Upravljanje mobilnog robota konstruiranog na platformi invalidskih kolica

Krešić, Martin

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka, Faculty of Engineering / Sveučilište u Rijeci, Tehnički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:190:509851

Rights / Prava: Attribution 4.0 International/Imenovanje 4.0 međunarodna

Download date / Datum preuzimanja: **2024-07-25**

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Engineering

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij strojarstva

Diplomski rad

UPRAVLJANJE MOBILNOG ROBOTA KONSTRUIRANOG NA PLATFORMI INVALIDSKIH KOLICA

Rijeka, rujan 2022.

Martin Krešić

0035199484

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij strojarstva

Diplomski rad

UPRAVLJANJE MOBILNOG ROBOTA KONSTRUIRANOG NA PLATFORMI INVALIDSKIH KOLICA

Mentor: izv. prof. dr. sc. Kristina Marković

Komentor: izv. prof. dr. sc. Željko Vrcan

Rijeka, rujan 2022.

Martin Krešić

0035199484

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 16. ožujka 2022.

Zavod: **Zavod za konstruiranje u strojarstvu**
Predmet: **Konstrukcijski elementi robota**
Grana: **2.11.01 opće strojarstvo (konstrukcije)**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Martin Krešić (0035199484)**
Studij: Diplomski sveučilišni studij strojarstva
Modul: Konstruiranje i mehatronika

Zadatak: **Upravljanje mobilnog robota konstruiranog na platformi invalidskih kolica / Control system for wheelchair based mobile robot**

Opis zadatka

Zadatak je konstruirati pogonski sustav za prenamjenu invalidskih kolica u mobilnog robota. Potrebno je razviti upravljanje mobilnog robota pomjenom operacijskog sustava Robot Operating System ROS kako bi se mobilni robot mogao gibati autonomno.
Rad treba obuhvatiti pregled stanja u mobilnoj robotici te teorijsku osnovu pogonskih sustava mobilnih robota i operacijskog sustava za upravljanje mobilnih robota. Potrebno je izabrati i obrazložiti optimalno rješenje pogonskog sustava i upravljanja te izraditi i priložiti tehničku dokumentaciju.

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

Martin Krešić

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor

Doc. dr. sc. Kristina Marković

Doc. dr. sc. Željko Vrcan (komentor)

Predsjednik povjerenstva za diplomski ispit

Prof. dr. sc. Kristian Lenić

Izjava

Izjavljujem da sam ovaj završni rad izradio samostalno koristeći navedenu literature i znanje stečeno na Tehničkom fakultetu u Rijeci.

Martin Krešić

Zahvala

Ovim putem htio bih se zahvaliti svojoj mentorici izv. prof. dr. sc. Kristini Marković, komentoru izv. prof. dr. sc. Željku Vrcanu i asistentici Maji Dundović za pruženu pomoć i savjete prilikom izrade ovog diplomskog rada.

Također bih želio zahvaliti i svojoj obitelji, curi i prijateljima koji su mi bili neopisivo velika podrška tokom cijelog diplomskog studija.

Sadržaj

1 Uvod

Mobilni roboti su vrsta robota koji imaju sposobnost kretanja kroz svoju okolinu. Kretanje ovakvih robota može biti potpuno autonomno ili upravljano od strane čovjeka. Zadatak ovog diplomskog rada je prenamijeniti konstrukciju invalidskih kolica u mobilnog robota s mogučnošću autonomnog kretanja kroz prostor. Kako bi se od invalidskih kolica napravio autonomni mobilni robot potrebno je implementirati četiri sustava koja čine svakog mobilnog robota:

1) Lokomocijski
2) Percepcijski
3) Kognicijski
4) Navigacijski

Lokomocijski i percepcijski sustav se implementiraju dodavanjem komponenti na već postojeću konstrukciju invalidskih kolica. Lokomocijski sustav daje robotu mogućnost kretanja kroz prostor i za njegovu implementaciju se uglavnom koristi postojeća struktura kolica uz dodatak pogonskih aktuatora. Uloga percepcijskog sustava je prikupljanje podataka iz okoline pomoću različitih senzora. Za implementaciju ovog sustava potrebno je odabrati odgovarajuće senzore.

Kognicijski i navigacijski sustav implementiraju se pomoću ROS-a (engl. *Robot Operating System*). Pomoću ovih sustava omogućava se upravljanje robotom kao i autonomna navigacija kroz prostor. ROS je modularni sustav otvorenog koda koji mogućnosti upravljanja i autonomne navigacije može ostvariti na razne načine. Za realiziranje upravljanja i autonomne navigacije u okviru ovog diplomskog rada koristit će se navigacijski stog. Navigacijski stog je skup algoritama i funkcionalnosti unutar ROS-a specifično namijenjen za autonomnu navigaciju robota.

2 Mobilni roboti

Mobilni roboti su oni roboti koji su sposobni kretati se kroz svoju radnu okolinu. Njihovo kretanje može se ostvariti teleoperacijom (upravljanjem od strane čovjeka) ili može biti potpuno autonomno. Kod autonomnog kretanja robot mora biti dovoljno „inteligentan“ da samostalno reagira i donosi odluke na temelju podataka iz okoline koje dobiva putem senzora [1]. Za razvijanje i proizvodnju ovakvih robota potrebno je udružiti znanja iz polja strojarstva, elektrotehnike, računarstva i informatike. Razvojem ove grane robotike nastali su brojni primjeri različitih mobilnih robota poput: mobilnih robota na kotačima, humanoidnih robota, bespilotnih letjelica (dronova), svemirskih rovera itd (Slika 1.1).

Slika 2.1 Različite vrste mobilnih robota

Zbog svojih sposobnosti mobilni roboti mogu zamijeniti rad čovjeka u mnogim poljima. Mobilni roboti tako se mogu koristiti za sljedeće primjene: nadziranje, patroliranje, svemirska istraživanja, petrokemijske primjene, industrijska automatizacija, transport, građevina, zabava itd. [1]

Mobilna robotika kao disciplina može se raščlaniti na četiri jednostavnije grane [1]:

1) Lokomocija
2) Percepcija
3) Kognicija
4) Navigacija

Lokomocija se bavi problematikom mehanizmima pokretanja, kinematikom, dinamikom i teorijom upravljanja [1]. Jednostavnije rečeno, lokomocija se bavi svim aspektima odgovornim za mogućnost gibanja robota u prostoru.

Percepcija se sastoji od područja analize signala, računalnog vida, te senzorskih tehnologija [1]. Kombinacijom spomenutih područja ostvaruje se mogućnost prikupljanja različitih signala iz robotove okoline. Prikupljanjem signala stvara se veza između robota i njegove okoline, odnosno može se reći da je grana percepcije zadužena za robotova „osjetila".

Kognicija je odgovorna za analizu ulaznih signala, dobivenih od percepcije, i izvršavanje odgovarajućih akcija s ciljem ostvarivanja robotovog zadatka [1]. Kognicija vrši ulogu središnjeg kontrolnog sustava robota, te je primarno odgovorna za ponašanje autonomnih mobilnih robota.

Navigacije je sačinjena od područja teorije informacije, umjetne inteligencije i algoritama planiranja [1]. Uloga navigacije je osmišljanje najoptimalnije rute kretanja autonomnog mobilnog robota. Kako bi mogla izvršiti svoj zadatak, navigacija mora biti dobro povezana sa kognicijom. Navigacija planira rutu kretanja na temelju obrađenih ulaznih podataka dobivenih od kognicije. Nakon što je ruta određena šalje se u kognicju, koja ostvaruje gibanje robota po ruti izvršavanjem niza odgovarajućih akcija.

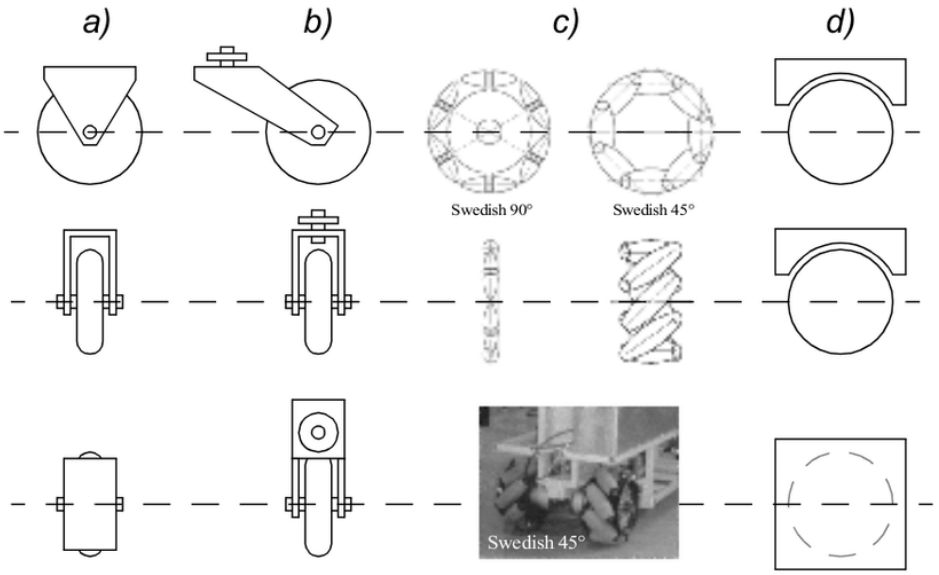
2.1 Lokomocija

Prvi sustav kojeg je potrebno konstruirati na svakom mobilnom robotu je lokomocijski sustav. On utječe na način na koji će se robot kretati, te na njegovu kinematiku i dinamiku. Izbor lokomocijskog sustava prvenstveno ovisi o mediju (kopno, voda, zrak) kroz koji se robot kreće. Dodatni uvjeti koji utječu na izbor lokomocijskog sustava su teresnki uvjet, manevrabilnost, mogućnost kontroliranja (engl. *controllability*), stabilnost, iskoristivost i mnogi drugi [1]. S obzirom na lokomocijski sustav mobilni roboti mogu se podijeliti na:

1) Kopneni mobilni roboti
    a. Mobilni roboti s kotačima
    b. Hodajući mobilni roboti (mobilni roboti sa nogama)
    c. Mobilni roboti s gusjenicama (gusjeničari)
    d. Hibridni mobilni roboti
2) Zračni mobilni roboti
3) Podvodni mobilni roboti

Mobilni roboti s kotačima najčešća su varijanta kopnenih mobilnih robota. Glavna prednost kotača nad nogama i gusjenicama je njihova jednostavnost. Iz tog razloga robote s kotačima jednostavnije je konstruirati, izgraditi i programirati, u usporedbi sa drugim mobilnim robotima. Glavni nedostatak uporabe kotača je otežana mobilnost po neravnim površinama i površinama s malim koeficijentom trenja. Postoje četiri vrste kotača koji se koristi u mobilnoj robotici (Slika 1.2):

1) Standardni fiksni kotač – konvencionalni kotač s jednim stupnjem slobode gibanja
2) *Caster* kotač – najčešće pasivni kotači sa dva stupnja slobode gibanja (rotacije oko glavne osi vrtnje i oko rotacijskog zgloba)
3) Švedski kotač – kotač sa slobodno rotirajućim valjcima na obodu, ima tri stupnja slobode gibanja
4) Sferični kotač – omnidirekcijski kotač [2]



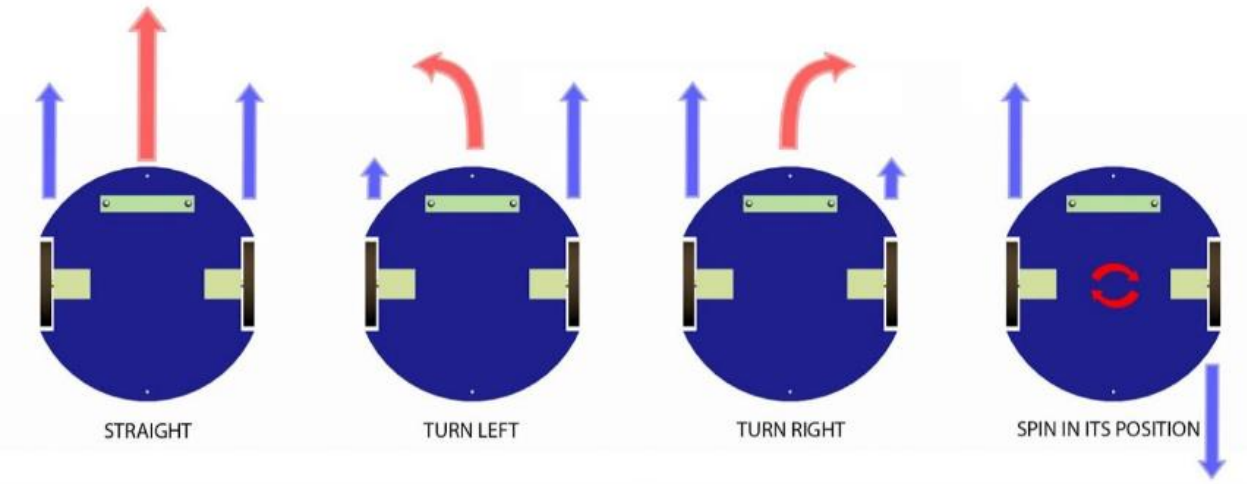
Slika 2.2 Vrste kotača; a) standardni fiksni b) caster kotač c) švedski kotač d) sferični kotač [2]

Vrsta i količina kotača na robotu bitno utječu na njegovu kinematiku i dinamiku, te je stoga pri konstruiranju potrebno obratiti posebnu pažnju pri izboru kotača. Najčešće su u uporabi roboti s dva, tri ili četiri kotača. Ovakve izvedbe nude najbolji omjer stabilnosti, manevrabilnosti i jednostavnosti upravljanja naspram cijene i kompleksnosti izrade. Izvedbe sa više kotača koriste se ako je potreban rad robota na nepirstupačnijim vanjskim terenima [1].

Osim po vrsti kotača, mobilne robote s kotačima moguće je podijeliti i po vrsti pogona. Mogući pogoni za mobilne robote s kotačima su [1]:

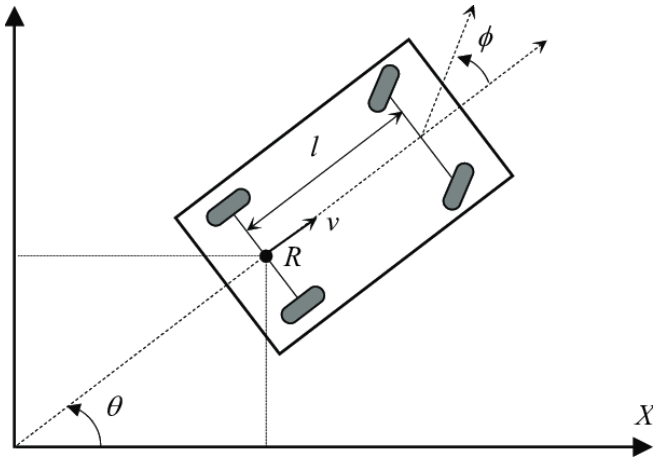
1) Diferencijalni pogon
2) Automobilski pogon
3) Omnidirekcijski pogon
4) Sinkroni pogon

Diferencijalni pogon koristi dva aktivna (pogonska) kotača sa zasebnim aktuatorima. Gibanje se kod diferencijalnog pogona ostvaruje kombiniranjem brzina svakog od kotača. Ukoliko kotači imaju jednake brzine rezultirajuće gibanje je pravocrtno, a ukoliko su im brzine različite dolazi do skretanja robota (Slika 1.3). Ovakav pogonski sustav je jednostavan za konstruirati i programirati i iz tog razloga se izrazito često koristi u praksi.



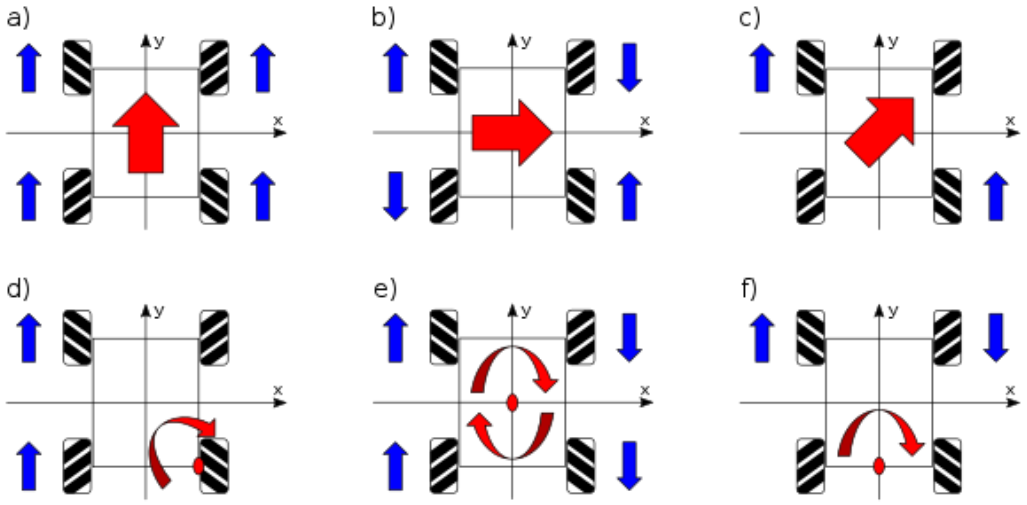
Slika 2.3 Način kretanja robota sa diferencijalnim pogonom [3]

Kako mu i ime govori automobilski pogon mobilnih robota oblikovan je po uzoru na automobile. Ovakav pogon (najčešće) se sastoji od četiri kotača, pri čemu su dva kotača pogonska, a preostala dva omogućuju skretanje robota (Slika 1.4). Ovakav pogonski sustav jednostavan je za konstruirati, te ima odlčinu stabilnost i robusnost [1]. Glavni nedostatak ovoga sustava je slabija manevrabilnost, odnosno nemogućnost promjene orijentacije bez promjene pozicije (robot se ne može okretati na mjestu).



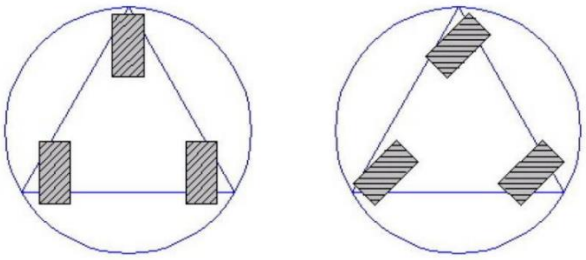
Slika 2.4 Mobilni robot s automobilskim pogonom [3]

Omnidirekcijski pogon omogućava kretanje mobilnog robota u svim smjerovima. Robot se može kretati naprijed/nazad, postrance lijevo/desno, te se rotirati na mjestu. Kako bi ovakav raspon gibanja bio moguć potrebno je koristiti švedske ili sferične kotače. Izvedbe sa švedskim kotačima uglavnom imaju četiri kotača, a izvedbe sa sferičnim kotačima najčešće imaju po tri kotača. Neovisno o količini kotača svaki od njih je pogonski i pokretan zasebnim aktuatorom. Različitom kombinacijom brzina i smjerova kretanja pojedinih kotača moguće je ostvariti sva ranije navedena gibanja (Slika 1.5). Roboti sa omnidirekcijskom pogonom mogu ostvariti jako složena gibanja i treba im znatno manje prostora nego ostalim vrstama robota. Najveći nedostatak ovakvog pogona je velika razina kompleksnosti i relativno visoka cijena.



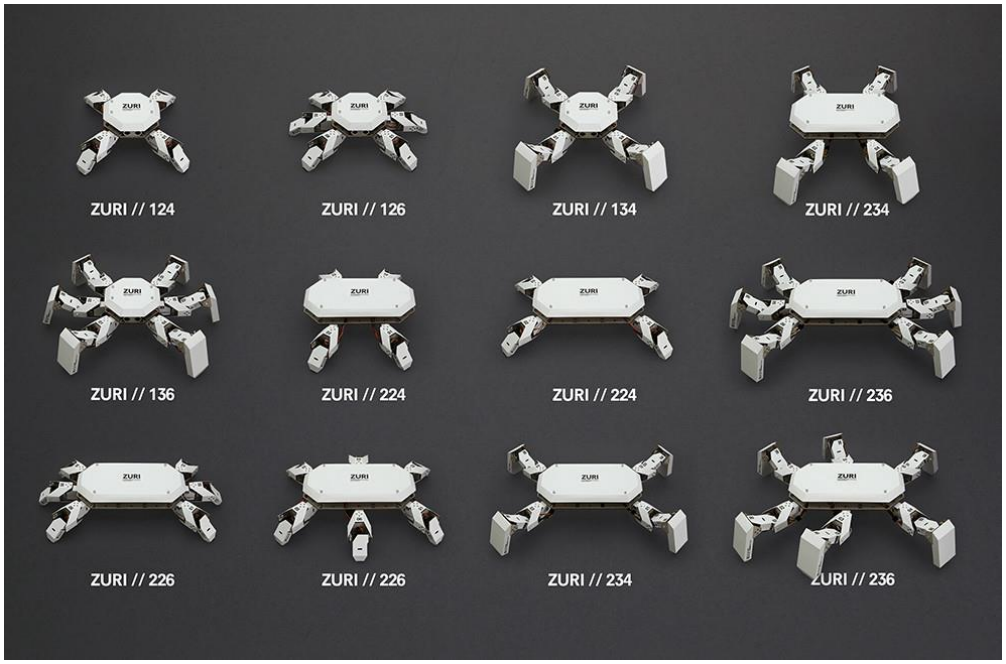
Slika 2.5 Omnidirekcijski pogon sa švedskim kotačima [3]

Sinkroni pogon sastoji se od tri ili više kotača koji su pogonjeni pomoću dva aktuatora. Ime je dobio zbog sinkroniziranosti brzina i orijenatacija kotača. Sinkronizacija se postiže konstrukcijskom izvedbom u kojoj je jedan aktuator odgovoran za kontrolu brzine, a drugi za kontrolu orijentacije svih kotača. Ovakav pogon relativno je teško konstruirati, ali ga je zato lako programirati.



Slika 2.6 Sinkroni pogon [3]

Hodajući roboti najčešća su alternativa robotima s kotačima. Za kretanje koriste noge koje povećavaju manevrabilnost i prilagodljivost robota neravnom terenu [1]. Korištenjem nogu robotu se omogućava penjanje i prelaženje preko prepreka, čime se ostvaruje nova dimenzija gibanja u odnosu na robote s kotačima. Glavni problem hodajućih robota je stabilnost, odnosno kompliciranost postizanja iste. Postoje dvije vrste stabilnosti, statička i dinamička. Statička stabilnost odnosi se na održavanje konfiguracije robota u mirovanju, a dinamička na održavanje konfiguracije u pokretu [1]. Statička stabilnost predstavlja najveći problem kod humanoidnih robota (dvije noge). Dodavanjem više nogu statička stabilnost se znatno poboljšava. Najčešće izvedbe imaju između dvije i osam nogu, s time da se preferira paran broju nogu radi jednostavnijeg programiranja i upravljanja (Slika 1.7).



Slika 2.7 Robot "Zuri" i njegove različite konfiguracije [4]

Roboti s gusjenicama (Slika 1.8) koriste gusjenice i diferencijalni pogon kako bi povećali manevrabilnost na neravnom, mekom terenu. Korištenjem gusjenica ostvaruje se veća kontaktna površina sa podlogom, što ovakve robote čini idealnima za kretanje po površinama s manjim koeficijentom trenja (blato, makadam). Ovakvi roboti proklizavaju prilikom skretanja, te stoga nije preporučljiva njihova primjena na tvrdim podlogama (veliki faktor trenja povećava trošenje). Također zbog proklizavanja mogu se javljati problemi s odometrijom odnosno lokalizacijom robota [1].

Slika 2.8 Roboti gusjeničari [5]

2.2 Percepcija

Za ispravan rad autonomnog mobilnog robota ključni su podaci koje robot posjeduje o svojoj okolini, ali i o samom sebi. Ove podatke robot dobiva pomoću senzora, točnije, pomoću izdvajanja relevantnih mjernih podataka dobivenih pomoću senzora. Korištenje senzora omogućava robotu izvršavanje zadataka vezanih za pozicioniranje, lokalizaciju, mapiranje i reprezentaciju . Također, korištenje senzora omogućava i neke dodatne napredne funkcije poput prepoznavanja objekata [1].

Postoji velik broj senzora koji se mogu primjeniti u mobilnoj robotici za potrebe prikupljanja podataka. Radi lakše kategorizacije uvode se sljedeće podjele senzora:

1) Proprioceptivni – eksteroceptivni
2) Pasivni – aktivni

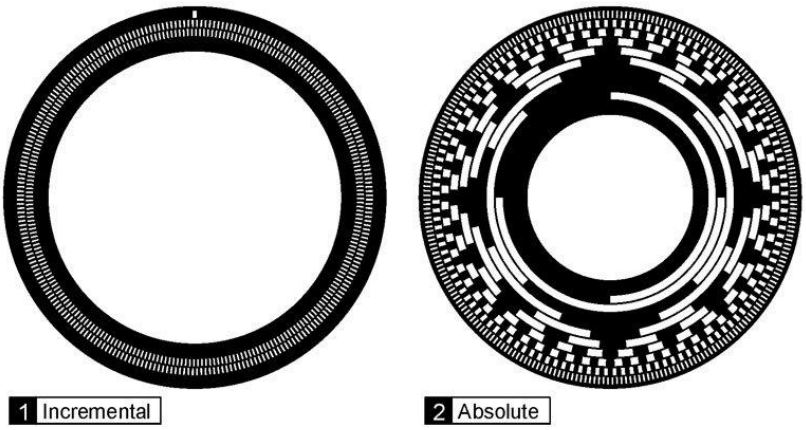
Proprioceptivni senzori su oni senzori koji mjere vrijednosti unutar samog robota. Pod te vrijednosti ubrajaju se: brzina motora, opterećenje kotača, kutevi zglobova, napon baterije itd. Eksteroceptivni senzori prikupljaju podatke iz okoline u kojoj se robot nalazi. Pod te podatke ubrajaju se: udaljenost, intenzitet svjetla, amplituda zvuka itd. [1]

Pasivni senzori mjere ambijentalnu energiju koja dolazi na senzor. Primjer takvog senzora je temperaturna sonda, koja dobiva mjerenja koristeći toplinsku energiju okoliša u kojem se nalazi. Aktivni senzori emitiraju određeni oblik energije u okoliš, te mjere povratnu reakciju. Primjer aktivnog senzora je sonar, koji emitira ultrazvučne valove i mjeri vrijeme njihove refleksije nazad do prijemnika, te na taj način određuje udaljenost između senzora i objekta [1]

Neki od najčešće korištenih senzora u mobilnoj robotici su:

1) Enkoderi
2) Akcelerometri
3) Žiroskopi
4) Ultrazvučni senzori
5) Laserski daljinomjeri
6) Senzori blizine (engl. *Proximity sensors*)

Enkoderi su senzori koji mjere pomak, no mogu se koristiti i za mjerenje brzine zbog linearne ovisnosti frekvencije pulseva i brzine enkodera [1]. Enkoderi se dijele na linearne i rotacijske s obzirom na vrstu gibanja, te na inkrementalne i apsolutne (Slika 1.9). Inkrementalni enkoderi mjere pomak brojanjem impulsa, pri čemu svaki impuls predstavlja jednu jedinicu pomaka koja ovisi o razlučivosti enkodera. Svaka pozicija na inkrementalnom enkoderu je identična, te enkoder ne može bilježiti podatke o apsolutnoj poziciji. Apsolutni enkoderi, osim podataka o pomaku, mogu dati i podatke o apsolutnoj poziciji s obzirom na početnu poziciju. Praćenje pozcije je moguće zbog toga što svaka pozicija na enkoderu predstavlja jedinstveni *bit* informacija. U mobilnoj robotici najčešće se koriste rotacijski enkoderi na pogonskim elementima ili na zglobovima. Enkoderi na pogonskim elementima mogu biti inkrementalni, budući da se koriste prvenstveno za mjerenje brzine, dok na zglobovima trebaju biti apsolutni zbog važnosti poznavanja točne pozicije zgloba u svakom trenutku.

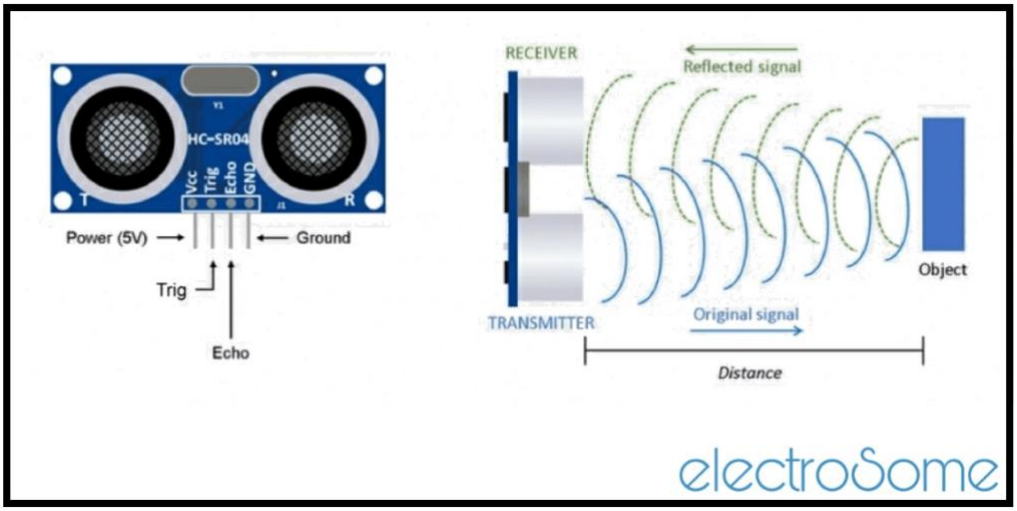


Slika 2.9 Inkrementalni i apsolutni rotacijski enkoderi [6]

Akcelerometar je senzor koji mjeri ubrzanje tijela na kojem se nalazi. Najčešće se osjetni element sastoji od inercijske mase ovješene na opruge. Prilikom ubrzanja dolazi do djelovanja sile na inercijska masu. Zbog djelovanja sile dolazi do promjene položaja inercijske mase, koja se pomiče sve dok se ne uravnoteže sila zbog ubrzanja i sila u opruzi. Veličinu pomaka inercijske mase potrebno je pretvoriti u električni signal kako bi se rezultat mogao očitati na nekom od odgovarajućih uređaja [7]. U mobilnoj robotice najčešće se primjenjuju zajedno sa žiroskopima unutar IMU-a (engl. *Internal Measurement Unit*).

Žiroskop je senzor koji se koristi za mjerenje kutne brzine i određivanje orijentacije. Klasični žiroskopi sastoje se od rotirajućeg diska montiranog unutar dva ili tri kardanska prstena, no ovakvi žiroskopi se ne koriste u robotici. Za potrebe robotike koriste se MEMS (engl. *Micro-Electro-Mechanical Systems*) žiroskopi koji koriste vibrirajuće strukture i rotirajuće okvire. Uslijed rotacije okvira vibrirajuća struktura djeluje silom na okvir, ova sila je poslijedica Coriolisovog efekta [8]. Mjerenjem nastale sile moguće je ustanoviti rotacijsku brzinu. U mobilnoj robotice najčešće se primjenjuju zajedno sa akcelerometrima unutar IMU-a.

Ultrazvučni senzor (Slika 1.10) je uređaj koji mjeri udaljenost od objekta koristeći ultrazvučne valove. Senzor emitira zvučne valove, te ih potom i očitava nakon što se reflektiraju od površine nekog tijela. Udaljenost se određuje pomoću vremenske razlike između emitiranog i primljenog reflektiranog signala. Glavne komponente koje sačinjavaju ultrazvučni senzor su odašiljač koji odašilje signal pomoću piezoelektričnih kristala i prijamnk koji očitava reflektirani signal [1]. Ultrazvučni senzor koristi se u mobilnoj robotici za detekciju prepreka i određivanje udaljenosti između robota i prepreke.



Slika 2.10 Ultrazvučni senzor i njegov princip rada [9]

Laserski daljinomjer koristi se za mjerenje udaljenosti i funkcionira na sličan način poput ultrazvučnog senzora, samo umjesto zvučnih valova emitira i detektira svjetlost. Preciznost ovakvih senzora je otprilike 25 puta veća od ostalih senzora za mjerenje udaljenosti. Veća preciznost proizlazi iz toga da laserski senzori za područje od 180° generiraju 180 očitanja, dok senzori poput sonara generiraju manji broj očitanja za isti prostor. Također zbog visoke fokusiranosti laserskih zraka one se ne distorziraju i propuštaju kroz medij, te se time postiže manje lažno pozitivnih detekcija prepreka [10].

Senzori blizine su senzori koji detektiraju prisustvo nekog objekta unutar određenog područja. Ovi senzori ne mogu odrediti točnu udaljenost objekta, nego samo da li se objekt nalazi u njihovom mjernom području [1]. Senzori blizine se mogu podijeliti na kontaktne i beskontaktne. Kako im i ime kaže kontaktni senzori blizine moraju biti u direktnom kontaktu sa objektom kako bi ga prepoznali dok beskontaktni senzori mogu prepoznati objekte na određenoj daljini. Detektiranja objekta prije kolizije veome je značajno u polju mobilne robotike, te se iz tog razloga preferira korištenje beskontaktnih senzora. Najčešće su u uporabi kapacitivni senzori, čiji se princip rada temelji na promjenjivom kapacitetu kondenzatora ovisno o dielektričnosti prostora ili predmeta ispred senzora [1]. U mobilnoj robotici koriste se za detekciju objekata u blizini robota, s ciljem planiranja rute i izbjegavanja kolizija.

2.3 Kognicija

Kako bi robot mogao izvršiti zadane zadatke potrebno je na određen način kontrolirati njegovu mehaničku strukturu. Kognicija je dio mobilnog robota koji je odgovoran za određivanje načina na koji će robot izvršiti zadatke. Kako bi se odredio najoptimalniji način izvršavanja zadataka potrebni su podaci o stanju robota, robotovoj okolini i odnosu između robota i okoline. Spomenuti podaci dobivaju se analizom i obradom podataka dobivenih iz senzora, odnosno percepcije [1]. Na temelju podataka dobivenih iz percepcije i zadanog zadatka kognicija planira najbolji način ostvarivanja rute kretanja robota, te šalje upute kontrolnom sustavu koji potom pravovremeno aktivira pojedine aktuatore.

Kako bi kognicija mogla izvršiti navedene funkcije potrebno je napraviti odgovarajući kognitivni model. Zadaća kognitivnog modela je prikazivanje robota, njegove okoline i njihove interakcije. Kognitivni modeli koriste tehnologije računalnog vida i prepoznavanja uzoraka za prepoznavanje i praćenje objekata u robotovoj okolini. Razni algoritmi za mapiranje koriste se za kreiranje mapa koje stvaraju prikaz robotove okoline, na temelju podataka dobivenih iz senzora [1].

Za pokretanje robota, odnosno njegovih aktuatora, direktno je odgovoran kontrolni sustav, a indirektno kognicija. Kognicija je ta koja kontrolnom sustavu daje informacije kada i kako pokrenuti pojedini aktuator. Kako bi kognicija mogla davati točnije upute potrebno je pratiti položaj robota za vrijeme gibanja po ruti, odnosno potrebno je odrediti njegovo eventualno odstupanje od zadane rute. Tokom godina razvijeni su mnogi algoritmi i metode za praćenje i minimiziranje/eliminiranje odstupanja, ti algoritme i metode nazivaju se još i strategije kontrole. Neke od poznatijih i češće korištenih strategija kontrole su:

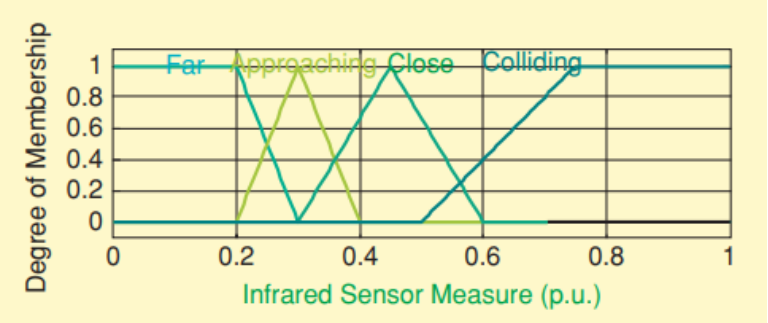
1) Robusne metode kontrole
2) Adaptivne metode
3) Metoda neuronskih mreža
4) Metode *fuzzy* logike

Robusne metode kontrole se zasnivaju na stvaranju robusnih sustava kontrole. Sustav kontrole je robusan kada zadovoljava postavljene uvjete i uz prisustvo nesigurnosti. Što je veća granica tolerancije na nesigurnosti, to je sustav robusniji. Kontroleri napravljeni pomoću robusnih metodi garantiraju ispravan rad dok god su odstupanja od njihovog modela unutar datih granica. Robusne metode kontrole u mobilnoj robotici koriste se za određivanje okretnog momenta na temelju drugih derivacija trenutne pozicije i željenje pozicije, te matrice masa [1]. Ove metode obavezno koriste povratnu petlju za određivanje i implementaciju proporcionalnih i derivacijskih (PD) pojačanja.

Sustavi bazirani na adaptivnim metodama kontrole automatski kompenziraju varijacije u dinamici sustava. Kompenzacija se vrši mijenjanjem karakteristika kontrolera s ciljem da radna svojstva sustava ostanu ista, odnosno optimalna. Svaki sustav baziran na adaptivnim metodama sadrži elemente za mjerenje ili procjenu dinamike sustava i za promjenu karakteristika kontrolera [11]. Adaptivne metode, u usporedbi s robusnim, zahtijevaju manje poznavanje dinamike robota, također dopuštaju i linearnu pretpostavku parametara [1].

Neuronske mreže su dio polja strojnog učenja, inspirirane radom ljudskog mozga. Sastoje se od određenog broja međusobno povezanih čvorova (neurona) koji su zaduženi za obavljanje računskih operacija nad podacima. Podaci koji uđu u neuronsku mrežu prolaze od neurona do neurona i mijenjaju se sukladno računskim operacijama pojedinih neurona. Na izlazu iz mreže potom se dobiva novi set podataka [12]. Ovaj izlazni set podataka može se koristiti kao ulazni set podataka kontrolera robota, koji na temelju tih podataka pomiče robota. Nakon pomicanja robota može se provjeriti odstupanje od željene putanje, koje zatim postaje novi ulaz u neuronsku mrežu. Ovaj proces se potom ponavlja dok se ne zadovolji tražena putanja. Što više puta neuronska mreža prođe ovaj proces „učenja“, tim se povećava brzina dolasaka na pravu putanju kao i ispravnost praćenja putanje [13].

Fuzzy logika, za razliku od standardne *boolean* logike, dozvoljava vrijednosti parametara između 0 i 1 (*boolean* dozvoljava isključivo ili 0 ili 1). Ovakav način bilježenja parametara omogućava jednostavan način uvođenja nesigurnosti u samu logiku kontrolera, npr. vrijednost od 0.9 predstavlja sigurnost u točnost podatka u iznosu od 90%. U primjeni *fuzzy* logike često se koriste nenumeričke vrijednosti za izražavanje pravila i činjenica. Iz tog razloga, za korištenje *fuzzy* logike, potrebno je prvo fuzificirati (engl. *fuzzify*) ulazne podatke, odnosno podijeliti ih u *fuzzy* setove. Nakon podjele podataka u setove moguće je nad njima provoditi *fuzzy* logiku, koja načešće poprima oblik „ako – tada“ (engl. *if – then*) funkcija [14]. Primjer raspodjele podataka o blizini prepreke prikazan je na Slici 1.11. Na apcisi su prikazani podaci infracrvenog senzora blizine (veća vrijednost znači da je prepreka bliže), a na ordinati pripadnost setu. Napravljena je podjela na sveukupno četiri *fuzzy* seta, između kojih ima određenih preklapanja radi glađeg prijelaza između setova. Na temelju ovakvih setova potom se može napisati pravilo „ako je prepreka blizu („close“) tada uspori“, pri čemu treba prvo pravilno definirati naredbu „uspori“ kako bi je robot mogao razumijeti.



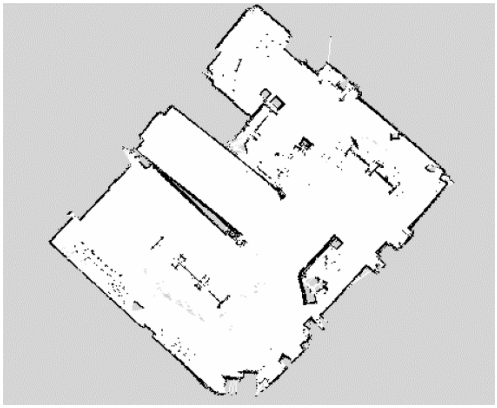
Slika 2.11 Fuzzy setovi za izbjegavanje prepreke [15]

2.4 Navigacija

Glavni zadatak svakog mobilnog robota je pomicanje od početne do zadane točke/pozicije u prostoru. U većini slučajeva, zbog postojanja prepreka, robot ne može doći do cilja najdirektnijom ili najbržom rutom. Za stvaranje optimalne rute, temeljene na mapi prostora i podacima iz senzora, odgovoran je navigacijski sustav [1]. Zadaće navigacijskog sustava su:

1) Mapiranje
2) Lokalizacija
3) Planiranje rute

Mapiranje je proces stvaranja pojednostavljenih, virtualnih reprezentacija (mapa) robotova okruženja u formatu koji je robotu razumljiv. Pod pojmom „mapa“ najčešće se misli na 2D tlocrt robotovog okruženja, moguće je napraviti i 3D reprezentaciju robotova prostora, ali taj proces je znatno kompliciraniji i najčešće nepotreban. Mape se stvaraju na temelju podataka dobivenih putem primjerenih senzora (npr. laserski daljinomjer). Poželjeno je da korišteni senzori imaju čim veću točnost i preciznost kako bi se napravila što točnija i preciznija mapa. Točnost i preciznost mape imaju značajan utjecaj na ispravnost i kompleksnost lokalizacije i planiranja rute. Primjer izgleda mape prikazan je na Slici 1.12. Mapa je dobivena korištenjem laserskog daljinomjera, crno označena područja predstavljaju prepreke, odnosno objekte od kojih se laserska zraka reflektirala.

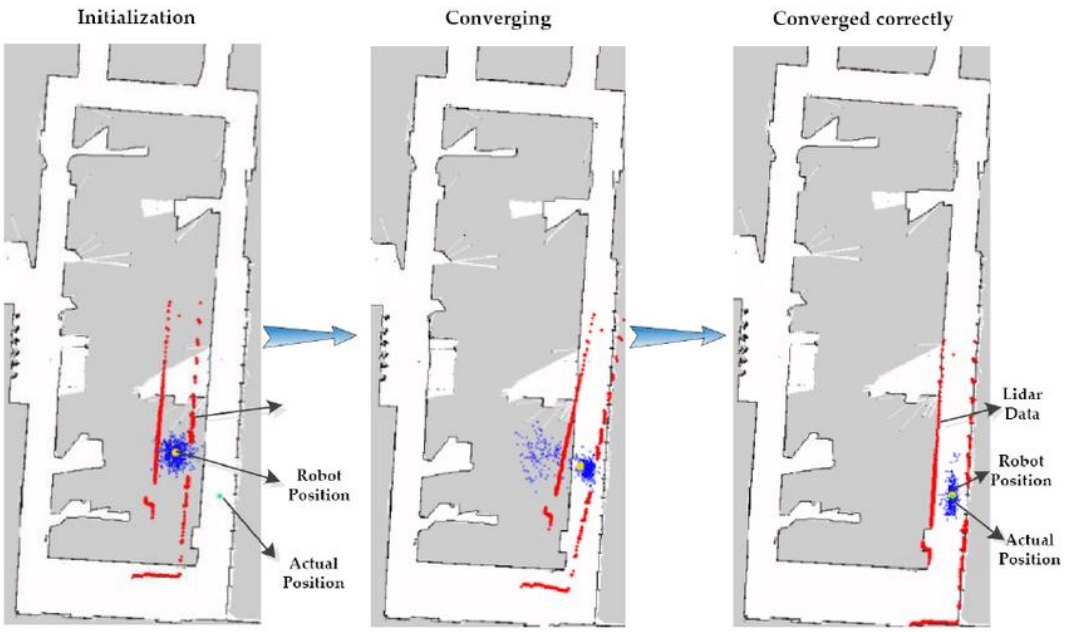


Slika 2.12 Prikaz mape u programu rviz [16]

Lokalizacija je proces određivanja lokacije robota s obzirom na njegovu okolinu. Robotova lokacija unutar okoline je jedan od najbitnijih podataka u mobilnoj robotici. Početna robotova lokacija potrebna je za planiranje rute do cilja. Trenutna robotova lokacija bitna je za praćenje robotova gibanja i utvrđivanja eventualnih odstupanja od planirane rute . Postoje razne metode za lokalizaciju robota, no zajedničko im je da se većinski zasnivaju na podacima senzora, odometriji i poznavanju mape prostora. Neke od metoda lokalizacije su [1]:

1) Markova lokalizacija
2) Lokalizacija pomoću Kalmanova filtera
3) (Adaptivna) Monte Carlo lokalizacija (Slika 1.13)

Postoje metode koje omogućavaju lokalizaciju u prethodno nemapiranom prostoru. Ovakve metode skupno se nazivaju SLAM (engl. *Simultaneous Loaclization and Mapping*), te one istovremeno provode procese lokalizacije i mapiranja.



Slika 2.13 Adaptivna Monte Carlo lokalizacija [16]

Zadatak planera rute je pronalaženje najboljeg puta do zadanog cilja. Definicija „najboljeg“ puta ovisi o metodama i algoritmima na kojima je planer rute baziran. Tako će neki planeri preferirati rutu koja će robota najbrže dovesti do cilja, dok će neki veći prioritet staviti na sigurnije zaobilaženje prepreka.

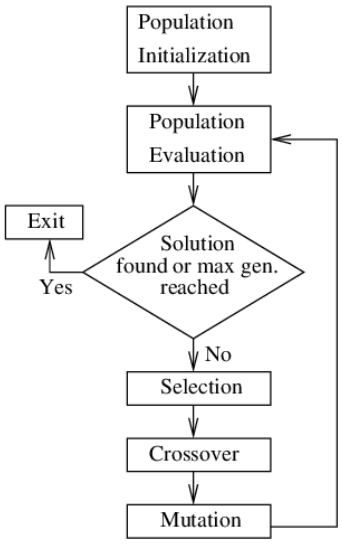
Kroz povijest je razvijen velik broj različitih metoda, algoritama i planera za planiranja rute, a oni se generalno mogu podijeliti u četiri kategorije [1]:

1) Klasične metode
2) Probabilističke metode
3) Heuristički planeri
4) Evolucijski algoritmi

Klasične metode su najstarije metode planiranja rute. Ove metode bazirane su na *roadmap* algoritmima, potencijalnim funkcijama i dekompoziciji ćelija. Neke od klasičnih metoda pokazale su se problematičnima zbog prevelikih memorijskih zahtjeva za dekompoziciju ćelija i postojanja lokalnih minimuma unutar polja potencijalnih funkcija. Rješavanjem pojedinih problema klasičnih metoda nastaju probabilističke metode. Probabilističke metode značajno se ne razlikuju od klasičnih. Glavna razlika je u dodatku slučajnog odabira u određenim fazama planiranja rute [1].

Heuristički planeri su skupina planera koja prioritizira brzinu planiranja rute. Heuristički planeri generalno žrtvuju točnost i preciznost kako bi što brže isplanirali rutu. Kako bi uštedili na vremenu heuristički planeri rade sa većim brojem jednostavnijih opcija, te se konačni rezultat temelji na najboljem mogućem rješenju dobivenom kombinacijom tih opcija.

Evolucijski algoritmi su računalni procesi koji imitiraju evolucijski proces u prirodi i primjenjuju ga na apstraktne jedinke. Algoritam u početku nasumično generira određen broj jediniki, pri čemu svaka jedinka predstavlja potenijalno rješenje problema. Svako potencijalno rješenje ima određenu mjeru kvalitete, a funkcija koja tu kvalitetu određuje naziva se funkcija cilja. Proces odabiranja najboljeg rješenja podijeljen je u određen broj ciklusa. Na kraju svakog ciklusa potencijalna rješenja se uspoređuju i odabire se određeni broj najboljih jedinki koje ulaze u idući ciklus. Nad nekim od jedinaka se tada provode operacije križanja i/ili mutiranja kako bi se stvorila nova nasumična populacija jedinki. Ovi ciklusi se ponavljaju dok se ne prođe kroz prethodno određeni broj ciklusa ili dok se ne zadovolji određeni uvjet [17].



Slika 2.14 Princip rada evolucijskog algoritma [18]

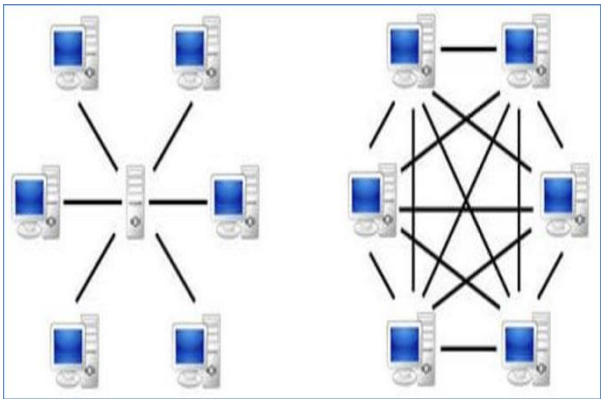
3 Robot Operating System (ROS)

Robot Operating System (ROS) nije operacijski sustav u tradicionalnom smislu upravljanja procesima i vremenskog planiranja. Umjesto toga ROS pruža strukturirani komunikacijski sloj koji se nalazi iznad glavnog operacijskog sustava heterogenog računalnog klastera [19]. Stoga je pravilnije reći da je ROS fleksibilni okvir za razvoj robotskog software-a, a ne operacijski sustav, iako mu ime kaže suprutno.

ROS je dizajniran kako bi zadovoljio specifičan set uvjeta prilikom razvoja uslužnih robota na velikoj skali, u sklopu STAIR projekta na sveučilištu u Stanfordu, i razvoja osobnih robota u tehnološkom inkubatoru Willow Garage. Dobivena arhitektura ROS-a pokazala se pogodnom za rad ne samo u domeni uslužnih robota i mobilno-manipulatorskoj domeni, već i za rad na širem polju robotike [1]. Glavni ciljevi pri razvijanju ROS-a bili su:

1) *Peer-to-peer* topologija
2) Baziranost na alatima (engl. *Tool-based*)
3) Višejezičnost
4) Mora biti „mršav“ (engl. *„thin“*)
5) Mora biti besplatan i otvorena koda (eng. *Open-Source*)

Peer-to-peer (P2P) topologija je oblik povezivanja i komunikacije više računala, a da pritom nije potrebno centralno računalo kao kod klasične *server-client* topologije (Slika 2.1). Ovakva topologija je povoljna za robote koji na/u sebi imaju ugrađeno više računala, odgovornih za jednostavnije zadatke, koja su dodatno povezana s vanjskim računalima odgovornim za složenije zadatke. Najčešće su ova dva tipa računala povezana bežično putem spore *Wi-Fi* mreže. Korištenjem *server-client* topologije došlo bi do nepotrebno velikog prijenosa podataka preko spore mrežo, što bi negativno utjecalo na rad robota. Kod P2P topologije sva računala su međusobno direktno povezana, te se time smanjuje količina podataka koja se prenosi bežično, što pozitivno utječe na brzinu prijenosa podataka između računala.



Slika 3.1 sporedba server-client topologije (lijevo) i P2P topologije (desno) [20]

ROS je okvir baziran na mikro jezgrama, odnosno sačinjen je od velikog broja manjih alata čija je uloga izgradnja i pokretanje raznih ROS komponenti. Neki od primjera alata su alati za: konfiguraciju parametara, vizualizaciju P2P topologije, autogeneriranje dokumenata itd [19]. Jednostavnije rečeno ROS nije monolitan sustav, nego modularan tj. „baziran na alatima“.

Kako bi omogućio jednostavno korištenje što većem broju korisnika ROS je napravljen kao višejezičan, odnosno neovisan o programskom jeziku. Ovo je moguće zbog korištenja XML-RPC protokola za uspostavljanje veza i konfiguriranja P2P mreže i činjenice da svi veći programski jezici imaju u sebi implementiranu potporu za ovaj protokol [19]. Također ROS arhitektura omogućava interakciju modula razvijenih pomoću dva različita programska jezika, npr. modul napisan u C++ i modul napisan u Python-u mogu neometano komunicirati i funkcionirati unutar istog projekta. Za ovakvu interakciju zaslužan je IDL (engl. *Interface Definition Language*) koji opisuje poruke koje se šalju među modulima. IDL koristi kratke tekstualne datoteke za opise poruka. Generatori koda pojedinog programskog jezika potom na temelju tih tekstualnih datoteka „prevode“ poruku na traženi programski jezik.

Većina *software*-a u robotičkim projektima sadrži algoritme i *driver*-e koji bi se mogili koristiti i van tog projekta, odnosno prenamjeniti za uporabu u nekom drugom projektu. Nažalost zbog različitih razloga korisni kod ostaje „zapetljan“ u *middleware*-u (*software*-u između operacijskog sustava i aplikacija) i ne može biti korišten van projekta [21]. Kako se ovakve stvari nebi događale u ROS-u je usvojena „mršava“ ideologija izgradnje. Pri razvoju *software*-a za ROS moguće je kompletan razvoj algoritama i *driver*-a napraviti u zasebnim knjižnicima koje su neovisne o ROS-u. Knjižnice je potom moguće povezati s ROS-om pomoću Catkin sustava izgradnje, koji stvara male izvršne datoteke koje omogućuju rad knjižnica u ROS-u [19]. Ovakva „mršava“ izgradnja koja zaobilazi uporabu *middleware*-a omogućava laganu prenamjenu koda za razne projekte i podržava modularni princip rada na kojem se temelji ROS.

Cjelokupni izvorni kod ROS-a je javno dostupan i BSD licenciran. BSD licenca omogućava korištenje ROS-a i za nekomercijalne i za komercijalne projekte. Zbog modularnosti samog sustava, razni moduli i paketi nisu BSD licencirani. Odluku o vrsti licence pojedinog modula donosi njegov kreator.

3.1 Glavne značajke ROS-a

Glavnim značajkama nazivaju se one značajke koje su neophodne za ispravan rad i funkcionalnost ROS-a, i čija je implementacija obavezna u svim ROS modulima i projektima baziranim na ROS-u. Pod glavne značajke ubrajamo:

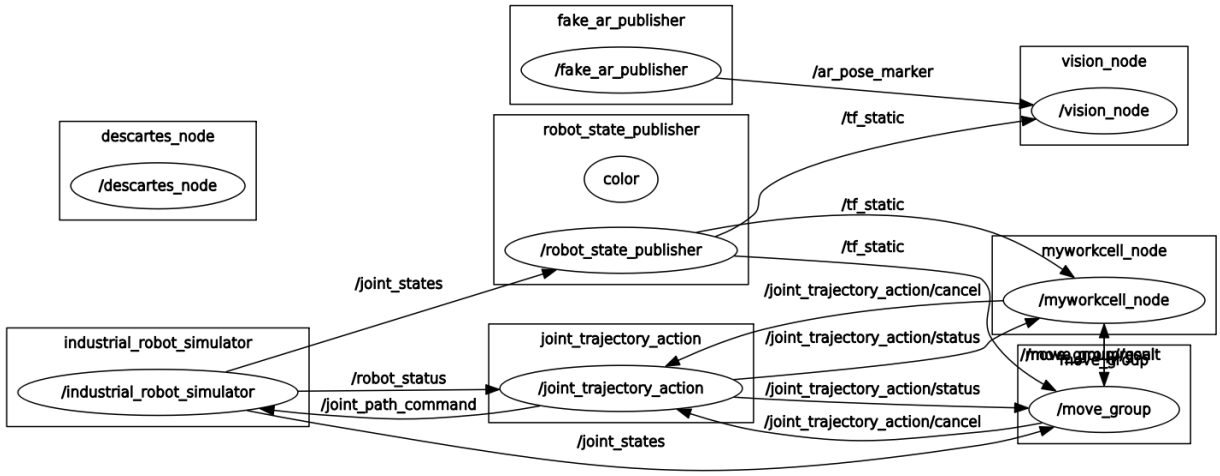
1) Čvorove (engl. *Nodes*)
2) Teme (engl. *Topics*)
3) Usluge (engl. *Services*)
4) Poruke (engl. *Messages*)
5) Pakete (engl. *Packages*)
6) Sustav izgradnje (engl. *Build System*)

3.1.1 ROS čvorovi

ROS čvor je proces koji obavlja izračun. Čvorovi su napravljeni za rad na malim skalama odnosno odgovorni su za pojedinu komponentu ili proces. Naprimjer jedan čvor će biti odgovoran za sonar, drugi za pogonske kotače, a treći za lokalizaciju robota. Ovakav princip rada znači da će robot moći obaviti zadatak tek onda kada su svi potrebni čvorovi za taj zadatak pravilno povezani. Prednost ovakvog načina rada je što su greške lokalizirane tj. pad jednog čvora neće uzrokovati pad cijelog sustava. Također ovakav sustav je baziran na kombiniranju više jednostavnijih programskih kodova, što mu pruža dodatnu prednost nad monolitnim sustavima. [22]

Čvorovi međusobno komuniciraju preko tema, RPC usluga i parametarskog servera. Kako bi se komunikacija mogla nesmetano odvijati potrebno je svaki čvor definirati pomoću dva imena. Prvo ime je „graph resource name“ koje predstavlja čvor na grafu povezivanja, ovo ime osim imena samog čvora uzima u obzir i okolinu gdje se čvor nalazi. Drugo ime je „package resource name“ koje predstavlja paket u čijem sklopu se čvor nalazi. Koristeći ovakav sustav imenovanja ne dolazi do konflikata ukoliko dva čvora imaju isto ime. Naprimjer ako robot ima dvije kamere kojima se upravlja pomoću različitih paketa „kamera1“ i „kamera2“, ime čvora za svaki od njih može biti samo „kamera“. Do konflikta neće doći pošto ih ROS registrira kao „kamera1/kamera“ i „kamera2/kamera“ [23].

Za detaljne podatke o pokrenutim čvorovima koriste se „rosnode“ komandne linije, a s dodatkom paketa „rqt_graph“ moguće je vizualizirati strukturu pomoću grafičkog sučelja (Slika 2.2).



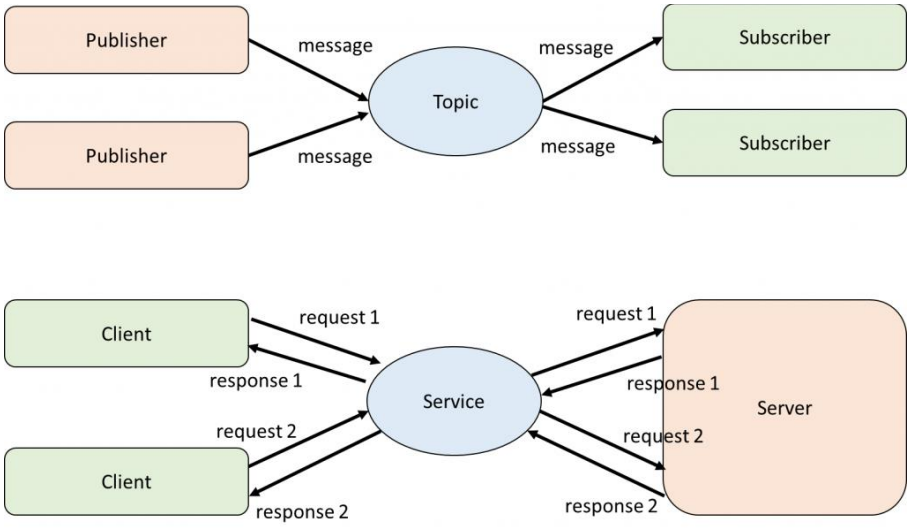
Slika 3.2 Prikaz grafa strukture pomoću rqt_graph paketa

3.1.2 ROS teme, usluge i poruke

ROS teme su imenovane sabirnice preko kojih čvorovi izmijenjuju poruke. Teme koriste anonimnu semantiku za objave i pretplate, što znači da čvorovi koji koriste temu ne znaju s kojim čvorovima komuniciraju. No za ovakav oblik komunikacije nije ni bitno od kuda informacije dolaze, već je bitan sadržaj informacija. Čvorovi se na temelju informacija koje trebaju ili generiraju spajaju s određenom temom gdje informacije objavljuju (engl. *publish*) ili se na njih pretplaćuju (engl. *subscribe*). Jedna tema podržava veći broj objavljivača i pretplatnika. Teme su namijenjene za jednosmjernu komunikaciju od objavljivača do pretplatnika (Slika 2.3) [24].

Jednosmjerni komunikacijski model objavljivanja/pretplate veoma je fleksibilan, ali nije pogodan za interakcije zahtjev/odgovor (engl. *request/reply*) preko RPC protokola. Kada je potrebna interakcija zahtjev/odgovor koriste se ROS usluge, koje pružaju dvosmjernu komunikaciju nalik modelu *client-server*. ROS usuluge definirane su pomoću para poruka, pri čemu jedna poruka opisuje zahtjev, a druga odgovor. Za razliku od tema gdje je više čvorova moglo objavljivati pod istu temu, samo jedan čvor može pružati uslugu (Slika 2.3). Ukoliko drugi čvor želi stupiti u kontakt sa čvorom koji pruža uslugu mora poslati odgovarajući zahtjev. Nakon zaprimanja i obrade zahtjeva čvor šalje odgovarajući odgovor nazad prema čvoru koji je uputio zahtjev [25].

ROS poruke su jednostavna podatkovna struktura sačinjena od primitivnih tipova podataka (integer, boolean, itd.) i njihovih matrica. Poruke mogu sadržavati i ugniježđene strukture koje nalikuje na klasične C strukture. Poruke prate identičnu konvenciju imenovanja kao i čvorovi. Bitno je pravilno imenovati poruke jer se time određuje njen tip, a samo se poruke s istim tipom mogu razmijenjivati među čvorovima. Osim tipa bitno je i podudaranje MD5 sume. [26]



Slika 3.3 Usporedba ROS tema i usluga

3.1.3 ROS paketi

ROS paketi su osnovna građevna jedinica svakog projekta. Paketi objedinjuju čvorove, nezavisne ROS knjižnice, *dataset*-ove, konfiguracijske datoteke i ostale stvari koje sačinjavaju koristan modul. Paketi se rade po „Zlatokosa“ principu: ne premalo da se ne izgubi funckionalnosti i ne previše da paket nebi bio prekompliciran i težak za uporabu sa drugim *software*-om. Primjenom ovog principa osigurava se mogućnost korištenja istog paketa u različitim sustavima u kombinaciji s različitim paketima, kao i mogućnost jednostavne prilagodbe paketa za potrebe korisnika. [27]

Budući da su paketi principijelno samo direktoriji, lako ih je napraviti potpuno manualno ili koristeći alat „catkin_create_pkg“. Kako bi ROS prepoznao da je direktorij zapravo paket, on mora sadržavati manifest u obliku XML datoteke. Datoteka manifesta naziva se „package.xml“ i mora sadržavati sljedeće podatke:

- ime paketa (<name>)
- verziju (<version>)
- opis paketa (<description>)
- ime održavatelja (<maintainer>)
- softwareske licence koda (<license>)
- ovisnost o drugim paketima

3.1.4 ROS sustav izgradnje

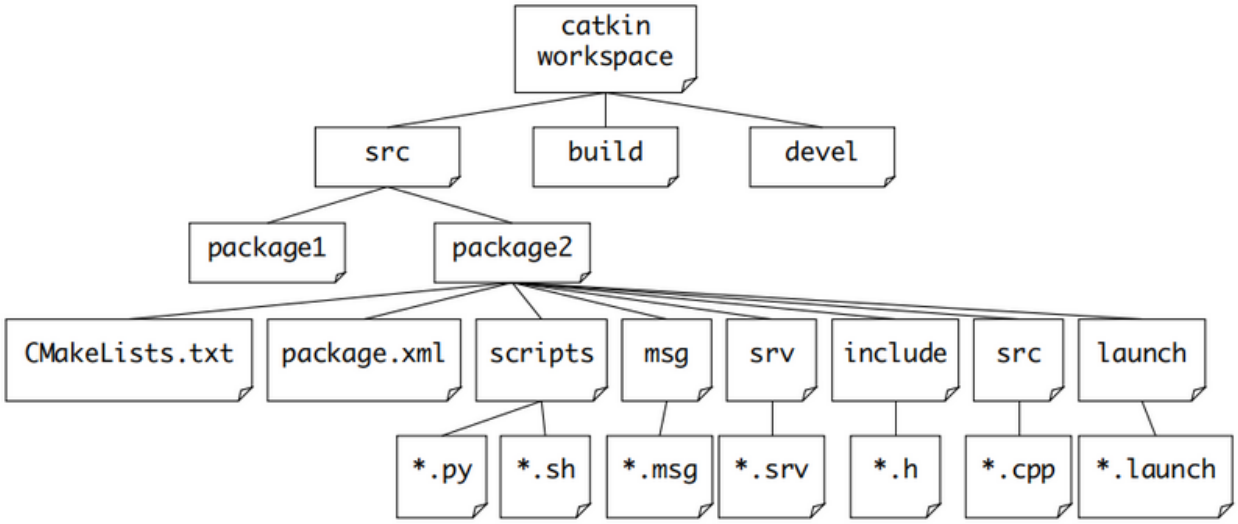
Sustav izgradnje (eng. build system) je sustav čija je zadaća stvaranje „meta“ na temelju izvornog koda. Ovaj korak je bitan kako bi se korisniku omogućila interakcija i korištenje paketa bez poznavanja i razumijevanja samog koda paketa. Dobivene mete mogu biti u obliku knjižnica, izvršivih programa, skripti ili bilo čega što nije statički kod. U ROS-u se za svaki paket nakon izgradnje stvara minimalno jedna meta.

ROS koristi catkin build system koji je napravljen specifično za njega. Catkin se bazira na CMake platformi na koju su dodane Python skripte koje omogućuju automatsko pronalaženje paketa i istovremenu izgradnju više zavisnih projekata. [28]

Za izgradnju više zavisnih paketa pomoću catkin-a potrebno definirati radni prostor (eng. workspace). Catkin radni prostor sastoji se od:

- izvornog prostora (eng. source space)
- prostora za izgradnju (eng. build space)
- razvojnog prostora (eng. development space)
- instalacijskog prostora (eng. install space)

Izvorni prostor se koristi za preuizimanje i pohranu ROS paketa. Osim samih paketa ovaj prostor mora sadržavati i „CMakeLists.txt“ datoteku koja sadrži podatke o izvornim datotekama i upute za stvaranje meta. U prostoru za izgradnju odvija se izgradnja putem „cmake <put do izvornog prostora> make“ naredbe. Izgrađeni paketi smještaju se u razvojni prostor koji služi kao okruženje za testiranje i razvoj prije same instalacije paketa. Kada je korisnik zadovoljan rezultatima gradnje pokreće se instalacija paketa u instalacijski prostor. Instalacija se pokreće naredbom „make install“. [28]



Slika 3.4 Struktura Catkin radnog prostora[28]

4 Prenamjena invalidskih kolica u mobilnog robota

Cilj ovog diplomskog rada je prenamjena invalidskih kolica u autonomnog mobilnog robota. Kako je opisano u poglavlju 1, mobilni roboti sastoje se od četiri glavna sustava:

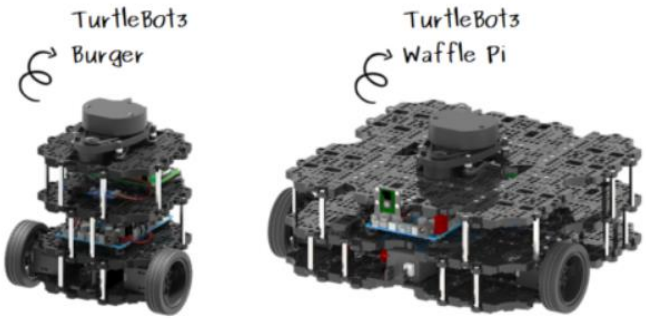
1) Lokomocijski sustav
2) Percepcijski sustav
3) Kognicijski sustav
4) Navigacijski sustav

Za osoposobljavanje lokomocijskog i percepcijskog sustava potrebno je napraviti fizički preinake na invalidskim kolicima, odnosno potrebno je dodati određene komponente.

Osnovni lokomocijski sustav invalidskih kolica sastoji se od četiri kotača, od čega su dva standardni fiksni kotači (veliki kotači), a preostala dva su *caster* kotači (mali kotači). Ovakvu postojeću konfiguraciju najjednostavnije je prenamjeniti u diferencijalno pogonjenog mobilnog robota s kotačima. Jedina stvar koju je potrebno promijeniti, odnosno dodati, je pogonski sustav. Diferencijalni pogonski sustav zahtjeva da se svaki od glavnih (velikih) kotača pogoni zasebno, te je za tu svrhu potrebno na kolica ugraditi dva motora.

Percepcijski sustav ne postoji u osnovnoj izvedbi invalidskih kolica, što znači da ga je za potrebe prenamjene u mobilnog robota potrebno izgraditi „od nule“. Percepcijski sustav mora biti u stanju prikupljati podatke o robotovoj okolini, ali i o samom stanju robota. Iz tog razloga potrebno je odabrati barem jedan eksteroceptivni (vanjski) senzor i barem jedan proprioceptivni (unutarnji) senzor. Kao eksteroreceptivni senzor odabire se laserski daljinomjer, a kao proprioceptivni senzor odabire se IMU (engl. *Inertial Measurement Unit*).

Odabir komponenti lokomocijskog i percepcijskog sustava vrši se prema primjeru TurtleBot3 mobilnog robota. TurtleBot3 (Slika 3.1) je autnomni mobilni robot s diferencijalnim pogonom, te je u potpunosti implementiran u ROS-u. Spomenuta svojstva TurtleBot-a se u potpunosti poklapaju sa svojstvima koja su zadana u sklopu ovoga diplomskog zadatka, te je iz tog razloga odličan primjer na kojem se može bazirati prenamjena invalidskih kolica u mobilnog robota.



Slika 4.1 TurtleBot3, verzije Burger i Waffle Pi [29]

Kognicijski i navigacijski sustav implementirani su pomoću ROS-a, te je njihova implementacija opisana u poglavlju 4.

4.1 Odabir motora

Za diferencijalni pogon mobilnog robota potrebno je odabrati dva identična motora, po jedan za svaki pogonski kotač. Za tu potrebu odabire se servomotor Dynamixel MX-64.

Slika 4.2 Dynamixel MX-64 [30]

Servomotori su rotacijski aktuatori koji dozvoljavaju preciznu kontrolu kutne pozicije, brzine i ubrzanja. Pojam servomotor ne odnosi se na specifičnu vrstu elektromotora, već na cijeli sustav izgrađen oko kompatibilnog elektromotora. Sustav od kojeg se sastoji servomotor sadrži elektromotor, kontroler, te senzor koji daje podatke o poziciji.

Dynamixel MX-64 sastoji se od istosmjernog (DC) Maxon motora, ARM Cortex-M3 mikrokontrolera i AMS-ovog apsolutnog enkodera. Detaljnije specifikacije prikazane su u Tablici 3.1.

Tablica 4.1 Specifikacije Dynamixel MX-64 servomotora

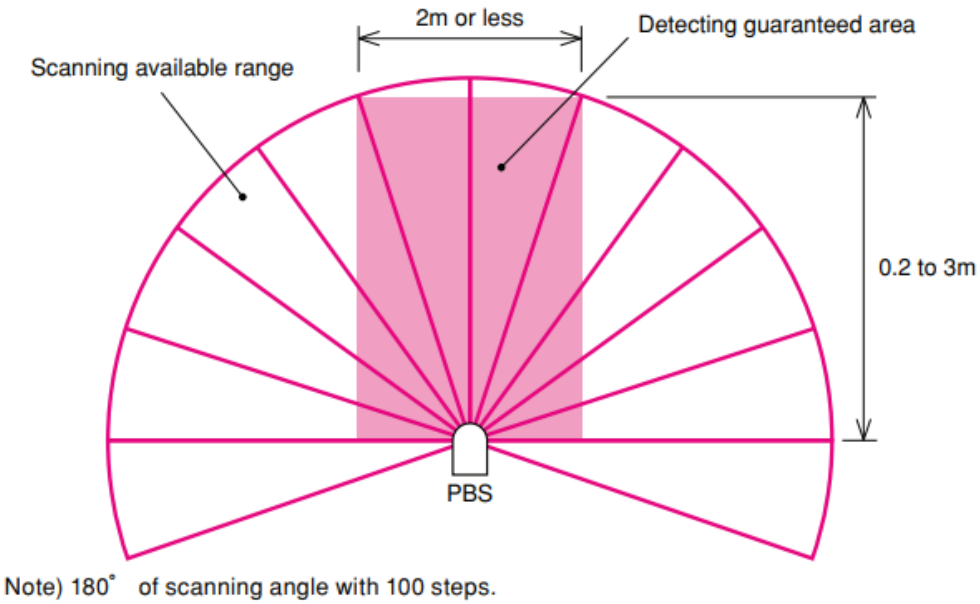
Veličina	Vrijednost
Razlučivost	4096 [puls/okr]
Nazivni napon	12 [V]
Brzina bez opterećenja	63 [min^{-1}]
Okretni moment	6 [Nm]
Prijenosni omjer	200:1
Dimenzije	40.2 x 61.1 x 41 [mm]
Masa	135 [g]

4.2 Odabir laserskog daljinomjera

Laserski daljinomjer je eksteroceptivni senzor, čija je uloga detekcija prepreka u robotovoj okolini. Princip rada i više detalja o senzoru opisani su u poglavlju 2.

Prilikom odabira laserskog daljinomjera za prenamjenu invalidskih kolica potrebno je, osim na specifikacije, pažnju obratiti na kompatibilnost senzora sa ROS sučeljem. Ukoliko laserski daljinomjer nije moguće implementirati unutar ROS-a, neće biti moguće ni operacije mapiranja i autonomnog navigiranja.

Za potrebe prenamjene invalidskih kolica odabire se Hokuyo PBS-03JN laserski daljinomjer. Ovaj laserski daljinomjer ima područje detekcije od 180° i domet od 3 metra. Treba napomenuti kako proizvođač garantira detektiranje objekata samo u području pravokutinka dimenzija 2 x 3 metra ispred senzora (Slika 3.3). Preostalo područje unutar dometa i dalje se skenira, no postoji mogućnost da objekti u tom području neće biti detektirani [31].



Slika 4.3 Domet PBS-03JN laserskog daljinomjera [3]

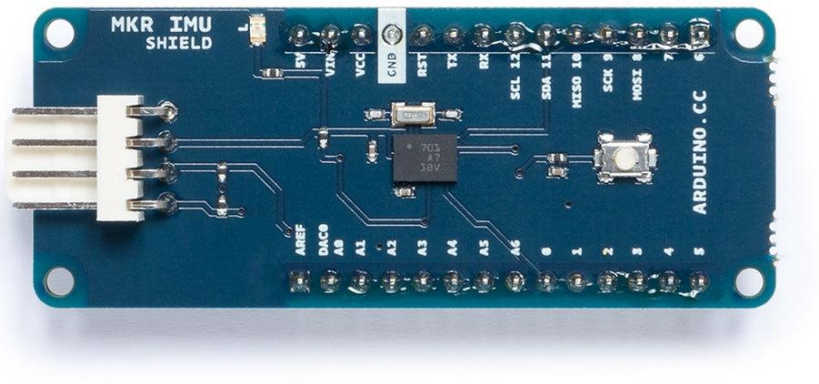
Hokuyo PBS-03JN, kao i svi drugi Hokuyo laserski daljinomjeri, ima ROS podršku u vidu Hokuyo ROS paketa. Ovaj paket omogućuje jednostavno prikupljanje podataka dobivenih laserskim mjerenjima, te pripremu za daljnju obradu unutar ROS-a. Također ovaj paket omogućava i stvaranje virtualnog laserskog daljinomjera za potrebe računalnih simulacija robotova modela unutar ROS-a.

Slika 4.4 Laserski daljinomjer Hokuyo PBS-03JN [31]

4.3 Odabir IMU-a

IMU je sustav, sastavljen od proprioceptivnih senzora, pomoću kojeg je moguće odrediti poziciju i orijentaciju robota. Budući da gibanja mobilnog robota mogu biti linearna i rotacijska, potrebni su posebni senzori za pojedinu vrstu gibanja. Za određivanje linearnog pomaka koriste se akcelerometri. Akcelerometri mjere linearnu akceleraciju robota, no računskim putem moguće je dobiti podatke o linearnom pomaku. Za određivanje kutnog pomaka i orijentacije koriste se žiroskopi [1]. Podatke o poziciji i orijentaciji također je moguće dobiti pomoću enkodera servomotora, no zbog postojanja odstupanja u mjerenjima i enkodera i IMU-a preporuča se korištenje oba sustava kako bi se dobili najtočniji mogući rezultati

Za potrebe prenamjene invalidskih kolica odabire se Arduino MKR IMU Shield, koji je opremljen sa tri akcelerometra i tri žiroskopa. Odabrani IMU može mjeriti pomake i akcleracije po šest osi (tri linearne i tri rotacijske). Također ovaj IMU kompatibilan je sa ROS-om i s dva najčešća upravljačka sustava za ovakve projekte, Arduinom i Raspebrry PI-jem [32].



Slika 4.5 Arduino MKR IMU Shield [32]

5 Autonomna navigacija pomoću ROS-a

Nakon odabira komponenti koje čine lokomocijski i percepcijski sustav mobilnog robota, potrebno je definirati kognicijski i navigacijski sustav kako bi robot imao mogućnost autonomne navigacije u prostoru. Ovi sustavi implementiraju se pomoću ROS-a, tj. pomoću pojedinih ROS paketa čijim povezivanjem i međudjelovanjem se ostvaruje mogućnost autonomne navigacije robota.

Implementacija autonomne navigacije pomoću ROS-a može se podijeliti na sljedeće korake:

1) Izrada URDF modela
2) Postavljanje Gazebo simulacije
3) Mapiranje prostora
4) Postavljanje AMCL-a
5) Postavljanje *Navigation* paketa

5.1 Izrada URDF modela

URDF (engl. *Universal Robot Description Format*) model predstavlja jednu ili više datoteka u XML formatu koje predstavljaju model robota. Za opisivanje modela koriste se razni *tag*-ovi koji modelu pridodaju odgovarajuća svojstva. Osnovni *tag*-ovi su *link* koji predstavlja fizički element robota i *joint* kojime se povezuju link-ovi. *Link* i *joint* označavaju samo vrstu, odnosno ulogu, pojedinog elementa modela. Za potpuno i ispravno definiranje modela robota potrebno je proširiti opise *link*-ova i *joint*-ova koristeći dodatne *tag*-ove.

URDF modeli najčešće predstavlja pojednostavljeni fizički model robota. Za prikaz se koriste jednostavni geometrijski oblici, te nije potrebno ulaziti u veliku razinu detalja pri izradi modela.

Prvi *link* koji se definira je „base_footprint“ (Slika 4.1). Ovaj *link* nema stvarni fizički značaj već se koristi kao glavni koordinatni sustav robota i kao veza sa vanjskim koordinatnim sustavima (odometrija i mapa). Jedini dodatni parametar koji je potrebno definirati za ovaj *link* je ishodište njegovog koordinatnog sustava, a taj parametar definira se pomoću *tag*-a *origin. Origin* definira poziciju ishodišta nekog koordinatnog sustava s obzirom na globalni sustav pomoću šest parametara: *x, y, z, r, p, y*. Parametri *x, y* i *z* predstavljaju translatorni pomak, po istomenim osima od ishodišta, dok parametri *r, p* i *y* (eng. *roll, pitch, yaw*) predstavljaju rotaciju oko glavnih osi koordinatnog sustava . Radi jednostavnosti preporuča se staviti glavni robotov koordinatni sustav u ishodište globalnog sustava.

```
<link name="base_footprint">
    <origin xyz="0 0 0" rpy="0 0 0" />
</link>
```

Slika 5.1 Definiranje "base_footprint" linka

Idući link koji se defnira je link nazavan „kostur“. Ovaj link predstavlja cjelokupnu strukturu kolica izuzev kotača. Za razliku od „base_footprint“ *link*-a, „kostur“ ima fizički značaj, te mu je zbog toga potrebno dodati fizikalna svojstva. Ovakva svojstva definiraju se unutar *tag*-a *collision* koji pomoću jednostavnih geometrijskih oblika opisuje područja robota koja mogu imati fizičku interakciju sa drugim objektima. Elementi definirani unutar *collision*-a ne moraju vjerno prikazivati stvarni izgled robota, već je dovoljno da prikažu cjelokupni prostor koji robot zauzima u prostoru. *Collision* svojstva *link*-a „kostur“ (Slika 4.2) definiraju se pomoću dva kvadra (eng. *box*) dimenzija 0,4 x 0,5 x 0,36 i 0,4 x 0,15 x 0,3 mm. U ovom slučaju *origin* tag opisuje poziciju definiranog elementa unutar koordinatnog sustava *link*-a. Koordinatni sustav *link*–a kostur definira se s obzirom na *link* „base_footprint“, a njegovo definiranje biti će opisano u nastavku poglavlja.

```
<link name="kostur">
    <collision>
        <origin xyz="0 0 -0.085" rpy="0 0 ${PI/2}" />
        <geometry>
            <box size="0.4 0.5 0.36"/>
        </geometry>
    </collision>
    <collision>
        <origin xyz="-0.3 0 0.25" rpy="0 0 ${PI/2}" />
            <geometry>
                <box size="0.4 0.15 0.3"/>
            </geometry>
    </collision>
```

Slika 5.2 Collision svojstva linka "kostur"

Ukoliko robota želimo prikazati i ispitati u nekom programu (npr. Gazebo) potrebno je definirati i njegovu vizualnu reprezentaciju. Vizualna svojstva robota (njegov izgled) definiraju se pomoću *tag*-a *visual*. URDF format unutar *tag*-a *visual* prihvaća .stl i .dae datoteke kao opis vizaulnih svojstava. Ova opcija nudi mogućnost prikaza robota pomoću mreža (engl. *mesh*), dobivenih pomoću nekog CAD programa, umjesto prikazivanja pomoću jednostavnih geometrijskih tijela (poput *collision*-a). Prilikom uvoza datatoteka u URDF format potrebno je pažnju posvetiti mjerilu, budući da je standardna mjerna jedinica u URDF formatu metar, a u većini CAD programa milimetar. Mjerilo se postavlja pomoću parametra *scale* koji je potrebno definirati za sve tri osi koordinatnog sustava.

```
<visual>
    <origin xyz="0 0.046 0" rpy=" ${PI/2} 0 ${PI/2} " />
    <geometry>
    <mesh filename="package://kolica_description/meshes/Kolica.stl" scale="0.001 0.001 0.001"/>
    </geometry>
</visual>
```

Slika 5.3 Visual svojstva linka "kostur"

Ukoliko se model robota želi testirati u nekom od simulatora potrebno je modelu dodati inercijalna svojstva. Inercijalna svojstva sastoje se od mase elementa i njegovih momenata tromosti. Momenti tromosti opisani su matricom

$$\begin{vmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{vmatrix}$$

Zbog simetričnosti matrice moguće je izostaviti elemente I_{zx}, I_{yx} i I_{zy}. Ova svojstva definiraju se unutar URDF formata pomoću *tag*-a *inertial* na način prikzan na Slici 4.4

```
<inertial>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <mass value="3.8"/>
    <inertia ixx="0.306" ixy="-0.016" ixz="0.007" iyy="0.276" iyz="0.111" izz="0.302"/>
</inertial>
```

Slika 5.4 Inertial svojstva linka "kostur"

Nakon opisivanja tijela robota, odnosno kolica, potrebno je još opisati kotače. Pogonski kotači opisani su pomoću *link*-ova „kotacDesni“ i „kotacLijevi“. Za svaki od kotača potrebno je definirati *collision*, *visual* i *inertial* svojstva, na isti način kako je napravljeno i za *link* „kostur“. Prikaz opisa desnog pogonskog kotača dan je na Slici 4.5, a lijevog na Slici 4.6.

```
<link name= 'kotacDesni'>
    <collision>
        <origin xyz="0 0 0" rpy="0 ${PI/2} ${PI/2}" />
        <geometry>
            <cylinder length="${kotaciSirina}" radius="${kotaciFi/2}"/>
        </geometry>
    </collision>

    <visual>
        <origin xyz="0 0 0" rpy="0 0 ${PI/2}" />
        <geometry>
            <mesh filename="package://kolica_description/meshes/KotacD.stl" scale="0.001 0.001 0.001"/>
        </geometry>
    </visual>

    <inertial>
        <origin xyz="0 0 0" rpy="0 ${PI/2} ${PI/2}" />
        <mass value="2.5"/>
        <inertia ixx="0.579" ixy="0" ixz="0" iyy="0.293" iyz="0" izz="0.293" />
    </inertial>
</link>
```

Slika 5.5 Opis linka "kotacDesni"

```
<link name= 'kotacLijevi'>
    <collision>
        <origin xyz="0 0 0" rpy="0 ${PI/2} ${PI/2}" />
        <geometry>
            <cylinder length="${kotaciSirina}" radius="${kotaciFi/2}"/>
        </geometry>
    </collision>

    <visual>
        <origin xyz="0 0 0" rpy="0 0 ${PI/2}" />
        <geometry>
            <mesh filename="package://kolica_description/meshes/Kotac.stl" scale="0.001 0.001 0.001"/>
        </geometry>
    </visual>

    <inertial>
        <origin xyz="0 0 0" rpy="0 ${PI/2} ${PI/2}" />
        <mass value="2.5"/>
        <inertia ixx="0.579" ixy="0" ixz="0" iyy="0.293" iyz="0" izz="0.293" />
    </inertial>
</link>
```

Slika 5.6 Opis linka "kotacLijevi"

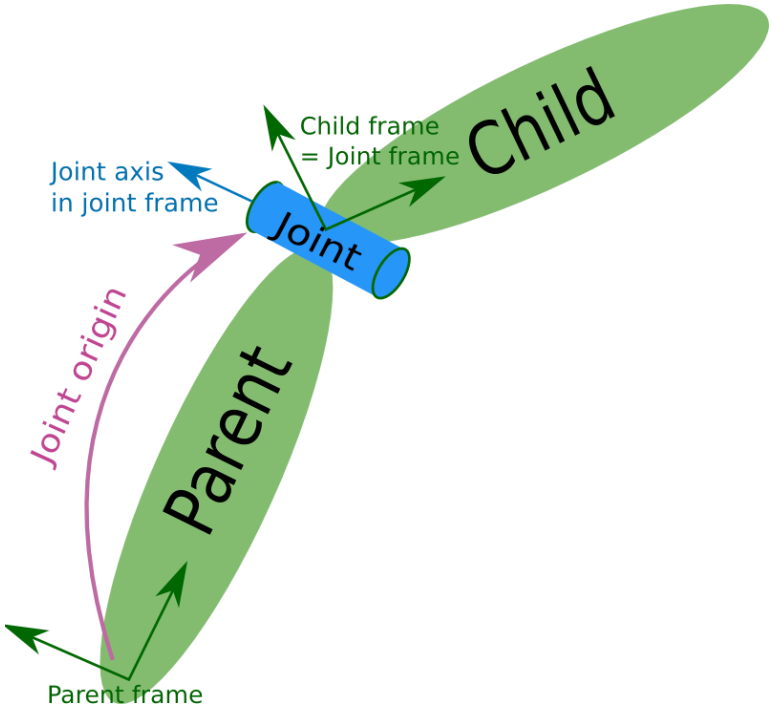
Prikaz pomoćnih kotača unutar URDF modela značajno je pojednostavljen. Prvo pojednostavljenje je smanjenje broja pomoćnih kotača. Umjesto dva bočna kotača postavlja se jedan centralni kotač. Ovo pojednostavljenje radi se kako bi se smanjio broj elemenata a samim tim i broj transformacija među koordinatnim sustavima. Drugo pojednostavljenje je korištenje sfere umjesto cilindra za reprezentaciju kotača. Ovo pojednostavljene napravljeno je iz razloga jednostavnijeg opisa sfere nego klasičnog *caster* kotača. Pomoćni kotač definiran je unutar *link*-a „kotacMali“, te je opisan pomoću *collision* i *inertial* svojstava (Slika 4.7). *Visual* svojstva su izostavljena budući da je vizualna reprezentacija pomoćnih kotača povučena iz 3D modela kolica unutar *link*-a „kostur“.

```
<link name = "kotacMali">
    <collision>
        <origin xyz="0 0 0" rpy=" 0 0 0" />
        <geometry>
            <sphere radius="0.05"/>
        </geometry>
    </collision>

    <inertial>
        <origin xyz="0 0 0" rpy=" 0 0 0" />
        <mass value="0.35"/>
        <sphere_inertia m="0.35" r="0.05" />
    </inertial>

</link>
```

Slika 5.7 Opis linka "kotacMali"

Nakon opisivanja kotača potrebno je još definirati elemente seznora robota. Kao i „base_footprint“ ovi elementima nije potrebno dodavati fizikalna ni vizualna svojstva. Elementi senzora unutar URDF modela potrebni su samo iz razloga kako bi čvorovi odgovorni za njihov rad znali njihovu poziciju na robotu i unutar radnog prostora.

Kada su sve pojedine komponente robota definirane u URDF modelu, potrebno ih je povezati pomoću *joint*-ova. Prilikom ovakvog povezivanje stvara se ne samo fizikalna povezanost komponenti već se povezuju i njihovi koordinatni sustavi. Za povezivanje i transformacije među koordinatnim sustavima odgovoran je ROS paket *tf*, koji je baziran na hijerarhijskog strukturi elemenata robota. Hijerarhijska struktura sastoji se od *parent* (nadređenog) i *child* (podređenog) člana. Pri povezivanju stoga treba paziti na pravilno definiranje *origin*-a, odnosno koordinata, *joint*-a, jer ovaj parametar određuje i poziciju *child* koordinatnog sustava u odnosu na *parent* koordinatni sustav [33]. Ovisnost koordinatnih sustava unutar definicije *joint*-a prikazana je na Slici 4.8.



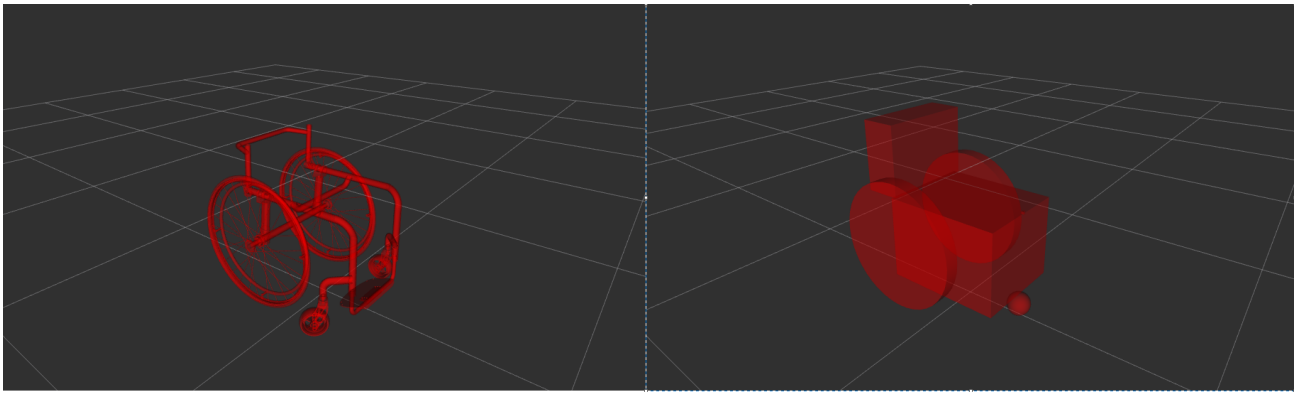
Slika 5.8 Definiranje jointa i koordinatnih sustava [33]

Za izradu URDF modela kolica koriste se dvije vrste *joint*-ova. Prva vrsta je fiksni (engl. *fixed*) *joint* koji nema nikakvih sloboda gibanja i koristi se za statičko povezivanje dvije komponente. Druga vrsta je rotacijski (engl. *continuous*) *joint* koji osim što povezuje komponente nudi i mogućnosti rotacije oko jedne od osi. Svi *joint*-ovi korišteni u izradi URDF modela kolica dani su u Tablici 4.1.

Tablica 5.1 Popis joint-ova modela

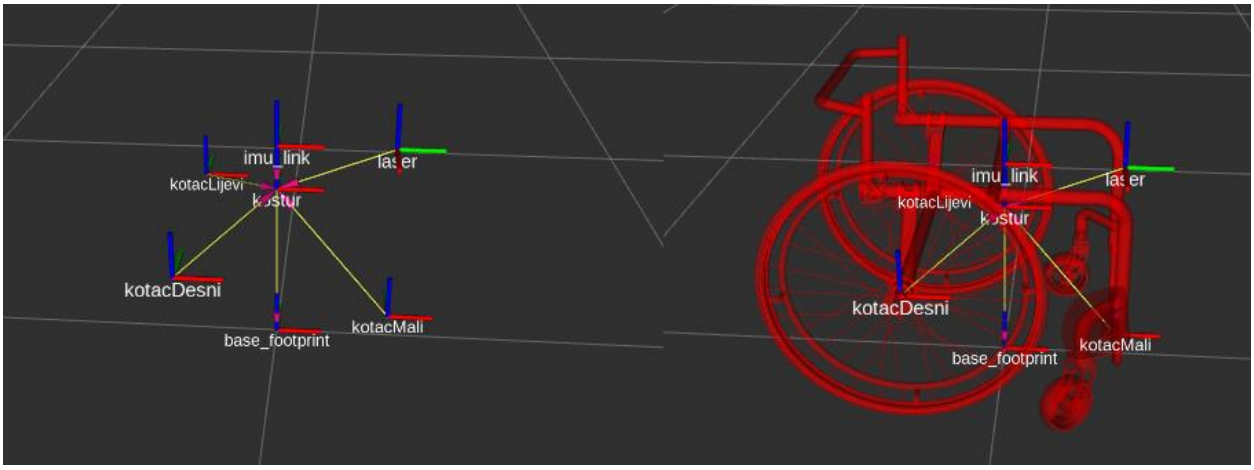
Naziv *joint*-a	**Vrsta *joint*-a**	***Parent***	***Child***	***Origin* (x y z r p y)**
base_joint	Fiksni	base_footprint	kostur	0 0 0,362 0 0 0
kostur_kotacDesni	Rotacijski	kostur	kotacDesni	-0,187 -0,23 -0,085 0 0 0
kostur_kotacLijevi	Rotacijski	kostur	kotacLijevi	-0,187 0,23 -0,085 0 0 0
kostur_kotacMali	Fiksni	kostur	kotacMali	0,25 0 -0,312 0 0 0
laser_joint	fiksni	kostur	laser	0,25 0 0,1 0 0 -π/2
imu_joint	fiksni	kostur	Imu_link	0 0 0,18 0 0 0

Definiranjem *joint*-ova dovršena je izrada URDF modela. Ispravnost modela može se prikazati pomoću ROS paketa Rviz. Rviz je program koji se koristi za vizualizaciju robota i podataka koje robot zaprima, odnosno može se reći da prikazuje robota onako kako robot sebe „vidi“. Za prikazivanje samog modela robota odgovorni su ROS čvorovi *joint_state_publisher* i *robot_state_publisher*. Prikaz robotova modela unutar rviza-a prikazan je na Slici 4.9, pri čemu je na lijevo prikaz vizualnog modela opisanog pomoću *visual* parametara, a na desno prikaz fizikalnog modela opisanog pomoću *collision* parametara.



Slika 5.9 Vizualizacija modela robota u rviz-u

Unutar rviz-a moguće je prikazati i podatke paketa *tf*, odnosno koordinatne sustave, njihovu povezanost i poziciju u prostoru. Također je moguće istovremeno prikzivati i robotov model i *tf* podatke.



Slika 5.10 Prikaz koordinatnih sustava pomoću rviz-a

5.2 Gazebo

Gazebo je simulacijsko okruženje unutar ROS-a koje omogućava relativno brzo i jednostavno razvijanje i ispitivanje robotskih modela. Gazebo sadržava brojne knjižnice pomoću kojih se vrše simulacije fizikalnih svojstva bitnih za gibanje mobilnog robota kroz prostor (npr. trenje i sudari). Gazebo također nudi implementaciju virtualnih senzora koji daju informacije o virtualnom prostoru kroz koji se robot kreće na isti način kako bi to radili i stvarni senzori u stvarnom prostoru.

Kako bi napravljeni URDF model ispravno radio unutar Gazebo simulacije potrebno je dodati fizikalna svojstva kotačima, dodati upravljač za diferencijalni pogon i definirati parametre senzora.

Fizikalna svojstva koja se dodaju kotačima su koeficijent trenja, krutost, koeficijent prigušenja, dubina penetracije i maksimalna korekcijska brzina kontakta. Za pravilno definiranje trenja potrebna su tri parametra: *mu1, mu2* i *fdir*. *Mu1* i *mu2* predstavljaju koeficijente trenja na glavnom pravcu kretanja i na njemu okomitom pravcu, dok je parametar *fdir* vektor koji definira glavni pravac kretanja. Krutost se definira pomoću parametra *kp*, te se u ovom slučaju za sva tri kotača postavlja velika vrijednost parametra kako bi se kotače tretiralo kao idealno krute. Faktor prigušenja definira se parametrom *kd*, te se on za sva tri kotača postavlja na malu vrijednost kako bi se kotače tretiralo kao idealno krute. Dubina penetracije je veličina unutar Gazebo simulatora kojom se definira pri kojoj dubini penetracije jednog tijela u drugo će se krenuti javljati reaktivne sile. Dubina penetracije postavlja se pomoću parametra *minDepth* i izražava se u metrima. Za potrebe ove specifične simulacije postavlja se mali *minDepth* kako bi se osigurao kontinuirani kontakt između kotača i površine i kako bi se pravilno definiralni potencijalni sudari robota i prepreka. Maksimalna korekcijska brzina kontakta je veličina koja definira maksimalnu brzinu reakcija nakon kolizije dvaju objekata, odnosno kojom se najvećom brzinom predmet smije „odbiti“ od drugog predmeta. Ova brina postavlja se pomoću parametra *maxVel* i izražava se u metrima po sekundi. Prikaz odabranih parametara za svaki kotač dan je na Slici 4.11.

```
<gazebo reference="kotacDesni">
  <mu1>0.5</mu1>
  <mu2>0.5</mu2>
  <kp>500000.0</kp>
  <kd>10.0</kd>
  <minDepth>0.001</minDepth>
  <maxVel>0.1</maxVel>
  <fdir1>0 1 0</fdir1>
</gazebo>

<gazebo reference="kotacLijevi">
  <mu1>0.5</mu1>
  <mu2>0.5</mu2>
  <kp>500000.0</kp>
  <kd>10.0</kd>
  <minDepth>0.001</minDepth>
  <maxVel>0.1</maxVel>
  <fdir1>0 1 0</fdir1>
</gazebo>

<gazebo reference="kotacMali">
  <mu1>0.1</mu1>
  <mu2>0.1</mu2>
  <kp>1000000.0</kp>
  <kd>100.0</kd>
  <minDepth>0.001</minDepth>
  <maxVel>0.1</maxVel>
</gazebo>
```

Slika 5.11 Fizikalni Gazebo parametri kotača

Kako bi se robot mogao kretati potrebno je dodati Gazebo *plugin* koji će upravljati motorima, na temelju uputa koje korisnik zadaje. U tu svrhu koristi se *differential_drive_controller* koji je namijenjen isključivo za upravljanje robotima sa diferencijalnim pogonom. Za ispravan rad ovog *plugin*-a potrebeno je definirati *joint*-ove pogonskih kotača, promjer i razmak pogonskih kotača, glavni koordinatni sustav robota, te ROS teme gdje se objavljuju odometrija i upute motorima. Parametri vezani za pogonske kotače daju informacije kojim *joint*-ovima se treba upravljati, te donekle vrše i ulogu enkodera budući da se podaci u promjeru i razmaku pogonskih kotača koriste za odometriju. Korištenjem ovog *plugin*-a automatski se stvara veza između glavnog koordinatnog sustava robota i koordinatnog sustava odometrije. Transformacije između ova dva sustava bitne su za lokalizaciju, odnosno određivanje robotovog položaja u prostoru, te je iz tog razloga bitno ispravno definirati ove koordinatne sustave. ROS tema za objavljivanje uputa bitna je u komunikaciji sa različitim ROS čvorovima koji se koriste za upravljanje robotom. Ova tema se može imenovati bilo kako dok god je unutar svih povezanih čvorova imenovana isto. Preporuka je ostaviti standardni naziv *cmd_vel*. Odabrani parametri za *differential_drive_controller* prikazani su na Slici 4.12.

```
<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <legacyMode>false</legacyMode>
    <alwaysOn>true</alwaysOn>
    <updateRate>20</updateRate>
    <leftJoint>kostur_kotacLijevi</leftJoint>
    <rightJoint>kostur_kotacDesni</rightJoint>
    <wheelSeparation>0.46</wheelSeparation>
    <wheelDiameter>0.55</wheelDiameter>
    <torque>3</torque>
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <odometrySource>world</odometrySource>
    <robotBaseFrame>base_footprint</robotBaseFrame>
    <publishWheelJointState>true</publishWheelJointState>

  </plugin>
</gazebo>
```

Slika 5.12 differential_drive_controller parametri

Implementacijom *differential_drive_controllera* omogućeno je kretanje robota u prostoru, no kako bi se omogućila autonomna navigacija potrebno je implementirati i senzore opisane u poglavlju 3.

Laserski daljinomjer se implementira pomoću *head_hokuyo_sensor plugin*-a koji napravljen kako bi simulirao rad Hokuyo senzora unutar virtualne okoline. Kako bi laser ispravno funkcionirao unutar ROS okruženja potrebno je definirati koji *link* na URDF modelu predstavlja laser i temu u koju će se dobiveni podaci objavljivati. Moguće je mijenjati razne parametre virtualnog lasera, poput dometa, razlučivosti, itd., kako bi njegov rad što bolje opisivao rad stvarnog lasera. Također moguće je dodati i umjetno stvorenu buku u očitanje laserskog daljinomjera. Uključivanjem ili isključivanjem buke stvara se razlika između realnog i idealnog modela lasera unutar Gazebo-a. Prikaz parametara za odabrani laserski daljinomjer prikazan je na Slici 4.13, a detaljan popis i opis parametara može se pronaći u [34].

IMU se implementira na sličan način poput laserskog daljinomjera. Za implementaciju se koristi *plugin imu_plugin*, za čiji je ispravan rad potrebno definirati *link* koji predstavlja IMU i imena ROS usluge i teme u koju se podaci objavljuju. Osim navedenog potrebno je još definirati buku kako bi se rezultati što vjernije simulirali unutar Gazebo-a. Implementacija IMU-a prikazana je na Slici 4.14.

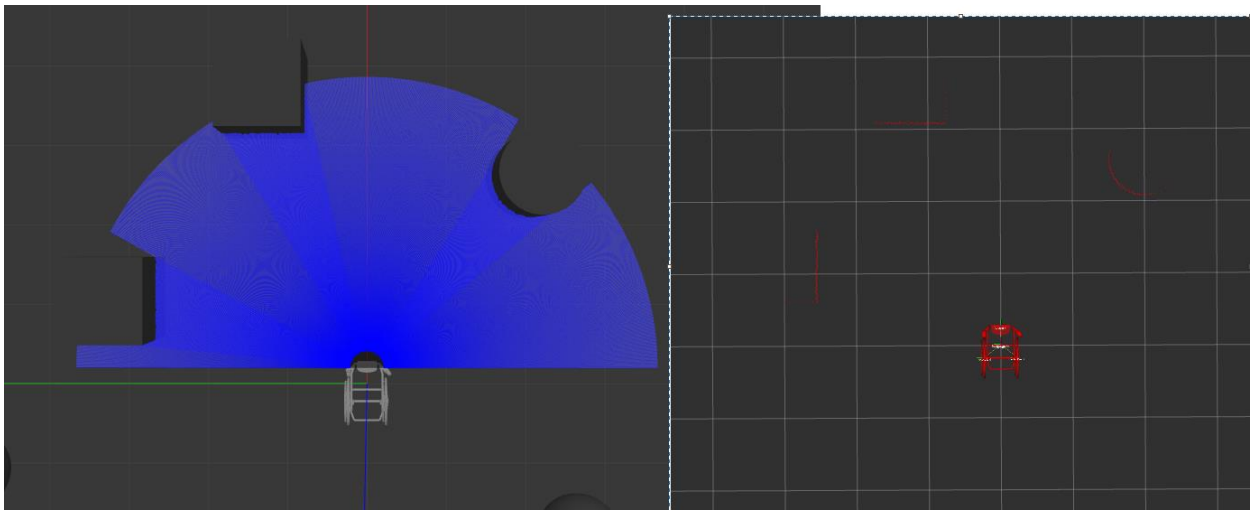
```
<gazebo reference="laser">
    <sensor type="ray" name="head_hokuyo_sensor">
      <pose>0 0 0 0 0 0</pose>
      <visualize>true</visualize>
      <update_rate>40</update_rate>
      <ray>
        <scan>
          <horizontal>
            <samples>720</samples>
            <resolution>1</resolution>
            <min_angle>0</min_angle>
            <max_angle>3.14</max_angle>
          </horizontal>
        </scan>
        <range>
          <min>0.2</min>
          <max>3.5</max>
          <resolution>0.01</resolution>
        </range>
        <noise>
          <type>gaussian</type>

          <mean>0.0</mean>
          <stddev>0.01</stddev>
        </noise>
      </ray>
      <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
        <topicName>/scan</topicName>
        <frameName>laser</frameName>
      </plugin>
    </sensor>
  </gazebo>
```

Slika 5.13 Parametri laserskog daljinomjera

```
<gazebo>
    <plugin name="imu_plugin" filename="libgazebo_ros_imu.so">
      <alwaysOn>true</alwaysOn>
      <bodyName>imu_link</bodyName>
      <frameName>imu_link</frameName>
      <topicName>imu</topicName>
      <serviceName>imu_service</serviceName>
      <gaussianNoise>0.0</gaussianNoise>
      <updateRate>0</updateRate>
      <imu>
        <noise>
          <type>gaussian</type>
          <rate>
            <mean>0.0</mean>
            <stddev>2e-4</stddev>
            <bias_mean>0.0000075</bias_mean>
            <bias_stddev>0.0000008</bias_stddev>
          </rate>
          <accel>
            <mean>0.0</mean>
            <stddev>1.7e-2</stddev>
            <bias_mean>0.1</bias_mean>
            <bias_stddev>0.001</bias_stddev>
          </accel>
        </noise>
      </imu>
    </plugin>
  </gazebo>
```

Slika 5.14 Implementacija IMU-a

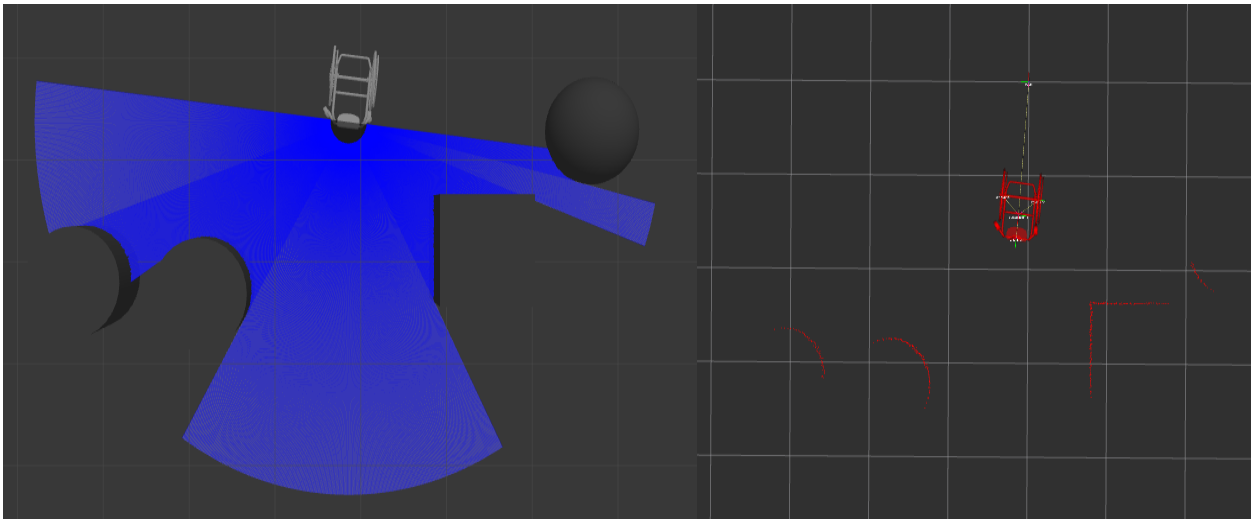
Implementacijom senzora model je u potpunosti pripremljen za Gazebo simulator. Prije nego što se krene u implementaciju autonomne navigacije potrebno je provjeriti ispravnost napravljenog modela. Provjera se vrši istovremenim pokretanjem Gazebo simulatora i programa za vizualizaciju rviz. Prikaz u Gazebo-u (Slika 4.15 lijevo) prikazuje simuliranu okolinu i robotovo ponašanje unutar te okoline. Prikaz u rviz-u (Slika 4.15 desno) prikazuje kako robot doživljava sebe i svoju okolinu. Ako je model dobro napravljen ova dva prikaza bi trebala biti jednaka, kao što je situacija na Slici 4.15.



Slika 5.15 Usporedni prikaz početnog stanja u Gazebo-u i rviz-u

Slika 4.15 pokazuje početno stanje u oba programa. Iz početnog stanja može se zaključiti ispravan rad senzora i odnosa glavnog i odometrijskog koordinatnog sustava. Robot se nalazi na istoj poziciji unutar oba prikaza, a očitanja laserskog daljinomjera dobro su prenesena u rviz.

Nakon provjere ispravnosti senzora potrebno je još provjeriti ispravnost upravljanja diferencijalnim pogonom. Ova provjera se vrši pomoću ROS čvora *teleop_twist_keyboard* pomoću kojeg je moguća teleoperacija robotom. Ovaj čvor omogućuje zadavanje komandi pomoću tipkovnice. Komande se objavljuju u temu *cmd_vel* na koju je kontroler motora pretplaćen i na taj način se ostvaruje gibanje robota kroz prostor. Moguće je i ručno unosti komande u temu *cmd_vel*, no korištenje čvora za teleoperaciju znatno olakšava upravljanje. Slika 4.16. prikazuje poziciju robota u Gazebo-u (lijevo) i rviz-u (desno) nakon pokretanja pomoću teleoperacije. Gibanje robota se pravilno odvijalo u oba programa, te je i konačna pozicija identična u oba programa.



Slika 5.16 Usporedni prikaz stanja nakon teleoperacije u Gazebo-u i rviz-u

5.3 Mapiranje

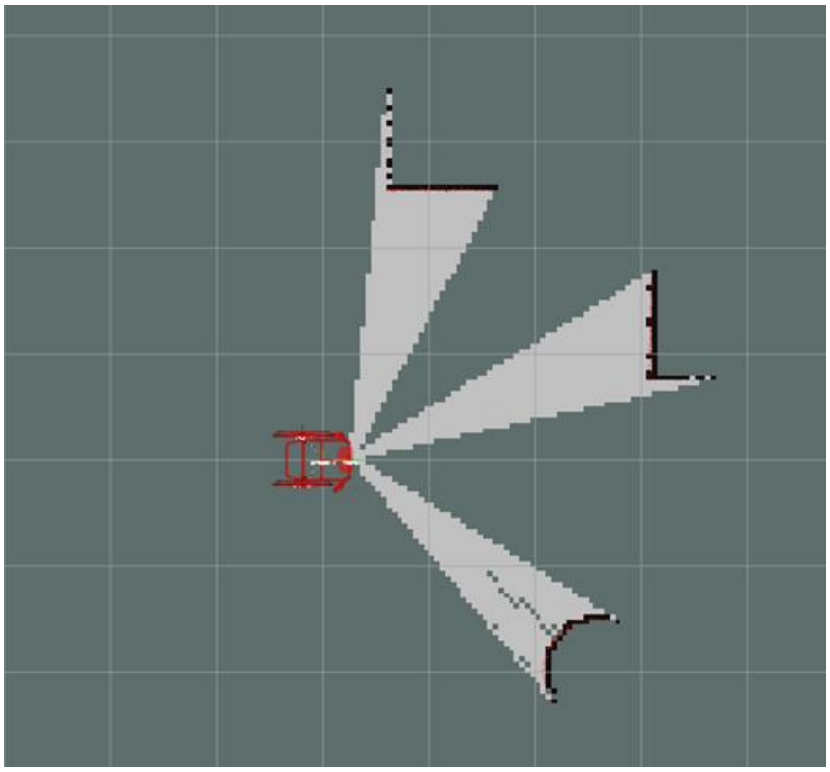
Nakon osposobljavanja diferencijalnog pogona i senzora moguće je krenuti u proces mapiranja, odnosno izrade mape robotovog okruženja. U ovu svrhu koristi se *gmapping* paket, konkretnije *slam_gmapping* čvor.

SLAM (eng. Simultaneous Localization And Mapping) je način mapiranja koji se koristi za mapiranje prije nepoznatih prostora uz istovremeno odvijanje lokalizacije. Ovaj proces može se odvijati potpuno autonomno od strane robota, ali pouzdanija i manje zahtjevna metoda je mapiranje pomoću teleoperacije robota. Za potrebe teleoperacije koristiti će se već spomenuti čvor *teleop_twist_keyboard*.

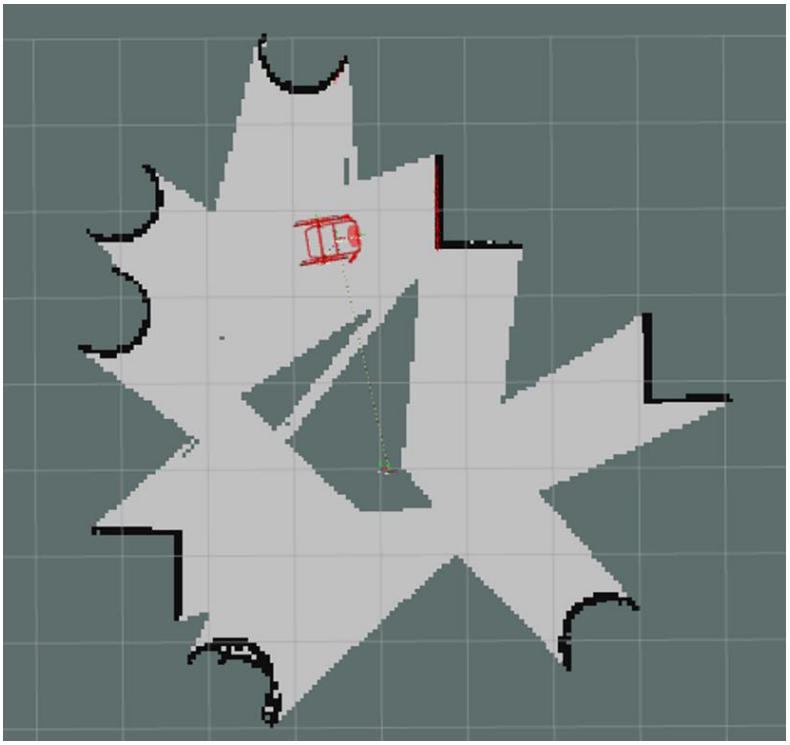
Za ispravan rad *gmapping*-a potrebno je pravilno definirati glavni koordinatni sustav robota, odnosno potrebno je čvoru dati informaciju uz koji *link* je vezan glavni koordinatni sustav robota. Osim *gmapping*-a potrebno je pokrenuti Gazebo, koji simulira okolinu za mapiranje, i rviz, pomoću kojeg je moguće gledati i kontrolirati kvalitetu mape u nastajanju.

Slika 4.17 prikazuje početak procesa mapiranja. Prije pokretanja, robot „vidi“ 3 prepreke i mjeri udaljenost do njih. Prostor, kojim putuju laserske zrake, između prepreke i robota se potom označava kao poznat prostor. Unutar rviz-a poznavanje prostora se označava bojama. Tako je tamnija siva nepoznat prostor, svijetlija siva je poznat prostor, a crna boja predstavlja prepreke.

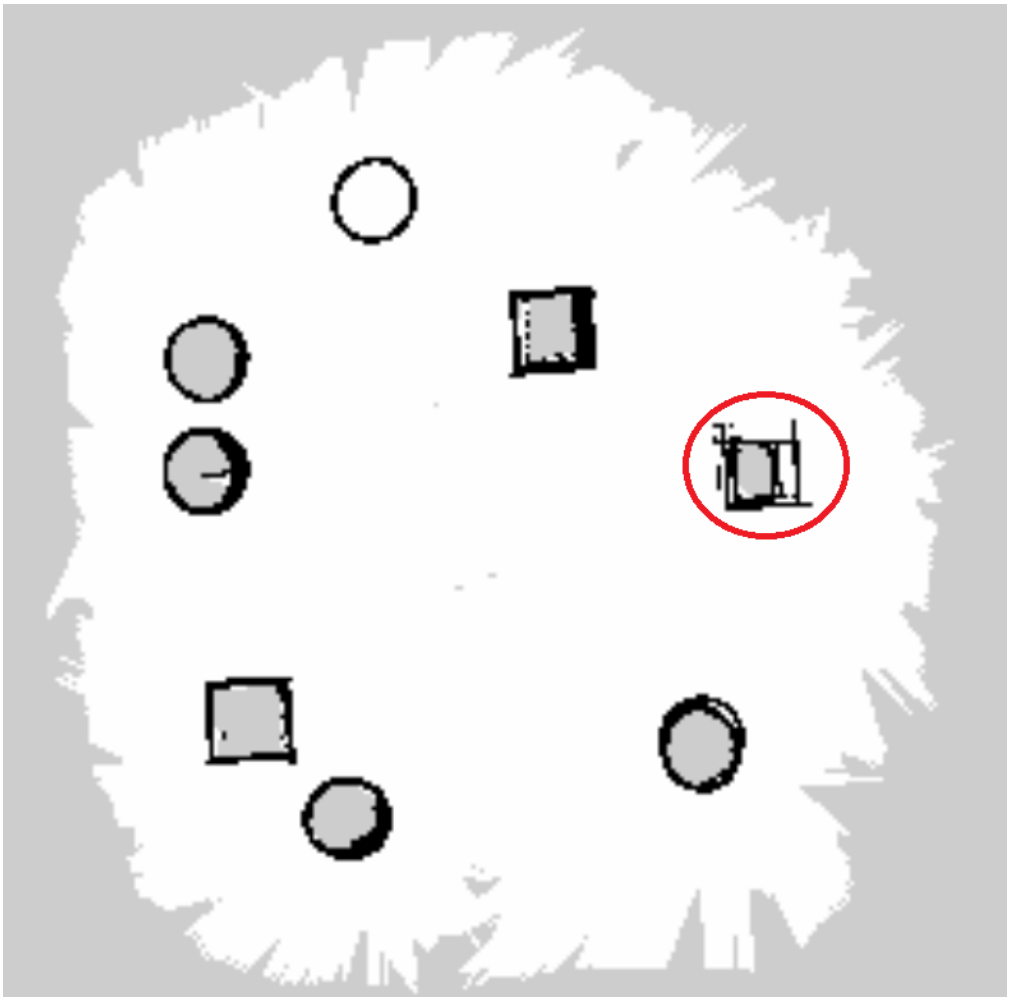
Kako bi se dobila kvalitetna mapa potrebno je temeljito obići prostor sa robotom, ponekad i više puta. Također preporuča se obilaženje prepreke sa svih strana kako bi se dobile što kvalitetnije informacije o njihovoj veličini i poziciji. Slika 4.18 pokazuje mapu nakon kraćeg prolaska robota s jedne strane prepreka, a Slika 4.19 prikazuje gotovu mapu prostora koja će se koristiti prilikom autonomne navigacije.



Slika 5.17 Početak mapiranja



Slika 5.18 Parcijalna mapa prostora

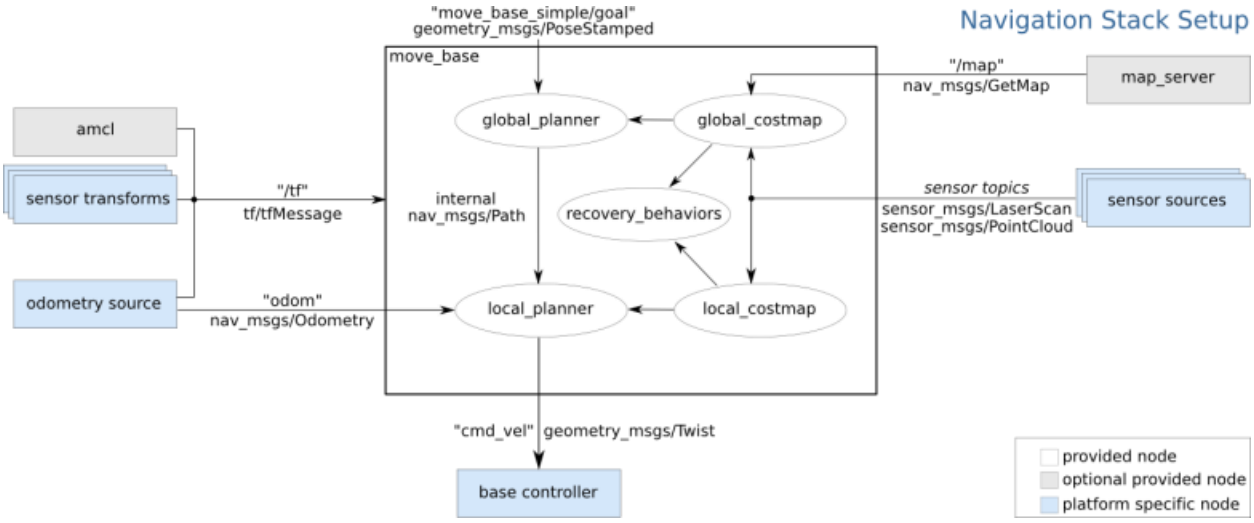


Slika 5.19 Mapa prostora

Procesom mapiranja uspješno su zabilježene sve prepreke postavljene u robotovo okruženje. Prepreka označena crveno na Slici 4.19 je izobličena iz razloga što je bila prva prepreka koju se mapiralo sa svih strana. U tom trenutku lokalizacija nije bila u potpunosti stabilna zbog manjka poznatih „orijentira“ u prostoru. Očitanja senzora su pravilno protumačena i zabilježena, no robot u tom trenutku nije u potpunosti točno znao svoju poziciju.

5.4 Navigacijski stog

Autonomno kretanje podrazumijeva robotovo samostalno kretanje u prostoru od polazene do zadane točke uz zaobilaženje prepreka na putu. Kako bi robot mogao izvršiti ovakav zadatak potrebno je pravilno postaviti navigacijski sustav koji je odgovoran za planiranje optimalne putanje i slijeđenje te iste putanje do odredišta. Navigacijski sustav implementirat će se pomoću ROS navigacijskog stoga (engl. *navigation stack*) čija je struktura prikazana na Slici 4.20.



Slika 5.20 Struktura ROS navigacijskog stoga [35]

Središnji dio navigacijskog stoga je *move_base* čvor. *Move_base* odgovoran je za stvaranje planova, odnosno ruta do zadanoga cilja. Razlikuju se dvije vrste planera unutar čvora, globalni i lokalni. Globalni planer odgovoran je za kreiranje rute od robotove trenutne pozicije do zadanog cilja. Lokalni planer na temelju rute dobivene iz globalnog planera kreira set uputa za kontrolu aktuatora, sa svrhom praćenja dobivene rute. Lokalni planer također može modificirati rutu, odnosno odstupati od rute globalnog planera, ukoliko se pokaže da je to optimalniji način za doći do cilja. Planeri svoje odluke donose na temelju podataka iz *costmap*-a, mapa prostora podijeljenih u manje ćelije kojima se dodjeljuju težinski faktori. Globalna *costmap*-a uzima u obzir cjelokupni mapirani prostor, dok lokalna *costmap*-a uzima u obzir samo komad mapiranog prostora koji okružuje robota.

Parametri vezani uz *move_base* čvor podešavaju su unutar YAML datoteka. Postoje sveukupno tri datoteke koje definiraju parametre *costmap*-a: *common_costmap*, *global_costmap* i *local_costmap*. U *common_costmap* datoteci definirani su robotov otisak (engl. *footprint*) i glavni koordinatni sustav, razlučivost mape i tema pod kojom se mapa objavljuje, te podaci o vrsti senzora i tema pod kojom se objavljuju podaci dobiveni senzorima (Slika 4.21). Navedeni parametri vrijede za obje *costmap*-e, a parametri specifični za pojedinu *costmap*-u definiraju se u posebnim datotekama (Slika 4.22 i 4.23). Parametri koji se definiraju za pojedinu *costmap*-u su referentni koordinatni sustav i *plugin*-ovi. *Plugin*-ovi koji se koriste vezani su za slojeve *costmap*-e. Koriste se tri glavna sloja: statički, inflacijski i sloj prepreka. Inflacijski sloj koristi se u obje *costmap*-e i pomoću njega se prividno uvećavaju prepreke kako bi se osigurala dovoljna udaljenost robota od prepreke. Statički sloj koristi se u globalnoj *costmap*-i i vezan je uz kartu dobivenu procesom mapiranja (poglavlje 4.3). Sloj prepreka koristi se u lokalnoj *costmap*-i i vezan je uz očitanja senzora, glavna uloga mu je uočavanja dinamičkih prepreka i prepreka koje nisu zabilježene na statičkoj mapi.

```yaml
footprint: [[-0.33, -0.2], [-0.33, 0.2], [0.33, 0.23], [0.33, -0.2]]
footprint_padding: 0.01

robot_base_frame: base_footprint
update_frequency: 2
publish_frequency: 1
transform_tolerance: 0.8

resolution: 0.05

obstacle_range: 2.5
raytrace_range: 2.0

#layer definitions
static:
    map_topic: /map
    subscribe_to_updates: true

obstacles_laser:
    observation_sources: laser
    laser: {data_type: LaserScan, clearing: true, marking: true, topic: scan, inf_is_valid: true}

inflation_radius: 0.55
```

Slika 5.21 common_costmap YAML datoteka

```yaml
global_frame: map
rolling_window: false
track_unknown_space: true

plugins:
  - {name: static, type: "costmap_2d::StaticLayer"}
  - {name: inflation_g, type: "costmap_2d::InflationLayer"}
```

Slika 5.22 global_costmap YAML datoteka

```yaml
global_frame: odom
rolling_window: true
width: 5
height: 5

plugins:
  - {name: obstacles_laser, type: "costmap_2d::ObstacleLayer"}
  - {name: inflation_l, type: "costmap_2d::InflationLayer"}
```

Slika 5.23 local_costmap YAML datoteka

Posljednja YAML datoteka definira parametre planera. Unutar *planner* datoteke definiraju se parametri poput maksimalne i minimalne dopuštene brzine robota, maksimalnog linearnog i kutnog ubrzanja, dopuštenog odstupanja od zadanog cilja i težinskih faktora za ocjenjivanje rute. Na temelju parametara unutar ove datoteke određuje se optimalna putanja, s obzirom na dodijeljene težinske faktore, te se ujedno generiraju i komande za pokretanje motora koje se potom prosljeđuju kontroleru putem teme *cmd_vel*. Prikaz odabranih parametara dan je na Slici 4.24, a detaljan popis parametara i opis njihove funkcije može se pronaći u [35].

```
# Robot Configuration Parameters
  max_vel_x: 0.2
  min_vel_x: 0.0
  max_vel_y: 0.0
  min_vel_y: 0.0

# The velocity when robot is moving in a straight line
  max_vel_trans:  0.22
  min_vel_trans:  0.11
  max_vel_theta: 0.4
  min_vel_theta: 0.1
  acc_lim_x: 2
  acc_lim_y: 0.0
  acc_lim_theta: 2

# Goal Tolerance Parametes
  xy_goal_tolerance: 0.2
  yaw_goal_tolerance: 0.2
  latch_xy_goal_tolerance: false

# Forward Simulation Parameters
  sim_time: 1.7
  vx_samples: 20
  vy_samples: 0
  vth_samples: 40
  controller_frequency: 10.0

# Trajectory Scoring Parameters
  path_distance_bias: 32.0
  goal_distance_bias: 20.0
  occdist_scale: 0.05
  forward_point_distance: 0.325
  stop_time_buffer: 0.2
  scaling_speed: 0.25
  max_scaling_factor: 0.2
```

Slika 5.24 planner YAML datoteka

Kako bi *move_base* čvor ispravno radio potrebni su mu podaci iz drugih čvorova. Popis potrebnih podataka zajedno sa listom čvorova koji ih objavljuju i tema preko kojih su objavljivani dan je u Tablici 4.2.

Tablica 5.2 Popis ulaznih podataka u move_base čvor

Podatak	Čvor	Tema
Transformacije između koordinatnih sustava	tf	tf
Izvor odometrije	Gazebo	odom
Mapa	map_server	map
Podaci senzora	Gazebo	scan

Svi podaci i pripadajući čvorovi iz Tablice 4.2 implementirani su u ranijim poglavljima ovoga diplomskog rada. Zadnji korak pri implementaciji autonomne navigacije je osposobljavanje AMCL (engl. *Adaptive Monte Carlo Localization*) čvora. AMCL se koristi za lokalizaciju robota, te je njegova upotreba unutar navigacijskog stoga neobavezna budući da se lokalizacija može vršiti i pomoću odometrije.

AMCL je metoda lokalizacije koja koristi podatke senzora i odometrije za određivanje trenutnog robotovog položaja i orijentacije na danoj mapi. Ova lokalizacijska metoda koristi čestični filter (engl. *particle filter*) u kojem svaka čestica predstavlja mogući položaj i orijentaciju robota. Prilikom pomaka robota podaci dobiveni senzorima i odometrijom se mijenjaju, te na temelju toga algoritam stvara nove pretpostavke (čestice) o poziciji i orijentaciji. Ovaj proces se odvija iteracijski odnosno nakon svakog, algoritmom određenog, iznosa pomaka vrši se nova procjena pozicije robota. Metoda se zove adaptacijska iz razloga što se kroz iteracije smanjuje broj čestice u filteru, tj. algoritam se adaptira na novu situaciju [36].

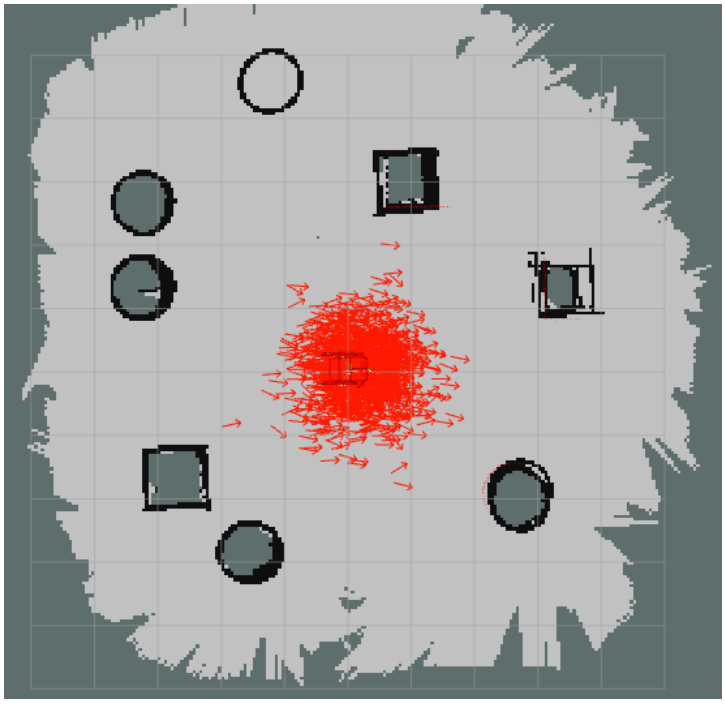
AMCL je u potpunosti implementiran u ROS pomoću čvora *amcl*. Prilikom postavljanja parametara *amcl*-a bitno je pravilno definirati odometrijski koordinatni sustavi, robotov glavni koordinatni sustav i vrstu pogona. Također čvor nudi opciju podešavanja tolerancija i očekivanih grešaka odometrijskih i senzorskih mjerenja. Korišteni parametri prikazani su na Slici 4.25, a potpuni popis parametara sa detaljnim objašnjenjima može se pronaći u [36].

```
<node pkg="amcl" type="amcl" name="amcl">
    <param name="odom_model_type" value="diff"/>
    <param name="odom_alpha5" value="0.1"/>
    <param name="transform_tolerance" value="0.5" />
    <param name="gui_publish_rate" value="10.0"/>
    <param name="laser_max_beams" value="30"/>
    <param name="min_particles" value="500"/>
    <param name="max_particles" value="2000"/>
    <param name="kld_err" value="0.05"/>
    <param name="kld_z" value="0.99"/>
    <param name="odom_alpha1" value="0.2"/>
    <param name="odom_alpha2" value="0.2"/>
    <!-- translation std dev, m -->
    <param name="odom_alpha3" value="0.8"/>
    <param name="odom_alpha4" value="0.2"/>
    <param name="laser_z_hit" value="0.5"/>
    <param name="laser_z_short" value="0.05"/>
    <param name="laser_z_max" value="0.05"/>
    <param name="laser_z_rand" value="0.5"/>
    <param name="laser_sigma_hit" value="0.2"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_model_type" value="likelihood_field"/>
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d" value="0.1"/>
    <param name="update_min_a" value="0.02"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="base_frame_id" value="base_footprint" />
    <param name="resample_interval" value="1"/>
    <param name="recovery_alpha_slow" value="0.0"/>
    <param name="recovery_alpha_fast" value="0.0"/>
</node>
```

Slika 5.25 Parametri amcl čvora

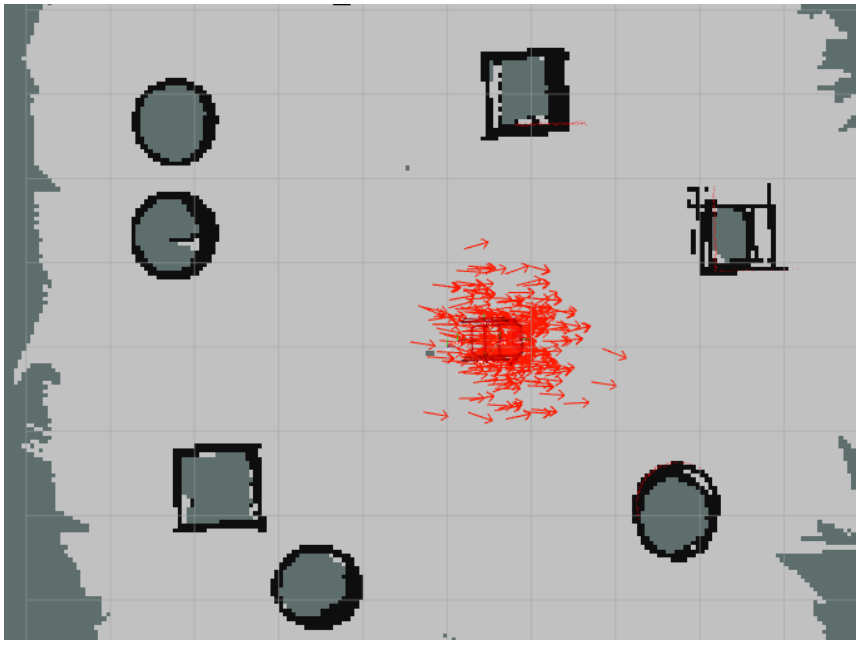
Provjeru ispravnosti AMCL-a najlakše je napraviti pomoću teleoperacije robota. Provjera se sastoji od pokretanja robota putem čvora *teleop_twist_keyboard* unutar Gazebo simulatora i praćenja rada čestičnog filtera pomoću vizualizacije u rviz-u. Potrebno je provjeriti da li filter konvergira, odnsono da li se kroz iteracije smanjuje broj čestica i da li čestice dobro pretpostavljaju poziciju robota.

Prikaz početne pozicije robota zajedno sa česticama, vizualiziranih kao crvene strelice, dan je na Slici 4.26. Svaka strelica i njena orijentacija označava moguću poziciju i orijentaciju robota. Grupiranost čestica kao na Slici 4.26 znak je dobro postavljenog AMCL-a, pogotovo ako se nakupina čestica okvirno podudara sa stvarnom pozicijom robota. Ukoliko se početna pretpostavljena pozicija dovoljno ne podudara sa stvarnom, moguće je pomoću rviz sučelja i opcije *2D Pose Estimate* ručno unesti novu poziciju oko koje će AMCL grupirati čestice.



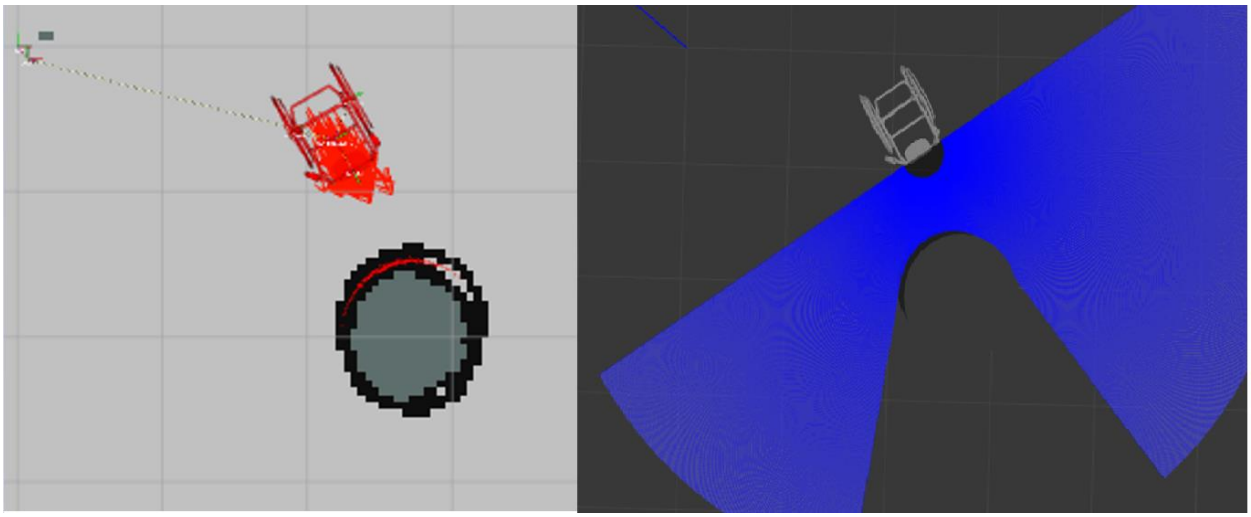
Slika 5.26 AMCL test – početna pozicija

Slika 4.27 pokazuje stanje AMCL-a nakon kratkog pomaka robota. Vidi se znatno smanjenje u broju čestica, što je znak dobre adaptivnosti algoritma. Također čestice su dobro grupirane oko stvarne pozicije robota i većina njih je orijentirana u dobrom smjeru.



Slika 5.27 AMCL test – stanje nakon kratkog pomaka

Slika 4.28 prikazuje konvergirano stanje AMCL-a i stvarnu poziciju robota unutar Gazebo simulatora. AMCL je dobro konvergirao uz znatno smanjenje broja čestice, te se pretpostavljena pozicija i stvarna pozicija podudaraju. Na temelju ovoga može se zaključiti kako je implementacija AMCL-a uspješno izvedena i da je robot spreman za autonomnu navigaciju.

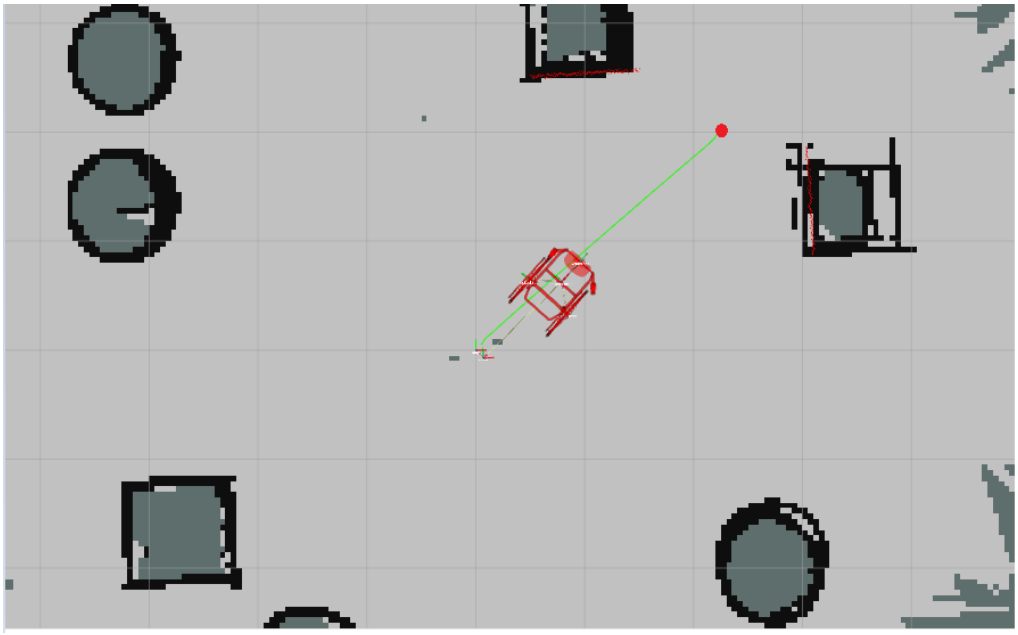


Slika 5.28 AMCL test - konačna pozicija

5.5 Autonomna navigacija

Postavljanje navigacijskog stoga sa AMCL-om zadnji je korak potreban za autonomno navigiranje robota po zadanoj mapi. Kao i kod prethodnih primjera kretanje robota odvija se unutar Gazebo simulatora uz praćanje pomoću vizualizacija unutar rviz-a. Preko rviz korisničkog sučelja omogućeno je jednostavno zadavanje cilja robotu putem opcije *2D Nav Goal*. Za potrebe ovog diplomskog rada zadavanje cilja i praćenje kretanja robota napravljeno je više puta, no zbog sažetosti rada biti će prikazana samo dva primjera.

Prvi primjer prikazan je na slici 4.29. i prikazuje gibanje do cilja (crvena točka) između dvije kvadratne prepreke. U ovom primjeru nije potrebno zaobilaziti nikakve prepreke, te planer iz tog razloga ucrtava direktnu rutu (zelena crta) od ishodišta do cilja. Gibanje za ostvarivanje rute uključuje jedan zakret i daljnje pravocrtno gibanje. Robot uspješno prati rutu uz manja odstupanja prilikom samog početka gibanja, odnosno za vrijeme početnog okreta. Praćenjem rute robot dolazi na zadanu poziciju unutar granica tolerancije propisanih parametrima planera.



Slika 5.29 Autonomana navigacija – zakret i pravocrtno gibanje

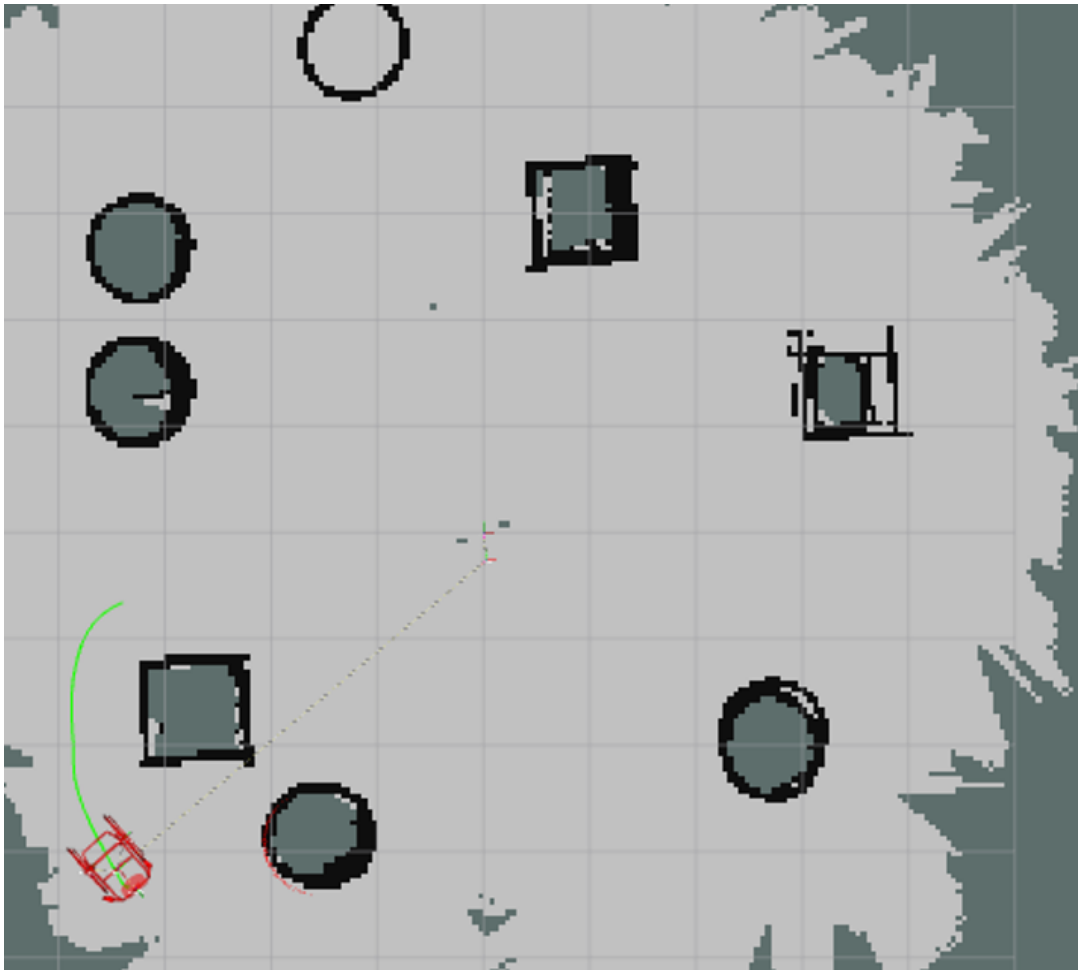
U drugom primjeru zadaje se cilj do kojeg je najbliža, direktna ruta blokirana preprekama. Slika 4.30 prikazuje početnu poziciju robota i zadani cilj na globalnoj *costmap*-i. Na slici se može vidjeti kako je najkraći direktni put do cilja blokiran jednom kvadratnom i jednom kružnom preprekom. Ukoliko je navigacijski stog dobro postavljen, robot bi trebao prepoznati prepreke i pronaći rutu koja će ih zaobići.

Slika 5.30 Prikaz početne pozicije i zadanog cilja na globalnoj costmap-i

Slika 4.31 prikazuje robota u pokretu i djelomičnu rutu do zadanog cilja. Ruta je samo djelomično određena zbog parametara planera koji određuju koliko daleko u budućnost se ruta planira. Nakon prolaska određenog vremenskog intervala brišu se podaci planera o početku rute i stvaraju se novi za nastavak gibanja do cilja. Ovakav pristup koristi se radi štednje računalnih resursa (RAM, CPU...). Ruta prikazana na Slici 4.31 zaobilazi prepreke, ali i vodi robota dovoljno blizu preprekama kako bi se ostvario što brži dolazak do cilja. Također na slici se može vidjeti kako robot ispravno prati zadanu rutu i kako se očitanja senzora poklapaju sa preprekama na mapi.

Slika 5.31 Prikaz robota na putu do cilja i parcijalne rute

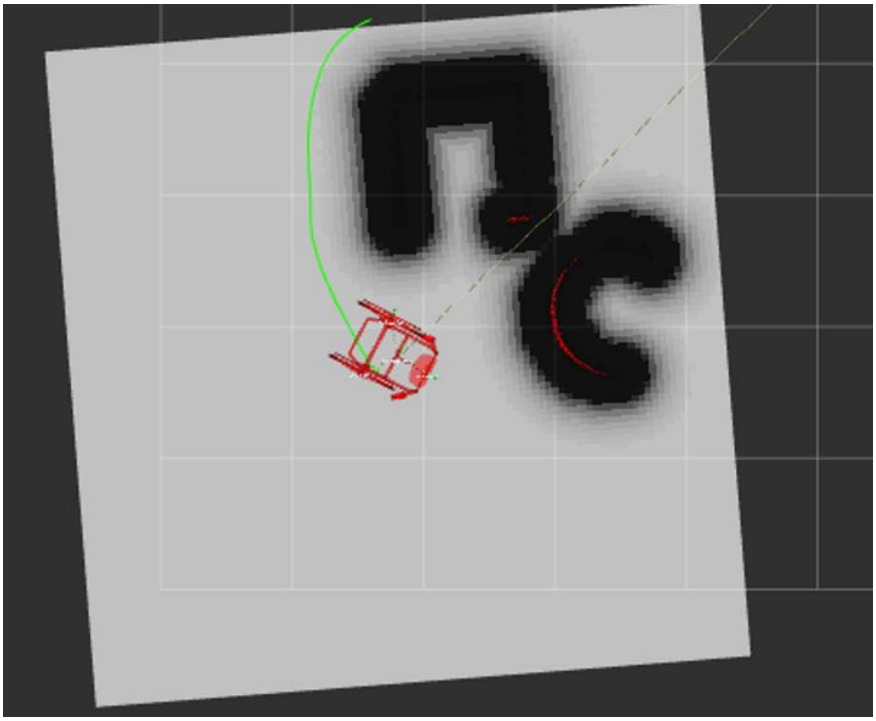
Slika 4.32 prikazuje robota na lokaciji zadanoj putem navigacijskog cilja i ostatak njegove rute. Tokom gibanja kroz prostor robot nije značajno odstupao od zadane rute, što je znak dobro postavljenog navigacijskog stoga.



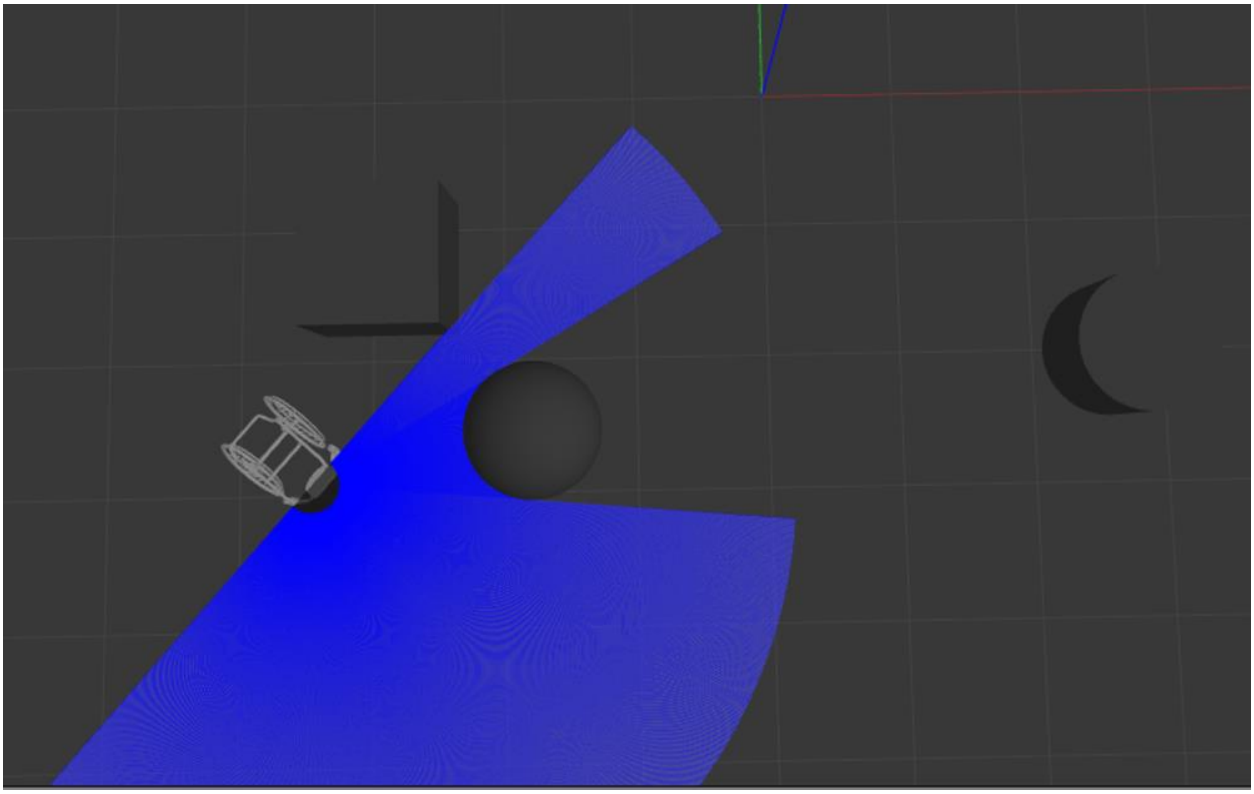
Slika 5.32 Prikaz konačne pozicije i ostatka rute

Slika 4.33 prikazuje lokalnu *costmap*-u na mjestu konačne pozicije robota. Lokalna *costmap*-a koristi se primarno za izbjegavanje prepreka i ne gradi se pomoću unaprijed zadane mape već pomoću očitanja senzora. Izgled mape na Slici 4.34 odgovara očekivanom, budući da prikazuje prepreke koje je robot zaobišao. Kontura prepreka nije u potpunosti zatvorena budući da ih robot nije obišao sa svih strana, odnosno nisu bile zabilježene laserskim daljinomjerom.

Zadnja provjera ispravnosti prikazana je na Slici 4.34 koja prikazuje lokaciju robota u Gazebo simulatoru. Budući da su pozicije prikazane na mapi i u Gazebo unutar granica tolerancija odstupanja, može se zaključiti da je autonomno navigiranje uspješno izvedeno.



Slika 5.33 Lokalna costmap-a na konačnoj poziciji



Slika 5.34 Prikaz konačne pozicije u Gazebo simulatoru

6 Zaključak

Zadatak ovog diplomskog rada bila je prenamjena invalidskih kolica u mobilnog robota sa sposobnoću autonomne navigacije kroz prostor. Autonomnu navigaciju, kao i sustav upravljanja, bilo je potrebno razviti unutar ROS-a.

Drugo poglavlje daje kratak uvid u polje mobilne robotike kroz podjelu na četiri glavna sustava svakog mobilnog robota. Na temelju pregleda ovih sustava stvara se uvid ne samo u način funkcioniranja mobilnih robota, već i u komponente potrebne za rad pojedinog sustava. Lokomocijski sustav odgovoran je za mogućnost kretanja robota u prostoru. Pregledom lokomocijskog sustava daje se uvid u razne načine pokretanja s obzirom na aktuatore i druge pogonske elemente, s time da je najveći fokus na robotima s kotačima budući da su usko vezani uz zadatak diplomskog rada. Percepcijski sustav ima ulogu robotovih „osjetila“. Pregledom percepcijskog sustava daje se uvid u razne vrste senzora koje se koriste u mobilnoj robotici. Kognicijski sustav odgovoran je za analizu ulaznih signala iz raznih izvora i generiranje izlaznih signala za pokretanje aktuatora. Pregledom kognicijskog sustava daje se uvid razne metode koje se koriste za upravljanje mobilnih robota. Navigacijski sustav odgovoran je za planiranje rute i „snalaženje“ u prostoru. Pregledom navigacijskog sustava objašnjeni su njegovi osnovni dijelovi i dan je uvid u neke od metoda planiranja ruta.

U trećem poglavlju opisuje se ROS, sustav pomoću kojega je potrebno upravljati robotom. ROS je modularan sustav koji počiva na interakciji usko specijaliziranih paketa. ROS pakete sačinjeni su od čvorova, tema, usluga i poruka. U poglavlju je dan uvid u karakteristike, funkcionalnost i međusobnu interakciju spomenutih značajki.

Četvrto poglavlje bavi se prenamjnom invalidskih kolica u mobilnog robota. Pod prenamjenom prvenstveno se misli na uspostavu lokomocijskog i percepcijskog sustava opisanih u prvom poglavlju. Prenamjena se radi po uzoru na TurtleBot3 mobilni robot, koji je komercijalni robot potpuno prilagođen za rad u ROS-u. Za invalidska kolica izabire se diferencijalni pogon kao način pokretanja, te se odabiru prikladni servomotori. Za percepcijski sustav odabiru se laserski daljinomjer, za vanjska očitanja, i IMU, za unutarnja očitanja.

Peto poglavlje objašnjava osposobljavanje autonomne navigacije mobilnog robota u ROS-u. Poglavlje je strukturirano tako da opisuje ovaj postupak korak po korak. Prvi korak je izrada URDF modela koji predstavlja robota i njegove aktuatore i senzore. Nakon izrade modela potrebno je napraviti simulacijski okolinu u kojoj će se testirati funkcionalnost robota. Za potrebe simulacija koristi se Gazebo simulator. Nakon izrade simulacijske okoline, istu je potrebno mapirati kako bi se robotu dali podaci o prostoru kroz koji će se kretati. Zadnji korak je postavljanje navigacijskog stoga koji vrši ulogu planiranja rute i kreiranja komandi, pomoću kojih se ostvaruje ruta. za aktuatore. Nakon postavljanja navigacijskog stoga napravljene su provjere ispravnosti, od kojih su dvije dokumentirane u sklopu ovog diplomskog rada.

Literatura

[1] Rubio, F.; i dr.: "A Review of Mobile Robots: Concepts, Methods, Theoretical Framework, and Applications." *International Journal of Advanced Robotic Systems*, Mar. 2019.

[2] Candido, C.P.: "Auxiliary mechanisms for telerobotics", diplomski rad, Universidade Nova de Lisboa, Faculdade de Ciencias e Tecnologia, 2008.

[3] S interneta, https://medium.com/manual-robotics/drives-76c2b2dac97c , pristupljeno 29.08.2022.

[4] S interneta, https://gigazine.net/gsc_news/en/20141106-zuri-01-paperbot-system/ , pristupljeno 29.08.2022.

[5] S interneta, https://www.gyrobot.com.au/index.php/robot/tracked-robot-chassis, pristupljeno 29.08.2022.

[6] S interneta, https://blog.orientalmotor.com/incremental-vs-absolute-systems , pristupljeno 29.08.2022.

[7] Panescu D.: „MEMS in medicine and biology", IEEE Engineering in Medicine and Biology Magazine, vol. 25, no.5, pp. 19-28, 2006.

[8] Trusov, A.A., i dr.: "Non-axisymmetric coriolis vibratory gyroscope with whole angle, force rebalance, and self-calibration", University of California, Irvine, 2012.

[9] S interneta, https://electrosome.com/interfacing-hc-sr04-ultrasonic-sensor-arduino-uno/ , pristupljeno 29.08.2022.

[10] ActivMedia Robotics, „PowerBot™ Operations Manual", 2003.

[11] Gupta, A., Yan, D.: "Mineral processing design and operations", Elsevier, Amsterdam, 2016.

[12] S interneta, https://www.ibm.com/cloud/learn/neural-networks , pristupljeno 29.08.2022.

[13] Velagic, J., Osmic, N., Lacevic, B.: "Neural Network Controller for Mobile Robot Motion Control" ,International Journal of Intelligent Systems and Technologies, vol. 3, no. 3, pp. 127-132, 2008.

[14] Zadeh, L.A.,: "Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems", World Scientific Publishing Co., Inc., New Jersy, 1996.

[15] Cupertino F., i dr.,: "Fuzzy Control of a Mobile Robot", IEEE Robotics & Automation Magazine, vol. 13, no. 4, pp. 74-81, 2006.

[16] S interneta, https://github.com/linorobot/linorobot/wiki/6.-Creating-a-Map , pristupljeno 29.08.2022.

[17] Vrcan Ž.,: Predavanje iz predmeta numeričke metode u konstruiranju, Tehnički fakultet Rijeka, Rijeka, 2022.

[18] Lukac, M., Krylov, G.: "Study of GPU Acceleration in Genetic Algorithms for Quantum Circuit Synthesis", 2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL), 2017, pp. 213-218

[19] Quigley, M., i dr.,: "ROS: an open-source Robot Operating System" ,ICRA Workshop on Open Source Software, 2009.

[20] Jaideep, G., Prakash Battula, "Detection of DDOS attacks in distributed peer to peer networks", International Journal of Engineering & Technology, vol. 7, no. 2.7, pp. 1051-1057, 2018.

[21] T. Kaupp, i dr.,: "Building a Software Architecture for a Human-Robot Team Using the Orca Framework," Proceedings 2007 IEEE International Conference on Robotics and Automation, 2007, pp. 3736-3741

[22] S interneta, http://wiki.ros.org/Nodes , pristupljeno 02.09.2022.

[23] S interneta, http://wiki.ros.org/Names , pristupljeno 02.09.2022.

[24] S interneta, http://wiki.ros.org/Topics , pristupljeno 02.09.2022.

[25] S interneta, http://wiki.ros.org/Services , pristupljeno 02.09.2022.

[26] S interneta, http://wiki.ros.org/Messages , pristupljeno 02.09.2022.

[27] S interneta, http://wiki.ros.org/Packages , pristupljeno 02.09.2022.

[28] S interneta, http://wiki.ros.org/catkin/conceptual_overview , pristupljeno 02.09.2022.

[29] S interneta, https://emanual.robotis.com/docs/en/platform/turtlebot3 , pristupljeno 03.09.2022.

[30] S interneta, https://emanual.robotis.com/docs/en/dxl/mx/mx-64/ , pristupljeno 03.09.2022.

[31] S interneta, https://www.hokuyo-aut.jp/search/single.php?serial=156#download , pristupljeno 03.09.2022.

[32] S interneta, https://store.arduino.cc/products/arduino-mkr-imu-shield?selectedStore=eu , pristupljeno 03.09.2022.

[33] S interneta, http://wiki.ros.org/urdf/XML/joint , pristupljeno 06.09.2022.

[34] S interneta, https://wiki.ros.org/hokuyo_node , pristupljeno 06.09.2022.

[35] S interneta, http://wiki.ros.org/navigation/Tutorials/RobotSetup , pristupljeno 10.09.2022.

[36] S interneta, http://wiki.ros.org/amcl#:~:text=amcl%20is%20a%20probabilistic%20localization,robot%20against%20a%20known%20map. , pristupljeno 10.09.2022.

Sažetak

U ovom diplomskom radu napravljen je upravljački sustav mobilnog robota konstruiranog na platformi invalidskih kolica. Upravljački sustav napravljen je koristeći Robot Operating System (ROS). Pomoću ROS-a implementirani su SLAM (eng. *Simultaneous Localization and Mapping*) i autonomni način rada robota. SLAM način rada koristi teleoperaciju za upravljanje i koristi se za mapiranje robotu nepoznatih okruženja. Autonomni način rada koristi AMCL (eng. *Adaptive Monte Carlo Localization*) i planere putanje za samostalnu navigaciju robota u prijašnje mapiranim okruženjima. Ispravnost sustava testirana je unutar Gazebo simulatora koristeći simulacijsko okruženje sa preprekama.

Ključne riječi: mobilni robot, Robot Operating System, SLAM, AMCL, Gazebo

In this master thesis control system for mobile robot for a wheelchair based mobile robot was made. Control system was made using the Robot Operating System (ROS). ROS was used to implement SLAM (*Simultaneous Localization and Mapping*) and autonomous mode of operation for the robot. SLAM mode uses teleoperation as means of control and it is used for mapping of environments which are previously unknown for the robot. Autonomous mode uses AMCL (*Adaptive Monte Carlo Localization*) and route planners for fully autonomous navigation of the robot in previously mapped environments. Validity of the system was tested via Gazebo simulator using a simulated obstacle course.

Keywords: mobile robot, Robot Operating System, SLAM, AMCL, Gazebo