarstvo**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Aleks Marković (0069069268)**
Studij: Diplomski sveučilišni studij računarstva
Modul: Računalni sustavi

Zadatak: **Upravljanje robotom i kartiranje okoline u Unity 3D**

Opis zadatka:

Izraditi aplikaciju za upravljanje mobilnim robotom i kartiranje okoline u kojoj djeluje u Unity 3D programskom okruženju. Aplikacija treba u stvarnom vremenu iscrtavati 3D kartu okoline i prikazivati video sa kamere na robotu.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 16. ožujka 2020.

Mentor:

Predsjednik povjerenstva za
diplomski ispit:

Izv. prof. dr. sc. Kristijan Lenac

Izv. prof. dr. sc. Kristijan Lenac

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad sukladno članku 9. pravilnika o diplomskom radu.

Rijeka, veljača 2021.

Ime Prezime

Zahvala

Zahvaljujem mentoru izv. prof. dr. sc. Kristijanu Lencu na podršci tijekom pisanja ovog rada i kroz diplomski studij.

Sadržaj

Popis slika	ix
Popis tablica	xi
1 Uvod	1
2 Opis problema	3
2.1 ROS#	3
2.1.1 Alternativni alati	4
2.2 Unity i aplikacija	5
2.2.1 Pogled s kamere	6
2.2.2 Upravljanje robotom	6
2.2.3 Kartiranje	7
3 Specifikacije robota	9
4 Programske alatne	14
4.1 ROS	15
4.2 Unity	17
4.2.1 Unity Editor	17
4.2.2 Skripte	19

Sadržaj

4.3	ROS#	22
4.3.1	RosBridgeClient	22
4.4	Gazebo	23
5	Opis rješenja	24
5.1	Opis sučelja i Unity objekata	26
5.1.1	Meni	26
5.1.2	Ros Connector	26
5.1.3	Plane	27
5.1.4	Model robota	27
5.1.5	Canvas	27
5.1.6	Multi Robot Control	29
5.1.7	Scena 2	29
5.2	Značajke i mogućnosti rješenja	30
5.2.1	Kamere	30
5.2.2	Laserski sken	31
5.2.3	2D kartiranje	31
5.2.4	Upravljanje	32
5.2.5	Prikaz s više robota	33
5.2.6	3D kartiranje	35
5.2.7	RQT graf sustava	36
5.3	Koraci pri korištenju	37
6	Rezultati	39
6.1	Faza 1 (Waffle Pi)	40
6.1.1	Linux i Windows	40
6.1.2	Android	43

Sadržaj

6.1.3	Više robotski prikaz	46
6.1.4	Problemi i poteškoće	47
6.2	Faza 2 (Waffle)	49
6.2.1	Problemi i poteškoće	51
7	Zaključak	55
Bibliografija		57
Sažetak		60
A Konfiguracija radne okoline		61
A.1	Dodatni koraci	62
A.1.1	Uvoz URDF modela u Unity	63
A.1.2	Dodatno	65
A.2	Glavna launch datoteka	66
A.3	Ostalo	67

Popis slika

3.1	Turtlebot 3 - Waffle [1]	9
3.2	Raspberry Pi 3 Model B [2]	10
3.3	Intel® Joule™ 570x na svojoj popratnoj ploči [3]	11
3.4	LDS-01 laserski skener [4]	11
3.5	Intel® Realsense™ R200 kamera [5]	12
3.6	Raspberry Pi kamera modul v2 [2]	12
4.1	Unity Editor	19
4.2	Gazebo	23
5.1	Model robota u Unity-ju	28
5.2	Primjer ROS tema s više robota	34
5.3	RQT graf sustava	37
5.4	Dijagram sustava [6]	38
6.1	Aplikacija u Unity Editoru	40
6.2	Windows aplikacija - 2D karta, robot i laserski sken	41
6.3	Windows aplikacija - prikaz iz kamere	42
6.4	Mobilna aplikacija - prikaz s kamere	43
6.5	Mobilna aplikacija - 2D karta, robot i laserski sken	44
6.6	Mobilna aplikacija - okomita orijentacija uređaja	45

Popis slika

6.7	Linux aplikacija - simulacija s dva robota	46
6.8	Koraci kartiranja	48
6.9	Unity Editor - 3D karta	49
6.10	Unity Editor - 3D karta (nastavak)	50
A.1	Launch datoteka za URDF uvoz	64
A.2	URDF uvoz	64

Popis tablica

4.1	ROS 1 i ROS 2 bitne razlike	16
6.1	Windows i Linux hardver računala	40
6.2	FPS usporedba	47
6.3	FPS usporedba - 3D karta	54

Poglavlje 1

Uvod

Robotika kao znanost moderne sadašnjice, prisutna je u raznim segmentima našeg života, što ukazuje potrebu i nužnost izrade aplikacija za upravljanje i interakciju čovjeka sa robotima.

Cilj ovog diplomskog rada je napraviti funkcionalnu aplikaciju za upravljanje robotima i kartiranje okoline koristeći Unity razvojno okruženje. Unity će znatno olakšati izradu univerzalnog i multiplatformskog softvera za upravljanje više vrsta robota.

Kartiranje okoline podrazumijeva sakupljanje podataka sa senzora robota i iskorištavanje tih podataka za izgradnju karte okoline. Izgradnja karata kod robota je vrlo bitna iz razloga što će je roboti moći koristiti za navigaciju i lokalizaciju u prostoru. Karta se izgrađuje postepeno kako se podaci sa senzora čitaju i procesiraju, a pošto robotski senzori imaju ograničeni domet, potrebno je kretati se robotom po prostoru tj. imati mogućnost upravljanja njime.

Mogućnost upravljanja robotom je vrlo bitni segment u robotici iz razloga što se ne može uvijek osloniti na autonomno upravljanje. Iako je autonomno upravljanje robotom kompleksniji aspekt robotike od ručnog upravljanja (putem aplikacije ili sl.), ručno je upravljanje obavezna značajka da bi se robota moglo usmjeriti točno gdje upravitelj želi. Dodatno, ako postoji autonomno upravljanje koje vodi robota u krivom smjeru, ručnim se upravljanjem može spriječiti potencijalna materijalna šteta. Upravljanje robotom podrazumijeva upravljanje njegovim kretanjem i ponašanjem

Poglavlje 1. Uvod

u prostoru.

Kao glavni alat za spajanje na robota i upravljanje koristi se ROS (Robot Operating System) [7] (dalje u tekstu: ROS), a da bi se omogućila komunikacija između ROS-a, tj. robota i Unity-ja, koristiti će se ROS# knjižnica. Implementacija i testiranje robota provest će se uz popularni Turtlebot 3 (slika 3.1) [1], za što će se koristiti simulirano okruženje, odnosno simulacija Turtlebota i njegovog modela.

Rad je razrađen u 5 poglavlja, od kojih početno poglavlje opisuje problematiku i potencijalne izazove ovog rada. Sljedeća dva poglavlja prikazuju specifikacije korištenog robota i njegovih komponenata te opisuju tehnologije i programske alate korištene pri implementaciji cjelokupnog rješenja. Zadnja dva poglavlja usko su vezana za sam razvoj rješenja. Poglavlje opisa rješenja navodi sve implementirane značajke sustava za upravljanje robotom i kartiranje okoline te kako iste funkcioniраju, dok poglavlje rezultata prikazuje i objašnjava krajnji rezultat razvoja (aplikaciju) te navodi probleme i izazove na koje se naišlo u cjelokupnom procesu razvoja.

Poglavlje 2

Opis problema

Cilj ovog rada je napraviti funkcionalnu aplikaciju za upravljanje robotom i kartiranje okoline koristeći Unity razvojno okruženje. Jedna od bitnih stavki u navedenom zadatku jest komunikacija i razmjena podataka između ROSa i Unityja. Potrebno je pronaći i testirati alat koji će imati najbolje značajke za svrhu ovog rada.

2.1 ROS#

ROS# knjižnica sadrži set alata za adaptaciju i interakciju ROSa s Unityjem. Ista većinski rješava problem komunikacije između ROSa i Unityja, no ne sasvim. Za kompleksnije stvari kao što je kartiranje, potrebno je proširiti mogućnosti ROS#a na način da se omogući intuitivno i efikasno čitanje ROS poruka. Također treba rješiti problem kako iskoristiti i prikazati dobivene podatke u Unity ekosustavu. Velika prednost ROS#a je što omogućava ROS i Unity komunikaciju bez posebnih implementacijskih detalja, što znači da ga je dovoljno uvesti u Unity te upisati IP adresu gdje se ROS nalazi da bi se otvorila konekcija između ROSa i Unityja.

Navedeni alat ima i mogućnost uvoza URDF (eng. Unified Robot Description Format) modela robota u Unity i ujedno to je i jedini način za efikasno dobivanje modela robota u Unityju. URDF datoteka je pisana u XML formatu i opisuje model robota. [8]

2.1.1 Alternativni alati

Uz ROS# postoje još dva potencijalna alata koji služe za istu svrhu:

1. **Unity Robotics Hub** - Službeni Unity alat koji se razvija od strane tvrtke *Unity Technologies*. Navedeni alat je manje popularan od ROS# alata no ubrzano raste. Dijelovi Unity Robotics Huba su proizašli iz ROS#a, a to su alati za uvoz URDF modela i TCP konektor koji su dodatno modificirani od strane Unityja. Iako navedeni alat još nije dostigao ROS# po mogućnostima i popularnosti, aktivno se razvija i izrađuju nove inačice pa bi u skoroj budućnosti mogao postati vodeći alat za spajanje ROSa i Unityja. [9]
2. **Unity ROS** - Alat razvijen od strane privatne osobe koji je zadnje ažuriran prije tri godine te nije doživio veliku popularnost i korištenje od strane ROS i Unity zajednice. Razlog toga jest loša podrška i limitirane funkcionalnosti alata. [10]

Iako svi navedeni alati imaju slični temelj za uspostavljanje komunikacije između ROSa i Unityja, ROS# i dalje vodi na ljestvici popularnosti svojom fleksibilnošću, većim mogućnostima te podrškom zajednice i programera.

2.2 Unity i aplikacija

S obzirom da je Unity poznat kao dobar alat za razvijanje softvera za više platforma odjednom, isti će se iskoristiti na način da se ciljna aplikacija napravi kako bi ista bila kompatibilna za:

1. Linux Ubuntu - Na prvom mjestu se nalazi Ubuntu iz razloga što se cijeli rad radi na ovom operacijskom sustavu, prvo bitno zbog ROSa.
2. Android - Jedan od većih zahtjeva danas je imati mobilnu aplikaciju. Posebno je to slučaj za upravljanje nekim robotom iz razloga što je nošenje prijenosnog računala teže. Mobilna aplikacija za upravljanje robotom olakšava i poboljšava korisničko iskustvo pri upravljanju robota.
3. Windows - Kao najkorišteniji operacijski sustav danas, prikladno je imati aplikaciju i za njega.

Mac OS aplikaciju je također moguće napraviti, no zbog tehničke ograničenosti pre skočit će se.

Pri izradi aplikacije za pojedinu platformu, potrebne su neke dorade da bi aplikacija radila na toj platformi. Zahvaljujući Unityju i Mono frameworku koji Unity koristi (.NET Standard 2.0) navedene dorade su dovoljno sitne da je prelazak na drugu platformu relativno bezbolan.

Unatoč tome, neke značajke znaju bit nekompatibilne. Kao opći primjer mogu se navesti načini interakcije korisnika s aplikacijom, npr. mobilni će uređaj imati na raspolaganju razne senzore, zaslon na dodir i sl., ali neće imati tipkovnicu i miš (osim kod eksplicitnog spajanja dodatnog hardvera), dok se osobno računalo uvijek koristi tipkovnicom i mišem ali većinom nema iste senzore kao mobilni uređaj, a i zaslon na dodir nije uvijek prisutan. Iz tog se razloga ne može uvijek implementirati isti način interakcije korisnika s aplikacijom za razne platforme i uređaje a da bude uvijek funkcionalan.

Još jedan mogući problem kod implementacije svih potrebnih značajki jest mogućnost prevelikog protoka podataka, gdje bi moglo doći do efekta uskog grla (eng. bottleneck). Potrebno je dobro se informirati o tome koliko će potrebna komunikacija

Poglavlje 2. Opis problema

između ROSa i Unityja za pojedinu značajku slati podataka (koliko svaka ROS tema šalje podataka i po kojoj frekvenciji). Ovisno o tome, postoji mogućnost da treba: promjeniti kvalitetu emitirane slike s kamere, smanjiti frekvenciju slanja podataka o kartiranju ili čak i promjeniti način stvaranja podataka (npr. druga metoda mapiranja). Postoje slučajevi koje se neće moći procjeniti dok se ne iste pokuša implementirati.

2.2.1 Pogled s kamere

Implementacija prikaza pogleda s kamere robota u realnom vremenu treba biti napravljena na način da se slika nesmetano prikazuje na ekranu, bez obzira na razlučivost. U ovom će dijelu zadatka trebati paziti na omjer slike u aplikaciji da se izbjegne potencijalno izobličenje slike tj. prikaz slike treba biti skalabilno. Sukladno time, potrebno je istražiti i isprobati na koji način navedenu sliku s kamere prikazati u Unityju.

2.2.2 Upravljanje robotom

Upravljanje robotom podrazumijeva upravljanje njegovim motoričkim sposobnostima. U ovom će se radu to ograničiti na kretanje robota kroz prostor. Potrebno je implementirati da se upravljanje robotom može izvršavati na svim navedenim platformama. Ovdje najveći problem predstavlja prijašnje navedene moguće razlike u hardveru uređaja (računalo i mobilni uređaj). Iz tog razloga postoji mogućnost da će se trebati implementirati više metoda upravljanja robotom.

Navedena značajka se treba implementirati na način da se šalju naredbe na određenu ROS temu koja upravlja motoričkim sposobnostima robota. Osnovna ROS tema za tu svrhu jest **/cmd_vel** s prioritetom 90 na koju se objavljuje upravljačke poruke. Postoje još dvije ROS teme koje se mogu koristiti:

1. **/joy_vel** koja ima najveći prioritet od 100 i prima upravljačke naredbe s joysticka.
2. **/nav_vel** s najmanjim prioritetom od 80 koju koristi *move_base* paket za slanje

Poglavlje 2. Opis problema

upravljačkih naredbi.

Svi navedeni načini primaju poruke formata *geometry_msgs/Twist* kao upravljačke naredbe. *geometry_msgs/Twist* se sastoji od dva Vector3 tipa podataka, jedan za linearnu brzinu robota (npr. upravljanje naprijed i unatrag), a drugi za kutnu brzinu (okretanje robota). Zbog najjednostavnije primjene koristit će se popularniju */cmd_vel* ROS temu. [11]

2.2.3 Kartiranje

Najveći implementacijski problem predstavlja kartiranje. Pošto je korišteni robot edukacijske svrhe, isti nema kompleksne komponente za interakciju s prostorom ali opremljen je laserskim skenerom i kamerom. Sukladno tome, potrebno je istražiti dostupne metode i omogućiti 2D i 3D kartiranje. Postoje popularni alati u ROS sustavu koji to omogućuju uz pretpostavku da je robot opremljen potrebnim komponentama, npr. laserski skener za 2D kartiranje ili kamera s mogućnosti 3D percepcije (percepcija dubine prostora) za 3D kartiranje prostora.

Kartiranje okoline može se opisati kao modeliranje okoline robota u specifičnim dimenzijama prostora. Karta okoline može biti u dvije ili tri dimenzije (2D ili 3D), gdje 2D karta izgleda kao tlocrt zgrade, a 3D karta dodatno opisuje dubinu prostora oko robota te se više poistovjećuje s pravim svjetom.

Izgradnja 2D karte prostora odradjuje se SLAM (eng. Simultaneous Localization And Mapping) metodom - simultana lokalizacija i kartiranje. Robot se kreće u prostoru prikupljajući podatke iz svojih senzora pomoću kojih istovremeno izrađuje 2D kartu prostora i određuje vlastiti položaj na toj karti prostora. Određivanje položaja robota naziva se lokalizacija. Postoji mnogo SLAM algoritama, stoga je potrebno istražiti koji su prikladni za hardverske mogućnosti Turtlebota.

U drugu ruku, 3D kartiranje se neće koristiti za lokalizaciju već samo za izgradnju 3D karte. Lokalizacija u 3D prostoru je također moguća, ali će se u ovom radu više raditi na samoj izgradnji karata. Također, Turtlebot se u simulaciji nalazi u zgradite je stoga 2D lokalizacija dovoljna zbog pravilnog terena. Smisao 3D karte jest bolja interpretacija detalja okoline pošto su u njoj bolje prikazane veličine i oblici objekata

Poglavlje 2. Opis problema

u prostoru. Ovdje također postoji više algoritama za implementaciju 3D kartiranja te je također potrebno pronaći metodu prikadnu za Turtlebota. Osim toga, potrebno je paziti na koji način odabrana metoda objavljuje rezultirajuću 3D kartu da bi se je moglo uspješno koristiti u Unityju.

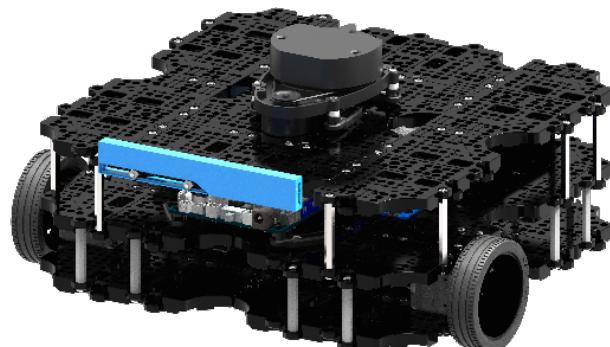
Kada se kartiranje uspješno pokrene u ROS sustavu, potrebno je rezultirajuće podatke (kartu) moći prenijeti u Unity. Ovdje će trebati analizirati rezultirajuće podatke da bi se došlo do zaključka gdje i kako se trebaju procesirati. Podaci će se svakako morati procesirati u Unityju prije korištenja, no moguće je da će se trebati prije i formatirati na određeni način da bi dohvati podataka u Unityju bio efikasniji. Za formatiranje podataka najprikladnije je napraviti ROS čvor kojega se može isprogramirati da čita i formatira podatke ROS tema koje objavljuje alat za mapiranje kojeg se bude odabralo kao najprikladnijim. Formatirane podatke se tada može ponovno objaviti na isti način.

Korištenje dobivenih podataka u Unityju podrazumijeva kako će se te podatke pretvoriti u vizualni element u aplikaciji. U slučaju 3D kartiranja cijeli će se proces trebati optimizirati iz razloga što su 3D elementi računski skuplji - svaki 3D element u aplikaciji je zasebni objekt kojeg treba stvoriti i držati u memoriji. Dok 2D elementi se mogu postavljati kao npr. slike ili teksture što drastično smanjuje potrebu za resursima računala jer predstavljaju jedan objekt ili komponentu objekta u aplikaciji.

Poglavlje 3

Specifikacije robota

U ovom se radu koristio trenutno iznimno popularan svjetski robot za edukacijske svrhe - Turtlebot 3. Od svoje dvije glavne inačice, Burger i Waffle, korišten je Waffle (slika 3.1) zbog boljih mogućnosti od kojih je najbitnija bila kamera. U prvoj fazi rada se koristio Waffle Pi, inačica koja koristi Raspberry Pi kao ugrađeno računalo, ali se u drugoj fazi rada odlučilo koristiti običnu verziju Wafflea. Pi verzija koristi Raspberry Pi kameru koja nema 3D mogućnosti dok obična verzija koristi Intel® Joule™ 570x s Intel® Realsense™ R200 kamerom koja omogućava 3D percepцију простора. Obična Waffle inačica se više ne proizvodi iz razloga što je Intel prestao s proizvodnjom Intel Joulea, no ostali Turtlebotovi su još uvijek ažurni.



Slika 3.1 Turtlebot 3 - Waffle [1]

Poglavlje 3. Specifikacije robota

Bitne karakteristike korištenog modela Turtlebot-a su sljedeće:

1. Raspberry Pi 3 model B, koje služi za pokretanje ROS sustava i svih ROS značajki (slika 3.2). Raspberry Pi se izvršava na Linux operacijskom sustavu, ima četverojezgreni 1.2GHz Broadcom BCM2837 64-bitni procesor, 1 GB RAM memorije i ugrađeni Wi-Fi i bluetooth modul. [2]



Slika 3.2 Raspberry Pi 3 Model B [2]

ili

Intel® Joule™ 570x ugrađeno računalo, koje također služi za pokretanje ROS sustava i svih ROS značajki (slika 3.3). Ovo se računalo također izvršava na Linux operacijskom sustavu, koristi četverojezgreni 1.7GHz Atom T5700 procesor, 16 GB ugrađene memorije, 4 GB LPDDR4 RAM memorije, grafičkim procesorom Intel® HD Graphics: HDMI 1.4B i ugrađenim 802.11ac Wi-Fi modulom. [12]

Poglavlje 3. Specifikacije robota



Slika 3.3 Intel(R) Joule™ 570x na svojoj popratnoj ploči [3]

2. Laserski skener LDS-01 (slika 3.4) skenira prostor u 360 stupnjeva i omogućuje SLAM kartiranje i navigaciju. Maksimalni domet od 3,5 metra i brzina skeniranja od 300 okretaja po minuti čini ga prikladnim za Turtlebota. [4]



Slika 3.4 LDS-01 laserski skener [4]

3. Intel(R) Realsense™ R200 kamera (slika 3.5) sa značajkom 3D percepcije omogućuje korištenje alata za izgradnju 3D karte. Njezine male dimenzije od samo 101.6mm x 9.6mm x 3.8mm omogućuju montiranje na Turtlebot te mala dubinska rezon-

Poglavlje 3. Specifikacije robota

lucija od 480 x 360 je prikladna za tekući prijenos slike zbog manjeg protoka podataka. [13]



Slika 3.5 Intel® Realsense™ R200 kamera [5]

ili

Raspberry Pi kamera modul v2 (slika 3.6) koji se koristi u paru s Raspberry Pi ugradbenim računalom. Ova kamera je još manja od Intelove s kvadratnom veličinom od 25 x 24 x 9 mm. Navedena kamera ne podržava 3D percepciju no njezinih 8 megapiksela čini ju odličnom za ostale edukacijske svrhe. [2]



Slika 3.6 Raspberry Pi kamera modul v2 [2]

Poglavlje 3. Specifikacije robota

4. Težinom od 1.8kg i veličinom 281mm x 306mm x 141mm dostiže brzinu od 0.26m/s s brzinom okretanja od 104.27 stupnjeva po sekundi.
5. S maksimalnom visinom penjanja od manje od deset centimetara ograničen je na ravne, većinom zatvorene prostore.

Poglavlje 4

Programski alati

Prije samog rješavanja problematike kako napraviti navedenu aplikaciju, potrebno je objasniti što su i kako funkcioniraju korišteni softverski alati. Definirati će se i koji su preduvjeti, tj. knjižnice ili alati koji svaki od njih zahtjeva da se može odraditi funkcija koja im je zadana za prethodno navedenu svrhu.

Korištene su najnovije dostupne te stabilne inačice korištenih alata:

1. ROS Noetic Ninjemys - datum izlaska 23. svibnja 2020.
2. Unity 2019. LTS - izašlo polovicom 2020. godine.
3. ROS# 1.6 - datum izlaska 20. prosinca 2019.

Sve to na najnovijoj tada dostupnoj LTS verziji Linux Ubuntu operacijskog sustava, 20.04 LTS.

4.1 ROS

Robotski Operacijski Sustav (ROS) je radni okvir (eng. framework) koji se instalira u Linux operacijski sustav i unatoč tome što sadrži riječi operacijski sustav, on to nije. Iako postoji i eksperimentalna verzija za Windows 10 i OS X, ovaj će se rad usredotočiti na razvoj na Linuxu radi kompatibilnosti s raznim alatima. Jedna od najbitnijih karakteristika ROS sustava jest da je omogućena komunikacija i upravljanje hardverom robota preko softverskih alata ROS-a bez posebnih znanja o istima.

ROS se ponajviše koristio u znanstvene i obrazovne svrhe, ali se zbog svoje praktičnosti i potencijala ubrzano proširio i u ostale grane robotike. Prije prelaska na ROS, svaki proizvođač robota je trebao razvijati svoj API (Application Programming Interface) za komunikaciju i upravljanje svojim robotima. Sada roboti diljem svijeta većinom koriste ROS kao svoj primarni sustav za komunikaciju i upravljanje, te je zbog toga vrlo korisno naučiti služiti se istime te poznavati njegovu primjenu. Radi unificirane primjene ROS sustava, stečenim znanjima i vještinama moguće je razviti softver koji će poslužiti na različitim robotima, odnosno robotima različitih proizvođača.

ROS sadrži razne alate i knjižnice, koji su razvijeni i posloženi po određenoj ROS konvenciji. Sve zajedno kako pojednostavljuje razvoj novih robotskih softvera i omogućava kompleksno ponašanje robota. Također ROS sadrži razne upravljačke programe i algoritme, i sve je otvorenog koda - besplatno.[7]

Nedavno je razvijen i novi ROS, ROS 2. Prva službena verzija izdana je krajem 2017. Iako je preporučljivo koristiti tehnologije otporne na budućnost (future-proof), za ovaj rad odabran je ROS 1 iz razloga što neki od kritičnih alata za uspjeh rada nisu podržani na najnovijim verzijama ROSa 2. Unatoč tome što je ROS 2 više otporan na budućnost i noviji, ROS 1 se i dalje razvija te nova verzija izlazi svake godine. Neke od razlika između ROSa 1 i 2 navedene su u tablici 4.1.

Poglavlje 4. Programske alati

Tablica 4.1 ROS 1 i ROS 2 bitne razlike

Značajka	ROS 1	ROS 2
Testirane platforme	Ubuntu	Ubuntu, OS X, Windows 10
C++	C++11	C++14, C++17
Python	Python 2 (Noetic Python 3)	Python 3.5
Roslaunch	XML	Python - kompleksnija logika
Čvor u procesu	1	Više od 1
Ostalo	Velika zajednica s puno stabilnih i odličnih paketa. Puno literature i tutorijala.	Minimalne ovisnosti, bolja prenosivost, pouzdanost, persistencija i rad u stvarnom vremenu (real-time).

Glavni način komunikacije kroz ROS vrši se putem ROS tema (eng. topic) [14]. ROS teme su *"kanali"* putem kojih ROS čvorovi (eng. nodes) izmjenjuju podatke. ROS čvor je proces koji odrađuje određeni zadatak [15]. Čvor objavljuje podatke na određenu temu a drugi se čvor pretplaćuje na istu temu da bi dobivao objavljene podatke te jedna tema može imati više čvorova koji su pretplaćeni ili objavljuju na nju. Svaka ROS tema ima određeni tip podataka i može primati i slati podatke samo tog formata - ROS poruka. ROS poruka je jednostavni oblik strukture podataka s određenim tipovima varijabla (primitivni tipovi, polja primitivnih tipova i ugnježđeni tipovi podataka) [16].

roslaunch je ROS alat koji omogućuje pokretanje više ROS čvorova odjednom iz *.launch* datoteka XML formata [17]. U njemu se mogu definirati sve potrebne stvari potrebne za pokretanje simulacije, npr. pokretanje i učitavanje simuliranog okruženja, pokretanje potrebnih čvorova te definiranje potrebnih argumenata. *roslaunch* automatski pokreće i *roscore* - ROS značajka koja služi kao jezgra svih ROS čvorova koja mora postojati da bi ROS čvorovi mogli komunicirati. [18]

Za programiranje novih ROS čvorova podržani su Python i C++ programski jezici.

4.2 Unity

Unity je multi-platformsko (cross-platform) razvojno okruženje, primarno za razvijanje računalnih igara. Sveukupni razvoj Unity razvojnog okruženja radi američka korporacija Unity Technologies. Također, sam softver je besplatan za edukacijske i osobne svrhe te komercijalne svrhe do 100.000 američkih dolara prihoda. Postoji veliki izbor besplatnih paketa i onih koji se naplaćuju koji se mogu koristiti za ubrzati i olakšati određene zadatke, a to se sve nalazi na tzv. *Unity Asset Store*. [19]

Unity LTS (Long Term Support) verzija se u tekućoj godini dovrši za prošlu - Unity 2019. LTS je došao sredinom 2020. godine, a nova, 2020. verzija je već dostupna i ona se postepeno nadograđuje. LTS verzija se svakog tjedna ažurira novim zakrpama (patch).

U slučaju da je potrebno testirati aplikaciju na drugoj platformi, potrebno je imati instaliran Unity alat za željenu platformu. U slučaju da je to Mac OS, tada je potrebno imati zasebnu instalaciju Unity-ja na željenom operacijskom sustavu da bi se mogla izgraditi aplikacija za taj sustav.

4.2.1 Unity Editor

Unity Editor je glavni alat za razvoj (slika 4.1). U njemu se radi većina razvoja oko vizualnih elemenata i interakcije između istih. Glavne komponente uređivača su sljedeće:

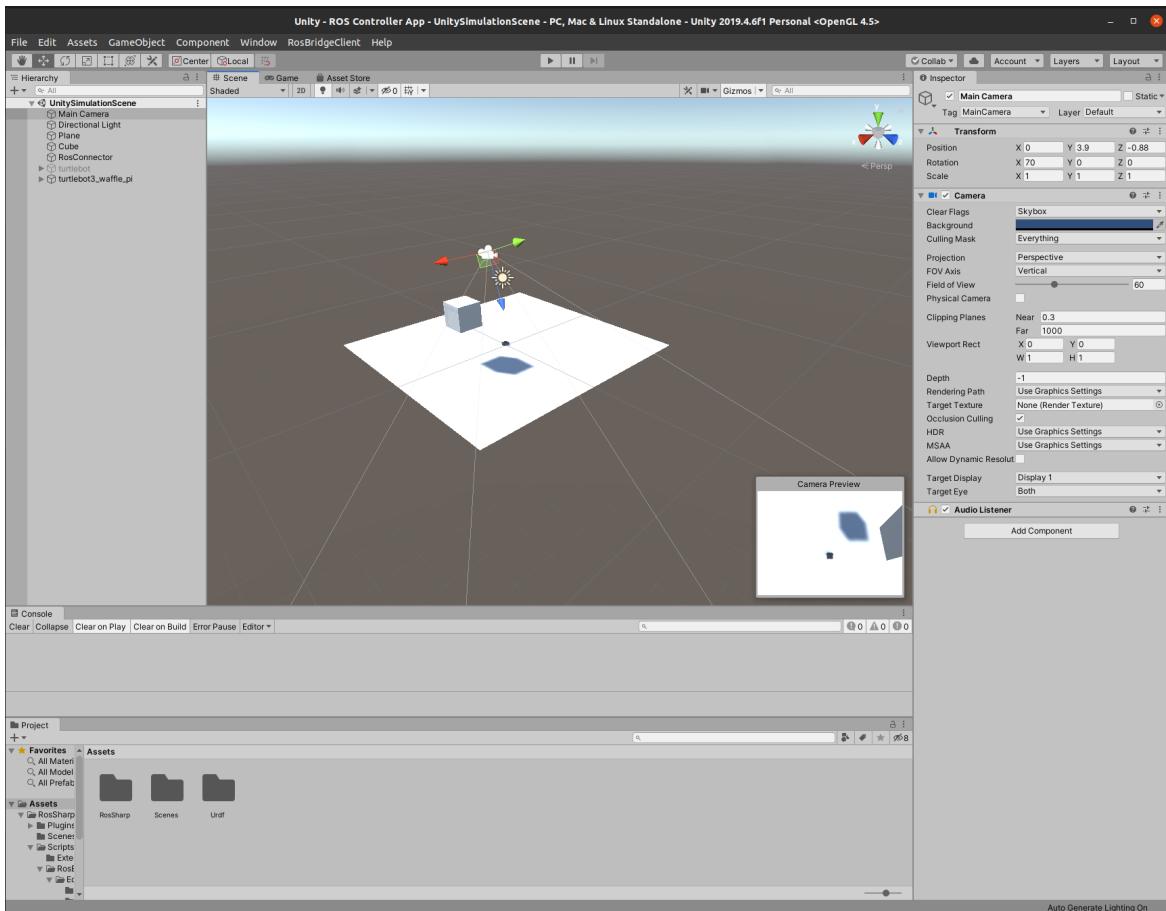
1. Hierarchy (hijerarhija) - Hijerarhijski prikaz svih elemenata u trenutnoj sceni. Unity svoje elemente u sceni naziva GameObject (igrači objekt).
2. Scene (scena) - Glavni prozor gdje se radi sve u vezi s elementima koji se mogu grafički prikazati.
3. Game (igra) - Prozor koji služi kao emulacija igre. Pritisom na "Play" gumb pokreće se igra i ovaj prozor gdje se može igrati i vidjeti što i kako radi nakon izrade u scenskom prozoru.
4. Inspector (inspektor) - Nakon odabira određenog elementa iz hijerarhije (element se također može odabrati i u sceni), ovdje se prikazuju sve komponente

Poglavlje 4. Programski alati

nadodane na taj odabrani element tj. GameObject. Mogu se dodavati razne komponente koje Unity pruža, tipa prikaz slika i teksta, animacije, efekti i još mnogo toga, ali i skripte koje sama osoba koja razvija softver može napisati.

5. Console (konzola) - Mjesto gdje se ispisuju sve poruke, upozorenja i greške.
6. Project (projekt) - Ovime se može upravljati arhitekturom projekta. Od klasičnog stvaranja mapa i datoteka, imenovanja i ostalog, do dodavanja specifičnih Unity dodataka.
7. Ostalo - Meni traka u kojoj se mogu pristupiti svim ostalim postavkama (uključujući i već navedenim stavkama). Dodatni alati se mogu postaviti kao aktivni i vidljivi, npr. pristup Asset Storeu, panel za animacije i animator te ostale stvari koje nisu vezane za ovaj rad.

Poglavlje 4. Programski alati



Slika 4.1 Unity Editor

4.2.2 Skripte

Unity također svojim korisnicima omogućava programiranje vlastitih ili korištenje tuđih skripti. Programiranje se vrši u C# programskom jeziku, koji se izvršava na *Mono* razvojnog okruženju. Također, moguće je skripte programirati u *JavaScriptu*, ali je podrška znatno lošija.

Cilj *Mono* razvojnog okruženja je da omogući korištenje .NET Frameworka na razne operacijske sustave, jer je inače .NET framework razvijen od strane Microsofta samo za Windows OS. Komponente *Mono* frameworka uključuju:

Poglavlje 4. Programske alati

1. C# kompajler - Potpun kompajler, sa svim značajkama od C# 1.0 do 6.0.
2. Mono Runtime [20] - Glavne komponente:
 - Just-in-Time (JIT) kompajler - kompilacija koda tokom izvršavanja programa.
 - Ahead-of-Time (AOT) kompajler - kompilacija koda u nativni kod stroja gdje će se izvršavati, prije njegovog izvršavanja.
 - Čitač knjižnica - omogućuje nam učitavanje vanjskih programske knjižnica.
 - Sakupljač smeća (Garbage collector) - automatsko brisanje objekata u memoriji koji se više ne koriste.
 - Dretveni sustav - omogućava izvršavanje više dretva (threads) istovremeno, pospješuje brzinu i produktivnost određenog programa ako je programirano na optimalan način.
 - Interoperabilnost - karakteristika da više programske sustave mogu bez poteškoća međusobno funkcionirati.
3. .NET Framework knjižnica klasa - *Mono* platforma pruža set klasa kao temelj za razvijanje aplikacija. Navedene su klase kompatibilne s Microsoftovim .NET Framework klasama.
4. *Mono* pruža i dodatne klase koje nisu uključene u Microsoftovim baznim klasama koje su posebice korisne za razvijanje Linux aplikacija, npr. Gtk+, Zip, OpenGL, Cairo, POSIX i druge.

Glavne značajke *Mono* radnog okruženja:

- Multi-platformske radne okruženje - Linux, macOS, BSD, Microsoft Windows.
- Multi-jezičan - C#, VB 8, Java, Python, Ruby i još mnogi.
- Kompatibilan s binarnim kodom.
- Kompatibilan s Microsoft APIjem - pokreće ASP.NET, ADO.NET, Silverlight i Windows.Forms.
- Otvorenog koda i besplatan - sav *Mono* sadržaj, uključujući i knjižnice, distri-

Poglavlje 4. Programske alati

biran je koristeći MIT licencu.

- Opsežna pokrivenost - implementacije mnogih popularnih knjižnica i protokola.

Programiranje prilagođenih skripti omogućava kompleksna ponašanja i radnje u igri/softveru kojega se razvija. Gotovo sve akcije koje se može napraviti u Editoru, može se i u skripti, te se time može drastično povećati opseg mogućnosti.

4.3 ROS#

ROS# je set programskih knjižnica i alata otvorenog koda u C#u koja omogućuje komunikaciju između ROS-a i .NET aplikacija - Unityja, razvijen od strane Siemens kompanije. Siemens je kompanija koja se bavi razvijanjem tehnologije za bitne ekonomski industrije (transport, tvornice i sl.).

ROS# je bio primarno razvijen za Windows operacijski sustav, ali se uspešno može koristiti i na druge operacijske sisteme. Paket se može preuzeti s Unity Asset Storea ili direktno sa službenog Github repozitorija. [21] ROS# je ujedno i glavni alat koji će se koristiti u ovom radu za stvaranje mosta između ROSa i Unitya.

Glavni sadržaj ovog paketa je sljedeći:

1. .NET rješenje za RosBridgeClient (knjižnica koja omogućuje spajanje vanjskih aplikacija na ROS sustav), URDF (robotski modeli) i MessageGeneration (generiranje poruka).
2. ROS paketi korišteni od strane ROS#a.
3. Unity projekt koji sadrži primjer scenu te RosBridgeClient, Urdf i MessageGeneration ekstenzije za Unity.

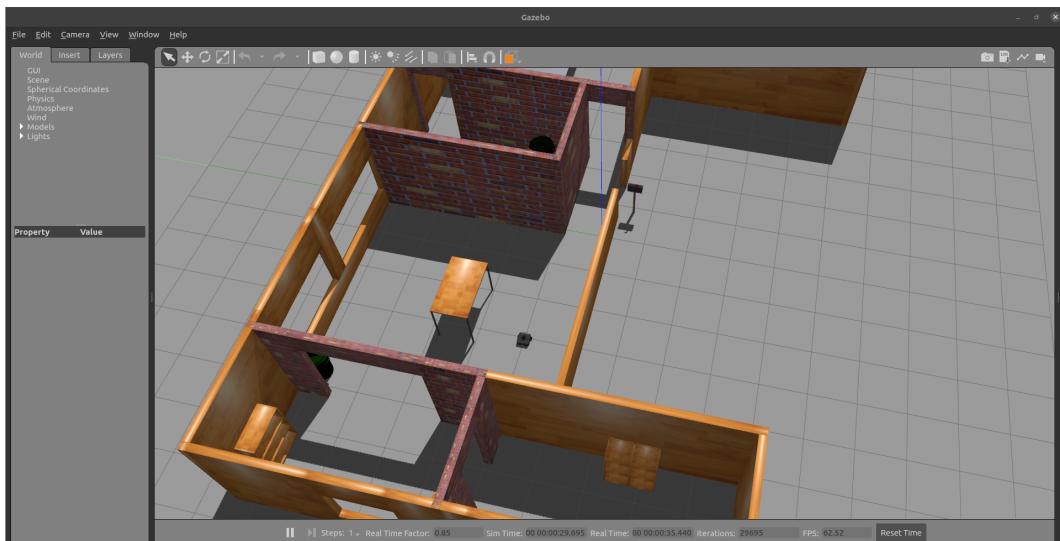
4.3.1 RosBridgeClient

RosBridgeClient za spajanje .NET baziranih aplikacija na ROS koristi *rosbridge_suite*. *rosbridge_suite* pruža ROS API funkcionalnost za programe koji su izvan ROS sustava. Isti koristi naredbe bazirane na JSON formatu. Transportni sloj je implementiran u *rosbridge_server* paketu koji pruža WebSocket poveznici. [22] WebSocket je konstantna poveznica između klijenta i servera gdje se promet odvija u oba smjera te koristi TCP/IP protokol.

4.4 Gazebo

Gazebo je alat za 3D simulaciju robota i njegovog okruženja kojega se će se u radu koristiti za pokretanje Turtlebot simulacije (slika 4.2). Neke od bitnijih značajki ovog alata su: realistična fizika u simulaciji, napredna 3D grafika i efekti, stvaranje senzornih podataka, prikaz robotskih modela i slično. [23]

Gazebo simulaciju će se pokretati *roslaunch* naredbom nad datotekom ekstenzije *.launch* koja sadrži potrebne parametre simulacije, od kojih je za Gazebo bitna lokacija *.world* datoteke. *World* datoteka opisuje model svijeta tj. okoline koja će se simulirati. [24]



Slika 4.2 Gazebo

Poglavlje 5

Opis rješenja

Nastavno na dosad navedeno opisat će se aplikacija i njezina struktura, kako je ista napravljena i na kojem principu funkcionira. Zahtjevi aplikacije bili su omogućiti upravljanje robotom i kartiranje okoline koristeći Unitz Tri glavne komponente ovog sustava, kao što je već navedeno, su: ROS, ROS# i Unity. Na slici 5.4 prikazan je dijagram kako sustav izgleda gdje su nam navedene komponente i glavni dijelovi sustava.

Preduvjet koji treba biti ispunjen prije pokretanja sustava jest imati model robota u Unityju da prikaz i osjećaj pri upravljanju budu kao u simulaciji. Jednom kad Unity sadrži funkcionalnu ROS# knjižnicu, dobiva se pristup alatu za uvoženje URDF modela robota, što je ujedno i prvi korak u izradu konkretne aplikacije. Potrebno je uvesti model korištenog robota što uključuje i njegove fizičke i motoričke karakteristike kako bi se ga moglo ispravno prikazati u Unityju te upravljati istim. U tu svrhu potrebno je pokrenuti *publish_description_turtlebot.launch* datoteku koja je detaljnije opisana u Dodatku A ovog rada. Prije samog pokretanja ove *launch* datoteke potrebno je navesti u kojem će se ROS paketu i gdje unutar paketa pronaći i učitati URDF model željenog robota. Pokretanjem *launch* datoteke učitava se URDF model robota te se kroz *file_server* alat objavljuje kao ROS čvor kojega ROS# URDF alat može pročitati i uvesti u Unity.

Na samom početku bilo je potrebno istražiti koji ROS sustav koristiti, s obzirom da postoje dva sustava, odnosno ROS 1 i ROS 2 sustav. ROS 1 sustav se i dalje

Poglavlje 5. Opis rješenja

razvija te se kontinuirano svake godine izrađuju nove verzije istog, dok ROS 2 sustav polako zamjenjuje ROS 1 sustav zbog novijih tehnologija i bogatijih značajka. Zaključno je ipak bolje trenutno odabratи ROS 1. Najveći razlog za tu odluku stoji u tome što ROS# paket trenutno službeno podržava samo ROS 1 sustav. Moguće je ROS# koristiti i sa ROS 2 sustavom, no potrebne su mnoge adaptacije koje nisu dokumentirane od strane službenih programera ROS#a pa je i vjerojatnost nailaženja na probleme i greške visoka. Sukladno navedenom koristit će se ROS 1 sustav.

5.1 Opis sučelja i Unity objekata

U ovoj će se sekcijsi opisati koji su i čemu služe glavni objekti u korištenim Unity scenama.

5.1.1 Meni

Napravljen je jednostavni inicijalni meni s nekoliko polja za upis:

1. IP adresa - polje za unos IP adrese ROSa, tj. robota. IP adresu, ako je nepromjenjiva, dovoljno je upisati jednom jer se spremo pomoću Unity značajke *PlayerPrefs* koja služi za spremanje korisničkih postavka.
2. Prefiks robota - opcionalno polje u slučaju da postoji više robota u simulaciji.

Postoje i dva gumba gdje svaki vodi na jednu scenu gdje je implementirana neka značajka. Na svakoj od ovih scena se vrši spajanje na ROS i započinje interakcija:

1. Gumb koji vodi na scenu (dalje u tekstu scena 1) gdje je implementirana kontrola robota, pogled u prvom licu s kamere, 2D kartiranje i prikaz podataka laserskog skena.
2. Gumb koji vodi na scenu (dalje u tekstu scena 2) gdje se vrši 3D kartiranje prostora.

Sljedeće će se navesti svi elementi (objekti) u sceni 1 te čemu služe.

5.1.2 Ros Connector

Na sceni 1 najbitniji element, tj. objekt, jest **RosConnector** koji prvobitno služi za otvaranje veze prema ROSu, ali i za sadržavanje svih skripta kojima je zadaća pretplata ili izdavanje poruka na ROS temu. **RosConnector** sadrži sljedeće relevantne skripte:

1. *Image Subscriber* koji se pretplaćuje na jednu ROS temu kamere, u ovom slučaju se koristi */camera/rgb/image_raw/compressed*.

Poglavlje 5. Opis rješenja

2. *Laser Scan Subscriber* koji se pretplaćuje na ROS temu laserskog skena (*/scan*).
3. *Map Subscriber* koji se pretplaćuje na ROS temu */map* koju izdaje čvor *slam_gmapping*.
4. *Odometry Subscriber* koji se pretplaćuje na temu */odom* i dohvaća odometrijske podatke robota.
5. *Joystick Publisher* izdaje poruke na */joy* ROS temu, koja služi kao joystick kontrola robota.
6. *Twist Publisher Static* učitava ulaz gumbova na glavnoj sceni za upravljanje smjerom robota. Isti izdaje poruke na */cmd_vel* ROS temi koja upravlja linearnom i kutnom brzinom robota.

5.1.3 Plane

Plane je 3D objekt koji služi kao podloga (pod ili teren) robotu. Bez podloge bi se trebalo isključiti svojstvo gravitacije u Unityju, no na taj način se može podloga iskoristiti za projekciju generirane 2D karte okoline robota. Na način kada postavimo prikaz scene iznad navedenog objekta i robota, dobiva se efekt robota koji ide kroz kartu koja se paralelno izrađuje.

5.1.4 Model robota

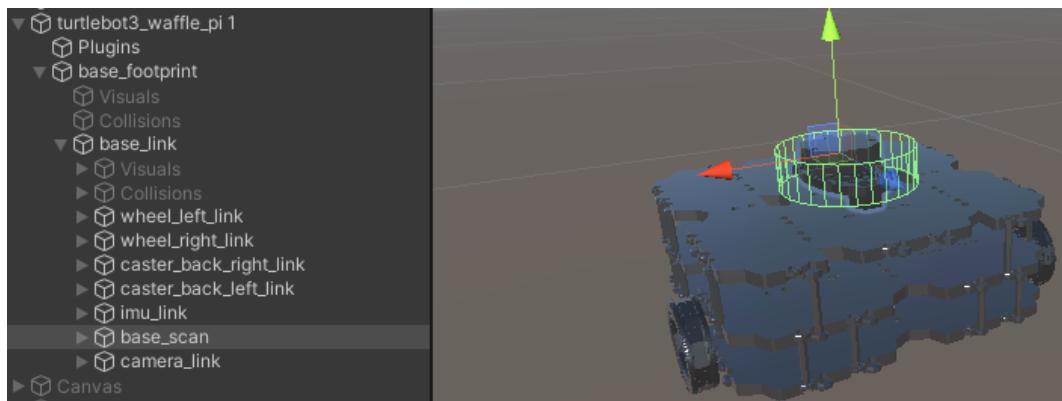
Model robota se prikazuje kao zaseban objekt u sceni, u ovom slučaju objekt **turtlebot3_waffle_pi**. Isti je generiran prijašnje navedenim alatom za uvoz URDF modela i sadrži podobjekte po svojstvima robota kao na slici 5.1.

Pomični podobjekti, npr. kotači, se vrte isto kao i u simulaciji zahvaljujući pomoćnim skriptama ROS# a koje učitavaju stanja kotača sa simulacije.

5.1.5 Canvas

Canvas u Unityju je 2D objekt koji većinom služi za slaganje grafičkih komponenta sučelja, npr. gumbovi, tekst ulazi i sl. Jedna od najvećih prednosti Canvasa je responzivnost. Canvasu se može definirati bazna rezolucija, koja se onda može

Poglavlje 5. Opis rješenja



Slika 5.1 Model robota u Unity-ju

skalirati ovisno o rezoluciji uređaja gdje je aplikacija pokrenuta. Isto vrijedi za sve elemente u Canvasu. Elementi se mogu "usidriti" u nekom kutu ekrana, mogu se širiti i smanjivati ovisno o rezoluciji, mijenjati poziciju ovisno o veličini ekrana i sl.

U ovoj aplikaciji Canvas se sastoji od sljedećih elemenata:

1. *Camera Image* kao što ime govori, je element Canvasa gdje nam se prikazuje slika s kamere. Slika kamere učitava se kao tekstura te se ista preslikava kao *Sprite* (grafički element u Unityju - slika) na navedeni element koji sadrži polje za sliku. Element je proširen po cijeloj dužini i širini Canvasa, tako da se ovisno o rezoluciji ekrana skalira.
2. *Control* je objekt koji sadrži podobjekte koji su gumbovi za određivanje linarne i kutne brzine robota.
3. *Laserscan Button* je gumb koji uključuje i isključuje prikaz objekata generiranih laserskim skenom.
4. *Camera Button* je gumb koji mjenja prikaz kamere (topografski prikaz karte i robota te prikaz u prvom licu s kamere robota).
5. *2D Map Button* je gumb koji uključuje i isključuje crtanje i prikaz 2D karte na *Plane* objekt.
6. *Switch Robot Camera* mjenja prikaz kamere u prvom licu na drugog robota (u

Poglavlje 5. Opis rješenja

slučaju da ima više robota u simulaciji). Ako u simulaciji postoji samo jedan robot, tada je taj gumb isključen.

7. *FPS* koji prikazuje trenutni FPS (broj ažuriranja slike u sekundi) u sceni.

5.1.6 Multi Robot Control

Multi Robot Control objekt služi samo za pokretanje skripte koja omogućuje prikaz kamera dva robota odjednom.

5.1.7 Scena 2

Scena 2 sadrži iste elemente kao scena 1 osim što od Canvas gumbova ima samo *Control*, *FPS* i novi *3D Map Button* koji služi za aktiviranje i deaktiviranje crtanja 3D objekata u sceni - 3D karte.

5.2 Značajke i mogućnosti rješenja

U sljedećoj će se sekciji navesti i opisati značajke i mogućnosti koje se implementiralo u ovom radu.

5.2.1 Kamere

U sceni postoje dvije kamere. Kamera u Unityju označava kako se scena prikazuje. Ona može biti iz različitih kuteva, sa različitom širinom i dubinom snimanja, bilo kojom pozicijom snimanja i još mnogo postavka. Jedna će kamera biti postavljena iznad 3D objekata, a druga će biti namještена da snima Canvas - u ovom slučaju pozicija kamere nije bitna jer fiksno snima 2D Canvas.

Prikaz kamere iz prvog lica - Prva značajka na koju se nailazi prilikom pokretanja aplikacije je prikaz kamere iz prvog lica. Slika se u stvarnom vremenu dohvaća sa robota te se kao tekstura postavlja na prijašnje navedenom Canvas objektu. Testiran je još jedan način na koji se može prikazivati slika kamere, a to je postaviti 3D objekt ispred robota koji služi kao platno gdje se postavlja slika. Platno bi trebalo biti fiksne pozicije u odnosu na poziciju i orientaciju robota. Zbog nemogućnosti efikasnog skaliranja slike na različite rezolucije i uređaje oву se metodu odbacilo

Prikaz kamere iz ptičje perspektive - U ovoj se perspektivi prikazuje 3D model robota na 2D karti koja se izrađuje s dobivenim podacima iz `/map` ROS teme, gdje se izrađuje nova tekstura koja se precrtava na *Plane* objekt. U ovoj su perspektivi također vidljive sfere (također 3D objekti) koji se crtaju shodno podacima iz `/scan` ROS teme (laserskog skena).

U početnoj fazi implementacije slike s kamere robota pokušano je postaviti sliku na 3D objekt fiksne pozicije u odnosu na poziciju i rotaciju robota. Taj objekt služio je kao platno na kojem se projecira slika. Međutim, navedeni je objekt, kao i pogled na njega, bilo teško i neefikasno skalirati na različitim rezolucijama, pa je to dovelo do zaključka da se implementira gore navedenu značajku - prikaz slike na Canvasu.

Poglavlje 5. Opis rješenja

5.2.2 Laserski sken

ROS teme laserskog skena šalju poruke formata *sensor_msgs/LaserScan* u kojem se nalazi više vrsta podataka, ali je u ovom slučaju bitno float32 polje *ranges*. Problem je što ovo polje zna sadržavati beskonačne brojeve te tada ROS# izbacuje grešku i ne nastavlja s procesiranjem tog polja. Iz tog razloga bilo je potrebno napraviti ROS čvor koji će čitati temu laserskog skena, */scan*, filtrirati podatke koji su beskonačni te izdati novu ROS temu sa ispravljenim podacima. Dobiveni podaci se pomoću ROS# skripte *Laser Scan Writer* i *Laser Scan Visualizer* stvaraju kao sfere u 3D prostoru oko robota.

5.2.3 2D kartiranje

Pošto postoji mnoštvo SLAM metoda za 2D kartiranje, navesti će se nekoliko njih koje su prikladne za Turtlebotove mogućnosti:

1. Cartographer
2. Hector
3. Karto
4. Gmapping

Svi navedeni algoritmi koriste podatke 2D lidar senzora - laserski skener, kao glavni izvor podataka za izgradnju karata. Podatke laserskog skenera dobivaju pretplatom na ROS temu *sensor_msgs/LaserScan* koja sadrži polje podataka koje opisuje rezultirajući raspon u prostoru. Svaki element u polju označava rezultat skeniranja u nekom smjeru, gdje inkrement smjera ovisi o specifikaciji senzora. Iz razloga što je najčešće prisutan u edukacijske svrhe, u ovom će se radu koristit *gmapping* algoritam, koji svoju rezultirajuću kartu objaljuje na ROS temu */map*.

2D kartiranje u Unityju se vrši na način da se učitavaju podaci iz ROS teme */map*. Ista šalje poruke tipa *nav_msgs/OccupancyGrid*. Najrelevantniji podaci ove teme su nam širina i dužina karte te int8 polje podataka u kojem se nalaze podaci vjerojatnosti u rasponu od 0 do 100 - vjerojatnost da je jedno skenirano polje zauzeto nekim objektom u prostoru. Kada se vjerojatnost ne može izračunati, u polju

Poglavlje 5. Opis rješenja

podataka bude broj -1. Proces crtanja karte u Unityju je sljedeći:

1. Učitava se dužina i širina karte.
2. Inicijalizira se nova varijabla za kartu koja će biti tipa *Color* i biti će iste duljine kao i polje podataka iz */map* teme.
3. Iterira se po tom dobivenom polju podataka te se ovisno o dobivenom podatku u novoinicijalizirano polje boja dodaje nova definirana boja.
4. Nakon iteriranja je potrebno postaviti zastavicu (flag) u *true* koja predstavlja da je potrebno nacrtati novu kartu na sučelju.
5. Unity funkcija *Update()* koja se izvršava svako osvježavanje ekrana, provjerava navedenu zastavicu i izrađuje novu teksturu na temelju definiranog polja boja, dužine i širine karte. Nova se tekstura spremi na glavnu teksturu *Renderer* komponente na *Plane* objektu.

Korištena metoda kartiranja je *SLAM gmapping* koja je bazirana na podacima laserskog skena i pozicije robota. Ista je stvoritelj prijašnje navedenih *Occupancy Grid* podataka (mreža popunjenošću) tj. karte. Karta se ažurira u intervalu od 4 sekunde.

Ispunjavanje novog polja boja u petlji je vrlo loše optimiziran pristup. U aplikaciji se osjećao trzaj u izvršavanju tijekom izrade karte, pa se kao rješenje na to izvršavanje funkcije koja sadrži petlju izvršava kao novi *Task* koji u C# označuje asinkronu operaciju na novoj dretvi. Time se drastično smanjio trzaj u aplikaciji.

5.2.4 Upravljanje

Upravljanje robotom se može vršiti na dva načina:

- Korištenjem gumbova na sučelju - Svaki gumb inkrementira ili dekrementira brzinu za definirani korak. Trenutno definirani korak za linearnu brzinu iznosi 0.05, a za kutnu 0.02. Svakom promjenom se na */cmd_vel* ROS temi šalje poruka tipa *geometry_msgs/Twist* s novom brzinom. Poruka sadrži Vector3 tip varijable za linearnu i kutnu brzinu. */cmd_vel* ROS tema proslijeđuje naredbe brzine samom robotu.

Poglavlje 5. Opis rješenja

- Kombinacijom tipka (w, a, s, d) ili tipka strelica - Unity pomoću ROS# pomoćnih skripti koje učitavaju promjenu u ulazu *Input Managera*, pretvara ulaze u upravljačku naredbu za */joy* ROS temu. Nakon toga ROS# čvor *joy_to_twist* pretvara poruke */joy* ROS teme u poruke formata *geometry_msgs/Twist* te šalje na ROS temu */cmd_vel*. Unity *Input Manager* je dio postavka gdje se mogu konfigurirati načini ulaza naredba u aplikaciju.

Samo se jedna od navedenih metoda može koristiti odjednom iz razloga što obe metode šalju poruke na istu ROS temu (*/cmd_vel*) pa će u slučaju korištenja obe metode, metoda joysticka ili tipka premostiti metodu gumbova.

Povratna informacija s */odom* ROS teme se koristi za ažuriranje odometrijskih podataka robotskog modela u Unity prostoru.

5.2.5 Prikaz s više robota

U svrhu istraživanja, omogućeno je da se mogu prikazivati slike kamera u simulaciji gdje postoje dva robota iste vrste. Kao primjer je uzeta i adaptirana Turtlebot 3 simulacija s više robota.

U početnom izborniku potrebno je upisati prefiks ROS tema ispred teme robota, npr. postoji simulacija s više Turtlebotova gdje svaki Turtlebot ima svoje ROS teme koje su inače istog naziva, njima je potrebno dodati prefiks ispred naziva ROS teme da bi se znalo na kojeg Turtlebota se ta ROS tema odnosi (slika 5.2). Prefiks je potrebno definirati u glavnoj *launch* datoteci simulacije kao što je u sljedećem isječku XML koda gdje je prefiks string *tb* u *default* svojstvu argumenata.

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type
[burger, waffle, waffle_pi]"/>
  <arg name="first_tb3" default="tb1"/>
  <arg name="second_tb3" default="tb2"/>
</launch>
```

Poglavlje 5. Opis rješenja

Ova je funkcija napravljena na način da u sceni postoji dodatni neaktivni *Ros-Connector* i model robota, koji se ovisno o ulazu na glavnому izborniku uključuje. Pri preplaćivanju pojedinačnog robota na ROS teme, u glavnoj se ROS# *Unity Subscriber* skripti provjerava broj robota te se u ovom slučaju dodaje spomenuti prefiks u ROS temu. Promjena robota (kamere) se vrši na način da se uključuje kamera jednog robota a isključuje kamera drugog robota a obe kamere su namještene kao glavni prikaz.

Pri pokretanju i spajanju na simulaciju, prijašnje navedenim gumbom moguće je mijenjati pogled s jednog robota na drugi. U ovoj se simulaciji roboti kreću sami te su ostale funkcije onemogućene.

```
/tb1/camera/parameter_updates
/tb1/camera/rgb/camera_info
/tb1/camera/rgb/image_raw
/tb1/camera/rgb/image_raw/compressed
/tb1/camera/rgb/image_raw/compressed/parameter_descriptions
/tb1/camera/rgb/image_raw/compressed/parameter_updates
/tb1/camera/rgb/image_raw/compressedDepth
/tb1/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/tb1/camera/rgb/image_raw/compressedDepth/parameter_updates
/tb1/camera/rgb/image_raw/theora
/tb1/camera/rgb/image_raw/theora/parameter_descriptions
/tb1/camera/rgb/image_raw/theora/parameter_updates
/tb1/cmd_vel
/tb1 imu
/tb1/joint_states
/tb1/odom
/tb1/scan
/tb2/camera/parameter_descriptions
/tb2/camera/parameter_updates
/tb2/camera/rgb/camera_info
/tb2/camera/rgb/image_raw
/tb2/camera/rgb/image_raw/compressed
/tb2/camera/rgb/image_raw/compressed/parameter_descriptions
/tb2/camera/rgb/image_raw/compressed/parameter_updates
```

Slika 5.2 Primjer ROS tema s više robota

5.2.6 3D kartiranje

Pronašlo se dvije metode za 3D kartiranje koje su prikladne za manje budžetne robote i ROS:

1. **Octomap** - Alat za izradu potpunog 3D modela okoline koji implementira 3D mrežu zauzetosti. Karta se postepeno generira čitajući podatke s robotske kamere koja ima mogućnost 3D percepcije (dubina prostora). [25] Octomap koristi *octree* stablastu strukturu podataka koja se koristi u svrhu prikazivanja 3D prostora, gdje svaki čvor sadrži točno osam čvorova ispod sebe (opisani prostor se dijeli na osam dijelova). [26]
2. **RTAB-Map** - SLAM metoda koja provjerava koja je vjerojatnost da nova slika (podaci) dolaze iz prethodne lokacije ili neke nove lokacije. Pri prihvaćanju navedene hipoteze dodaje se novo ograničenje u graf karte. Za izgrađivanje karte RTAB-Map koristi podatke formata *Point Clouds* koje objavljuje kamera. [27]

Zbog lakšeg pokretanja na Turtlebotu i boljeg formata rezultirajuće 3D karte, odabrana je Octomap metoda. Koristit će se ROS *octomap_mapping* paket koji izrađuje 3D mrežu popunjenošću (eng. 3D occupancy grid). Navedena metoda se koristi pokretajući *octomap_server* čvor koji omogućava inkrementalno izgrađivanje i spremanje karte te distribuira kartu ostalim čvorovima preko svojih ROS tema. [25]

Analizom dostupnih *octomap* čvorova odlučeno je da bi najprikladniji čvor za Unity bio */octomap_point_cloud_centers* koji šalje poruke formata *sensor_msgs/PointCloud2*. *PointCloud2* je prikladan format za Unity iz razloga što sadrži kolekciju N-dimenzionalnih točaka, u tom slučaju 3 dimenzije (x, y, z). Navedeni čvor objavljuje točke stvorene 3D karte napravljene od strane *octomap* alata. Naime, odabrani čvor je potrebno procesirati da bi se iz njega dobilo x, y, z točke. U tom cilju potrebno je napraviti dodatni čvor koji se pretplaćuje na odabrani čvor, procesira podatke te ih procesirane objavljuje u novi čvor na kojega će se Unity pretplaćivati. Za procesiranje podataka korištena je gotova funkcija za tu svrhu iz *ros_numpy* knjižnice koja adaptira poznati Pythonov *numpy* za ROS, *point_cloud2.pointcloud2_to_array* funkcija. Dobiveno polje podataka nije moguće objaviti bez da postoji ROS poruka tog formata. U tu svrhu

Poglavlje 5. Opis rješenja

napravljene su dvije nove ROS poruke, *CustomPointCloud* i *CustomPointCloudMsg*.

CustomPointCloud sadrži točke (x, y, z) u float32 formatu dok *CustomPointCloudMsg* sadrži polje *CustomPointCloud* poruka. Kombiniranjem dvije poruke jedini je način za napraviti intuitivni format podataka za lakše naknadno procesiranje u Unityju.

Imajući na raspolaganju prilagođene ROS poruke za potrebe ovog rada sada je moguće u prijašnje navedenom novom čvoru napraviti novu poruku formata *CustomPointCloudMsg* u kojem će se spremiti sve točke u 3-dimenzionalnom prostoru te objavljivati na novi čvor */octomap_point_cloud_centers_filtered*.

Sljedeće je potrebno izmijeniti ROS# knjižnicu, tj. dodati nove ROS poruke jer inače Unity nebi mogao primati poruke stvorenog čvora. U ROS# .NET projekt dodaju se dvije nove klase u mapu *MessageTypes* koje implementiraju isti format podataka kao stvorene ROS poruke. Projekt treba ponovno izgraditi te izmjenjene .dll datoteke dodati u Unity projekt.

U aplikaciji se vrši klasičnu pretplatu na novi čvor */octomap_point_cloud_centers_filtered* te dobivene poruke spremamo u novostvoreni format. Stvaranje i prikaz 3D karte se vrši na sljedeći način:

1. Pri primitku nove poruke, pokreće se funkcija za stvaranje objekata u sceni koji će reprezentirati dobivene točke 3D karte.
2. Petljom prolazimo kroz sve dobivene točke. Za svaku točku stvara se kocku kojoj se dodijeljuje veličina i (x, y, z) pozicija u Unity prostoru.
3. Primitkom nove poruke, ponovno se pokreće funkcija za stvaranje objekata (ažuriranje karte) samo ako je prijašnje pozivanje funkcije gotovo.

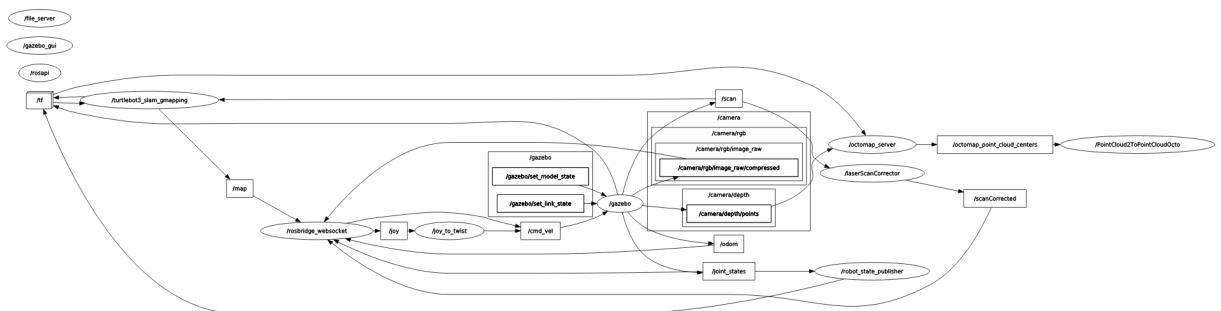
Ovim koracima stvara se 3D karta u Unity aplikaciji.

5.2.7 RQT graf sustava

Na slici 5.3 prikazan je cjelovit RQT graf sustava. *rqt-graph* je grafički alat za prikaz ROS grafa pokrenutog sustava/aplikacije. U njemu su prikazane poveznice između pokrenutih ROS tema i čvorova.

Poglavlje 5. Opis rješenja

U grafu je vidljivo da sve ROS teme s povratnim informacijama idu u `/ros_websocket` a iz njega odlaze poruke u ROS teme za upravljanje robotom.



Slika 5.3 RQT graf sustava

5.3 Koraci pri korištenju

Prvi korak sustava jest pokretanje Gazebo simulacije u ROSu gdje se simulira robota i njegovo okruženje. Alternativa u pravom svijetu bi bila ista osim što ne bismo trebali pokretati Gazebo već pravog robota na kojemu se također izvršava ROS.

Drugi korak jest Unity odnosno Unity Editor i njegove samostalne te mobilne aplikacije koje su implementirane i adaptirane sa istom svrhom i mogućnostima. Što znači da se može bilo koja aplikacija pokrenuti i spojiti na navedeni robotski sustav.

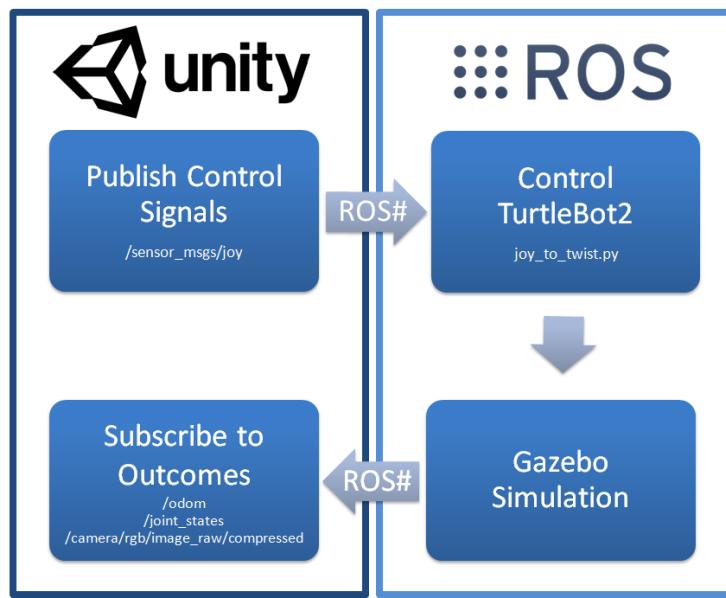
ROS# paket, odnosno most koji spaja ROS i Unity pomoću RosBridgeClienta otvara vezu između prethodno navedena dva koraka, a koji funkcioniра na temelju pretpplate i izdavanja poruka na ROS teme. Jednom kad se veza otvorila, Unity se može pretplaćivati na ROS teme i izdavati nove poruke na ROS teme.

Na slici 5.4 prikazan je glavni princip prethodno navedenih mogućnosti:

1. Unity se pretplaćuje na ROS teme - povratne informacije, npr. odometrija, slika kamere, laserski sken.

Poglavlje 5. Opis rješenja

2. Unity izdaje nove poruke na ROS teme - kontrolne poruke, npr. upravljanje robotom.



Slika 5.4 Dijagram sustava [6]

Poglavlje 6

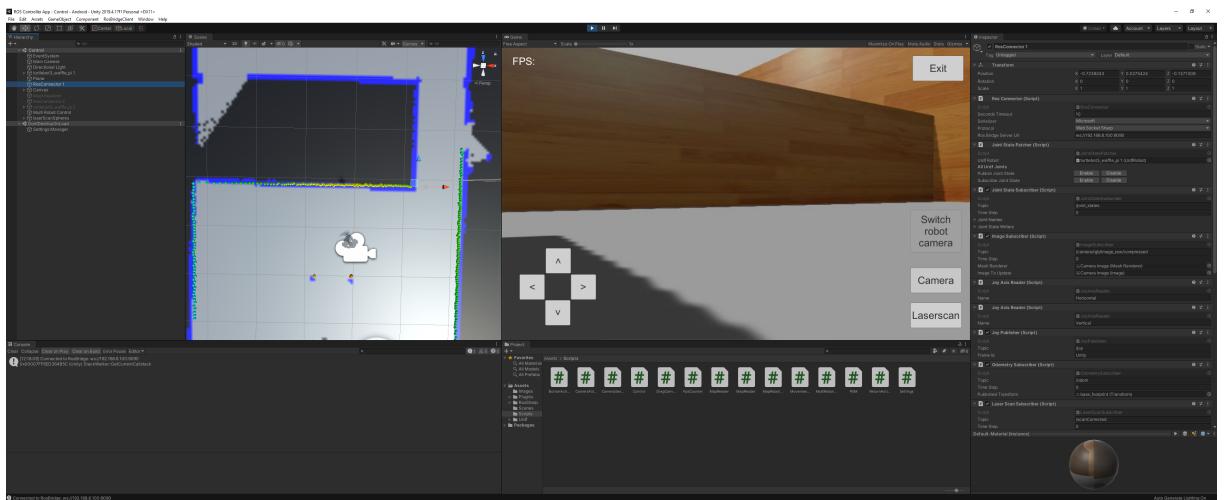
Rezultati

Implementacija se može podijeliti u 2 dijela, faza 1 i faza 2. Faza 1 podrazumijeva upravljanje Turtlebot Waffle Pi robotom, prikaz s kamera, prikaz podataka laserskog skena i 2D kartiranje. Faza 2 je isključivo 3D kartiranje, uz upravljanje Turtlebot Waffle robota. Ona je zasebno implementirana u drugoj sceni iz razloga što je stvaranje i prikaz 3D karte resursno zahtjevnije, ali ovom podjelom je i sam prikaz uredniji.

Uspješno su implementirane značajke iz ove faze za sve željene platforme (Windows, Linux i Android). Razvijanje sveukupnog sustava vršilo se na Linux Ubuntu operacijskom sustavu, a za ostale platforme napravilo se samo potrebne promjene, izgrađivanje i testiranje aplikacije. Na slici 6.1 prikazana je aplikacija u Unity Editoru.

Unity podržava i WebGL, ali pokušaj izrade aplikacije za tu platformu bio je neuspješan. Prepostavka ovog neuspjeha leži u tome što Unity WebGL koristi *WebAssembly* tehnologiju, koja ne podržava uvijek sve module koje se želi koristiti. *WebAssembly* je noviji tip koda koji se može izvršavati na modernim web preglednicima. Isti je low-level programski jezik sličan *asembleru* u binarnom formatu koji se izvršava sa sličnim performansama kao u nativnoj aplikaciji. On omogućuje aplikacije koje su napisane u Cu, C++u, C#u i Rustu da se prekompajlaju i omogući im se izvršavanje na web pregledniku [28].

Poglavlje 6. Rezultati



Slika 6.1 Aplikacija u Unity Editoru

6.1 Faza 1 (Waffle Pi)

6.1.1 Linux i Windows

Linux i Windows desktop aplikacije su po izgledu iste, ali se razlikuju u performansama. Windows se pokazao najlošiji u izvedbi aplikacije, unatoč puno superiorijim hardverom. Linux Ubuntu u drugu ruku, unatoč što se na njemu izvršavala i simulacija, performanse nisu padale. U tablici 6.1 vidljive su razlike u hardveru, gdje su Windows komponente, komponente stolnog računala pa su puno moćnije te rade na višoj frekvenciji.

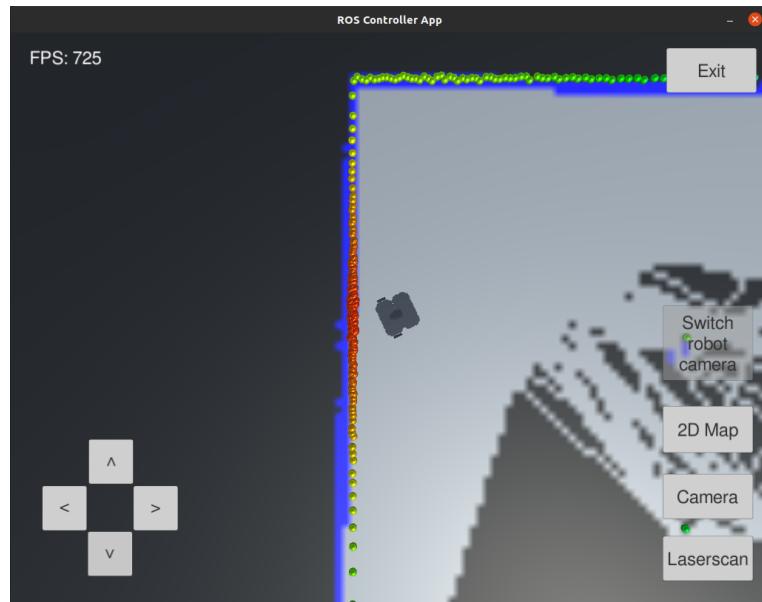
Tablica 6.1 Windows i Linux hardver računala

Komponenta	Windows	Linux Ubuntu (prijenosno računalo)
Procesor	Intel Core i7-10700 (8 jezgri, 16 dretvi)	Intel Core i7-9750H (6 jezgri, 12 dretvi)
Grafički procesor	NVIDIA GeForce GTX 1660 6GB Super	NVIDIA GeForce RTX 2060 6GB mobile
RAM	16GB XMP	16GB

Na slici 6.2 prikazan je pogled iznad objekata u sceni gdje obojane sfere prikazuju podatke laserskog skena kojima je intenzitet boje sve jači što je robot bliže predmetu u prostoru, dok je podloga ispod robota i sfera generirana karta. Slika 6.3 prikazuje

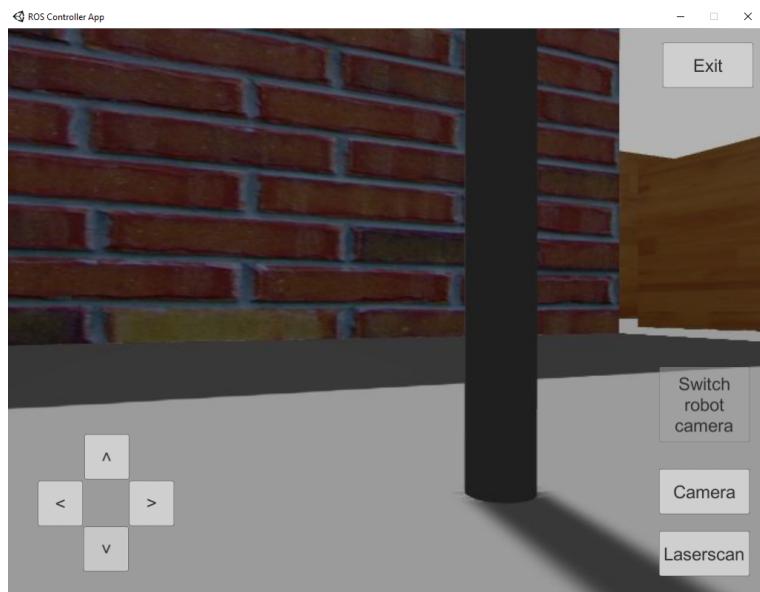
Poglavlje 6. Rezultati

pogled iz kamere robota - u ovom načinu rada aplikacija bude najfluidnija zbog konstantnog dotoka informacija (slike) koja zahtjeva minimalno dodatno procesiranje od strane aplikacije. Rezolucija slike koju ROS objavljuje iznosi 640 x 480 piksela.



Slika 6.2 Windows aplikacija - 2D karta, robot i laserski sken

Poglavlje 6. Rezultati



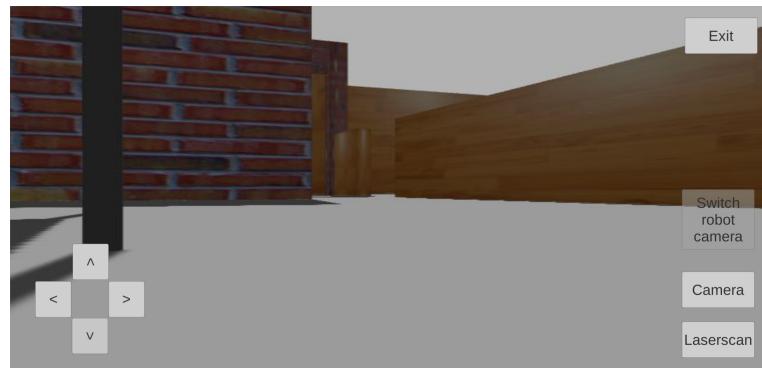
Slika 6.3 Windows aplikacija - prikaz iz kamere

Poglavlje 6. Rezultati

6.1.2 Android

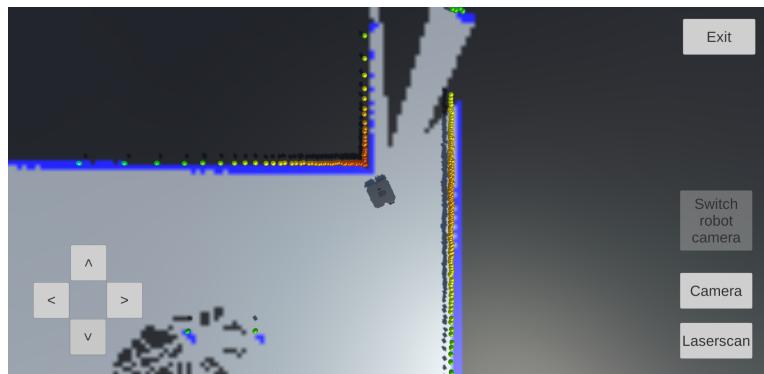
Android verzija aplikacije zahtjevala je dodatnu adaptaciju da bi uspješno funkcionalala. Preplate na ROS teme su radile bez dodatnih modifikacija, no upravljanje robotom tj. slanje poruka na ROS teme zahtjevalo je novu vrstu upravljanja - tu se kao rješenje prikazala metoda upravljanja gumbovima. S obzirom da spajanje tipkovnice na pametni telefon nije najbolje rješenje, donesena je odluka da se naprave gumbovi za upravljanje na samom grafičkom sučelju aplikacije. Još jedno moguće rješenje bilo bi koristiti senzore mobilnog telefona tj. brzinomjer i žiroskop. Vrlo zanimljiva stvar kod Android verzije je to što radi bolje od Windows verzije, unatoč limitiranom hardveru. Aplikacija je testirana na Samsung Galaxy A8 uređaju koji raspolaže s 4GB RAMa, Mali-G71 grafičkim procesorom i procesorom od 8 jezgri, 2 na 2.2GHz a ostale na 1.6Ghz, što je neusporedivo s dva navedena računala. Razlog u dobim performansima na mobilnom uređaju leži u tome što se Unity jako usredotočuje na razvoj mobilnih igara, stoga su dobro optimizirati proces izrade i izdavanja svojih produkata na mobilnim platformama.

Na slikama 6.4 i 6.5 prikazane su snimke zaslona s mobilnog uređaja, gdje se vidi iste značajke kao kod računalnih verzija.



Slika 6.4 Mobilna aplikacija - prikaz s kamere

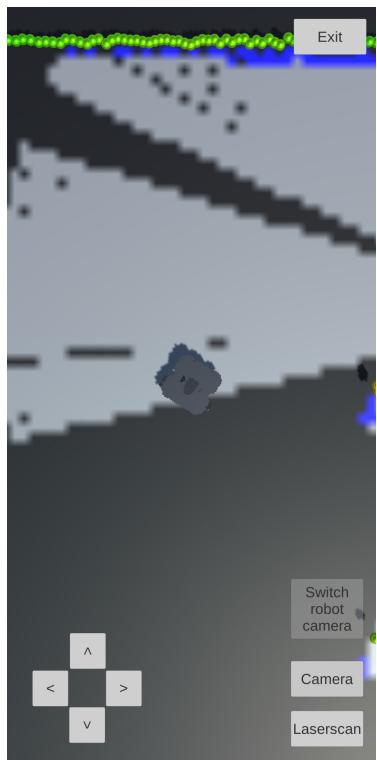
Poglavlje 6. Rezultati



Slika 6.5 Mobilna aplikacija - 2D karta, robot i laserski sken

Na slici 6.6 prikazana je ista scena samo u okomitom radu mobilnog uređaja, gdje se vidi kako elementi sučelja drže omjer i prilagođavaju se ekranu. Potrebno je napomenuti i da se nije mijenjalo ništa na sučelju pri adaptaciji desktop i mobilne aplikacije već je to značajka Unityja koji skalira i adaptira sučelje temeljem veličine ekrana, uz pretpostavku da su postavke dobro namještene od strane osobe koja je razvijala frontend u Unityju.

Poglavlje 6. Rezultati

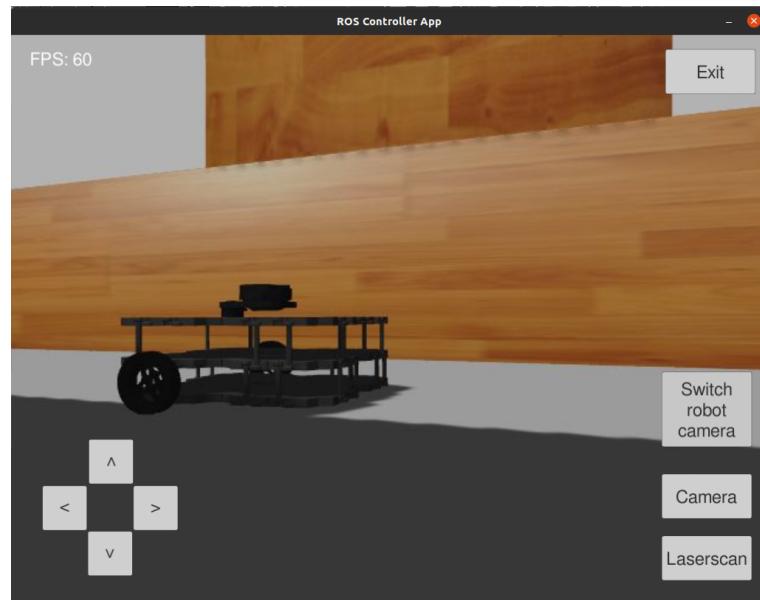


Slika 6.6 Mobilna aplikacija - okomita orijentacija uređaja

Poglavlje 6. Rezultati

6.1.3 Više robotski prikaz

Na sljedećoj slici, 6.7, prikazan je prikaz s kamere jednog od dvaju robota u simulaciji. Značajka prikaza kamera s oba robota radi isto na svim platformama.



Slika 6.7 Linux aplikacija - simulacija s dva robota

Poglavlje 6. Rezultati

6.1.4 Problemi i poteškoće

Tokom cijelog razvijanja faze 1 sustava bilo je više problema i poteškoća, od raznih inkompatibilnosti pa do ne funkciranja određenih elemenata, ali najbitnija bi bila dva problema u krajnjem rezultatu: performanse i pozicioniranje karte u Unityju.

Jedan od ostalih problema bio bi Unity Editor u Linux operacijskom sustavu. Unatoč tome što je kroz godine podrška za Linux verziju Unityja postala dobra te napravljena je stabilna verzija, teško je ne primjetiti poneke manjkavosti. Jedna od tih je slučaj da u procesu implementiranja rješenja, projekt se u Linuxu više nije mogao pokrenuti, dok isti projekt u Windowsu je radio bez grešaka.

1. Performanse

Kako se već napomenulo, performanse u Windows operacijskom sustavu su najlošije, dok su u ostala dva operacijska sustava solidne, no nisu savršene.

Analiza broja ažuriranja slike u sekundi (FPSova - Frames Per Second), upućuje da nije problem u tome (tablica 6.2). Mogući problem bi bio u tome kako pojedina aplikacija procesira dobivene podatke zbog pretplaćivanja na ROS teme koje stalno šalju nove poruke, no procesiranje tih poruka vrši ROS# i njegove skripte.

Tablica 6.2 FPS usporedba

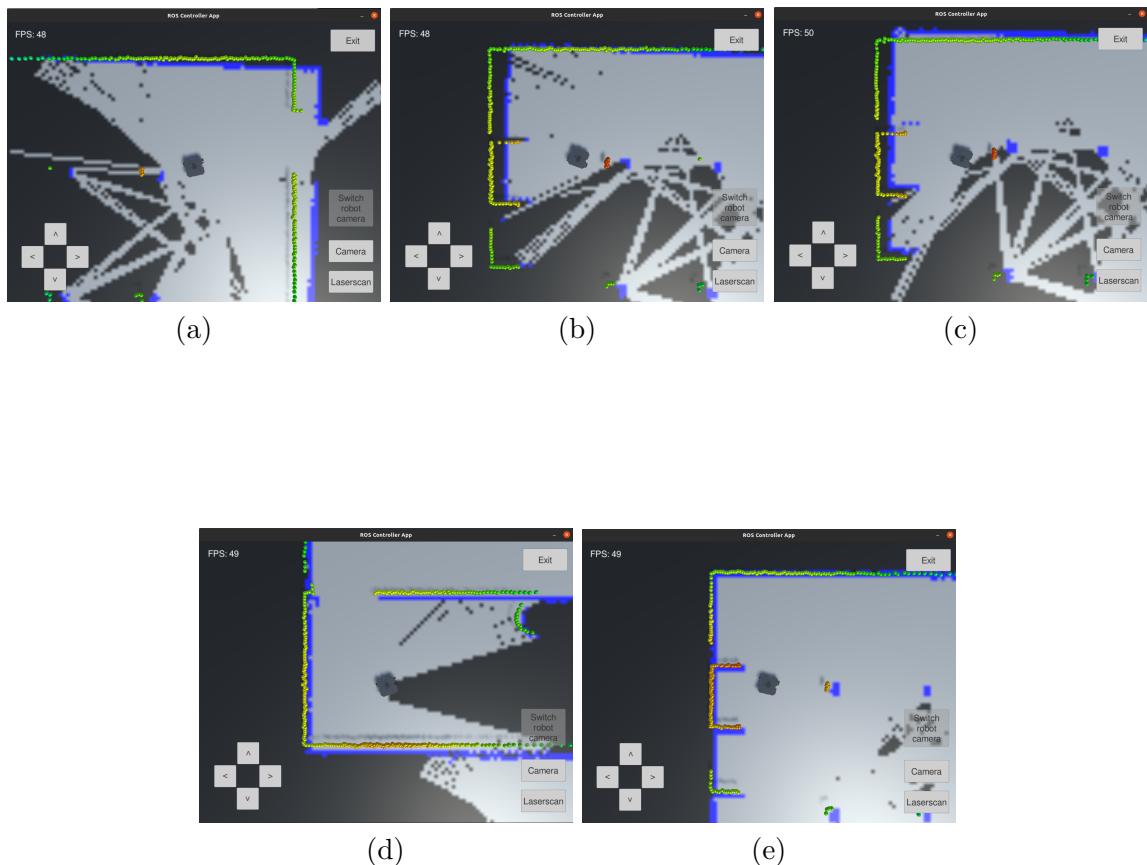
Android	Windows	Linux Ubuntu (prijenosno računalo)
10 - 100	Konstantno oko 99	100 - 1000 (integrirani ekran), manje od 100 uz vanjski monitor

2. Pozicioniranje 2D karte u Unityju

Pozicioniranje karte u Unityju podrazumijeva na koji se način slika karte nacrtana na *Plane* objekt, gdje se je relativno ograničeno zbog načina čitanja *nav_msgs/OccupancyGrid* podataka te slijedno tome crtanjem teksture gdje je vrlo teško napraviti precizan pomak.

Karta se generira sukladno SLAM gmappingu pa je ona većinom točna. No kad se pozicija karte poremeti, dalnjim skeniranjem (šetanjem) prostora ona se u većini slučajeva vrati u normalnijim okvirima (ilustracija 6.8). Također,

Poglavlje 6. Rezultati



Slika 6.8 Koraci kartiranja

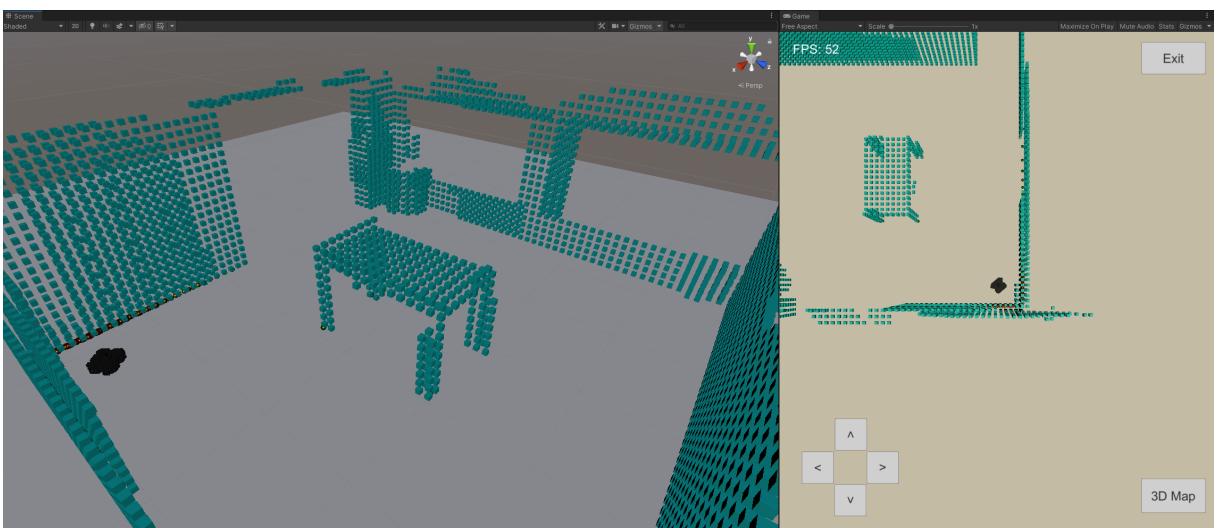
testiranjem se utvrdilo da je pozicija robota točna u Unity prostoru.

Poglavlje 6. Rezultati

6.2 Faza 2 (Waffle)

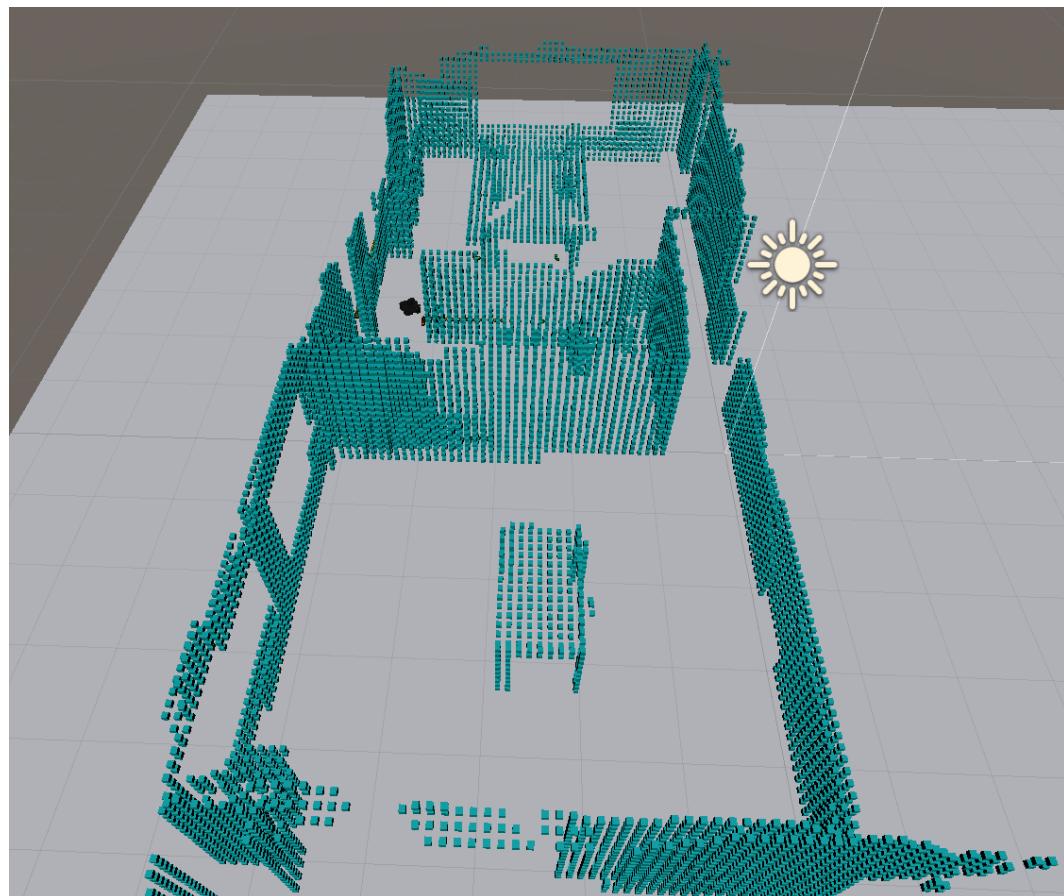
U ovoj je fazi promjena modela robota bila potrebna iz razloga što je bilo obavezno imati kameru koja ima sposobnost 3D percepcije. Bitno je navesti da su sve značajke iz faze 1, gdje je korišten drugi hardver, funkcionalne i na hardveru faze 2 pa se iz tog razloga neće ponovno navesti.

Cilj ove faze bila je izrada 3D karte. Na slikama 6.9 i 6.10 prikazana je 3D karta prostora u Unity Editoru. Slobodno šetanje po karti moguće je samo u Editoru, dok u samostalnim aplikacijama kamera slijedi robota.



Slika 6.9 Unity Editor - 3D karta

Poglavlje 6. Rezultati



Slika 6.10 Unity Editor - 3D karta (nastavak)

6.2.1 Problemi i poteškoće

Izgrađivanje 3D karte se vrši isključivo po skripti. Trebalo je osmisliti način kako da se karta kontinuirano generira i ažurira bez velikog pada u performansama i fluidnosti aplikacije. Bilo je više slučajeva i pokušaja optimizacije:

1. Slučaj 1 - Generiranje objekata sukladno dobivenim podacima. U ovom se slučaju svakim dobivenim skupom podataka generirao novi skup objekata koji su sačinjavali 3D kartu. Navedeni pristup nije bio optimiziran te se svakim pokretanjem funkcije za generiranje karte stvorio novi skup objekata koji je sadržavao nove točke i točke prijašnje poruke. To je u kratkom broju ažuriranja zablokiralo aplikaciju zbog masovnog broja objekata u sceni i velikog skupa podataka kroz koje se trebalo iterirati.
2. Slučaj 2 - Jedna mogućnost bila je napraviti collider. Collider u Unityju omogućuje fizička svojstva objekta tj. njegovo sudaranje s drugim objektom. Ideja je bila detektirati sudar dvaju objekata ako se nalaze na istoj točki u 3D prostoru i jednog od njih uništiti. Problem ovog rješenja jest taj što je uništavanje u Unityju skupa operacija, pogotovo ako je u pitanju puno objekata.
3. Slučaj 3 - Generiranje nove 3D karte svakim novim skupom podataka. Ovaj pristup imao je bolje performanse od prijašnjih jer se prijašnja karta brisala pozivanjem funkcije za uništavanje objekata samo jednom na objektu koji je sadržavao sve točke (objekt roditelj), ali je svako ponovno generiranje bilo loše vizualno rješenje (nestajanje nekoliko tisuća objekata odjednom pa generiranje novih nekoliko tisuća objekata). Osim toga, kada bi 3D karta sadržavala puno točaka, dogodilo bi se značajno zamrzavanje aplikacije u trenutku ponovnog generiranja.
4. Slučaj 4 - Generiranje karte u novoj dretvi. Ova je ideja nemoguća zbog Unityja iz razloga što isti ne dopušta generiranje ni rad s objektima u zasebnoj dretvi, već samo u glavnoj.
5. Slučaj 5 - Zadnja ideja pred rješenjem implementirala je spremanje generiranih objekata u listu gdje bi se u sljedećem ažuriranju karte uspoređivalo novodobi-

Poglavlje 6. Rezultati

vene podatke s listom te bi se podudarnosti ignorirale a nove točke iskoristilo za generiranje novih objekata i dodavanje istih u istu listu. Ideja je dobra, no kako lista raste, provjeravanje postaje vrlo skupa operacija.

Konačno rješenje kombinacija je slučaja 4 i 5 i podijeljeno je u nekoliko funkcija:

1. Asinkrona funkcija *MapPrep()* - Pokreće sljedeće dvije funkcije.
2. Funkcija provjere postojećih točaka *CheckExistingPoints()* - Provjerava listu postojećih imena generiranih objekata s novo dobivenim podacima. Ime se generira konkatenacijom stringova x, y i z koordinata točke što osigurava jedinstvenost i točnost usporedbe. Pri detekciji nove točke ista se stavlja u novu listu koja sadrži podatke prijašnje navedenog prilagođenog formata *CustomPointCloud*.
3. Funkcija za generiranje novih objekata u prostoru *Create3DMap()* odnosno ažuriranje karte - generira nove objekte u prostoru po koordinatama u novoj listi koja sadrži samo nove točke. Također dodaje imena novih objekata u prikladnu listu.

Primitkom nove poruke (samo u slučaju da je prijašnja iteracija tj. generiranje karte gotovo) poziva se asinkrona funkcija *MapPrep()* u kojoj se poziva novi C# *Task* koji pokreće funkciju *CheckExistingPoints()*. Izvršavanje navedene funkcije se uspješno odraduje u novoj dretvi jer navedena funkcija ne radi ništa s Unity objektima. Kraj izvršavanja zadatka se iščekuje C# *await* naredbom koja služi za čekanje kraja izvršavanja asinkronih funkcija te se nakon toga pokreće *Create3DMap()* koja generira nove objekte.

Na taj se način drastično poboljšalo performanse aplikacije pri generiranju 3D karte. Provjera liste generiranih objekata se izvršava bez ikakvog utjecaja na performanse zbog izvršavanja u novoj dretvi, a utjecaj generiranja novih objekata na performanse je minimiziran iz razloga što se pri svakoj iteraciji generira relativno mali broj objekata odjednom.

U slučaju velike karte, performanse polako opadaju zbog velikog broja objekata u sceni. Broj FPSova u testu s mapiranim prostorom od dvije prostorije navedeni su u tablici 6.3 gdje je rezultat prirodniji i sukladan hardverskim sposobnostima

Poglavlje 6. Rezultati

korištenog uređaja.

Poglavlje 6. Rezultati

Tablica 6.3 FPS usporedba - 3D karta

Android	Windows	Linux Ubuntu (prijenosno računalo)
Promjenjivo oko 10	Konstantno oko 99	Promjenjivo oko 60 na ugrađenom monitoru

Još jedan problem povezan je s Android verzijom aplikacije. U sceni 3D kartiranja gubi se mogućnost upravljanja robotom bez obzira na istu programsku logiku kao u prvoj sceni. Analizom zapisnika grešaka utvrdilo se da je problem povezan sa ROS# knjižnicom jer se pogreška prikazuje u baznom i već korištenom programskom kodu. Greška ukazuje na to da se komponenta koja je zadužena za pretplatu na ROS temu */cmd_vel* kojoj je svrha primanje naredba za upravljanje robotom ne inicijalizira ispravno.

Zadnji bitan problem za naglasiti u ovoj fazi jest promjena hardvera tj. promjena kamere. Slika kamere ima vidno jako nisko ažuriranje slike u sekundi (ispod 10). Problem nije u Unity aplikaciji jer se isti problem javlja i pri učitavanju slike u RVizu (RViz je ROS alat za 3D vizualizaciju. U njemu se može čitati i vizualizirati sve ROS teme: odometrija, kamera, kartiranje i sl.). Iako su postavke kamere iste kao kod Raspberry kamere (ista rezolucija, isti broj ažuriranja slike u sekundi) i boljim ugradbenim računalom, rješenje za ovaj problem se nije uspjelo pronaći.

Poglavlje 7

Zaključak

U ovom se radu razvila aplikacija za upravljanje robotom i kartiranje prostora koristeći *SLAM gmapping* za 2D kartu te *Octomap mapping* za 3D kartu. Paralelno s upravljanjem robota generira se 2D karta prostora, nalik tlocrtu zgrade. Također, omogućeno je generiranje sfera iz podataka laserskog skena, kao i prikaz iz prvog lica kamere robota. U slučaju više robota u simulaciji, moguća je promjena perspektive s jedne kamere robota na kameru drugog robota. Napravljeno je i generiranje 3D karte prostora u stvarnom vremenu koje se odrađuje u zasebnoj sceni.

Aplikacija je rađena isključivo koristeći Turtlebot 3 - Waffle simuliranog robota. Za adaptaciju na drugu vrstu robota dovoljno je učitati novi URDF model te eventualno izmijeniti ROS teme u *RosConnectoru* u slučaju da se razlikuju.

Aplikacija je izgrađena i testirana na tri platforme: Windows, Linux Ubuntu i Android. Moguće je uz minimalne promjene aplikaciju izgraditi i za Mac OS. Performanse aplikacija nisu savršene kao što je navedeno u zadnjem poglavlju rada, gdje iste ovise o fazi rada i korištenom hardveru. U fazi 1 gdje je korištena *Waffle Pi* inačica Turtlebota aplikacija za Windows je drastično lošija od ostalih iako je sve praktički isti kod. Naime, u fazi 2 je korištena *Waffle* inačica Turtlebota s kamerom koja omogućava 3D percepciju prostora gdje je fluidnost slike kamere drastično pala. Međutim, scena koja generira 3D kartu sačuvala je logičnost između korištenog hardvera i performansa aplikacije gdje su računalne verzije održale normalan i konstantan broj ažuriranja slike u sekundi (FPS), a mobilna se verzija uvelike usporila.

Poglavlje 7. Zaključak

ROS# se pokazao kao alat s vrlo velikim potencijalom za izradu konkretnijih aplikacija, kao i za edukacijske svrhe te za entuzijaste koji rade s Unity razvojnim okruženjem.

Zaključno, glavni cilj ovog rada - spajanje robota na Unity i implementiranje multiplatformske aplikacije, dodatno podiže mogućnosti ROSa i Unityja koji zajedno mogu dovesti do vrlo zanimljivih i korisnih rezultata.

Bibliografija

- [1] IEEE. Turtlebot 3. , s Interneta, <https://spectrum.ieee.org/automaton/robotics/robotics-hardware/review-robotis-turtlebot-3> , 07. siječnja 2021.
- [2] raspberrypi.org. Raspberry pi. , s Interneta, <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/> , 08. siječnja 2021.
- [3] prirucnici.hr. Intel® joule 570x na popratnoj ploči. , s Interneta, <https://www.prirucnici.hr/intel/joule-570x-developer-kit/priru%C4%8Dnik> , 07. siječnja 2021.
- [4] Robotis. Lds-01. , s Interneta, <https://www.robotis.co.uk/360-laser-distance-sensor-lds-01.html> , 07. siječnja 2021.
- [5] Intel. Intel® realsense camera r200 slika i specifikacije. , s Interneta, <https://software.intel.com/content/www/us/en/develop/articles/intel-realsense-data-ranges.html> , 07. siječnja 2021.
- [6] Siemens. Ros# gazebo simulacija. , s Interneta, https://github.com/siemens/ros-sharp/wiki/User_App_ROS_GazeboSimulationExample , 30. prosinca 2020.
- [7] Ros službene stranice. , s Interneta, www.ros.org , 17. rujna 2020.
- [8] ROS. Urdf. , s Interneta, <http://wiki.ros.org/urdf> , 31. siječnja 2021.
- [9] U. Technologies. Unity robotics hub. , s Interneta, <https://github.com/Unity-Technologies/Unity-Robotics-Hub> , 28. siječnja 2021.
- [10] M. Jenkin. Unity ros. , s Interneta, <https://github.com/michaeljenkin/unityros> , 31. siječnja 2021.
- [11] ROS.org. Upravljanje robotom. , s Interneta, <http://wiki.ros.org/robotican/Tutorials/Commandyourobotwithsimplemotioncommands> , 29. siječnja 2021.

Bibliografija

- [12] Intel. Intel® joule 570x. , s Interneta, <https://ark.intel.com/content/www/us/en/ark/products/96414/intel-joule-570x-developer-kit.html> , 07. siječnja 2021.
- [13] ——. Intel® realsense camera r200 specifikacije. , s Interneta, <https://ark.intel.com/content/www/us/en/ark/products/92256/intel-realsense-camera-r200.html> , 07. siječnja 2021.
- [14] ROS. Ros teme. , s Interneta, <http://wiki.ros.org/Topics> , 28. veljače 2021.
- [15] ——. Ros čvorovi. , s Interneta, <http://wiki.ros.org/Nodes> , 28. veljače 2021.
- [16] ——. Ros poruke. , s Interneta, <http://wiki.ros.org/Messages> , 28. veljače 2021.
- [17] ——. roslaunch. , s Interneta, <http://wiki.ros.org/roslaunch> , 28. veljače 2021.
- [18] ——. Ros jezgra. , s Interneta, <http://wiki.ros.org/roscore> , 28. veljače 2021.
- [19] U. Technologies. Unity asset store. , s Interneta, <https://assetstore.unity.com> , 22. rujna 2020.
- [20] M. Project. Mono. , s Interneta, https://www_mono-project.com/docs/about-mono/ , 21. rujna 2020.
- [21] Siemens. Ros#. , s Interneta, <https://github.com/siemens/ros-sharp> , 23. rujna 2020.
- [22] ROS. Rosbridgesuite. , s Interneta, http://wiki.ros.org/rosbridge_suite , 03. siječnja 2021.
- [23] Gazebo. Gazebo. , s Interneta, <http://gazebosim.org/> , 22. veljače 2021.
- [24] ——. Gazebo roslaunch. , s Interneta, http://gazebosim.org/tutorials?tut=ros_roslaunch , 28. veljače 2021.
- [25] ROS. Octomap alat. , s Interneta, http://wiki.ros.org/octomap_server , 07. siječnja 2021.
- [26] Wikipedia. Octree. , s Interneta, <https://en.wikipedia.org/wiki/Octree> , 28. veljače 2021.
- [27] Introlab. Rtab-map. , s Interneta, <http://introlab.github.io/rtabmap/> , 31. siječnja 2021.
- [28] MDN. Webassembly. , s Interneta, <https://developer.mozilla.org/en-US/docs/WebAssembly> , 03. siječnja 2021.

Bibliografija

- [29] Microsoft. .net core instalacija. , s Interneta, <https://docs.microsoft.com/en-us/dotnet/core/install/linux-ubuntu#2004-> , 24. rujna 2020.
- [30] ——. Visual studio code konfiguracija. , s Interneta, <https://code.visualstudio.com/docs/other/unity> , 24. rujna 2020.
- [31] M. Project. Instalacija mono radne okoline. , s Interneta, <https://www.monoproject.com/download/stable/#download-lin> , 24. rujna 2020.
- [32] Robotis. Instalacija turtlebot 3 paketa. , s Interneta, https://emanual.robotis.com/docs/en/platform/turtlebot3/pc_setup/#pc-setup , 24. rujna 2020.
- [33] R. W. Tools. Rosbridge. , s Interneta, https://github.com/RobotWebTools/rosbridge_suite , 28. rujna 2020.

Sažetak

Cilj ovog rada bio je napraviti funkcionalnu aplikaciju za upravljanje robotom i mapeiranje prostora koristeći Unity razvojno okruženje. Za spajanje ROS sustava i Unity razvojnog okruženja korišten je ROS# alat. Kartiranje se vrši paralelno uz upravljanje robotom. Izrađuje se 2D karta koja se prikazuje ispod robota. Omogućen je prikaz podataka s laserskog skena u obliku sfera i prikaz iz prvog lica kamere robota. U svrhu istraživanja napravljeno je da se u slučaju dva robota u simulaciji može mijenjati pogled kamere s jednog robota na drugi. Implementirano je i generiranje 3D karte, ali u zasebnoj sceni. Aplikacija je funkcionalna na tri platforme: Windows, Linux i Android.

Ključne riječi — ROS, Unity, ROS#, upravljanje robotom, kartiranje

Abstract

The objective of this thesis was to make a functional application for robot control and mapping using the Unity game engine. The connection of ROS and Unity game engine has been made with the ROS# tool. Mapping is done in parallel with the robot control. A 2D map is being made which is shown under the robot. Laser scan data is displayed in a form of spheres and a first person view from the robot camera was also enabled. For research purposes in the case of two robots in the same simulation, a feature was created in order to switch the camera view from one robot to another. 3D mapping is also implemented, but on a separate scene. The application is functional on three platforms: Windows, Linux and Android.

Keywords — ROS, Unity, ROS#, robot control, mapping

Dodatak A

Konfiguracija radne okoline

U ovom će se poglavlju opisati postupak konfiguriranja radne okoline. Ovo je potrebno iz razloga što većina alata nije testirana na najnovijim Ubuntu i ROS verzijama pa su određene adaptacije bile potrebne.

Instalaciju i konfiguraciju se može podijeliti u nekoliko koraka, od kojih će se pojasniti samo one koji su zahtjevali dodatne korake:

1. Instalacija Ubuntu 20.04 OS-a.
2. Instalacija ROS-a Noetic Ninjemysa gdje je potrebno pratiti upute na službenim stranicama. Vrši se instalacija potpunog paketa.
3. Instalacija Unityja. Dobro je povremeno instalirati noviju verziju jer sadrži korisna ažuriranja.
4. Instalacija Visual Studio Code-a - text editor koji će se koristiti uz Unity za programiranje skripta. Razlog odabira Visual Studio Code-a je taj što je potrebno imati podršku za Unity i mogućnost spajanja i pokretanja alata za otklanjanja pogreška (debugger).
5. Instalacija .NET Core radne okoline za omogućiti rad Unity-ja s Visual Studio Code - podrška za C#. [29]
6. Visual Studio Code konfiguracija za Unity. [30]
7. Instalacija *Mono* radne okoline. [31] Preporučljivo je nakon ovog koraka po-

Dodatak A. Konfiguracija radne okoline

krenuti i naredbu *sudo apt install mono-complete* da bi se instalirali eventualni segmenti koji fale.

8. Instalacija Turtlebot 3 paketa za ROS 1. [32]

A.1 Dodatni koraci

Ovdje će biti navedeni i eventualno pojašnjeni dodatni koraci. Neki od tih su jednostavna instalacija dodatnog paketa, a neki promjene da bi određena stvar mogla proraditi.

Kada je neki alat ili knjižnicu u ROSu potrebno instalirati iz izvornog koda, to se u osnovi radi na sljedeći način (eventualni alati imaju specificirana potrebna dodatna podešavanja):

1. Preuzimanje i spremanje mape s datotekama izvornog koda u ROS radni folder (workspace - */catkin_ws/src/*) - ova mapa je proizvoljna ali ROS standard je *catkin_ws* u *home* direktoriju Linux Ubuntu OS-a.
2. U korijenskom *catkin_ws* direktoriju pokreće se naredba *catkin_make* koja izgradi sav kod u njemu. Tada se pojave dvije nove mape - *build* i *devel*. Nakon toga treba pozvati izgenerirani *setup.bash* sljedećom naredbom koja omogućava korištenje novo izgrađenih paketa:

```
source /home/user/catkin_ws/devel/setup.bash ili source /opt/ros/noetic/setup.bash
```

- Za omogućiti kartiranje robotske okoline potrebno je dodatni instalirati *slam* pakete za kartiranje. Instalacija se vrši iz izvornog koda kojeg je moguće dohvatiti s git repozitorija paketa [?] gdje se nalazi i drugih paketa za robotsku percepцију.
- Osnovna spona koja omogućuje slanje podataka između ROSa i Unityja je *RosBridge* paket kojeg inače dohvaćamo naredbom *sudo apt-get install ros-noetic-rosbridge-server*. Istoga koristi ROS#. ROS Noeticu instalacija ovog paketa na klasičan način ne radi, ali više o tome u sljedećem odlomku.
- Za korištenje željenog robota u Unity, prvi korak je unijeti njegov model. Ko-

Dodatak A. Konfiguracija radne okoline

risti se URDF (Universal Robotic Description Format - univerzalni robotski opisni format). URDF je pisan u XML formatu i koristi se za opisati sve elemente opisanog robota.

A.1.1 Uvoz URDF modela u Unity

Za uvoz URDF modela potrebno je pokrenuti određenu *launch* datoteku koja se nalazi u ros-sharp (datotečno prihvatljiv naziv za ROS#) paketu. Datoteka se nalazi u *ros-sharp/ROS/file-server/launch*, odnosno *catkin_ws/src/file-server/launch* direktoriju koja se pokreće *rosaunch publish_description_turtlebot.launch* no prije samog pokretanja potrebno ju je urediti iz razloga što je to samo primjer za Turtlebot 2 robot, koji ima drugačije specifikacije od Turtlebot 3 robota.

Launch datoteke su također pisane u XML formatu i one služe za pokretanje više čvorova odjednom. U datoteci se može podesiti i dodatne parametre za pokretanje određenih čvorova. *Roslaunch* je korišten za pokretanje tih datoteka i to se može učiniti pozivanjem paketa i specifične *launch* datoteke ili direktno pozivanjem *launch* datoteke definiranjem njezine datotečne putanje:

- *roslaunch ime_paketa launch_datoteka*
- *roslaunch ../catkin_ws/src/paket/launch/launch_datoteka*

Osim uređivanja robotskih vrijednosti, potrebno je i urediti i argument *urdf_file* na način da se iz *xacro.py* izbaci *.py* te doda flag *-inorder*. To se mora iz razloga što su sve prijašnje verzije ROSa bile na Python 2, ali je ROS Noetic na Pythonu 3. (slika A.1)

Još jedna promjena koja je potrebna a povezana je s verzijama Python-a je za uspješno pokretanje *RosBridge* poslužitelja. Naime, ako se izvrši instalacija standardnom *apt install* naredbom, RosBridge neće raditi jer se kose vrste varijabla (*str* i *byte*) koje su drugačije definirane u svakoj verziji Python-a, pa je iz tog razloga potrebno instalirati cijeli RosBridge iz izvornog koda [33] uz namještenu točnu verziju Pythona (2).

Nakon ovih koraka može se pokrenuti Unity i u izborniku odabrati *RosBridgeClient - Transfer URDF from ROS* gdje se otvara prozorčić (slika A.2) u kojemu

Dodatak A. Konfiguracija radne okoline

```
<launch>
  <include file="$(find file_server)/launch/ros_sharp_communication.launch">
    <arg name="port" value="9090" />
  </include>

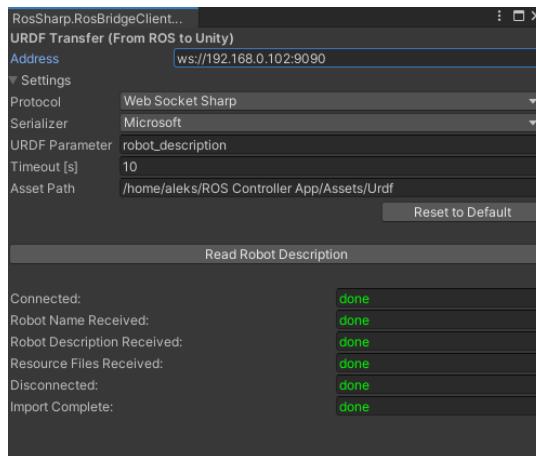
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" />
  <arg name="urdf_file" default="$(find xacro)/xacro --inorder '$(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro'" />

  <!--<param name="robot/name" value="Turtlebot3"/>-->
  <param name="robot_description" command="$(arg urdf_file)" />

</launch>
```

Slika A.1 Launch datoteka za URDF uvoz

je potrebno izmijeniti IP adresu gdje će se pronaći RosBridge poslužitelj. Ostale vrijednosti bi trebale biti dobre po zadanim vrijednostima. Prije spajanja, potrebno je pokrenuti TurtleBot simulaciju i gore navedenu *launch* skriptu. Dovršetkom ovog koraka, u Unity projekt se sprema URDF model kojega se može prikazati i koristiti u scenama.



Slika A.2 URDF uvoz

Za korištenje simuliranog okruženja koristi ste *Gazebo* softver, no iz razloga limitirane kompatibilnosti umjesto zadnjeg Gazebo 11, u slučaju da simulacija ne radi, potrebno je instalirati Gazebo 9 pakete:

Dodatak A. Konfiguracija radne okoline

```
sudo apt install gazebo9-common  
sudo apt-get install libgazebo9-*  
sudo apt install ros-noetic-gazebo-ros-pkgs
```

A.1.2 Dodatno

Dodatne informacije o konfiguriranju dostupne su na službenom ROS# repozitoriju u sekciji wiki stranica. [21]

Također, u službenom repozitoriju nalaze se Unity testne scene s kojima se može testirati osnovne funkcionalnosti.

Dodatak A. Konfiguracija radne okoline

A.2 Glavna launch datoteka

```
<launch>
    <include file="$(find
        rosbridge_server)/launch/rosbridge_websocket.launch">
        <param name="port" value="9090"/>
    </include>
    <include file="$(find turtlebot3_slam)/launch/turtlebot3_slam.launch">
        <param name="slam_methods" value="gmapping"/>
    </include>

    <!-- One robot -->
    <include file="$(find
        turtlebot3_gazebo)/launch/turtlebot3_house.launch">
    </include>

    <!-- Multiple robots
    <include file="$(find
        turtlebot3_gazebo)/launch/multi_turtlebot3.launch">
    </include> -->

    <node name="file_server" pkg="file_server" type="file_server" />

    <node name="joy_to_twist" pkg="gazebo_simulation_scene"
        type="joy_to_twist.py"/>

    <node name="rqt_graph" pkg="rqt_graph" type="rqt_graph" />

    <node name="laserScanCorrector" pkg="gazebo_simulation_scene"
        type="laserScanCorrector.py" />
</launch>
```

Dodatak A. Konfiguracija radne okoline

A.3 Ostalo

Ostale relevantne launch datoteke, Unity projekt i drugo biti će priloženi kao zasebne datoteke.