SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski studij računarstva

Diplomski rad

Upravljanje robotom i mapiranje okoline u Unity 3D (Robot control and mapping with Unity 3D)

Rijeka, siječanj 2021.

Aleks Marković
0069069268

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski studij računarstva

Diplomski rad

Upravljanje robotom i mapiranje okoline u Unity 3D (Robot control and mapping with Unity 3D)

Mentor: prof.dr.sc. Kristijan Lenac

Rijeka, siječanj 2021.

Aleks Marković
0069069268

Umjesto ove stranice umetnuti zadatak za završni ili diplomski rad

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, siječanj 2021.

_____

Ime Prezime

Zahvala

Zahvaljujem xxxxxx na podršci tijekom pisanja ovoga rada i korisnim raspravama i savjetima. Zahvaljujem xxxxx na podršku tijekom studiranja.

Sadržaj

Popis slika

Popis tablica

Poglavlje 1

Uvod

Tema ovog diplomskog rada je napraviti funkcionalnu aplikaciju za upravljanje robotom i mapiranje okoline koristeći Unity 3D razvojni program. Zahvaljujući Unity-ju biti će lakše ostvariti cilj da se napravi univerzalni i multiplatformski softver s kojim će se moći upravljati s više vrsta robota.
Kao glavni alat za spajanje i upravljanje na robota koristi se ROS (Robotski Operacijski Sustav) 1. Za omogućavanje komunikacije između ROS-a, tj. robota i Unity aplikacije, koristiti ćemo ROS# knjižnicu. Za svrhu implementacije i testiranja kao testnog robota odabran je popularni Turtlebot 3. Konkretnije koristit ćemo simulirano okruženje (simulaciju) Turtlebot-a i njegovog modela..

Poglavlje 2

Opis problema ??

Cilj ovog rada je napraviti funkcionalnu aplikaciju za upravljanje robotom i mapiranje okoline koristeći Unity razvojni program. Glavna stavka ili problem navedenog zadatka jest komunikacija između ROS-a i Unity-ja. Komunikacija će se odvijati pomoću ROS# knjižnice koja sadrži set alata za adaptaciju ROS-a Unity-ju.

Na samom početku bilo je potrebno istražiti koji ROS sustav koristiti. Postoji ROS 1 i ROS 2 sustav. ROS 1 se i dalje razvija i rade se nove verzije svake godine, no ROS 2 ga polako zamjenjuje zbog novijih tehnologija i bogatijih značajka. Zaključno je ipak bolje trenutno odabrati ROS 1. Najveći razlog za tu odluku stoji u tome što ROS# paket trenutno službeno podržava samo ROS 1. 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.

Nakon instalacije potrebnih alata, potrebno je iste konfigurirati na način da se mogu koristiti sa ROS# i Unity-jem. U to je uključena i potreba da se ROS# knjižnica uvede u Unity. Završetkom ovih osnovnih koraka omogućeno je kretanje implementacije glavnog cilja - aplikacija za upravljanje robotom i mapiranje okoline.

2.1 Unity i aplikacija

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 da bi se ga moglo ispravno prikazati i upravljati njime.

Pošto je Unity poznat kao dobar alat za razvijanje softvera za više platforma odjednom, ovim će se putem to iskoristiti na način da se ciljna aplikacija napravi da bude kompatibilna za:

1. Linux Ubuntu - Na prvom mjestu se nalazi Ubuntu iz razloga što se cijeli rad radi na ovom operacijskom sustavu, prvobitno zbog ROS-a.
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 preskočit će se.

Pri izradi aplikacije za svaku platformu, potrebne su neke dorade da bi aplikacija radila na toj platformi. Zahvaljujući Unity-ju i Mono framework-u na kojem Unity radi (.NET Standard 2.0) navedene dorade su dovoljno sitne da je prelazak na drugu platformu relativno bezbolan.

Poglavlje 3

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 se zada za prethodno navedenu svrhu.

Korištene su najnovije dostupne a 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.

3.1 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. Postoji i eksperimentalna verzija za Windows 10 i OS X, no ovaj će se rad usredotočiti na razvoj na Linux-u. Jedna od najbitnijih karakteristika ROS sustava jest da je omogućena komunikacija i upravljanje hardverom robota preko softverskih

alata ROS-a bez da se treba imati posebno znanje o korištenom hardveru.

ROS se ponajviše koristio u znanstvene i obrazovne svrhe, ali se zbog svoje praktičnosti i potencijala ubrzo proširio i u ostale grane robotike. Prije prelaska na ROS, svaki proizvođač robota je 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 ROS. Sa istim znanjem i vještinama moguće je razvijati softver koji će poslužiti na različitim robotima, različitih proizvođača, upravo radi ROS unificiranja.

ROS sadrži razne alate i knjižnice, koji su razvijeni i posloženi po određenoj ROS konvenciji. Sve zajedno jako pojednostavljuje razvoj novih robotskih softvera i omogućava kompleksno ponašanje robota. Također ROS sadrži razne upravljačke programe i algoritme, i sve je otvorenog koda - besplatno.[2]

Nedavno je razvijen i novi ROS, ROS 2. Prva službena verzija izdana je krajem 2017. Iako je preporučljivo koristiti više future-proof tehnologije, za ovaj rad odabran je ROS 1 iz razloga što neki od kritičnih alata za uspjeh rada nisu podržani na najnovijim verzijama ROS-a 2. Unatoč tome što je ROS 2 više future-proof i noviji, ROS 1 se i dalje razvija te nova verzija izlazi svake godine. Neke od razlika između ROS-a 1 i 2 navedene su u tablici 3.1.

Tablica 3.1 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, perzistentnost i rad u stvarnom vremenu (real-time).

3.2 Unity

Unity je cross-platform (multi-platformsko) 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 i ne besplatnih paketa koji se može koristiti za ubrzati i olakšati određene zadatke, a to se sve nalazi na tzv. "Unity Asset Store". [3]

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.

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

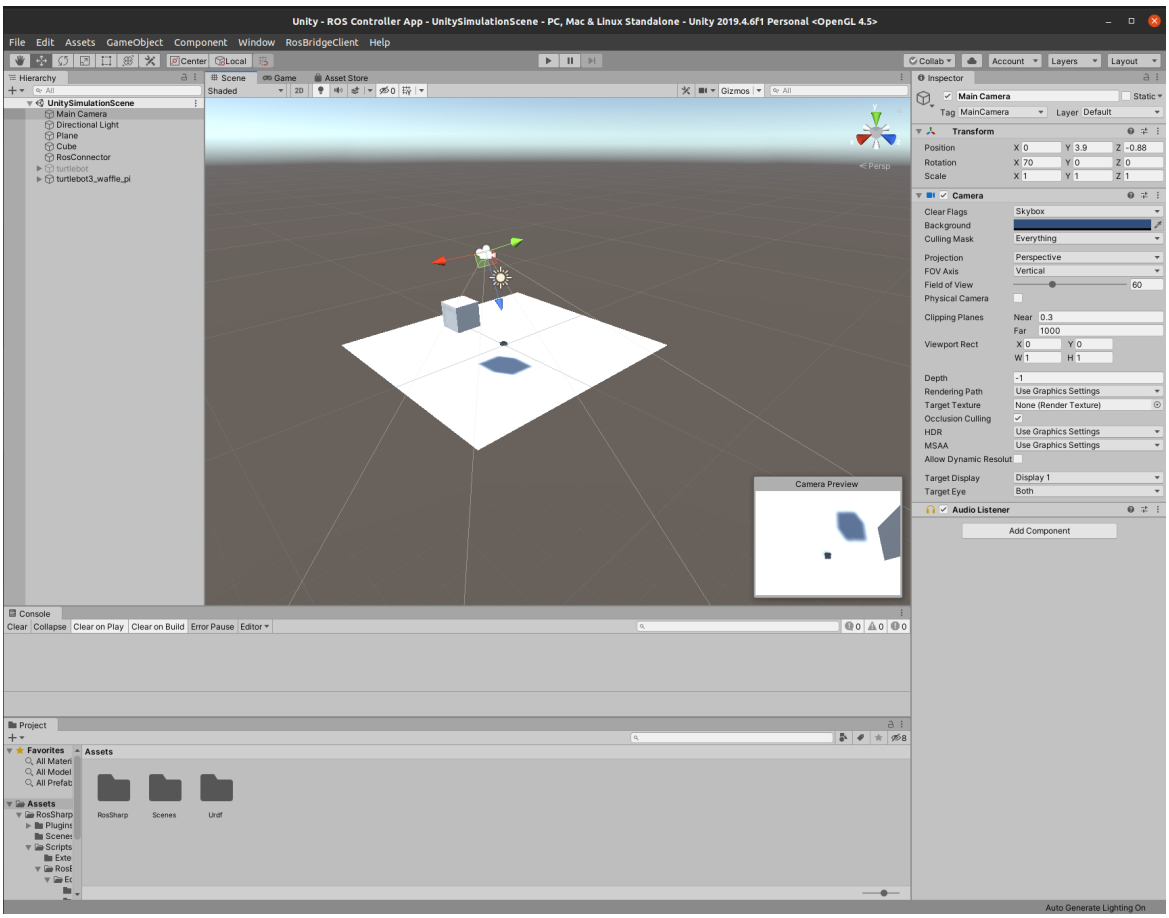
3.2.1 Editor

Unity Editor je glavni alat za razvoj (slika 3.1). U njemu se radi većina razvoja oko vizualnih elemenata i interakcije između njih. Glavne komponente uređivača su sljedeće:

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

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.

- Console (konzola) - Mjesto gdje se ispisuju sve poruke, upozorenja i greške.
- Project (projekt) - Ovime se može upravljati arhitekturom projekta. Od klasičnog stvaranja mapa i datoteka, imenovanja i ostalog, do dodavanja specifičnih Unity dodataka.
- 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 Store-u, panel za animacije i animator te ostale stvari koje nisu vezane za zadatak ovog rada.



Slika 3.1 Unity Editor

3.2.2 Skripte

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

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

- C# kompajler - Potpun kompajler, sa svim značajkama od C# 1.0 do 6.0.
- Mono Runtime [4] - 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 programskih 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 isprogramirano na pametan način.
  - Interoperabilnost - karakteristika da više programskih sustava mogu bez poteškoća međusobno funkcionirati.
- .NET Framework knjižnica klasa - *Mono* platforma pruža set klasa kao temelj za razvijanje aplikacija. Navedene su klase kompatibilne s Microsoftovim .NET Framework klasama.
- *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-platformsko radno 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-

buiran 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.

3.3 ROS#

ROS# je set programskih knjižnica i alata otvorenog koda u C#-u koja omogućuje komunikaciju između ROS-a i .NET aplikacija - Unity-ja, razvijen od strane Siemens kompanije. ROS# je bio primarno razvijen za Windows OS, ali se uspješno može koristiti i na druge operacijske sustave. Paket se može preuzeti s Unity Asset Store-a ili direktno sa službenog Github repozitorija. [5] Ovo je ujedno i glavni alat koji će se koristiti u ovom radu za stvaranje mosta između ROS-a i Unity-a.

Glavni sadržaj ovog paketa je sljedeći:

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

3.3.1 RosBridgeClient

RosBridgeClient za spajanje .NET baziranih aplikacija na ROS koristi *rosbridge_suite*. *rosbridge_suite* pruža ROS-u 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 poveznicu. [6] WebSocket je konstantna poveznica između klijenta i servera gdje se promet odvija u oba smjera te koristi TCP/IP protokol.

Poglavlje 4

Opis rješenja

U ovom će se poglavlju primarno opisati aplikacija i njezina arhitektura. Opisati će se kako je aplikacija napravljena i na kojem principu ona radi. Tri glavne komponente ovog sustava, kao što je već navedeno, su: ROS, ROS# i Unity. Na slici 4.1 prikazan je dijagram kako sustav izgleda gdje su nam navedene komponente glavni dijelovi dijagrama.

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

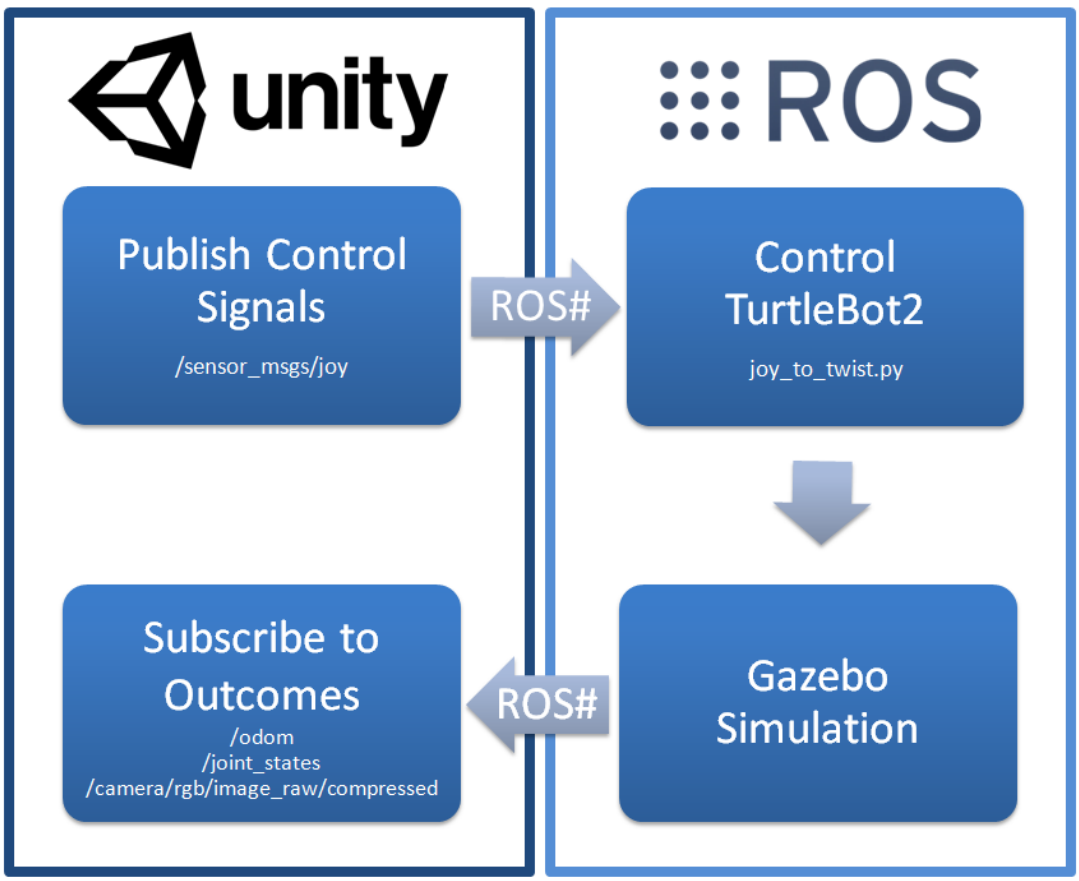
Drugi korak jest Unity. Pod Unity se misli na Editor i njegove standalone te mobilne aplikacije koje su implementirane i adaptirane sa istom svrhom i mogućnostima. Znači da koju god aplikaciju se pokreće, moći će se spojiti na navedeni robotski sustav.

Između navedena dva koraka, nalazi se ROS#, most koji spaja ROS i Unity. ROS# pomoću RosBridgeClient-a otvara vezu između njih. Isti radi na temelju pretplate i izdavanja poruka na ROS teme (topic). Jednom kad se veza otvorila, Unity se može pretplaćivati na ROS teme i izdavati nove poruke na ROS teme.

Na slici 4.1 prikazan je glavni princip prethodno navedenih mogućnosti:

- Unity se pretplaćuje na ROS teme - povratne informacije, npr. odometrija, slika kamere, laserscan,

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



Slika 4.1 Dijagram sustava [1]

4.1 Opis sučelja i Unity objekata

U ovoj će se sekciji opisati čemu služe glavni objekti u korištenim Unity scenama.

4.1.1 Meni

Napravljen je jednostavni inicijalni meni s nekoliko polja za upis:

- IP adresa - polje za unos IP adrese ROS-a, tj. robota. IP adresu, ako je nepromjenjiva, je dovoljno upisati jednom jer se sprema pomoću Unity značajke *PlayerPrefs* koja služi za spremanje igračevih (korisničkih) postavka.
- Broj robota - opcionalno polje u slučaju da ima više robota u simulaciji.
- Prefiks robota - opcionalno polje u slučaju da postoji više robota u simulaciji.

Connect gumb vodi na glavnu scenu gdje se vrši spajanje na ROS i gdje započinje sva interakcija. Sljedeće će se navesti svi elementi (objekti) u glavnoj sceni i čemu služe.

4.1.2 Ros Connector

Na glavnoj sceni najbitniji element, tj. objekt, jest **RosConnector** koji prvobitno služi za otvaranje veze prema ROS-u. Osim toga, njegova druga glavna uloga jest sadržavanje svih skripta kojima je zadaća pretplata ili izdavanje poruka na ROS temu. **RosConnector** sadrži sljedeće relevantne skripte:

- *Image Subscriber* koji se pretplaćuje na jednu ROS temu kamere, u ovom slučaju se koristi */camera/rgb/image_raw/compressed*.
- *Laser Scan Subscriber* koji se pretplaćuje na ROS temu laserskog skena (*/scan*).
- *Map Subscriber* koji se pretplaćuje na ROS temu */map* koju izdaje čvor *slam_gmapping*.
- *Odometry Subscriber* koji se pretplaćuje na temu */odom* i dohvaća odometrijske podatke robota.
- *Joystick Publisher* izdaje poruke na */joy* ROS temu, koja služi kao joystick

kontrola robota.

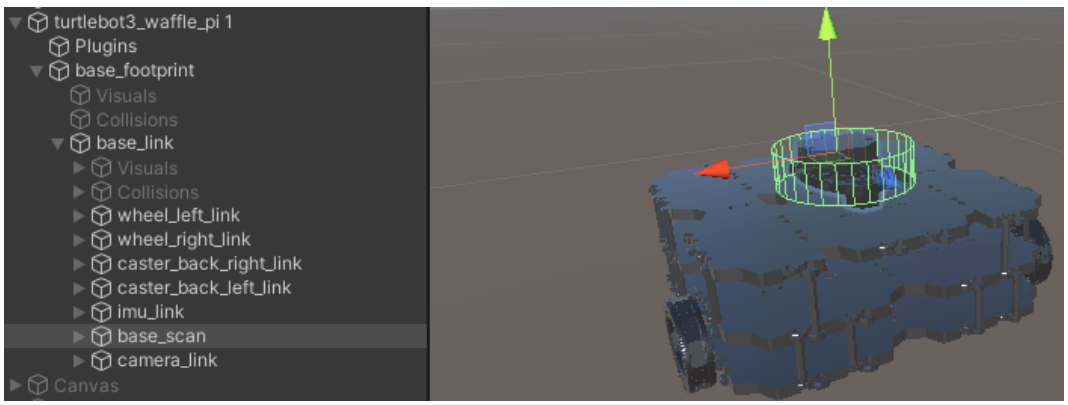
- *Twist Publisher Static* učitava ulaz gumbova na glavnoj sceni za upravljanje smjerom robota. Isti izdaje poruke na */cmd_vel* ROS temi koja upravlja linearnom i kutnom brzinom robota.

4.1.3 Plane

Plane je 3D objekt koji služi kao podloga (pod ili teren) robotu. Bez podloge bi se trebala isključiti svojstvo gravitacije u Unity-ju, no na taj način se može podloga iskoristiti za projekciju generirane mape okoline robota. Na način kada stavimo prikaz scene iznad navedenog objekta i robota, dobiva se efekt robota koji ide kroz mapu koja se paralelno izrađuje.

4.1.4 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 4.2.



Slika 4.2 Model robota u Unity-ju

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

4.1.5 Canvas

Canvas u Unity-ju je 2D objekt koji većinom služi za slaganje grafičkih komponenta sučelja, npr. gumbovi, tekst ulazi i sl. Jedna od najvećih prednosti Canvas-a je responzivnost. Canvas-u se može definirati bazna rezolucija, koja se onda može skalirati ovisno o rezoluciji uređaja gdje je aplikacija pokrenuta. Isto vrijedi za sve elemente u Canvas-u. 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:

- *Camera Image* kao što ime govori, ovo je element Canvas-a gdje nam se prikazuje slika s kamere. Slika kamere se učitava kao tekstura te ista preslikava kao *Sprite* na navedeni element koji sadrži polje za sliku. Element je proširen po cijeloj dužini i širini Canvas-a, tako da se ovisno o rezoluciji ekrana skalira.
- *Control* je objekt koji sadrži podobjekte koji su gumbovi za određivanje linearne i kutne brzine robota.
- *Laserscan Button* je gumb koji uključuje i isključuje prikaz objekata generiranih laserskim skenom.
- *Camera Button* je gumb koji mjenja prikaz kamere (topografski prikaz mape i robota te prikaz u prvom licu s kamere robota).
- *Switch Robot Camera* mjenja prikaz kamere u prvom licu na drugog robota (u slučaju da ima više robota u simulaciji). Ako u simulaciji postoji samo jedan robot, tada je taj gumb isključen.

4.1.6 Multi Robot Control

Navedeni objekt služi samo za pokretanje skripte koja omogućuje prikaz kamera dva robota odjednom.

4.2 Značajke i mogućnosti rješenja

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

4.2.1 Kamere

U sceni postoje dvije kamere. Kamera u Unity-ju 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 namještena 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.
- Prikaz kamere iz ptičje perspektive - U ovoj se perspektivi prikazuje 3D model robota na 2D mapi koja se izrađuje s dobivenim podacima iz */map* ROS teme, gdje se izrađuje nova tekstura koja se precrtava na *Plane* objekt. Također su u ovoj perspektive vidljive sfere (također 3D objekti) koji se crtaju shodno podacima iz */scan* ROS teme.

4.2.2 Laserski sken

ROS teme laserskog skena šalju poruke tipa *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 toga razloga je bilo potrebno napraviti ROS čvor koji će čitati temu laserskog skena, ROS tema */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* crtaju kao sfere u 3D prostoru oko robota.

4.2.3 Mapiranje

Mapiranje u Unity-ju se vrši na način da se učitavaju podaci iz ROS tema */map*. Ista šalje poruke tipa *nav_msgs/OccupancyGrid*. Najrelevantniji podaci ove teme su nam širina i dužina mape te int8 polje podataka u kojem se nalazi 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 podataka bude broj -1. Proces crtanja mape u Unity-ju je sljedeći:

1. Učitava se dužina i širina mape.
2. Inicijalizira se nova varijable za mapu 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 mapu 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 mape te se ista nova tekstura sprema na glavnu teksturu *Renderer* komponente na *Plane* objektu.

Korištena metoda mapiranja je *SLAM (Simultana Lokalizacija i Mapiranje) gmapping* koja je bazirana na podacima laserskog skena i pozicije robota. Ista je kreator prijašnje navedenih *Occupancy Grid* podataka (mreža popunjenosti) tj. mape.

Ispunjavanje novog polja boja u petlji je vrlo ne optimiziran pristup. U aplikaciji se osjećao trzaj u izvršavanju tijekom izrade mape, 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.

4.2.4 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 iznosti 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.
- Kombinacijom tipka (w, a, s, d) ili tipka strelica - Unity pomoću ROS# pomoćnih skripti koje učitavaju promjenu u ulazu *Input Manager-a*, pretvara ulaze u upravljačku naredbu za */joy* ROS temu. Nakon toga ROS# čvor *joy_to_twist* pretvara poruke */joy* ROS teme u poruke tipa *geometry_msgs/Twist* te i 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.

4.2.5 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 Turtlebot3 simulacija s više robota.

U početnom izborniku potrebno je upisati prefiks ROS tema ispred teme robota. Npr. postoji simulacija s više Turtlebot-ova 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 Turtlebot-a se ta ROS tema odnosi (slika 4.3). 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>
```

Ova je funkcija napravljena na način da u sceni postoji dodatni neaktivni *RosConnector* i model robota, koji se ovisno o ulazu na glavnom izborniku uključuje. Pri pretplaćivanju 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.

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.

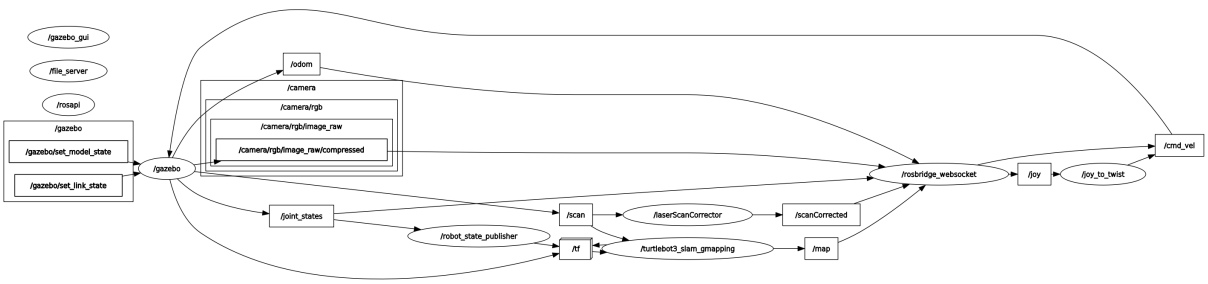
```
/tb1/camera/parameter_updates
/tb1/camera/rgb/camera_info
/tb1/camera/rgb/image_raw
/tb1/camera/rgb/image_raw/compressed
/tb1/camera/rgb/image_raw/compressed/parameter_descriptions
/tb1/camera/rgb/image_raw/compressed/parameter_updates
/tb1/camera/rgb/image_raw/compressedDepth
/tb1/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/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 4.3 Primjer ROS tema s više robota

4.2.6 RQT graf sustava

Na slici 4.4 prikazan je cjelovit RQT graf sustava. *rqt_graph* je grafički alat za prikaz ROS grafa pokrenutog sustava/aplikacije. U njemu se vidu poveznice između pokrenutih ROS tema i čvorova.

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 4.4 RQT graf sustava

Poglavlje 5

Rezultati

5.1 Aplikacija

Poglavlje 6

Rezultati

Bibliografija

[1] Siemens. Ros# gazebo simulacija. , s Interneta, https://github.com/siemens/ros-sharp/wiki/User_App_ROS_GazeboSimulationExample , 30. prosinca 2020.

[2] Ros službene stranice. , s Interneta, www.ros.org , 17. rujna 2020.

[3] U. Technologies. Unity asset store. , s Interneta, https://assetstore.unity.com , 22. rujna 2020.

[4] M. Project. Mono. , s Interneta, https://www.mono-project.com/docs/about-mono/ , 21. rujna 2020.

[5] Siemens. Ros#. , s Interneta, https://github.com/siemens/ros-sharp , 23. rujna 2020.

[6] ROS. Rosbridgesuite. , s Interneta, http://wiki.ros.org/rosbridge_suite , 03. sijecnja 2021.

[7] Microsoft. .net core instalacija. , s Interneta, https://docs.microsoft.com/en-us/dotnet/core/install/linux-ubuntu#2004- , 24. rujna 2020.

[8] ——. Visual studio code konfiguracija. , s Interneta, https://code.visualstudio.com/docs/other/unity , 24. rujna 2020.

[9] M. Project. Instalacija mono radne okoline. , s Interneta, https://www.mono-project.com/download/stable/#download-lin , 24. rujna 2020.

[10] Robotis. Instalacija turtlebot 3 paketa. , s Interneta, https://emanual.robotis.com/docs/en/platform/turtlebot3/pc_setup/#pc-setup , 24. rujna 2020.

[11] R. W. Tools. Rosbridge. , s Interneta, https://github.com/RobotWebTools/rosbridge_suite , 28. rujna 2020.

Sažetak

Ovo je tekst u kojem se opiše sažetak vašega rada. Tekst treba imati duh rekapitulacije što je prikazano u radu, nakon čega slijedi 3-5 ključnih riječi (zamijenite dolje postavljene općenite predloške riječi nekim suvislim vlastitim ključnim riječima).

Ključne riječi **— ključna riječ 1, ključna riječ 2, ključna riječ 3**

Abstract

This is a text where a brief summary of your work is outlined. The text should have a sense of recap of what was presented in the thesis, followed by 3-5 keywords (replace the general keyword templates below with some meaningful keywords of your own) .

Keywords **— keyword 1, keyword 2, keyword 3**

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 Ninjemys-a gdje je potrebno pratiti upute na službenim stranicama. Vrši se instalacija potpunog paketa.
3. Instalacija Unity-ja. 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#. [7]
6. Visual Studio Code konfiguracija za Unity. [8]
7. Instalacija *Mono* radne okoline. [9] Preporučljivo je nakon ovog koraka pokre-

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

8. Instalacija Turtlebot 3 paketa za ROS 1. [10]

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 ROS-u 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/deve/setup.bash ili *source /opt/ros/noetic/setup.bash*

- Za omogućiti mapiranje robotske okoline potrebno je dodatni instalirati *slam* pakete za mapiranje. Instalacija se vrši iz izvornog koda kojeg je moguće dohvatiti s git repozitorija paketa [**?**] gdje se nalazi i drugih paketa za robotsku percepciju.
- Osnovna spona koja omogućuje slanje podataka između ROS-a i Unity-ja je *RosBridge* paket kojeg inače dohvaćamo naredbom *sudo apt-get install ros-noetic-rosbridge-server*. Istoga koristi ROS#. ROS Noetic-u 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-

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 *roslaunch 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 ROS-a bile na Python 2, ali je ROS Noetic na Python-u 3. (slika A.1)

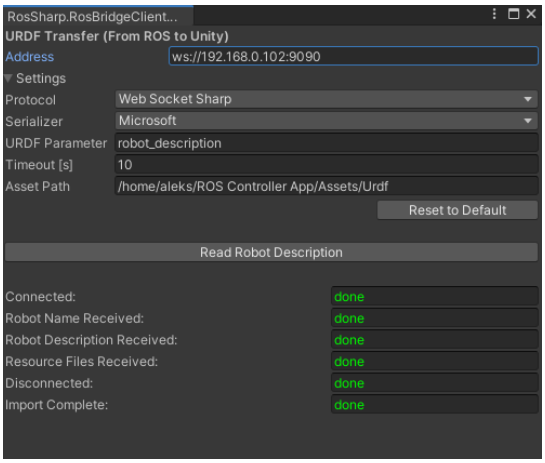
Još jedna promjena koja je potrebna a povezana je s verzijama Python-a je za uspješno pokretanje *RosBridge* poslužitelja. Naime, ako se izvrši instalacija standardnom *apt install* naredbom, RosBridge neće raditi jer se kose vrste varijabla (*str* i *byte*) koje su drugačije definirane u svakoj verziji Python-a, pa je iz tog razloga potrebno instalirati cijeli RosBridge iz izvornog koda [11] uz namještenu točnu verziju Python-a (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

```
<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:

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-sharp repozitoriju u sekciji wiki stranica. [5]

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