Navigacija mobilnog robota pomoću RGB-D kamere

Vartušek, Emil

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij računarstva

NAVIGACIJA MOBILNOG ROBOTA POMOĆU RGB-D KAMERE

Diplomski rad

Emil Vartušek

Osijek, 2019.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 21.09.2019.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Emil Vartušek
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-926R, 24.09.2018.
OIB studenta:	17871468413
Mentor:	Prof.dr.sc. Robert Cupec
Sumentor:	Doc.dr.sc. Damir Filko
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Emmanuel-Karlo Nyarko
Član Povjerenstva:	Doc.dr.sc. Damir Filko
Naslov diplomskog rada:	Navigacija mobilnog robota pomoću RGB-D kamere
Znanstvena grana rada:	**Automatizacija i robotika (zn. polje elektrotehnika)**
Zadatak diplomskog rada:	Korištenjem programskih alata raspoloživih u okviru ROS (Robot Operating System), izraditi računalni program za upravljanje mobilnim robotom, koji na temelju slike snimljene RGB-D kamerom izgrađuje kartu okoline robota, lokalizira mobilnog robota na toj karti te planira putanju od trenutne do zadane ciljne pozicije izbjegavajući prepreke na putu. Sumentor: Damir Filko
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene mentora:	21.09.2019.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 02.10.2019.

Ime i prezime studenta:	Emil Vartušek
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-926R, 24.09.2018.
Ephorus podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Navigacija mobilnog robota pomoću RGB-D kamere**

izrađen pod vodstvom mentora Prof.dr.sc. Robert Cupec

i sumentora Doc.dr.sc. Damir Filko

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Tablica sadržaja

1. UVOD

Roboti su već odavno među ljudima. Kako bi si čovjek olakšao rad, ali i svakodnevni život, stvorio je strojeve kako bi mu pomogli pri tome ili ga čak i zamijenili pri obavljanju određenih poslova. Kako su oni među nama već dugo vremena, isto toliko vremena trajalo je i pomicanje granica u njihovoj izradi te u njihovom usavršavanju. Danas postoji mnogo vrsta robota koji su sposobni obavljati svakakve funkcije, od onih i ne baš toliko bitnih, do toliko bitnih da čak i ljudski životi ovise o njima. Baš zbog toga, njihova primjena široko je raširena i moguće ih je pronaći u gotovo svakom segmentu današnjeg života. Zbog široke primjene i ovisno o funkcijama koje trebaju obavljati, pojavilo se mnogo različitih tipova robota: od robotskih manipulatora i mobilnih robota pa sve do humanoidnih robota. Kao zaključak može se reći da su roboti sve više i više prisutni u svakodnevnom životu ljudi, a rad na njima, tj. njihova izgradnja i programiranje, sve su raširenji diljem svijeta. U duhu toga, izabrana je i tema ovoga diplomskoga rada, Navigacija mobilnog robota pomoću RBG-D kamere.

Cilj ovoga diplomskoga rada bio je uspješno izvršiti navigaciju mobilnoga robota u prostoru čija karta je prethodno izrađena korištenjem RGB-D kamere. Prema tome, bilo je potrebno uspješno provesti proces mapiranja, odnosno izrade karte prostora te procese lokalizacije i navigacije mobilnog robota na toj karti.

Sam rad podijeljen je na dva dijela, teorijski dio i praktični dio. Teorijski dio rada opisan je kroz dva poglavlja, dok je praktični dio opisan unutar jednog poglavlja. Što se tiče poglavlja s teorijskim dijelom, prvo poglavlje nakon uvodnog opisuje softver korišten za rješavanje problema diplomskog rada, Robotski operacijski sustav, dok drugo poglavlje opisuje hardverske dijelove, odnosno sam mobilni robot i RGB-D kameru. Treće poglavlje posvećeno je praktičnom dijelu samoga rada, dok posljednje poglavlje predstavlja zaključak cijeloga rada.

1.1. Cilj diplomskoga rada

Korištenjem programskih alata raspoloživih u okviru *ROS* (engl. *Robot Operating System*), izraditi računalni program za upravljanje mobilnim robotom, koji na temelju slike snimljene RGB-D kamerom izgrađuje kartu okoline robota, lokalizira mobilnog robota na toj karti te planira putanju od trenutne do zadane ciljne pozicije izbjegavajući prepreke na putu.

2. ROBOTSKI OPERACIJSKI SUSTAV (ROS)

2.1. Zašto ROS?

Nakon što je sam zadatak diplomskog rada definiran, važno je odabrati alat kojim će se taj zadatak i izvršiti.

U današnjem svijetu postoji nekoliko alata, odnosno operacijskih sustava opće namjene za rad s robotima, kao što su *Microsoft Robotics Developer Studio, NAOQI, URBI* itd. No, najpoznatiji od njih, a u mnogo čemu i najpogodniji za rad, jest Robotski operacijski sustav (engl. *Robot Operating System)*, odnosno skraćeno *ROS*.

ROS, kao *open source* radni okvir (engl. *framework*) za izradu robota, predstavlja zajedničku softversku platformu za ljude koji izrađuju, programiraju i koriste robote. Na taj način, putem *ROS*-a, svi ti ljudi mogu podijeliti svoj napisani kod i ideje za rješavanje određenog zadatka, ali isto tako i preuzeti već neka gotova rješenja koja će im uvelike olakšati početak rada na njihovom robotu.

ROS, kao alat za izradu robota, veoma je uspješan te je do 2015. godine imao u svom sastavu više od 2000 paketa programa te je podržavao oko 80 komercijalnih robota, a do danas je taj broj samo rastao.

Sam *ROS* sastoji se od nekoliko dijelova koji ga čine veoma pristupačnim i pogodnim za korištenje, a samim time i veoma zastupljenim. Prvi dio *ROS*-a su skupovi *drivera* koji omogućuju čitanje podataka sa senzora i slanje naredbi aktuatorima i motorima na veoma sažet i dobro definiran način. Drugi dio predstavlja rastući, ali i već veliki skup osnovnih algoritama za izradu i programiranje robota. Treći dio odnosi se na samu infrastrukturu *ROS*-a koja omogućuje rad s podacima i njihovo prebacivanje s računala na računalo (ili na bilo koju drugu platformu) te povezivanje različitih komponenti složenog robotskog sustava te uključivanje vlastitih algoritama u taj sustav. Četvrti dio čine brojni alati koji omogućuju laku vizualizaciju robota i algoritama, spremanje podataka sa senzora te otklanjanje pogrešaka (engl. *debug*). Posljednji, peti dio *ROS*-a, odnosi se na velik broj izvora za pomoć pri radu u *ROS*-u, kao što su službena *wiki* stranica, forum gdje se mogu postaviti pitanja i podijeliti mišljenja te na veliku zajednicu uspješnih korisnika i razvijača *ROS*-a.

Zbog svih mogućnosti koje ovaj alat nudi te zbog samog načina na koji on funkcionira, *ROS* je postao vodećim alatom za stvaranje i programiranje svih vrsta robota, pa tako i mobilnih robota. Uz veliku podršku za programiranje mobilnih robota, *ROS* također pruža i velike mogućnosti pri izradi karti, tj. mapiranju, te pri samoj lokalizaciji i navigaciji robota u prostoru.

Slika 2.1. Logo *ROS*-a

2.2. Što je ROS?

ROS je radni okvir (engl. *framework*) namijenjen za pisanje robotskog softvera, tj. programa uz pomoć kojih će se realizirati rad robota [1]. *ROS* predstavlja skup alata, biblioteka i pravila koje su tu kako bi olakšale cijeli problem izrade kompleksnog i robusnog robota, jer takav je problem veoma teško riješiti.

Proces izrade ovakvoga robota najbolje je objasniti na primjeru, na kojemu će se također objasniti i zašto je *ROS* pogodan za rješavanje takvih problema.

Uzmimo primjer koji je za čovjeka veoma jednostavan: prenijeti neki predmet s jedne lokacije na drugu [2]. Recimo da se radi o šalici koju je potrebno prenijeti od kuhinjskog stola do perilice posuđa. Kada čovjek dobije taj zadatak, odlazi do kuhinjskog stola, uzima šalicu, odlazi do perilice posuđa i stavlja šalicu unutar perilice. No što kada se robotu da takav zadatak? Kako uopće dati robotu takav zadatak? Prvi problem već se tu pojavljuje, jer potrebno je smisliti neki način na koji će robot shvatiti što uopće je njegov zadatak. Nadalje, robot kao takav mora znati nalazi li se uopće u ispravnoj prostoriji, tj. prostoriji u kojoj se nalazi kuhinjski stol te mora moći prepoznati i predmete kao što su šalica i perilica posuđa. Čak i kada prepoznaje sve to, mora znati gdje se točno nalazi, kako i kojim putem doći do određene pozicije te koju radnju napraviti kada dođe do tamo. Sve ove stvari za čovjeka nisu nikakav problem, ali za jednog robota ovakve stvari traže veoma kompleksna rješenja. Baš zbog kompleksnosti tih rješenja, kako ovoga primjera, tako i ostalih primjera iz stvarnoga svijeta, *ROS* se pokazao kao izvanredan alat za njihovo rješavanje.

Svako rješavanje ovakvih problema, bilo za pojedinca ili grupu ljudi, bez ikakvih prijašnjih rješenja, odnosno „od nule“, bilo bi veoma otežano. Tu onda nastupa *ROS*! *ROS* je zamišljen kao operativni sustav koji podržava zajedničko razvijanje robotskoga softvera, što znači da se svaki kompleksan problem može podijeliti na manje kompleksne cjeline koje se zatim odvojeno rješavaju. Na razini gore navedenoga primjera, to bi značilo da će svaki segment toga primjera

rješavati određena skupina ljudi: skupina koja će se baviti mapiranjem prostora, skupina koja će se baviti prepoznavanjem objekata, skupina koja će se baviti navigacijom samoga robota itd. Zbog samog načina na koji *ROS* funkcionira i na koji je zamišljen, sva ova rješenja kasnije se ujedinjuju u jedinstveno rješenje, tj. u rješenje cjelokupnoga problema i na kraju se dobije sasvim funkcionalan robot koji radi ono što bi i trebao, a u ovom slučaju to je prenošenje šalice od kuhinjskog stola do perilice posuđa.

Iz navedenog primjera [2], moguće je vidjeti koliko *ROS* zapravo olakšava sav posao vezan uz izgradnju i programiranje robota. Naravno, ove probleme moguće je riješiti i bez *ROS*-a, ali zašto ih rješavati na teži i duži način kada *ROS* toliko toga olakšava i omogućava od samoga početka rješavanja problema? Uz to, zajednica ljudi koji koriste *ROS* je veoma široka te samim time postoji velik broj već napisanih algoritama koji se mogu iznova primijeniti na neki novi problem. Prema tome, moglo bi se reći da je *ROS* alat koji znatno štedi vrijeme izrade i programiranja robota, baš zato što nudi sve dijelove izrade i programiranja robota koje bi inače trebalo samostalno napisati pri rješavanju svakog problema, te isto tako omogućuje i da se njegovi korisnici posvete stvarima koje ih zanimaju, a ne onima koje im nisu potrebne.

2.3. Povijest ROS-a

ROS je nastao kao potreba za zajedničkim radnim okvirom koji će ujediniti mnoge ljude koji su se bavili istraživanjem robotike te kao takav ima mnogo preteča i suradnika, tj. ljudi koji su doprinijeli njegovom stvaranju. Prve od tih preteča predstavljali su brojni projekti na sveučilištu Stanford sredinom 2000-ih gdje su se razvijale tehnologije umjetne inteligencije kao što su *Stanford AI Robot* (*STAIR*) te *Personal Robots* (*PR*) program i u kojima se koristio sličan softver *ROS*-u. Nadalje, 2007. godine, tvrtka *Willow Garage Inc*. uvelike je doprinijela proširenju koncepata započetih na sveučilištu Stanford izradom i testiranjem određenih implementacija. Najveća priznanja ovdje odlaze mnogim istraživačima koji su posvetili svoje vrijeme i stručnosti kako bi pomogli pri izradi temeljnog dijela, tj. osnovnih paketa softvera koji zajedno čine *ROS*.

Ovakav način razvoja softvera za izgradnju i programiranje robota, u početku, činio se besmislenim i veoma teškim, baš zbog toga što se softver razvijao na nekoliko lokacija istovremeno. No, kasnije se to pokazalo kao najveća prednost *ROS*-a, jer je na taj način bilo tko mogao započeti svoj projekt na svom serveru i raditi na njemu bez traženja ikakve dozvole i bez ikakvog nadzora. Baš zbog toga, *ROS* danas i je toliko raširen i popularan te je to razlog zbog kojega danas postoji više desetaka tisuća korisnika *ROS*-a diljem svijeta.

Od 2010. godine, kada je izašla prva distribucija *ROS 1* operacijskog sustava pod nazivom *Box Turtle*, izašlo je još 11 dostribucija, od kojih su neke *Diamonback, Groovy Galapagos, Hydro Medusa, Indigo Igloo, Kinetic Kame, Lunar Loggerhead, Melodic Morenia* itd. Od 2015. počele su izlaziti i *ROS 2* distribucije, a do sada izašlo ih je samo nekoliko: *Ardent Apalone, Bouncy Bolson, Crystal Clemmys*.
U sklopu izrade diplomskog rada, izabrana je distrubucija operacijskog sustava *ROS 1, Kinetic Kame*.

2.4. Filozofski aspekti ROS-a

Kao i svaki operacijski sustav, i *ROS* koristi određene filozofije i prakse, tj. obilježja, kojima se vodi pri svome radu. U slučaju ROS-a, te filozofije i prakse prate filozofije i prakse *Unix* operacijskih sustava, što *ROS* čini „lakšim“ za korištenje *Unix* korisnicima, odnosno svima koji imaju pozadinu rada i razvoja aplikacija u *Unix* okolini.

Što se tiče samih filozofskih aspekata na kojima *ROS* počiva, ima ih 5*: peer to peer, tools-based, multilingual, thin* te *free and open source*.

Peer to peer aspekt odnosi se na sam sustav *ROS*-a koji je sastavljen od velikog broja računalnih programa koji su međusobno povezani i koji neprestano međusobno izmjenjuju poruke koje putuju direktno iz jednog programa u drugi. Ovaj aspekt čini *ROS* veoma kompleksnim zbog načina na koji se podaci dijele između programa i komponenti, ali isto tako osigurava i bolji rad s velikim količinama podataka koje se moraju poslati ili primiti.

Tools-based aspekt odnosi se na velik broj zasebnih alata, odnosno programa koji zapravo omogućuju rad *ROS*-a. Svi ovi alati zaduženi su za obavljanje određenog zadatka, ali zajedno čine *ROS*. Iz ovoga segmenta najlakše je i vidjeti sličnosti s *Unix* operacijskim sustavima, jer i oni sami funkcioniraju na isti način. Za razliku od ostalih softverskih radnih okvira za rad u robotici, *ROS* nema integrirano razvojno okruženje za rješavanje problema. Svaki dio *ROS*-a, tj. alat, zasebno se implementira u projekt koji se trenutno rješava, što također predstavlja prednost *ROS*-a. Što se tiče samih alata, oni su brojni i omogućuju mnoge stvari kao što su navigacija izvornim kodom programa, vizualizacija svih veza sustava, grafički prikaz protoka podataka, generiranje dokumentacije itd. Jedan od razloga zbog kojih je ovakav način rada dobar je neprestana potreba za implementacijama novih alata, tj. alata koji će više odgovarati određenom projektu od onih već postojećih. Važno je reći i kako neke od posljednjih verzija *ROS*-a

dozvoljavaju spajanje više alata u isti proces rješavanja projekta, ali i dalje su ti svi alati zasebni i ne utječu jedni na druge.

Multilingual aspekt odnosi se na mogućnost korištenja bilo kojeg programskog jezika pri pisanju *ROS* modula, tj. programa. Jedini uvjet je da za taj programski jezik postoji već napisana biblioteka koju će korisnici koristiti kako bi pisali u tom željenom programskom jeziku. Ovaj aspekt jedan je od najvažnijih aspekata *ROS*-a jer mnogi korisnici dolaze iz različitih pozadina programiranja: od onih koji programiraju u visoko produktivnim programskim jezicima kao što su *Python* i *Ruby*, do onih koji programiraju u puno brže izvodljivim programskim jezicima kao što je *C++* ili u programskim jezicima kao što su *Lisp* i *MATLAB*. Baš zbog tih korisničkih preferenci pri odabiru programskog jezika, ali i zbog nedostatka jednog programskog jezika koji bi predstavljao „najbolje" rješenje pri programiranju u *ROS*-u, *ROS* je omogućio da svaki korisnik sam izabere bilo koji od programskih jezika za koji postoje napisane korisničke biblioteke. Do 2015. godine, korisničke biblioteke bile su napisane za programske jezike *Python*, *C++*, *Java*, *JavaScript*, *Lisp*, *MATLAB*, *Ruby* i mnoge druge. Važno je napomenuti i da sve ove korisničke biblioteke međusobno komuniciraju, od kojih svaka ima način na koji prosljeđuje određenu poruku ostatku mreže, tj. ostalim bibliotekama te da će u ovom diplomskom radu biti korištena korisnička biblioteka za rad u programskom jeziku *Python*, zbog ranijeg iskustva programiranja u tom programskom jeziku, ali i zbog jednostavnosti koju taj programski jezik pruža pri radu.

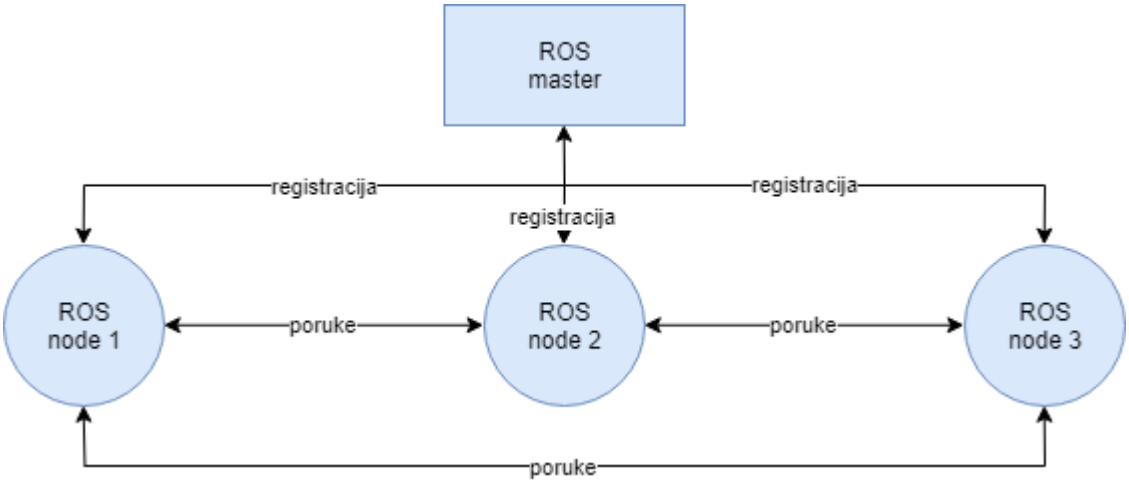
Thin aspekt odnosi se na mogućnost i poticaj korisnicima na pisanje zasebnih biblioteka napisanih na način da mogu primati poruke od ostalih *ROS* modula te slati poruke ostalim *ROS* modulima. Na ovaj način, te zasebno napisane biblioteke mogu se koristiti i izvan *ROS*-a, tj. unutar drugih aplikacija, ali i uvelike olakšavaju izgradnju automatiziranih testova kroz korištenje standardnih, ranije u *ROS* ugrađenih alata. Također, svaka od ovih napisanih biblioteka može se iznova iskoristiti, a sam način na koji su pisane, svaka za sebe, čini ih manjima obujmom te manje kompleksnima.

Free and open source aspekt odnosi se činjenicu da je *ROS* besplatan za korištenje u komercijalne i ne-komercijalne svrhe, a to mu omogućuje *BSD* licenca kojom je „zaštićen". Ovakva licenca omogućuje minimalnu zabranu toga što se s *ROS*-om i u *ROS*-u može raditi te omogućuje nesmetanu distribuciju softvera napisanog u *ROS*-u. Za slanje podataka između različitih modula, *ROS* koristi međuprocesnu komunikaciju (engl. *inter-process communication* ili IPC), što uvelike doprinosi njegovom *open source* konceptu. Na kraju, kako je ovaj diplomski rad akademske prirode i samim time neće biti primijenjen u komercijalne svrhe, valja reći da je ovaj aspekt zaslužan i za odabir *ROS*-a kao softvera za rješavanje samoga rada.

2.5. Osnovni pojmovi ROS-a

Osnovni cilj *ROS*-a je obavljati što veći broj programa istovremeno, odnosno u paraleli koja omogućuje izmjenu podataka sinkrono i asinkrono. Na primjeru navedenom u poglavlju 2.2. moglo se vidjeti koliko kompleksan može biti jedan jednostavan primjer samo u smislu obavljanja problema zadanog u tom primjeru. Ta kompleksnost višestruko se povećava kada na red dođe rješavanje svakog od pojedinih dijelova problema. Samo kretanje mobilnog robota predstavlja problem kojega treba riješiti i u kojemu je najvažnija stavka da sam mobilni robot istovremeno obavlja nekoliko funkcija koje su važne za taj dio problema. U ovom slučaju, neke od tih funkcija bi bile pokretanje i nastavak rada senzora, pohrana podataka koji pristižu od senzora, obrada tih podataka, upravljanje motorima za pokretanje mobilnog robota itd. Prema tome, za ispravan rad ovoga primjera važno je da se svi ovi procesi obavljaju neprekidno i paralelno, a u isto vrijeme i efikasno. Kako bi se to sve omogućilo, koristi se *ROS* i skup njegovih koncepata koji se svi zajedno nalaze pod nazivom *Computation graph model*.

Computation graph model je *ROS*-ov model koji se koristi pri izgradnji i programiranju robota. Ovaj model sastoji se od nekoliko koncepata koji obilježavaju rad u *ROS* operacijskom sustavu, a sam model radi na sljedeći način: svaki *ROS* proces, odnosno program, predstavlja jedan čvor (engl. *node*) ovoga grafa koji je povezan s drugim čvorovima putem tema koje se nazivaju *topics*. Čvorovi putem ovih tema mogu izmjenjivati poruke s ostalim čvorovima, mogu zatražiti usluge od drugih čvorova ili pružiti usluge drugim čvorovima, mogu zatražiti podatke od servera parametara ili mu ih predati itd. Ono što sve ovo čini mogućim je *ROS* proces koji se naziva *master*, a njegova zadaće je da omogući komunikaciju između čvorova putem tema te da kontrolira ažuriranje podataka na serveru parametara. Same poruke i usluge između čvorova ne prolaze kroz *master*, ali je *master* taj koji uspostavlja *peer to peer* mrežu između svih čvorova koji su registrirani na taj *master*. Ovakav način rada *ROS*-a, kroz *computation graph model*, pokazao se veoma pogodnim za izgradnju i programiranje robota, kako su u većini slučajeva roboti sastavljeni od više podsustava, pa čak i od više komada hardvera ili se njihovi programi nalaze na više računala istovremeno.



Slika 2.2. Primjer *computation graph modela* [3]

Iz objašnjenja na koji način *computation graph model ROS*-a radi, moguće je vidjeti da postoji nekoliko glavnih koncepata koji ga izgrađuju. Ti koncepti su: *nodes* (čvorovi), *topics* (teme), *master* (glavni čvor), *messages* (poruke), *services* (usluge), *actions* (akcije), *bags* (skupovi podataka) i *parameter server* (server parametara).

Čvorovi predstavljaju pokrenute *ROS* procese, odnosno programe. Oni mogu predstavljati procese vezane uz senzore, motore, obradu podataka itd., a svaki od tih čvorova ima svoje ime kojim se registrira u *master* čvoru ili može biti anoniman, ali u tom slučaju pridružuje mu se identifikacijska oznaka kojom će se taj čvor moći prepoznati. Čvorovi predstavljaju središnji dio programiranja u *ROS* operacijskom sustavu, jer se većina napisanog koda nalazi u samim čvorovima koji tada obavljaju one radnje za koje su namijenjeni. Tako *ROS* čvorovi primaju podatke od drugih čvorova, šalju podatke drugim čvorovima te šalju i primaju zahtjeve za djelovanjem od drugih čvorova.

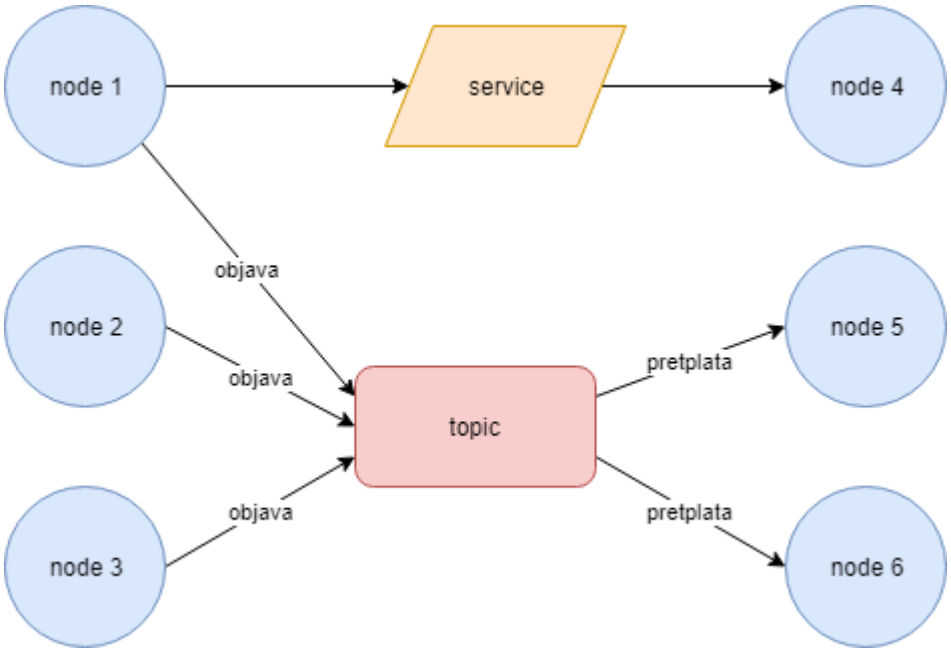
Teme predstavljaju poveznice između čvorova koje su bazirane na preplata/objava (engl. *subscribe*/*publish*) sustavu dijeljenja podataka. Putem njih, čvorovi razmjenjuju poruke na način da kada žele poslati poruku, oni moraju objaviti tu poruku određenoj temi, a kako bi primili određenu poruku, moraju biti pretplaćeni na temu kojom poruka dolazi. Sam prijenos podataka je anoniman, u smislu da čvor ne zna koji drugi čvor šalje ili prima određene podatke, nego samo zna kojom temom će poruka stići ili biti poslana do i od tog čvora. Sami podaci koji se šalju putem tema su asinkroni i uvijek jednako strukturirani, a primjeri podataka koji se šalju su: podaci sa senzora, naredbe za upravljanje motorima i aktuatorima itd.

Glavni čvor predstavlja čvor na kojega se svi ostali čvorovi registriraju. Njegova važnost je u tome što nakon deklaracije i registracije, ostali čvorovi mogu međusobno komunicirati,

odnosno slati i primati poruke i podatke. Prema tome, glavni čvor predstavlja čvor od kojega sve kreće i čvor bez kojega izrada i programiranje robota putem *ROS*-a ne bi bili mogući. Dio glavnog čvora je i još jedan od glavnih koncepata *computation graph* modela, a to je server parametara.

Poruke predstavljaju složenu podatkovnu strukturu koja se sastoji od jednostavnijih tipova podataka kao što su *integer*, *floating point*, *string*, *Boolean* itd. i samih poruka koje su predstavljene kao rekurzivne strukture.

Usluge predstavljaju radnje koje imaju jedinstveno rješenje, a čvorovi ih mogu pozvati ili pružiti bilo kada. Usluge se najčešće koriste kod radnji koje imaju definiran početak i kraj, odnosno kod radnji kod kojih će prijenos podatak biti sinkron, što je i glavna razlika u odnosu na teme.



Slika 2.3. Razlika između teme i usluge

Akcije, slično kao usluge, predstavljaju radnje koje čvorovi mogu bilo kada pozvati ili pružiti. No, za razliku od usluga, akcije se koriste kada je radnje potrebno obavljati kroz neko dulje vrijeme i kada glavni cilj izvršavanje samih radnji. Također, prijenos podataka kod akcija je asinkron.

Skupovi podataka predstavljaju format u kojemu je moguće spremiti sve poslane poruke između čvorova, a zatim ih i ponovno prikazati kada je to potrebno. Ovaj način spremanja poruka odnosno podataka koji se u tim porukama nalaze, može uvelike doprinijeti kod rješavanja određenih problema prilikom programiranja, kao što su simuliranje podataka i ispravljanje pogrešaka.

Server parametara predstavlja bazu podataka koji su bili podijeljeni između čvorova. Ovdje se najčešće spremaju podaci koji nisu podložni promjenama i podaci koji će se često koristiti pri izradi i programiranju robota.

2.6. Alati ROS-a

Kao što je već ranije spomenuto, jedan od pet glavnih aspekata *ROS*-a su alati, od kojih svaki ima određenu svrhu. Ovi alati uvelike doprinose mogućnostima *ROS*-a, jer olakšavaju i pojednostavljuju rješenja određenih problema pisanih u njemu. Kao i svi ostali algoritmi, ovi alati dolaze u tzv. *ROS* paketima u kojima se nalaze brojni alati za rješavanje zadataka vezanih uz bilo koji dio robota.

U *ROS*-u postoji mnogo alata koje je moguće koristiti pri izradi i programiranju robota, ali neki od najvažnijih su: *rviz, rosbag, catkin, rosbash* te *roslaunch*.
Alat *rviz* predstavlja alat koji omogućuje trodimenzionalnu vizualizaciju robota i okruženja u kojima se ti roboti nalaze i rade, ali isto tako i vizualizaciju podataka prikupljenih sa senzora. Prikaz tih podataka moguć je na razne načine, ovisno o potrebama.

Alat *rosbag* predstavlja alat kojim se prave, odnosno snimaju i zatim ponovno reproduciraju podaci koji se šalju porukama, a ti podaci se zatim spremaju u *bag* formatu. Ovi podaci mogu se iznova i iznova koristiti. Sam alat *rosbag* predstavljen je linijskom naredbom, ali moguće ga je prikazati i preko *GUI* (*Graphical User Interface*) sučelja korištenjem alata *rqt_bag*.

Alat *catkin* predstavlja sustav za izgradnju, odnosno „buildanje“ paketa u *ROS*-u, a njegov prethodnik, odnosno alat koji je *catkin* naslijedio, bio je *rosbuild*. Ovaj alat je *open source* te neovisan o jeziku u kojemu se radi.

Alat *rosbash* predstavlja skupinu alata koji povećavaju funkcionalnost tzv. *bash* ljuske, odnosno ljuske koja predstavlja dio *Unix* operacijskog sustava, kako je *ROS* usko povezan s tim programskim jezikom. Glavna svrha ovoga alata je omogućavanje korištenja određenih naredbi koje se inače koriste u *Unix* operacijskom sustavu te pokretanje nekih novih naredbi koje u tom

operacijskom sustavu nisu prisutne. Naredbe preuzete iz programskog jezika *Unix*, *ls*, skraćeno od *list*, *cd*, skraćeno od *change directory* i *cp*, skraćeno od *copy*, u *ROS*-u se pojavljuju u oblicima *rosls*, *roscd* i *roscp*, dok se kao potpuno nove naredbe ovdje pojavljuju naredbe *rosed* i *rosrun*, koje omogućuju izmjenu određenih datoteka u proizvoljnom uređivaču teksta, odnosno pokretanje programa napisanih u *ROS*-u.

Alat *roslaunch* omogućuje pokretanje više čvorova istovremeno, neovisno o tome pokreću li se lokalno ili s udaljenosti, te sprema određene parametre na server parametara. Korištenjem ovoga alata lako se može automatizirati veoma složen proces pokretanja svih programa napisanih u *ROS*-u i to do te mjere da se oni mogu pokrenuti samo jednom naredbom.

2.7. Paketi ROS-a

Osnovna građevna jedinica za organizaciju softvera unutar *ROS*-a naziva se paket (engl. *package*). Paket predstavlja direktorij u kojem se nalaze čvorovi, vanjske biblioteke, podaci i mnoge druge stvari koje su potrebne za izvođenje projekta u *ROS*-u. Mnogo paketa uključeno je u razne distribucije *ROS* operacijskog sustava, ali mnogo je i onih paketa koji u njih nisu uključeni i koje su napisali razni korisnici *ROS*-a diljem svijeta u svrhu obavljanja različitih zadataka. Više paketa na jednom mjestu čini stog (engl. *stack*), koji zatim nudi sve funkcionalnosti paketa koji se nalaze u njemu.

Postoji mnogo paketa s brojnim funkcionalnostima u sklopu *ROS* operacijskog sustava, ali i izvan njega. Tako postoje osnovni paketi *ROS*-a koji dolaze njegovim instaliranjem, paketi za mapiranje i lokalizaciju, paketi za navigaciju, percepciju, simulaciju itd. Svaki od ovih paketa sadrži određene alate i naredbe koje su karakteristične za taj paket i koje pružaju odgovarajuće funkcionalnosti.

Tako se među paketima sustava i alata mogu izdvojiti paketi *actionlib* za standardizaciju sučelja u kojemu se nalaze određeni zadaci, *nodelet* za pokretanje više algoritama u jedinstvenom procesu te *rosbridge* koji pružaju *ROS* funkcionalnosti ne-*ROS* programima.

Među paketima za mapiranje i lokalizaciju mogu se pronaći paketi *gmapping* koji omogućuje uporabu *SLAM* (*Simultaneous Localizaton And Mapping*) algoritama, *cartographer* koji omogućuju korištenje 2D i 3D *SLAM* algoritama, te *amcl* koji omogućava korištenje adaptivne *Monte-Carlo* lokalizacije.

Među paketima za navigaciju nalazi se paket *nav2d* za navigaciju mobilnog robota ravnim površinama, među paketima za percepciju nalazi se paket *vision_opencv* koji omogućuje

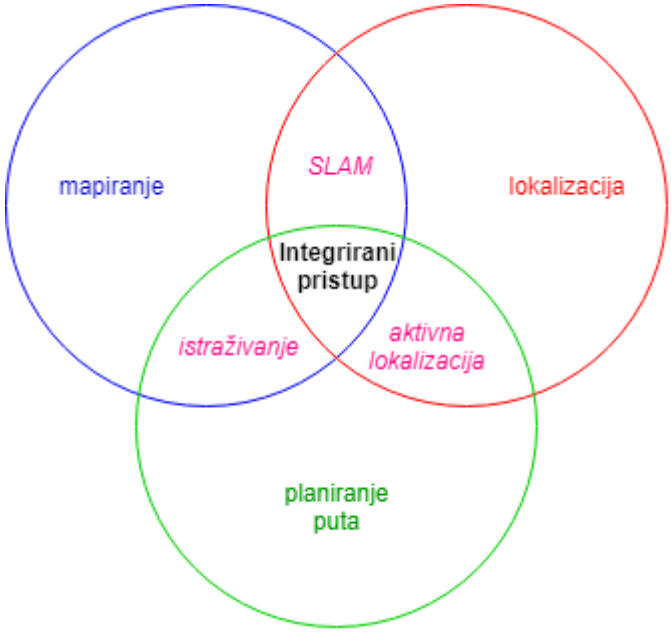
korištenje *OpenCV* biblioteka u *ROS*-u, dok se među paketima za simulaciju mogu pronaći paket *gazebo_ros_pkgs* koji omogućuje korištenje *Gazebo* simulatora u *ROS*-u i paket *stage* koji pruža sučelje za 2D *Stage* simulator.

2.8. Mapiranje, lokalizacija i navigacija

Kako je tema ovog diplomskog rada navigacija mobilnog robota RGB-D kamerom, važno je da *ROS* pruža određene mogućnosti za rješavanje tog problema. Kao što se moglo vidjeti iz prethodnog poglavlja, *ROS* sadrži pakete namijenjene baš za probleme mapiranja, lokalizacije i navigacije. No, što uopće znače ti pojmovi?

Mapiranje predstavlja izgradnju karte. U ovom slučaju, ono predstavlja izgradnju karte kojom će se mobilni robot kretati. No, kako bi se mobilni robot kretao, on mora znati na kojoj poziciji na toj karti se uopće nalazi. Taj dio problema naziva se lokalizacija. Na kraju, kada je karta izrađena, a pozicija robota određena, preostaje još samo kretanje robota tom kartom od točke A do točke B, odnosno planiranje puta (navigacija) mobilnog robota.

No, kako s izradom i programiranjem robota, a i samim *ROS*-om, ništa nije jednostavno, tako ni ove probleme nije jednostavno riješiti. Kako bi se ovaj zadatak ispravno izvršio, potrebno je da sam robot „razumije“ okolinu kojom se kreće. To znači da robot mora znati tumačiti određena očitanja sa senzora te znati kako promjena podataka sa senzora utječe na radnje mapiranja, lokalizacije i planiranja puta, odnosno mora znati kako to utječe na okolinu i njegov položaj u toj okolini.



Slika 2.4. Odnos mapiranja, lokalizacije i planiranja puta [4]

Sa slike 2.4. moguće je vidjeti odnose između 3 glavna problema: mapiranja, lokalizacije i planiranja puta. Tako je moguće vidjeti za što se sve treba pobrinuti i koje metode treba koristiti kako bi se problem riješio.

Karta predstavlja područje, odnosno okruženje u kojemu robot obavlja određene radnje i kao takva trebala bi se sastojati od dovoljnog broja informacija kako bi se određeni zadatak moga izvršiti. Karta može biti metrička, topološka ili hibridna. Kako bi se izgradila karta, potrebno je provesti proces mapiranja. No, kako bi se mapiranje izvršilo, u isto vrijeme potrebno je znati i točnu poziciju u kojoj se robot u trenutku mapiranja nalazi, ali isto tako važno je i planirati daljnji put tog robota u odnosu na kartu koja je do sada već izrađena. Prema tome, lokalizacijom bi se trebala odrediti točna pozicija u odnosu na neku referentnu poziciju, planiranjem puta trebala bi se odrediti ruta kojom će se robot kretati od točke A do točke B i pri tome prikupljati podatke potrebne za mapiranje tog prostora, te na kraju mapiranjem prostora izraditi kartu prostora kojim je prošao dok je bio na putu od točke A do točke B. Sva tri ova dijela od presudne su važnosti pri rješavanju problema navigacije robota, no 2 od 3 ova problema rješavaju se uz pomoć istoga alata, a to je tehnologija simultane lokalizacije i mapiranja (engl. *Simultaneous Localizaton And Mapping*), odnosno *SLAM*, dok se treći problem planiranja puta rješava kao poseban dio, ali istovremeno kada se rješavaju i preostala dva.

SLAM je računalno rješenje koje omogućuje izgradnju ili nadopunu karte nepoznatog prostora te u isto vrijeme, na toj karti, postavlja poziciju u kojoj se robot, ili bilo što drugo,

trenutno nalazi [5]. Iako ovo zvuči kontradiktorno, jer robot ili bi trebao imati već izrađenu kartu da zna gdje se nalazi ili bi trebao znati gdje se nalazi da zna da li taj prostor treba mapirati, postoje algoritmi koji ovaj problem rješavaju i to s veoma dobrim rezultatima. Svi *SLAM* algoritmi oslanjaju se na podatke primljene sa senzora. Senzori mogu biti 1D, 2D ili 3D, a ovisno o tome najčešće se koriste laseri ili kamere koje mogu snimati u dubinu, *kinect* ili RGB-D kamere. Ovi algoritmi već su pronašli svoju primjenu kod svih vrsta autonomnih vozila, od automobila do letjelica, te čak i kod nekih robota namijenjenih za rad u kućanstvu.
SLAM algoritmi rade tako da uz pomoć senzora uzimaju podatke u obliku oblaka točaka. Iz tih oblaka točaka uzimaju se tzv. „poligoni“ koji predstavljaju višekutne oblike unutar tog oblaka točaka [6]. Nakon toga, oblaci točaka se uzimaju na idućoj lokaciji te se i iz njih uzimaju „poligoni“ i tako dok se ne prođe određeni put koji treba mapirati. Za to vrijeme, poligoni se uspoređuju i uz pomoć njih se spajaju oblaci točaka u kojima se poligoni preklapaju te se na taj način gradi karta prostora. Nakon toga rade se brojne optimizacije kako bi karta što vjerodostojnije predstavila prostor koji je mapiran. Pojednostavljeno, ovi algoritmi rade na principu detekcije značajki lokalnog prostora u okolini trenutnog položaja mobilnog robota te uspoređivanja tih značajki s prethodno detektiranim značajkama prostora u kojemu je mobilni robot već bio. Nove značajke koje nisu sparene s prethodno detektiranim značajkama, dodaju se u proces i koriste se pri novim detekcijama.

Što se tiče samog *ROS*-a, on nudi mnoge pakete koji omogućavaju mapiranje, lokalizaciju i planiranje puta, odnosno navigaciju. Ovi paketi također nude i mnoge *SLAM* algoritme koji se koriste pri samom mapiranju i lokalizaciji. *SLAM* algoritam korišten u *ROS*-u je *gmapping*, dok se izgradnju karte koriste teme *odom*, *scan* i *tf*. Sama karta prikazana je uz pomoć shematskog prikaza, dok konfiguracijska datoteka karte sadrži razne podatke vezane uz samu kartu. Za lokalizaciju robota, *ROS* koristi adaptivnu *Monte-Carlo* lokalizaciju (engl. *Adaptive Monte-Carlo Localization*), odnosno *AMCL*. Sam *AMCL* je zamišljen za rad s laserom kao senzorom, ali moguće je koristiti i RGB-D kameru kao senzor.

2.9. Način rada ROS-a

Način rada *ROS*-a najlakše je objasniti uz pomoć primjera. Ranije je objašnjeno da se pri radu u *ROS*-u koriste čvorovi, teme, poruke itd., ali kako to izgleda kada se prikaže na pravom primjeru?

Primjer koji će se ovdje promotriti je primjer pokretanja simulacije *TurtleSim* i upravljanja njome uz pomoć tipkovnice [7].

Prije pokretanja čvorova simulacije i upravljanja, potrebno je pokrenuti sam *master* čvor:

```
emil@emil-ASUS:~$ roscore
```

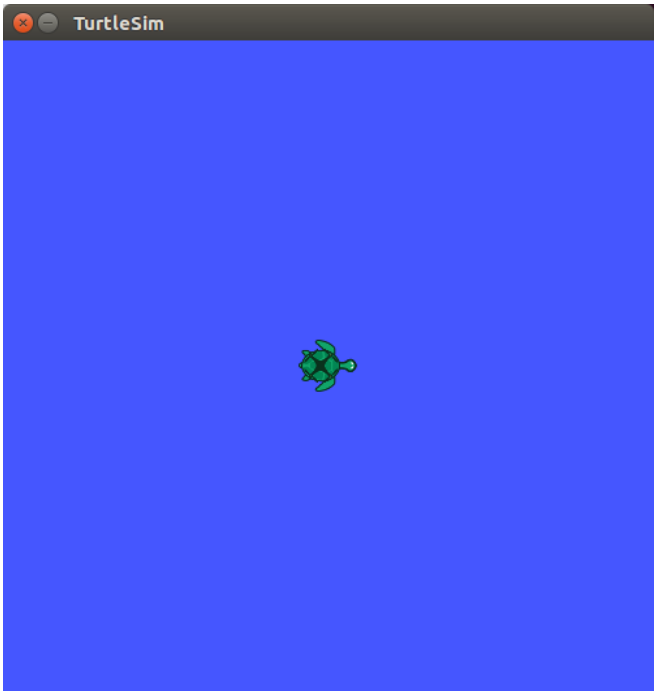
Pr. 2.1. Pokretanje *master* čvora

Nakon što je *master* čvor pokrenut, mogu se pokrenuti ostali čvorovi uz pomoć naredbe *rosrun*. U ovom primjeru, u novom prozoru konzole, prvo je potrebno pokrenuti *turtlesim_node* čvor iz paketa *turtlesim* kako bi se započela sama simulacija.

```
emil@emil-ASUS:~$ rosrun turtlesim turtlesim_node
[ INFO] [1568450234.078892680]: Starting turtlesim with node name /turtlesim
[ INFO] [1568450234.083199121]: Spawning turtle [turtle1] at x=[5,544445], y=[5,544445], theta=[0,000000]
```

Pr. 2.2. Pokretanje *turtlesim_node* čvora

Nakon pokretanja *turtlesim_node* čvora, otvara se prozor *TurtleSim* simulacije koji je moguće vidjeti na slici 2.5.

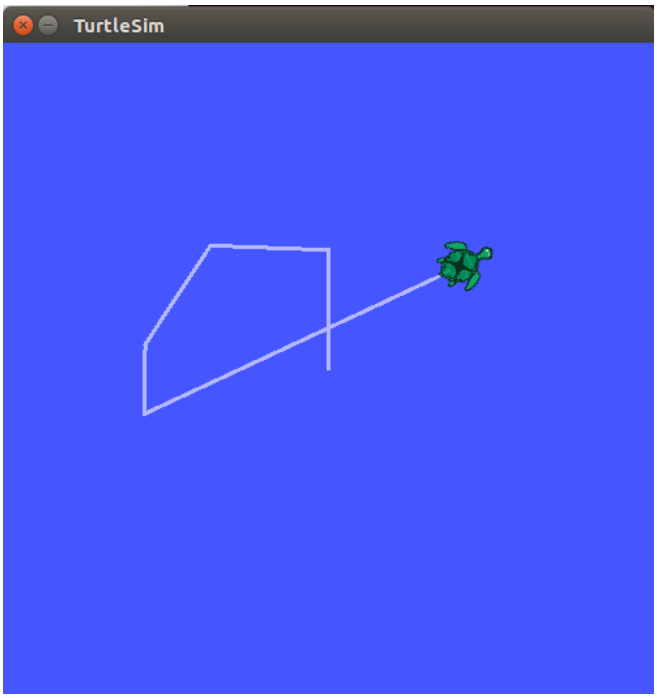


Slika 2.5. Prozor *TurtleSim* simulacije

Nakon što je *TurtleSim* simulacija pokrenuta, potrebno je omogućiti kretanje same kornjače. Kako bi se to omogućilo, potrebno je, također u zasebnom prozoru konzole, pokrenuti čvor *turtle_teleop_key* iz paketa *turtlesim* čijim će se pokretanjem omogućiti pomicanje kornjače u simulaciji uz pomoć tipkovnice. Rezultat pomicanja kornjače tipkovnicom moguće je vidjeti na slici 2.9.2.

```
emil@emil-ASUS:~$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

Pr. 2.3. Pokretanje *turtle_teleop_key* čvora za upravljanje tipkovnicom



Slika 2.6. Prozor *TurtleSim* simulacije nakon upravljanja kornjačom tipkovnicom

Korištenjem prethodnih naredbi njihovim upisivanjem u konzolu pokrenuti su *master* čvor te čvorovi *turtlesim_node* i *turtle_teleop_key*. Sve čvorove koji su korišteni moguće je vidjeti upisivanjem naredbe *rosnode list* u novu konzolu za vrijeme njihova izvođenja. Na primjeru 2.4., uz *teleop_turtle* i *turtlesim* čvorove, moguće je vidjeti i *rosout* čvor koji se pokreće pokretanjem *roscore* naredbe, a služi za zapisivanje poruka koje dolaze *rosout* temom.

```
emil@emil-ASUS:~$ rosnode list
/rosout
/teleop_turtle
/turtlesim
```

Pr. 2.4. Svi čvorovi ispisani korištenjem naredbe *rosnode list*

Poveznice između ovih čvorova predstavljaju teme, koje objavljuju i na koje se pretplaćuju sami čvorovi. Sve teme korištene u ovom primjeru moguće je vidjeti uz pomoć naredbe *rostopic list*, također za vrijeme izvođenja čvorova.

```
emil@emil-ASUS:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

Pr. 2.5. Sve teme ispisane korištenjem naredbe *rostopic list*

Za svaki čvor i za svaku temu moguće je vidjeti određene informacije o njima. Za čvorove je moguće vidjeti koje teme objavljuju te na koje teme i usluge su pretplaćeni, dok je za teme moguće vidjeti koji čvorovi ih objavljuju i koji čvorovi su pretplaćeni na njih te kojeg tipa su sami čvorovi. U primjeru 2.6. moguće je vidjeti informacije teme */turtle1/cmd_vel* pokretanjem naredbe *rostopic info*.

```
emil-ASUS:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist
Publishers:
 * /teleop_turtle (http://emil-ASUS:46241/)
Subscribers:
 * /turtlesim (http://emil-ASUS:37805/)
```

Pr. 2.6. Informacije teme */turtle1/cmd_vel* pokretanjem naredbe *rostopic info*

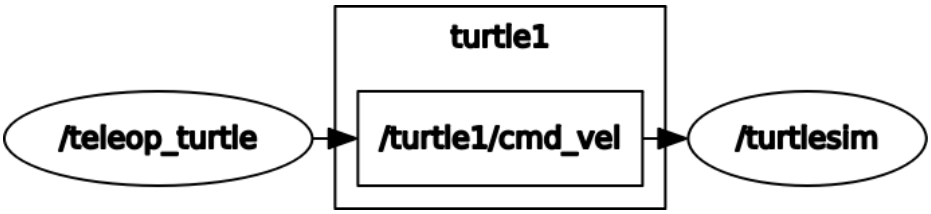
Ispisivanjem svih čvorova i teme te njihovih informacija može se dobiti uvid u način na koji sam *ROS* radi. No, kako bi to bilo još jasnije, za vrijeme izvođenja čvorova potrebno je u novoj konzoli upisati naredbu *rqt_graph* kako bi se dobio graf u kojem su prikazani svi čvorovi i sve

teme, te njihova međusobna ovisnost. Alat *rqt_graph* pokazao se veoma korisnim pri radu i rješavanju problema u *ROS*-u.

```
emil@emil-ASUS:~$ rqt_graph
```

Pr. 2.7. Pokretanje naredbe *rqt_graph*

Rezultate pokretanja naredbe *rqt_graph*, a samim time i odnose ranije ispisanih čvorova i tema, moguće je vidjeti na slici 2.7.



Slika 2.7. Rezultat pokretanja naredbe *rqt_graph*

Ovaj kratki primjer dobar je pokazatelj načina na koji sam *ROS* radi. Obuhvaća sve najvažnije dijelove *ROS*-a, iako je veoma jednostavan. Svakom projektu koji se rješava u *ROS*-u, pristupa se na ovaj način koji predstavlja jezgru samoga *ROS*-a.

3. MOBILNI ROBOT

Kako je odlučeno da će softver koji će se koristiti pri izradi diplomskog rada biti *ROS*, valja odlučiti i koje komponente hardvera će se koristiti kao bi se isti realizirao. Tema ovoga rada je navigacija mobilnog robota uz pomoć RGB-D kamere, što znači da je za njeno rješenje potrebno imati mobilni robot i RGB-D kameru. Za mobilni robot korišten je *3WD omni-wheel* mobilni robot, dok je za senzor, odnosno RGB-D kameru, korištena *Orbbec Astra S* kamera.

3.1. 3WD omni-wheel mobilni robot

Službeni naziv ovog mobilnog robota je *3WD 100mm Omni-Directional Triangle Mobile Robot*, a iz samog naziva može se vidjeti kakav je to uopće robot. Naime, radi se o mobilnom robotu trokutastog oblika s tri kotača koji se može pomicati u svim smjerovima. Pomicanje ovoga robota izvodi se promjenom brzine i orijentacije svakoga kotača bez da se promijeni orijentacija samog robota. Ovo je moguće zato što su sami kotači napravljeni tako da se mogu kretati u dva smjera. Kotači se mogu kretati kao normalni kotači, odnosno kotrljati, ali mogu se kretati i bočno zahvaljujući kotačima koji se nalaze na opsegu kotača. Ova tri kotača od ovog mobilnog robota čine robot koji ima sva 3 upravljiva stupnja slobode kretanja, za razliku od ostalih mobilnih robota koji imaju 2 od 3 upravljiva stupnja slobode kretanja, odnosno mogu ići naprijed/nazad i mijenjati rotaciju. Ovakav način rada robota, s 3 upravljiva stupnja slobode kretanja, čine mobilni robot bržim, pokretnijim i učinkovitijim.

Što se tiče samog mobilnog robota, on dolazi s *Arduino 328* mikroupravljačem baziranom na *Arduinu 168* [8]. Sam mikroupravljač je moguće programirati, a dolazi s 14 digitalnih ulazno/izlaznih pinova, 8 analognih pinova, USB-om itd. Ovaj mikroupravljač lako je koristiti, samo ga je potrebno kablom spojiti s računalom ili s izvorom struje ili ga jednostavno spojiti na bateriju, i on je spreman za rad.

Uz mikroupravljač, mobilni robot dolazi i s *Arduino IO Expansion* elementom, koji nudi proširenje ulazno/izlaznih jedinica koje olakšavaju spajanje raznih uređaja kao što su senzori, motori itd.

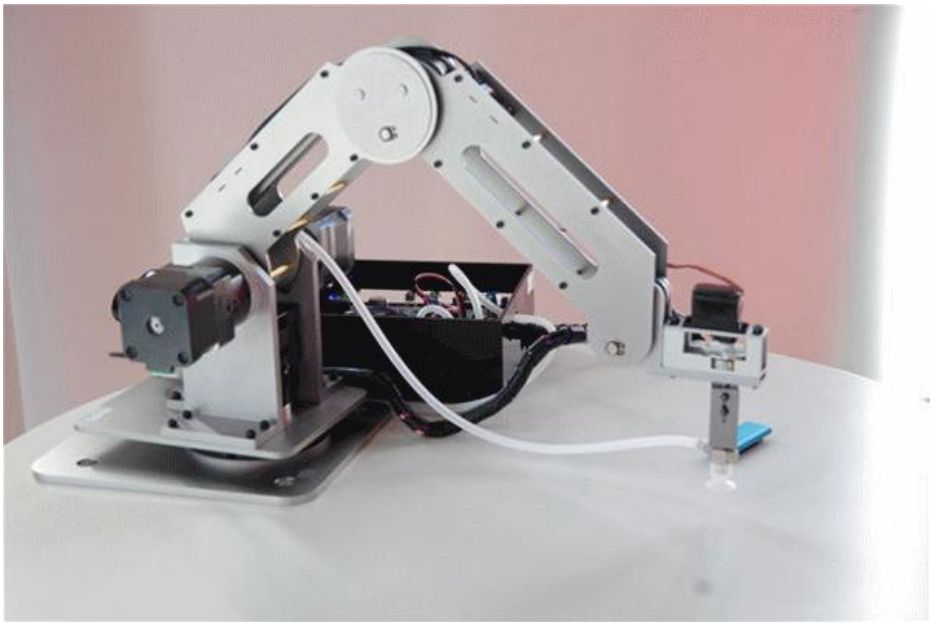
Što se tiče tehničkih podataka ovog mobilnog robota, on je dimenzija 330mm x 190mm x 108mm, a brzina kojom se kreće iznosi 0.6 m/s . Izrađen je od legure aluminija, teži 3.5 kg, a na sebi može ponijeti dodatnih 15 kg tereta.

Slika 3.11. *3WD omni-wheel* mobilni robot

Zajedno s ovim robotom, pričvršćena na njega, dolazi i robotska ruka. Ova robotska ruka pričvršćena je na gornji dio robota koji je i predviđen za takve stvari. Sama robotska ruka koja se nalazi pričvršćena na mobilnom robotu je raniji model *Dobot Magician* robotske ruke.

Dobot Magician robotska ruka je multifunkcionalna stolna robotska ruka namijenjena učenju, odnosno praksi rada na robotskim rukama. Neke od mogućnosti i namjena ove robotske ruke su 3D printanje, graviranje laserom, pisanje, crtanje itd. Uz to, ova robotska ruka podržava i razvoj u 13 sučelja i u preko 20 programskih jezika.

Što se tiče tehničkih podataka *Dobot Magician* robotske ruke, to je robotska ruka s 4 osi s maksimalnim dosegom od 320 mm [9]. Težina tereta koju može prenijeti je 500 g, dok se za komunikaciju s njom može koristiti USB kabel, *WiFi* ili *Bluetooth*. Napajanje za ruku je od 100 V do 240 V, a njena maksimalna potrošnja je 60 W. Robotska ruka je upravljana integriranim Dobot upravljačem. Što se tiče fizičkih obilježja, neto težina ove robotske ruke iznosi 3.4 kg, dok bruto težina iznosti 7.2 kg. Dimenzija baze robotske ruke je 158 mm x 158 mm, a izrađena je od legure aluminija i ABS inženjerske plastike. Softveri koji omogućuju korištenje robotske ruke su *DobotStudio, Repetier Host, GrblController3.6* i *DobotBlockly,* dok su kao SDK (*Software Develop Kit*) korišteni *Communication Protocol* i *Dobot Program Library*.



Slika 3.2. *Dobot V1* robotska ruka

3.2. Orbbec Astra S kamera

Orbbec Astra S je RGB-D kamera, odnosno 3D kamera opće namjene koja se može koristiti za mnoge stvari, a predstavlja alternativu *Kinect* kamerama koje imaju veću cijenu. Ova kamera najčešće se koristi u robotici, pri prepoznavanju gesta lica, 3D skeniranju, izgradnji interaktivnih sustava itd.

Postoji nekoliko tipova *Orbbec Astra* kamera. Ove kamere razlikuju se u određenim tehničkim podacima te razlikujemo *Astra, Astra S* i *Astra Pro* modele ove kamere. Pri rješavanju ovog diplomskog rada korišten je *Astra S* model kamere.

Što se tiče tehničkih podataka ove kamere, njezin domet snimanja je od 0.4 m do 2 m. Slike dobivene ovom kamerom su rezolucije 640x480, dok se video može dobiti u 30 fps-a [10]. Kako se ovdje radi o RBG-D kameri, i slike i video dobiveni kamerom su u boji, a dubinske slike i video su također prikazani u 640x480 rezoluciji, odnosno u 30 fps-a. Dimenzije ove kamere su 165mm x 30mm x 40mm, temperatura rada kamere je od 0 °C do 40 °C, a način spajanja kamere na napajanje je putem USB 2.0 konektora [11].

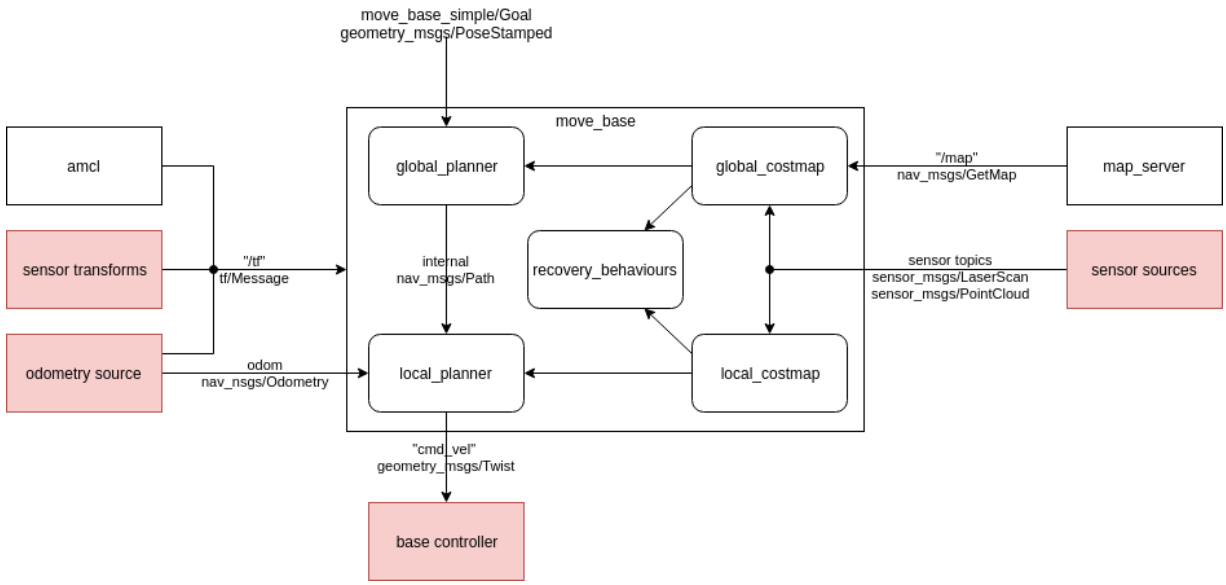
Slika 3.3. *Orbbec Astra S* RGB-D kamera

4. NAVIGACIJA MOBILNOG ROBOTA POMOĆU RGB-D KAMERE

Temeljni problem koji je bilo potrebno riješiti je problem navigacije mobilnog robota uz pomoć RGB-D kamere. U prethodnim poglavljima opisan je sam problem i svi aspekti potrebni za rješavanje tog problema. U ovom radu problem navigacije mobilnog robota riješen je primjenom navigacijskog stoga (engl. *navigation stack*). Navigacijski stog dolazi s već postojećom strukturom, ali i s određenim algoritmima potrebnim za rješavanje problema navigacije uz pomoć podataka prikupljenih sa senzora i sa samog mobilnog robota. Zbog svega toga, kao središnji dio rješavanja problema navigacije mobilnog robota RGB-D kamerom izabran je navigacijski stog.

4.1. Navigacijski stog

Navigacijski stog predstavlja strukturu koja iz određenih informacija, kao što su odometrija robota i očitanja sa senzora, generira i šalje poruke određenog tipa samoj bazi robota koja na taj način zna u kojem smjeru i kojom brzinom se treba kretati. Struktura navigacijskog stoga prikazana je na slici 4.1.

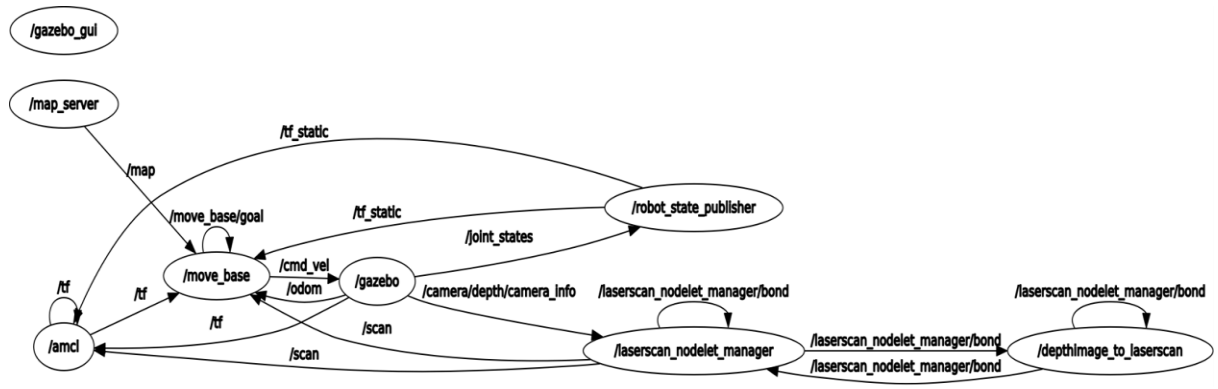


Slika 4.1. Struktura navigacijskog stoga [12]

Kako bi navigacijski stog radio ispravno, potrebno je zadovoljiti svaki dio njegove strukture prikazane na slici 4.1. Tako je moguće vidjeti da na toj slici pomicanje samog mobilnog robota, odnosno *move_base*, predstavlja središnji dio strukture, dok svi ostali dijelovi strukture, osim *base_controller* dijela, predstavljaju načine kojima se određuje sama pozicija mobilnog robota. Također, dijelovi stoga prikazani unutar crvenih pravokutnika predstavljaju dijelove koji su specifični za svaku platformu, dok su dijelovi unutar bijelih pravokutnika jednaki neovisno o platformi na koju se navigacijski stog implementira. No, uz sve ove dijelove, važan dio same navigacije je i izgradnja karte, odnosno tzv. mapiranje.

Svi dijelovi navigacijskog stoga će kroz iduća poglavlja biti zasebno pojašnjeni te će biti prikazano na koji način su implementirani pri rješavanju problema navigacije mobilnog robota pomoću RGB-D kamere.

Što se tiče samog rješenja problema navigacije mobilnog robota pomoću RBG-D kamere, ono je napisano u obliku *launch* datoteke kojom se pozivaju svi dijelovi navigacijskog stoga, odnosno čvorovi potrebni za rad navigacijskog stoga. Sve čvorove i teme kojima su povezani, odnosno izgled samog rješenja problema navigacije mobilnog robota pomoću RGB-D kamere moguće je vidjeti na slici 4.2. Rješenje je prikazano u obliku *RQT* grafa, pozvanog pokretanjem *rqt_graph* naredbe.



Slika 4.2. Prikaz *RQT* grafa procesa navigacije mobilnog robota RGB-D kamerom

4.2. Karta prostora

Da bi se mobilni robot uopće mogao lokalizirati, on se mora nalaziti na nekom njemu poznatom području. Zbog toga, jedan on najvažnijih elemenata je karta prostora u kojemu se mobilni robot nalazi. S obzirom na tu kartu, ali i ostale elemente navigacijskog stoga,

lokalizacija mobilnog robota postaje moguća. Prema tome, veoma je važno da je priložena karta točna, neovisno o tome na koji način je izrađena.

Zbog njene velike važnosti, kartu je bilo važno priložiti i prilikom rješavanja problema navigacije mobilnog robota RBG-D kamerom. U *launch* datoteku kojom je pokretan cijeli proces lokalizacije, bilo je potrebno dodati kod kojim se poziva čvor *map_server* uz pomoć kojega se procesu lokalizacije dodaje određena karta. Čvor *map_server* dio je istoimenog paketa, a određenu kartu pronalazi na osnovi njena imena i paketa u kojemu se nalazi.

```
<node name="map_server" pkg="map_server" type="map_server" args="$(find malac)/maps/willowgarage-refined.yaml"/>
```

Pr. 4.1. Pridruživanje određene karte uz pomoć *map_server* čvora

4.3. Izgradnja karte na temelju podataka sa senzora

Kako je sama karta veoma važan dio lokalizacije mobilnog robota, tako je isto toliko važna i izgradnja same karte. No, kao i za sve ostalo, i za izgradnju karte navigacijski stog ima rješenje. Ono dolazi u vidu paketa *gmapping* koji koristi *SLAM* algoritme pri izradi karte i lokaliziranju mobilnog robota u toj izrađenoj karti. Način rada *SLAM* algoritama opisan je u poglavlju 2.8., ali ovdje će biti detaljnije opisan sam *SLAM* algoritam kojega *gmapping* koristi te način na koji se *gmapping* mapiranje uvodi u problem izgradnje karte.

Algoritam korišten pri *gmapping* mapiranju je *Rao-Blackwellized Particle Filter* (*RBPF*) algoritam [13]. Ovaj algoritam svaku česticu (engl. *particle*) predstavlja kao zaseban položaj na karti prostora koji se mapira te se na taj način sastoji od velikog broja čestica koji je zatim potrebno smanjiti kako bi se dobila kompaktna karta prostora. Sama zamisao ovog algoritma je sljedeća: smanjiti broj varijabli koje se uzorkuju. Smanjenjem broja varijabli smanjuje se i dimenzija cijeloga sustava, odnosno sustav se pojednostavljuje. Prema tome, broj varijabli smanjuje se ovisno o modelu na koji se algoritam primjenjuje, na način da se za uzorkovanje uzimaju samo one varijable koje se ne mogu odrediti iz drugih varijabli, odnosno za uzorkovanje se ne uzimaju varijable koje je moguće odrediti iz drugih varijabli tog modela. Na primjeru modela koji sadrži varijable akceleracije *a*, brzine *v* i puta *s*, to bi značilo da je za uzorkovanje potrebno uzeti samo varijablu akceleracije *a*, jer se i brzina *v* i put *s*, zbog međusobnih ovisnosti između ove tri varijable, mogu odrediti iz varijable *a*. Zbog ovakvog načina rada, rezultati *RBPF*-a su znatno bolji u odnosu na običan čestični filtar, ali zahtijevaju veće računske troškove. U pogledu izrade karte, to bi značilo da ovaj algoritam ne uzima sve čestice koje predstavljaju

zaseban položaj na karti, nego nastoji smanjiti njihov broj pronalazeći zajedničke varijable za uzorkovanje unutar tih čestica. Na ovaj način, kao što je već rečeno, smanjuje se kompleksnost cijelog sustava i dobiva se kompaktna karta prostora.

RBPF u vidu *SLAM* algoritma dolazi kao dio *gmapping* paketa, odnosno kao dio *slam_gmapping* čvora, a da bi ovaj algoritam davao što bolje rezultate tijekom samog mapiranja, potrebno mu je proslijediti što točniju odometriju robota te što točnije podatke prikupljene sa senzora. Ovaj čvor također je potrebno pozvati u *launch* datoteci, ali također mu je potrebno zadati i određene parametre.

```
<node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
    <param name="base_frame" value="base_link"/>
  <param name="odom_frame" value="odom"/>
  <remap from="scan" to="scan"/>
  …
</node>
```

Pr. 4.2. Pozivanje *slam_gmapping* čvora s određenim parametrima

Iz primjera 4.2. moguće je vidjeti da je za rad *slam_gmapping* čvora potrebno postaviti određene parametre. Najvažniji od tih parametara su *base_frame*, *odom_frame* i *scan* parametri. Parametru *base_frame* potrebno je pridružiti parametar *base_link* koji predstavlja bazu samog mobilnog robota i koji je definiran u *URDF* datoteci modela mobilnog robota.

Parametru *odom_frame* potrebno je pridružiti odometriju samog mobilnog robota koja se izračunava preko enkodera kotača mobilnog robota i objavljuje se kao *odom* tema.

Parametru *scan* potrebno je pridružiti temu koju šalje sam senzor, u ovom slučaju RGB-D kamera. Ovdje je važno napomenuti kako RGB-D kamera ne šalje sama podatke tipa *LaserScan* koji su potrebni temi *scan*, nego se oni konvertiraju iz dubinske slike RGB-D kamere uz pomoć *depthimage_to_laserscan nodelet* čvora iz istoimenog paketa.

```
<node pkg="nodelet" type="nodelet" name="depthimage_to_laserscan"/>
```

Pr. 4.3. Pozivanje *depthimage_to_laserscan nodelet* čvora

Uz ove parametre, čvoru *slam_gmapping* moguće je postaviti velik broj ostalih parametara kojima se mapiranje može poboljšati, odnosno prilagoditi prema potrebi bilo kojega slučaja.

Nakon što su svi parametri podešeni u *launch* datoteci, tu *launch* datoteku potrebno je pozvati naredbom *roslaunch*, tako da se nakon naredbe pozovu imena paketa i datoteke koja se nalazi u tom paketu.

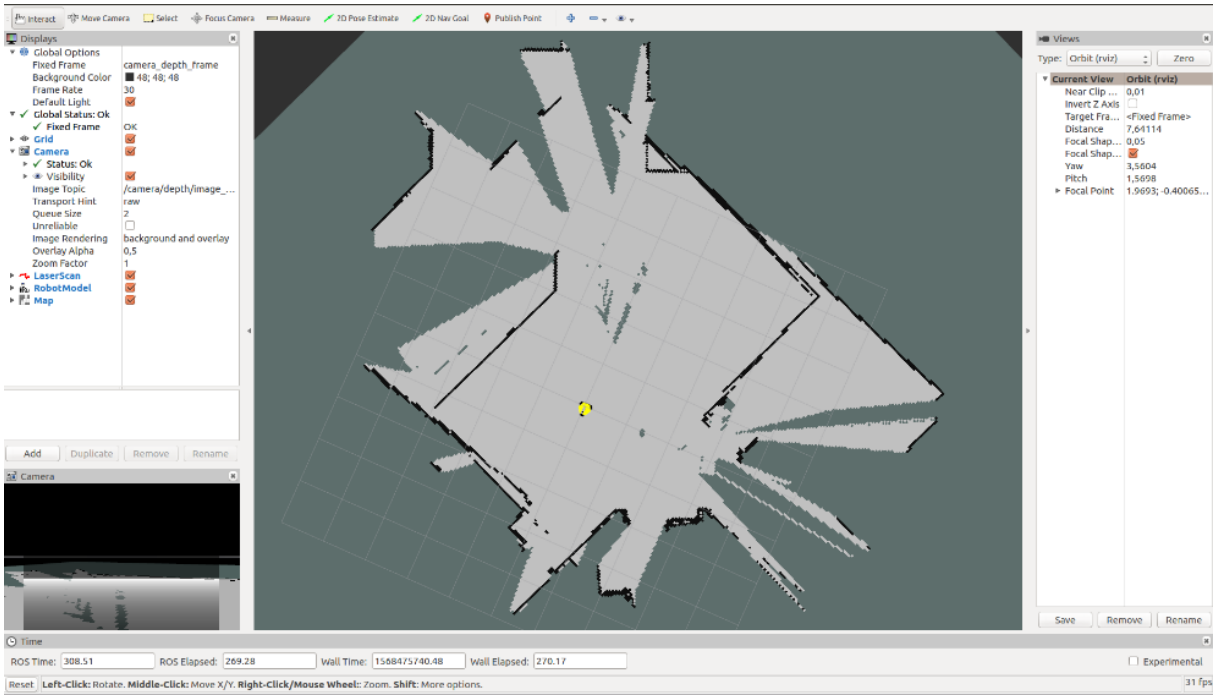
```
emil@emil-ASUS:~$ roslaunch malac malac_wg_gmapping.launch
```

Pr. 4.4. Pokretanje *launch* datoteke za mapiranje

Nakon pokretanja *launch* datoteke, potrebno je pozvati i alat *rviz* (Pr. 4.5.) u kojem će se vizualizirati robot i sam proces izgradnje karte *SLAM* algoritmom (slika 4.3.).

```
emil@emil-ASUS:~$ rviz
```

Pr.4.5. Pozivanja alata *rviz*



Slika 4.3. Proces izgradnje karte *SLAM* algoritmom prikazan u alatu *rviz*

4.4. Podaci sa senzora i odometrija

Kao i kod izrade karte, podaci sa senzora i odometrija su veoma važni i kod same lokalizacije. Uz pomoć njih, mobilni robot može, više ili manje uspješno, procijeniti gdje se nalazi na karti, ovisno o tome koliko su ovi podaci precizni.

Podaci sa senzora, u ovome slučaju RGB-D kamere, koriste se kao „oči“ mobilnog robota. Uspoređivanje podatka koji dolaze sa senzora s kartom kojom se mobilni robot kreće uvelike doprinosi procesu lokalizacije. Kako bi to bilo moguće, kao i kod izrade karte, podaci koji se šalju moraju biti *LaserScan* tipa. Zbog toga se i ovdje koristi *depthimage_to_laserscan* paket kako bi se dobio ovaj tip podataka iz dubinske slike RGB-D kamere i kako bi se podaci mogli slati putem teme *scan*.

```
emil-ASUS:~$ rostopic info /scan
Type: sensor_msgs/LaserScan
Publishers:
 * /laserscan_nodelet_manager (http://emil-ASUS:46045/)
Subscribers:
 * /move_base (http://emil-ASUS:42699/)
 * /amcl (http://emil-ASUS:45959/)
```

Pr. 4.6. Informacije o *scan* temi

Odometrija mobilnog robota izračunava se uz pomoć enkodera kotača. Putem enkodera, koji prikupljaju podatke o pokretanju samih kotača, može se dobiti pozicija mobilnog robota u odnosu na njegovu početnu poziciju. Kao i podaci sa senzora, odometrija također može biti manje ili više točna, ovisno o tome o kakvom mobilnom robotu je riječ i koliko su dobri enkoderi kotača. Što se tiče odometrije *3WD omni-wheel* mobilnog robota, ona i nije toliko točna, baš zbog toga što se radi o mobilnom robotu s 3 kotača koji se kreće u svim smjerovima te zapravo „klizi“ podlogom kada se kotači okreću, umjesto da se vozi. Zbog toga se lokalizacija ovog mobilnog robota oslanja se najviše na podatke pristigle sa senzora, dok je odometrija tu samo kako bi cijeli proces lokalizacije i izrade karte bio moguć. Sam proces lokalizacije koristi *odom* temu kako bi se prenijeli podaci o odometriji mobilnog robota do *move_base* dijela strukture navigacijskog stoga.

```
emil-ASUS:~$ rostopic info /odom
Type: nav_msgs/Odometry
Publishers:
 * /gazebo (http://emil-ASUS:45945/)
Subscribers:
 * /move_base (http://emil-ASUS:42699/)
```

Pr. 4.7. Informacije o *odom* temi

4.5. Transformacijsko stablo

Kako bi navigacijski stog radio, potrebno je na neki način opisati odnos između baze mobilnog robota i samog senzora kojime se prikupljaju podaci važni za sam proces lokalizacije. Ovaj problem se rješava uz pomoć transformacija. Same transformacije opisuju odnose između koordinatnih sustava svih dijelova mobilnog robota, od same baze robota, preko svih zglobova, pa do senzora. Mobilni robot i odnosi između svih njegovih zglobova i senzora opisani su u *URDF* datoteci modela robota, pomoću koje se može doći do odnosa baze robota i senzora, pa samim time i do transformacije između ta dva dijela. Sam skup transformacija naziva se transformacijsko stablo, a poruke između dijelova transformacijskog stabla prenose se putem *tf* teme.

```
emil-ASUS:~$ rostopic info /tf
Type: tf2_msgs/TFMessage
Publishers:
 * /robot_state_publisher (http://emil-ASUS:44899/)
 * /amcl (http://emil-ASUS:45959/)
 * /gazebo (http://emil-ASUS:45945/)
Subscribers:
 * /move_base (http://emil-ASUS:42699/)
 * /amcl (http://emil-ASUS:45959/)
```

Pr. 4.8. Informacije o *tf* temi

Iz primjera 4.8. moguće je vidjeti da su poruke koje se šalju *tf* temom *TFMessage* tipa, odnosno da se radi o porukama koje se isključivo šalju unutar transformacijskog stabla. Također, moguće je vidjeti i da je jedan od objavljivača ove teme *robot_state_publisher* čvor. Ovaj čvor

predstavlja središnji dio procesa podešavanja transformacija, jer je on taj koji zapravo uzima podatke iz *URDF* datoteke modela robota i iz tih podataka dolazi do samih transformacija, koje se zatim koriste pri lokalizaciji.

Ovdje je moguće vidjeti i koliko je zapravo važna sama *URDF* datoteka modela mobilnog robota. Bez ove datoteke, transformacije između zglobova robota, a ni sam prikaz robota u simulaciji i alatu *rviz*, ne bi bili mogući. Unutar ove datoteke, definira se oblik robota i svih njegovih komponenti te njihov međusobni odnos. Uz sam oblik, definira se i ponašanje tih komponenti, u vidu fizike tih komponenti i mogućnosti kolizije s ostalim predmetima. Što se tiče samih zglobova, postavlja se tip zgloba, a ovisno o tipu, postavljaju se i ostali parametri. Dio *URDF* datoteke mobilnog robota moguće je vidjeti na primjeru 4.9.

```
<robot name="malac">
  …
  <link name="right_wheel">
      <visual>
        <geometry>
          <cylinder length="0.04" radius="0.05"/>
        </geometry>
        <material name="black">
          <color rgba="0 0 0 1"/>
        </material>
      </visual>
      <collision>
        <geometry>
          <cylinder length="0.04" radius="0.05"/>
        </geometry>
      </collision>
      <inertial>
        <mass value="0.1667"/>
        <inertia ixx="0.000126" iyy="0.000126" izz="0.0002" ixy="0" ixz="0" iyz="0"/>
      </inertial>
  </link>
  <joint name="right_wheel_joint" type="continuous">
      <axis xyz="0 0 1"/>
      <parent link="base_link"/>
      <child link="right_wheel"/>
      <origin rpy="-1.5708 0 0.523" xyz="0.08 -0.1275 -0.015"/>
  </joint>
  …
</robot>
```

Pr. 4.9. Dio *URDF* datoteke mobilnog robota

Na primjeru 4.9. moguće je vidjeti dijelove *URDF* datoteke mobilnog robota kojima se definira desni kotač mobilnog robota te zglob tog kotača i baze mobilnog robota. Za oblik kotača korišten je valjak određenih dimenzija, materijala i boje. Kolizija tog kotača također se očituje u

obliku valjka istih dimenzija, a pridružena mu je i pripadna inercija, odnosno fizika samog valjka. Nadalje, zglob desnog kotača i baze mobilnog robota definiran je kao *continuous*, što znači da se radi o zglobu koji se okreće oko određene osi, u ovom slučaju oko osi *z*. Također, definirani su i parametri „roditelj“ i „dijete“, koji u ovom slučaju označavaju da se desni kotač, kao „dijete“, povezuje na bazu mobilnog robota, odnosno na „roditeljsku“ komponentu.

4.6. Adaptivna Monte Carlo lokalizacija (AMCL)

Adaptivna *Monte Carlo* lokalizacija (engl. *Adaptive Monte Carlo Localization*), odnosno *AMCL*, predstavlja probabilistički sustav za lokalizaciju koji koristi čestične filtre kako bi se odredila pozicija robota na određenoj karti [14].

Ovaj algoritam pogodan je za rješavanje problema lokalizacije zato što su rezultat rada ovoga algoritma svi mogući ishodi, odnosno ovaj algoritam uzima u obzir sva moguća rješenja. No, kako bi se došlo do rješenja, potrebno je provesti određeni postupak. Za početak, potrebno je dobiti matematički model procesa nad kojim se algoritam provodi. Nakon toga, pronalaze se varijable neizvjesnih vrijednosti te se određuju funkcije gustoće kojima se opisuju učestalosti kojima slučajne varijable poprimaju svoje vrijednosti. U slučaju korelacije varijabli, kreira se matrica korelacije. Na kraju se varijablama, u svakoj iteraciji, dodjeljuju slučajne vrijednosti koje proizlaze iz funkcija gustoće uzimajući u obzir matricu korelacije te se izračunavaju izlazne vrijednosti, odnosno rezultati, a ovaj postupak ponavlja se N puta. Na kraju, sva moguća rješenja su zapisana, a nad njima je moguće provesti analizu te odrediti najpovoljnije rješenje, što se događa i kod lokalizacije mobilnog robota, gdje se pronalazi rješenje koje najviše odgovara stvarnom položaju mobilnog robota, uzimajući u obzir odometriju mobilnog robota te još važnije, očitanja sa senzora, odnosno RGB-D kamere.

Uz poznatu, tj. prethodno izrađenu kartu, *AMCL* algoritmu je za njegov rad još potrebno pružiti i podatke sa senzora *LaserScan* tipa te određene transformacije iz transformacijskog stabla.

AMCL algoritam dolazi u obliku paketa kao dio *ROS*-a, a kako bi se koristio kao dio navigacijskog stoga, u *launch* detoteku potrebno je uključiti *amcl_omni.launch* datoteku koja je dio *amcl* paketa. Prije pokretanja *amcl* paketa, odnosno njegove *launch* datoteke, potrebno je postaviti određene parametre unutar te datoteke. Na primjeru 4.10. moguće je vidjeti neke od parametara *amcl* algoritma koji se pozivaju pokretanjem *amcl_omni.launch* datoteke. Ovdje se

mogu vidjeti parametri kojima se postavlja model odometrije mobilnog robota, maksimalan broj laserskih zraka senzora te minimalan i maksimalan broj čestica *amcl* algoritma.

```
<node pkg="amcl" type="amcl" name="amcl">
  <param name="odom_model_type" value="omni"/>
  <param name="laser_max_beams" value="30"/>
  <param name="min_particles" value="500"/>
  <param name="max_particles" value="5000"/>
  …
</node>
```

Pr. 4.10. Dio parametara *amcl_omni.launch* datoteke

```
<include file="$(find amcl)/examples/amcl_omni.launch"/>
```

Pr. 4.11. Uključivanje *amcl_omni.launch* datoteke u *launch* datoteku lokalizacije

Uključivanjem *amcl_omni.launch* datoteke u *launch* datoteku lokalizacije omogućuje se korištenje *AMCL* algoritma te se pokreće i *amcl* čvor koji objavljuje brojne teme. Isto tako, *amcl* čvor se i pretplaćuje na brojne teme. U primjeru 4.12. moguće je vidjeti da su neke od tema na koje se *amcl* čvor pretplaćuje teme *scan* i *tf*, čija je svrha u lokalizaciji ranije objašnjena.

```
emil@emil-ASUS:~$ rosnode info /amcl
--------------------------------------------------------------------------------
Node [/amcl]
Publications:
 * /amcl/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
 * /amcl/parameter_updates [dynamic_reconfigure/Config]
 * /amcl_pose [geometry_msgs/PoseWithCovarianceStamped]
 * /particlecloud [geometry_msgs/PoseArray]
 * /rosout [rosgraph_msgs/Log]
 * /tf [tf2_msgs/TFMessage]
Subscriptions:
 * /clock [rosgraph_msgs/Clock]
 * /initialpose [unknown type]
 * /scan [sensor_msgs/LaserScan]
 * /tf [tf2_msgs/TFMessage]
 * /tf_static [tf2_msgs/TFMessage]
```

Pr. 4.12. Informacije o *amcl* čvoru

4.7. Pokretanje mobilnog robota

Još u poglavlju 4.1. objašnjeno je kako je središnji dio navigacijskog stoga, odnosno same navigacije, zapravo pokretanje mobilnog robota. No, ono nikako ne bi bilo moguće bez svih prethodno opisanih komponenti sustava koje zapravo omogućuju pokretanje mobilnog robota. Tek kada su sve komponente navigacijskog stoga definirane, moguće je realizirati rješenje navigacije mobilnog robota. Komponente sustava predstavljaju čvorove koji šalju određene poruke putem određenih tema do *move_base* čvora kojim se onda mobilni robot pokreće.

Sam *move_base* čvor zapravo predstavlja cijeli proces navigacije mobilnog robota. Podaci sa senzora i odometrija do njega dolaze putem *scan*, odnosno *odom* teme, dok karta i određene transformacije do njega dolaze putem *map*, odnosno *tf* teme. Na osnovu svih podataka koji pristižu putem ovih tema, ali i na osnovu načina na koji je *move_base* čvor osmišljen i načina na koji on radi, moguće je provesti lokalizaciju i navigaciju samog mobilnog robota.

Rad *move_base* čvora temelji se na planiranju globalnog i lokalnog puta na osnovu globalne i lokalne karte. Da bi mobilni robot mogao doći od točke A do točke B na nekoj karti,

navigacijski stog mora isplanirati rutu kojom će mobilni robot ići. Pri planiranju te rute, navigacijski stog, uz *AMCL* algoritam za planiranje puta, koristi i globalnu kartu vrijednosti (engl. *global costmap*) kako bi dugoročno isplanirao rutu. No, uz globalnu kartu vrijednosti, navigacijski stog koristi i lokalnu kartu vrijednosti (engl. *local costmap*) kako bi kratkoročno planirao rutu. Dok se dugoročno planiranje temelji na samoj karti, kratkoročno se temelji na odometriji mobilnog robota i na podacima prikupljenim putem senzora.

No, prije planiranja rute, potrebno je postaviti globalne parametre navigacije, lokalnu parametre navigacije, vrijednosti zajedničkih parametara te bazni lokalni planer.

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[0.35, 0.15], [0.35, -0.15], [-0.35, -0.15], [-0.35, 0.15]]
robot_radius: 0.15
inflation_radius: 0.55
observation_sources: laser_scan_sensor point_cloud_sensor
laser_scan_sensor:
  sensor_frame: camera_link
  data_type: LaserScan
  topic: scan
  marking: true
  clearing: true
point_cloud_sensor:
  sensor_frame: camera_link
  data_type: PointCloud
  topic: camera/depth/points
  marking: true
  clearing: true
```

Pr. 4.13. Vrijednosti zajedničkih parametara

Primjer 4.13. prikazuje vrijednosti zajedničkih parametara (engl. *costmap common parameters*) potrebnih za planiranje rute mobilnog robota. Moguće je vidjeti da su postavljeni parametri koji opisuju sam mobilni robot, ali i parametri koji određuju s kojih tema i čvorova će se preuzimati i slati podaci.

```
global_costmap:
  global_frame: map
  robot_base_frame: base_link
  update_frequency: 5.0
  static_map: true
```

Pr. 4.14. Globalna parametri navigacije

Primjer 4.14. prikazuje globalne parametre navigacije (engl. *global costmap parameters*) potrebne za dugoročno planiranje rute. Moguće je vidjeti da je tema *map* korištena pri njenom planiranju te da je *base_link* tema postavljena kao bazni okvir robota. Na kraju, parametar *static_map* je postavljen na vrijednost *true* i time je omogućeno korištenje karte koja je ubačena uz pomoć *map_server* čvora.

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.05
```

Pr. 4.15. Lokalni parametri navigacije

Primjer 4.15. prikazuje lokalne parametre navigacije (engl. *local costmap parameteres*) iz kojih je moguće vidjeti kako je tema odom korištena pri kratkoročnom planiranju rute. Kao i kod globalnih parametara navigacije, *base_link* tema postavljena je kao bazni okvir robota, ali za razliku od globalnih parametara, ovdje je parametar *static_map* postavljen na vrijednost *false*, kako ovdje nije potrebno omogućiti korištenje prethodno ubačene karte. Na kraju, parametar *rolling_window* postavljen je na vrijednost *true*, što znači da će se lokalna karta vrijednosti gibati zajedno s mobilnim robotom, odnosno neće predstavljati potpunu okolinu mobilnog robota, nego samo određeni dio oko njega.

```
TrajectoryPlannerROS:
  max_vel_x: 0.45
  min_vel_x: 0.1
  max_vel_theta: 0.5
  min_in_place_vel_theta: 0.2

  acc_lim_theta: 3.2
  acc_lim_x: 2.5
  acc_lim_y: 2.5

  holonomic_robot: true
```

Pr. 4.16. Bazni lokalni planer

Primjer 4.16. prikazuje parametre baznog lokalnog planera. Uz razna ograničenja u vidu brzine i ubrzanja mobilnog robota, važan parametar ovdje je *holonomic_robot*. Ovaj parametar postavljen je na vrijednost *true*, što znači da se radi o robotu koji se može kretati u svim smjerovima.

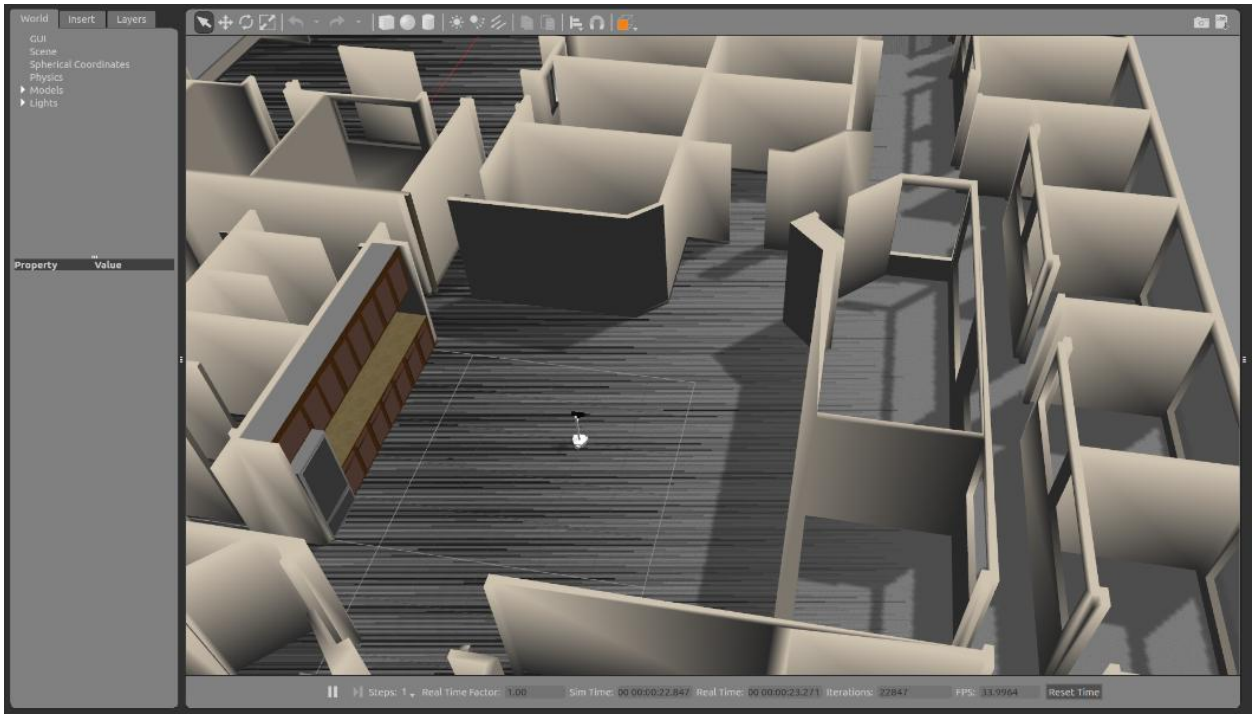
4.8. Navigacija mobilnog robota

Mobilni robot je spreman za izvršavanje zadatka. Karta prostora je ubačena, podaci dolaze se senzora, odnosno RGB-D kamere, u obliku pogodnom za lokalizaciju, a sam mobilni robot šalje podatke o odometriji. Sve transformacije unutar transformacijskog stabla su postavljene, a *AMCL* algoritam je spreman za provođenje procesa lokalizacije. Također, postavljeni su i parametri unutar lokalne i globalne karte vrijednosti, unutar karte vrijednosti zajedničkih parametara te parametri baznog lokalnog planera.

Jedino što preostaje je pokrenuti napisanu *launch* datoteku, te započeti proces navigacije mobilnog robota. Datoteka se pokreće naredbom *roslaunch*, nakon koje slijede ime paketa i ime datoteke koja se pokreće.

```
emil@emil-ASUS:~$ roslaunch malac malac_wg.launch
```

Pr. 4.17. Pokretanje *launch* datoteke za navigaciju mobilnog robota

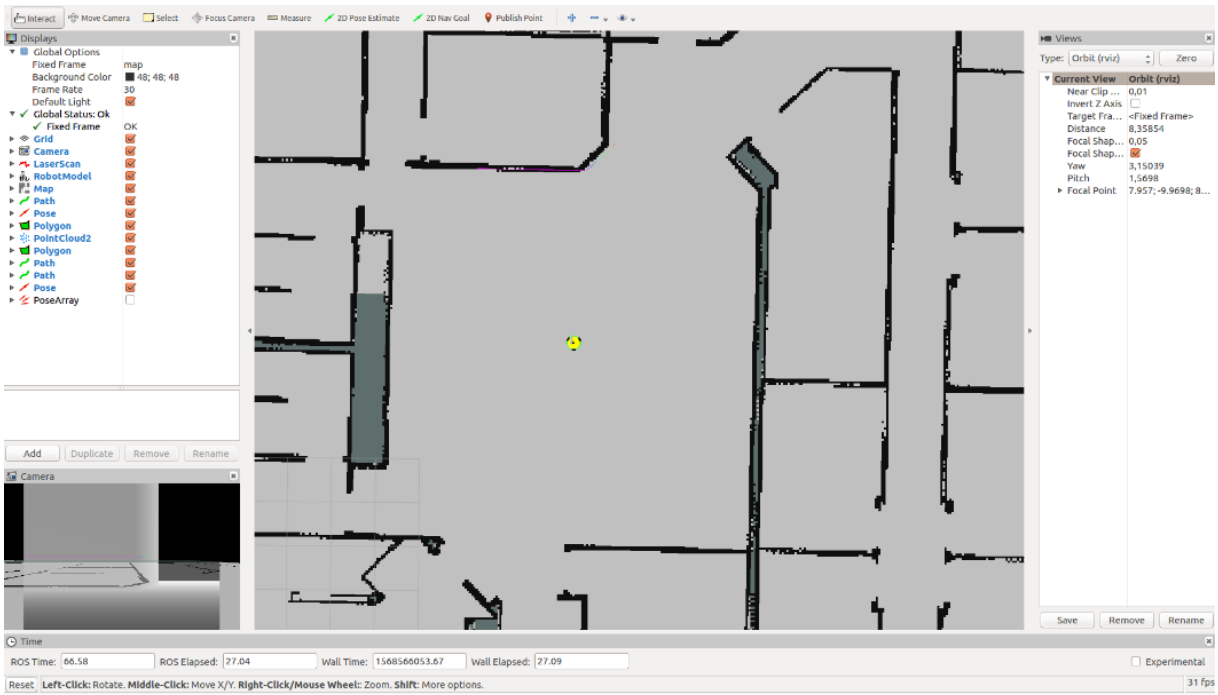


Slika 4.4. Izgled *Gazebo* simulacije i mobilnog robota u njoj

Kako bi se proces navigacije mogao nadgledati, potrebno je i pokrenuti alat *rviz* uz pomoću kojega će se vizualizirati sam mobilni robot te sam proces navigacije.

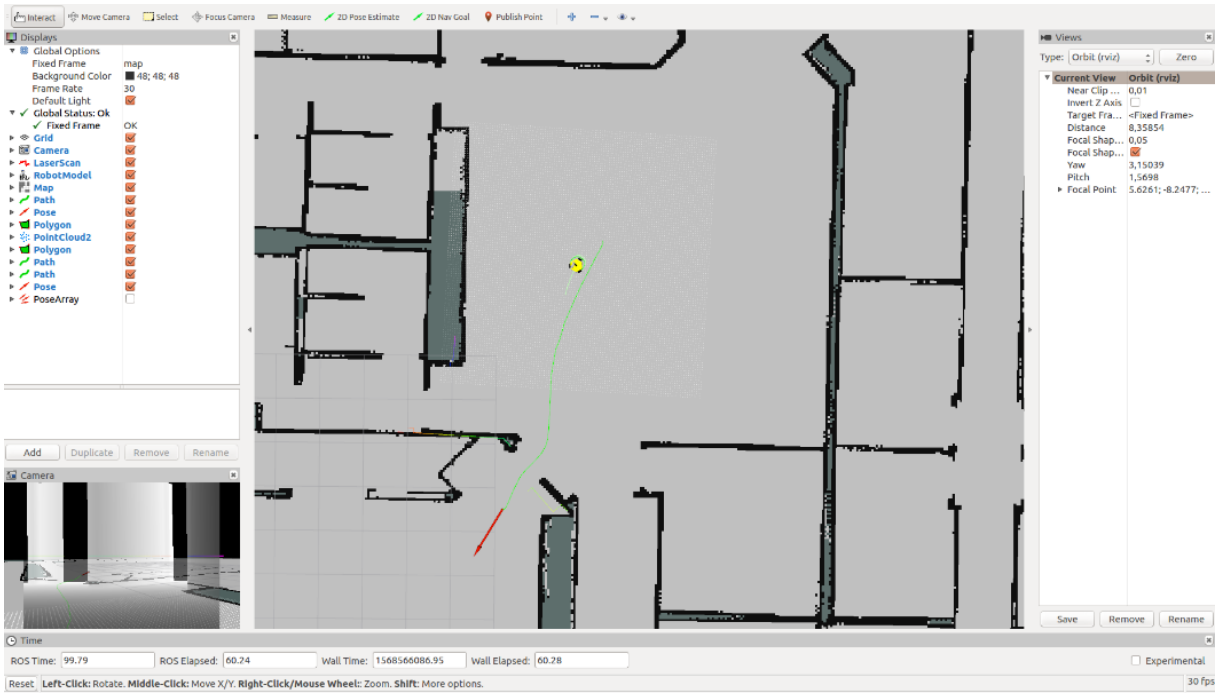
```
emil@emil-ASUS:~$ rviz
```

Pr. 4.18. Pokretanje alata *rviz*



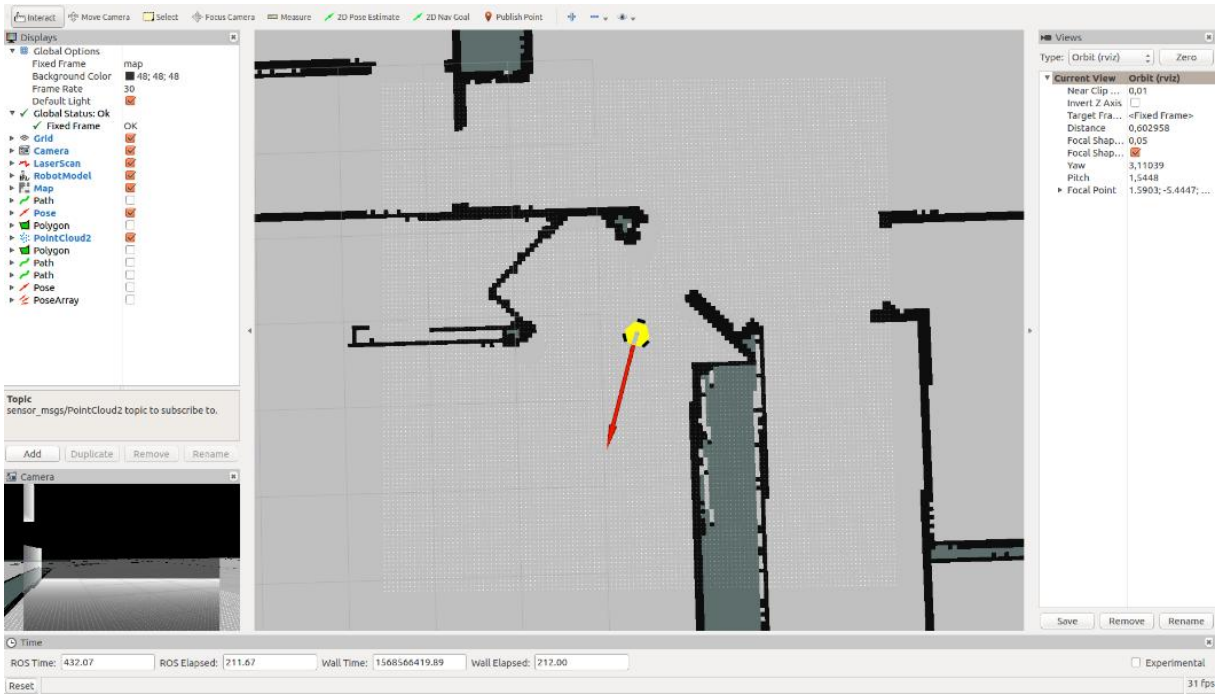
Slika 4.5. Prozor alata *rviz* s modelom robota

Na slici 4.5. moguće je vidjeti prozor alata *rviz* u kojemu je vizualiziran mobilni robot. Mobilni robot lokaliziran je s obzirom na poziciju u kojoj se on zapravo nalazi uz pomoć *2D Pose Estimate* naredbe alata *rviz*. Nakon što je ovako lokaliziran, mobilnom robotu je moguće zadati točku na karti, odnosno odredište do kojega treba doći. To je moguće uz pomoć *2D Nav Goal* naredbe alata *rviz*.



Slika 4.6. Mobilni robot navigira prostorom

Na slici 4.6. moguće je vidjeti kretanje robota kartom do točke određene naredbom *2D Nav Goal*. Crvena strelica, odnosno baza te strelice predstavlja tu točku, dok sama strelica predstavlja orijentaciju koju robot treba zauzeti kada stigne na odredište. Zelena linija predstavlja rutu kojom će se mobilni robot kretati do odredišta.



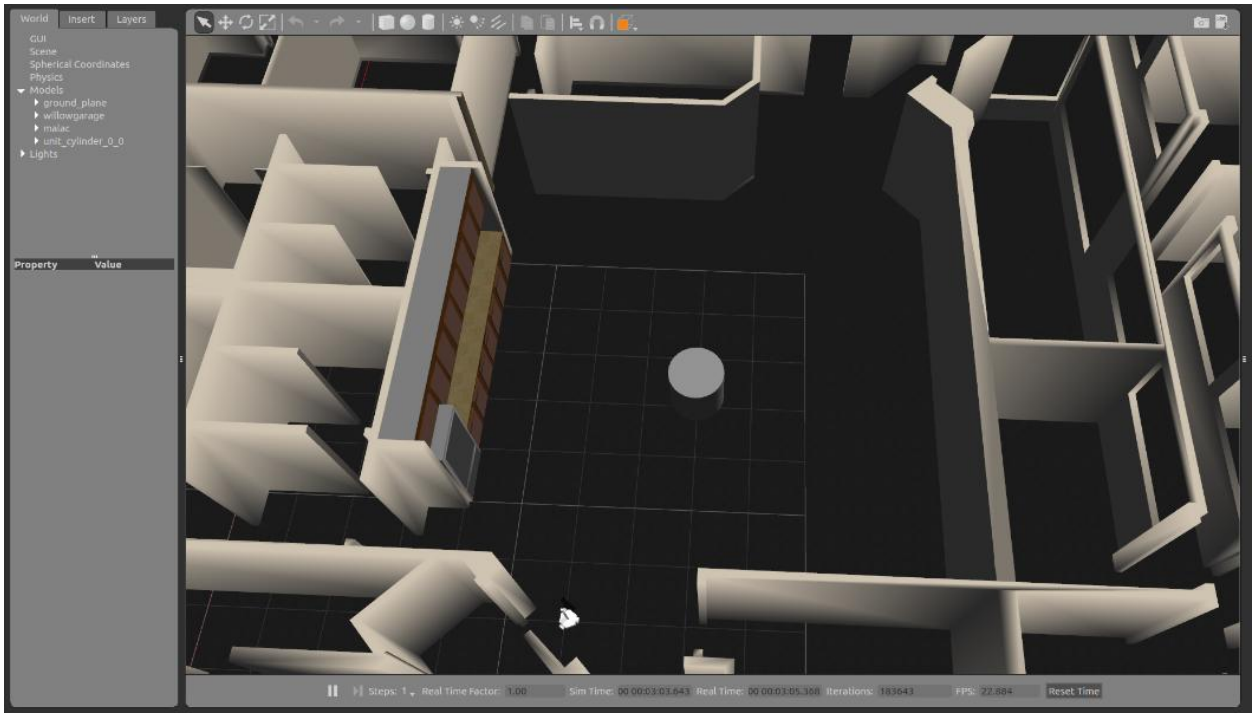
Slika 4.7. Vrijednosni oblak točaka

Vrijednosni oblak točaka (engl. *cost cloud*) sastoji se od skupa točaka koji signaliziraju mjesta gdje se sam mobilni robot može kretati. Na slici 4.7. moguće je vidjeti kako su bijele točke prisutne na prostoru sigurnom za kretanje mobilnog robota, dok izostaju u blizini zidova i ostalih prepreka, gdje kretanje mobilnog robota nije sigurno.

Slika 4.8. Oblak čestica

Na slici 4.8. moguće je vidjeti oblak čestica (engl. *particle cloud*) korištenih pri lokalizaciji te usmjerenje tih čestica.

Na slici 4.9. moguće je vidjeti ubačeni predmet na rutu mobilnog robota unutar *Gazebo* simulacije. Predmet je ubačen kako bi se demonstriralo kratkoročno i dugoročno planiranje rute mobilnog robota.



Slika 4.9. *Gazebo* simulacija s ubačenim predmetom ispred mobilnog robota

Slika 4.10. Kratkoročno i dugoročno planiranje rute

Razlika između kratkoročnog i dugoročnog planiranja rute objašnjena je u prijašnjim poglavljima. Na slici 4.10. moguće je vidjeti oba ova planiranja. Zelena crta označava dugoročan plan, dok kratka bijela crta odmah ispred robota označava kratkoročno planiranje. Na ovoj slici također je moguće vidjeti i očitanje sa senzora, odnosno RGB-D kamere, koje je uočilo prepreku ispred mobilnog robota. Ti podaci poslani su u navigacijski stog, gdje je zatim *AMCL* algoritam, u skladu s tim očitanjima, isplanirao dugoročan i kratkoročan plan rute.

5. ZAKLJUČAK

Problem navigacije mobilnog robota uz pomoć RGB-D kamere uspješno je riješen. Najvažnija stavka pri rješavanju ovog problema svakako je bio sam *ROS* koji je pružio svu potrebnu podršku za izvršavanje zadatka. Ta podrška došla je u vidu navigacijskog stoga koji sadrži sve što je potrebno kako bi se sam mobilni robot mogao kretati zadanom kartom uz zaobilaženje prepreka na toj karti. Uz to, omogućena je i izrada same karte bilo kojeg prostora u kojemu se mobilni robot nalazi.

Nakon što je sustav za navigaciju izrađen, njegov rad je testiran uz pomoć simulacije i modela mobilnog robota s pričvršćenom RGB-D kamerom. Samo testiranje uspješno je provedeno, a rezultati su pokazali kako su i sustav za navigaciju i sustav za izradu karte uspješno realizirani.

Nadalje, model mobilnog robota s RGB-D kamerom pričvršćenom na njega pokazao se veoma dobrim i uvelike je olakšao rad na samom problemu diplomskog rada. No, to se ne može reći i za stvarni mobilni robot. Prilikom implementacije rješenja diplomskog rada na stvarne sustave, došlo je do brojnih problema izazvanih lošijom opremom, pri tome misleći prvenstveno na sam model mobilnog robota za koji ne postoji skoro nikakva podrška, a i čija je izvedba jedna od lošijih među mobilnim robotima općenito. Ovi problemi nisu se uspjeli riješiti u vidu samog rada te je sam rad odrađen samo kao simulacija.

Na kraju, valja reći kako je sam rad moguće nadograditi, najviše u pogledu njegovog osposobljavanja na stvarnim sustavima. Također, zbog veoma loših enkodera kotača unutar samog *3WD omni-wheel* mobilnog robota, za daljnje rješavanje problema i njegovo osposobljavanje na stvarnim sustavima bilo bi poželjno koristiti drugačiji oblik mobilnog robota, što pri rješavanju ovog diplomskog rada nije bilo moguće.

LITERATURA

[1] *Robot Operating System, Wikipedia,* dostupno na*:* https://en.wikipedia.org/wiki/Robot_Operating_System, lipanj 2019.

[2] *Programmin Robots with ROS, Morgan Quigley, Brian Gerkey, William D. Smart, 2015.*

[3] *ROS 101: Introduction to the Robot Operating System, Robohub*, dostupno na: https://robohub.org/ros-101-intro-to-the-robot-operating-system/, lipanj 2019.

[4] *Giorgio Grisetti, Introduction to Navigation using ROS,* dostupno na: https://www.dis.uniroma1.it/~nardi/Didattica/CAI/matdid/robot-programming-ROS-introduction-to-navigation.pdf, lipanj 2019.

[5] *GeoSlam tehnologija, Geocentar*, dostupno na: https://geocentar.com/geoslam-tehnologija/, lipanj 2019.

[6] *Simultaneous localization and mapping, Wikipedia*, dostupno na: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping, lipanj 2019.

[7] *ROS Tutorials, ROS.org*, dostupno na: http://wiki.ros.org/ROS/Tutorials, rujan 2019.

[8] *(3WD 100mm Omni-Directional Triangle Mobile Robot Kit), Robotshop*, dostupno na: https://www.robotshop.com/en/3wd-100mm-omni-directional-triangle-mobile-robot.html, lipanj 2019.

[9] *Dobot Magician Specifications, Dobot*, dostupno na: https://www.dobot.cc/dobot-magician/specification.html, lipanj 2019.

[10] *Astra Stereo S, Orbbec*, dostupno na: https://orbbec3d.com/astrastereos/, lipanj 2019.

[11] *Orbbec Astra, ROS Components*, dostupno na: https://www.roscomponents.com/en/cameras/76-orbbec.html, lipanj 2019.

[12] *Setup and Configuration oft he Navigation Stack on a Robot, ROS.org*, dostupno na: http://wiki.ros.org/navigation/Tutorials/RobotSetup, rujan 2019.

[13] *Navigation, ROS.org*, dostupno na: http://wiki.ros.org/navigation, rujan 2019.

[14] *Monte Carlo metoda, Marko Posavec, 2009.*, dostupno na: http://www.rep.hr/vijesti/strucni-clanci/strucni-clanak-monte-carlo-metoda/211/

SAŽETAK

Ovaj diplomski rad bavi se rješavanjem problema navigacije mobilnog robota pomoću RGB-D kamere putem Robotskog operacijskog sustava (*ROS*-a). Objašnjen je način rada ROS-a te je opisan i sam mobilni robot. Za rješavanje praktičnog dijela korišten je navigacijski stog koji predstavlja dio ROS-a kojim se rješavaju problemi lokalizacije i navigacije mobilnog robota. Opisani su dijelovi navigacijskog stoga koji omogućuju uspješno rješavanje problema navigacije mobilnog robota pomoću RGB-D kamere.

Ključne riječi: mobilni, autonomnost, RGB-D, ROS, čvor, tema, navigacija, stog, mapiranje, lokalizacija

Mobile Robot Navigation Using RGB-D Camera

ABSTRACT

This thesis deals with the problem of navigating a mobile robot using an RGB-D camera and Robot Operating System (ROS). The ROS is explained and the mobile robot is described. The practical part is solved using the navigation stack. The navigation stack is a part of the ROS that solves localization and navigation problems of the mobile robot. The components of the navigation stack which were used to implement mobile robot navigation using an RGB-D camera are described.

Keywords: mobile, autonomy, RGB-D, ROS, node, topic, navigation, stack, mapping, localization

ŽIVOTOPIS

Emil Vartušek rođen je 1.8.1995. godine u Osijeku. Osnovnu školu završio je 2010. godine u Osnovnoj školi Ante Starčevića u Viljevu. Nakon toga upisuje opću gimnaziju u Srednjoj školi Donji Miholjac, te ju 2014. godine i završava. Iste godine upisuje preddiplomski studij na Elektrotehničkom fakultetu u Osijeku (kasnije preimenovanom u Fakultet elektrotehnike, računarstva i informacijskih tehnologija), smjer računarstvo, koji završava 2017. godine. Iste godine upisuje i diplomski studij na istom fakultetu, smjer Procesno računarstvo (kasnije preimenovan u Robotika i umjetna inteligencija), koji i danas pohađa.

Potpis:

_____