Planiranje kretanja i ispitivanje točnosti pozicioniranja mobilnog robota

Ćaran, Branimir

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Branimir Ćaran

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Doc. dr. sc. Marko Švaco, dipl. ing.

Student:

Branimir Ćaran

Zagreb, 2021.

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

Zahvaljujem se mentoru doc. dr. sc. Marku Švaci na potpori i stručnim savjetima tijekom studija i pisanja ovog rada te dr. sc. Josipu Vidakoviću na stalnoj dostupnosti i usmjeravanju.

Zahvaljujem se i svojim roditeljima Marinku i Miri, sestri Filipi te svim svojim prijateljima na bezuvjetnoj podršci tijekom studiranja i pisanja ovog rada.

Branimir Ćaran

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
x	m	Pozicija robota u smjeru osi X
y	m	Pozicija robota u smjeru osi Y
θ	rad	Orijentacija robota u X-Y
v	m/s	Linearna brzina robota
v_r	m/s	Linearna brzina desnog kotača
v_l	m/s	Linearna brzina lijevog kotača
ω	rad/s	Kutna brzina robota u X-Y ravnini
ω_r	rad/s	Kutna brzina desnog kotača
ω_l	rad/s	Kutna brzina lijevog kotača
r	m	Polumjer kotača robota
$\dot{x}$	m/s	Brzina robota u smjeru osi X
$\dot{y}$	m/s	Brzina robota u smjeru osi Y
$\dot{\theta}$	Rad/s	Rotacijska brzina robota u X-Y
dx	m	Apsolutna greška robota u smjeru osi X
dy	m	Apsolutna greška robota u smjeru osi Y
dxy	m	Apsolutna greška robota u smjeru osi X i Y
c_m	rad·m/okretaju	Koeficijent pretvorbe impulsa enkodera u linearni pomak kotača
D_n	m	Nazivni promjer kotača
C_e	impulsi/okretaj	Rezolucija
n	-	Omjer redukcije zupčanika
N_i	-	Inkrementalna promjena impulsa lijevog ili desnog motora
ΔU_i	m	Inkrementalni linearni pomak lijevog ili desnog kotača
b	b	Razmak među kotačima
$\Delta\theta_i$	rad	Inkrementalna orijentacija robota
E_d	-	Korekcijski faktor promjera kotača
E_b	-	Korekcijski faktor udaljenosti među kotačima
$x_{c.g.}$	m	Težište klastera pogrešaka u smjeru osi X
$y_{c.g.}$	m	Težište klastera pogrešaka u smjeru osi Y
$r_{c.g.}$	m	Apsolutna udaljenost težišta klastera od ishodišta
$E_{max,syst}$	m	Maksimalna sustavna pogreška robota
L	m	Duljina stranice kvadratne trajektorije
α	rad	Kut odstupanja
β	rad	Kut odstupanja
R	m	Polumjer kružnice koju robot opisuje prilikom gibanja ravno
x_{LIDAR}	m	Pozicija lidara u smjeru osi X koordinatnog sustava robota

SAŽETAK

Zadatak diplomskog rada je implementirati algoritme mapiranja, lokalizacije i navigacije u robotskom operativnom sustavu (ROS) na mobilnog robota. Osim implementacije navedenih procesa, detaljno su objašnjeni svi parametri koje je potrebno podesiti za adekvatan rad čvorova koji omogućavaju implementaciju navedenih algoritama. Zbog potrebe za što preciznijim mapiranjem, lokalizacijom i navigacijom, u ovom radu su detaljno objašnjeni i eksperimentalno provedeni procesi kalibracije odometrije mobilnog robota i pozicije lidar senzora. Osim kalibracije mobilnog robota u svrhu što točnije navigacije i pozicioniranja robota, detaljno su objašnjena, implementirana i testirana tri lokalna planera koji se mogu pronaći u sklopu ROS-a. U svrhu provjere tri lokalna planera, korišten je P regulator pozicije s kojim su se planeri na kraju usporedili.

Ključne riječi: mobilni robot, robotski operativni sustav, ROS, mapiranje, lokalizacija, navigacija, kalibracija odometrije, lokalni planeri

SUMMARY

The task of this Master's thesis is to implement mapping, localization and navigation algorithms in a robotic operating system (ROS) on a mobile robot. In addition to the implementation of these processes, all the parameters that need to be set for adequate operation of the nodes that allow the implementation of these algorithms are explained in detail. Due to the need for the most precise mapping, localization and navigation, this thesis explains in detail and experimentally performed the processes of mobile robot odometry calibration and lidar sensor position. In addition to the calibration of the mobile robot for the purpose of more accurate navigation and positioning of the robot, three local planners that can be found within the ROS were explained, implemented and tested in detail. In order to check the three local planners, a position P regulator was used with which the planners were eventually compared.

Key words: mobile robot, robotic operating system, ROS, mapping, localization, navigation, odometry calibration, local planners

1. UVOD

Mobilna i industrijska robotika doživjela je eksponencijalni rast u posljednjih deset godina ponajviše zbog zahtjeva za lakšim i bržim načinima za proizvodnju pojedinih dijelova. Osim industrijske primjene, robotika postaje dio svakodnevice svih ljudi što je još jedan od zahtjeva za njezini bržim i kvalitetnijim razvitkom.

U ovom radu će biti obrađeno mapiranje nepoznatog prostora i lokalizacija mobilnog robota u prethodno mapiranom prostoru. Za upravljanje robotom će se koristiti robotski operativni sustav (*engl. Robot Operating System – ROS*). U nastavku će biti objašnjen princip rada ROS-a, kao i paketi koji se koriste za upravljanje mobilnim robotom te paketi koji se koriste za mapiranje, lokalizaciju i navigaciju robota. Mobilni robot se danas sve više koriste u raznim segmentima života, od svakodnevne upotrebe u kućanstvu (roboti za usisavanje) do industrijskih mobilnih platformi koje na sebi imaju instalirane industrijske robote. Kombinacija mobilnih i industrijskih robota nude mogućnosti sve većeg stupnja autonomnosti proizvodnih sustava jer mobilni roboti nude mogućnost kretanja i snalaženja u prostoru dok industrijski roboti omogućuju manipulaciju predmetima rada. Kako bi krajnja manipulacija predmetom bila što bolja i točnija potrebno je što bolje pozicionirati mobilnog robota, stoga su problemi mapiranja i lokalizacije postali neizbježni. Osim navedenih procesa, za adekvatnu upotrebu mobilnih robota potrebno je osigurati sigurnu i točnu navigaciju po prostoru te provesti kalibracijske procese.

2. MOBILNA ROBOTIKA

Mobilna robotika je jedna od grana robotike koja se bavi robotima koji nisu stacionarno vezani nego se mogu kretati u prostoru. Takvi roboti koriste razne pogonske, senzorske i upravljačke sustave kako bi se kretali i snalazili u prostoru u kojem moraju obavljati svoje zadatke. Jedan od osnovnih motiva razvoja mobilne robotike je mogućnost rješavanja problema u okolinama koje su opasne za ljude.

U mobilnu robotiku se osim klasičnih mobilnih robota, mogu svrstati dronovi, autonomne podmornice, humanoidni roboti, roboti kućni ljubimci kao i mnogi drugi. Osnovna funkcija mobilnih robota je mogućnost kretanja po prostoru, najčešće nepoznatom, za što im je osim mehaničke konstrukcije i aktuatora potrebna inteligencija. Kako bi mobilni roboti što bolje percipirali okolinu i snašli se u istoj, koriste se razni senzorski sustavi koji omogućuju pretvorbu fizikalnih veličina iz okoline u signale koji se potom šalju na upravljački sklop robota na kojem se nalaze algoritmi koji služe za obradu tih signala, percipiranje prostora, planiranje putanja i izbjegavanje sudara.

Tendencija mobilne robotike je razvoj algoritama umjetne inteligencije koji će služiti za upravljanje mobilnim robotima kao i kopiranje načina kretanja iz prirode. Primjetan je razvoj mobilnih robota koji se kreću četveronoške kao psi ili lete kao ptice primjenom pneumatskih aktuatora.

Važno je reći kako je ipak najveći razvoj i primjena mobilne robotike u industriji jer su potrebe za mobilnim robotima sve veće. Tako se u postrojenjima i skladištima mogu pronaći mobilne platforme koje raznose proizvedene komade, u skladištima raznose pakete, pomažu industrijskim robotima prilikom manipulacije predmetima. Takvi mobilni roboti koriste inteligenciju isključivo za percipiranje i snalaženje u prostoru, međutim kako su industrijska postrojenja uređenog prostora, zahtjevi za inteligencijom nisu veliki.

2.1. Primjena mobilnih platformi

Kao što je već rečeno, pod pojmom mobilne platforme nalaze se roboti koji se mogu kretati po tlu, zraku i u vodi. Najčešća primjena mobilnih platformi je u vidu mobilnih robota, koji svoju primjenu nalaze u industriji, medicini i kućanstvima. Na slici 1. je prikazan mobilni robot tvrtke *Gideon Brothers*.

Slika 1. Autonomni mobilni robot za skladišta [2]

Robot prikazan na slici 1. je mobilni robot čija je glavna zadaća ispomoć u skladištu. Robot ima mogućnost autonomnog snalaženja u skladištu te transporta teških paketa iz točke A u točku B. Ovo je odličan primjer primjene mobilnih robota kao ispomoć ljudima, jer robot može prenositi puno teže terete od ljudi i to izvoditi znatno brže.

Ovakav tip robota je opremljen naprednim senzorskim sustavima koji se sastoje od lidar senzora i radara koji mu omogućuju mapiranje i navigaciju kroz iznimno dinamična skladišta. Konfiguracija ovog robota je diferencijalna, odnosno robot ima dva pogonska motora koji mu služe za kretanje, te se od aktuatorskih sustava na robotu nalazi i pogon za podizanje tereta.

Osim u skladištima, primjena mobilnih robota vidljiva je u industrijskim postrojenjima u vidu mobilnih robota s instaliranim industrijskim robotima u obliku robotske ruke. Takvi roboti imaju mogućnost pozicioniranja robotskih ruku u prostoru, dok robotska ruka odrađuje manipulaciju s predmetima rada. Na slici 2. je prikazan mobilni robot s industrijskim robotom tvrtke *Kuka*.

Slika 2. Kuka KMR iiwa – mobilni robot s industrijskom robotskom rukom [3]

Ovakav tip robota pogodan je za industrijska postrojenja iz razloga što može obavljati radnje na više radnih stanica, sam sebe opsluživati potrebnim predmetima rada, odlaziti samostalno u skladište po potrebne dijelove i potrošni materijal.

Primjena ovakvih robota je sve češća u industrijama koje se bave malo serijskim proizvodnjama posebnih dijelova kao što su motori s unutarnjim izgaranjem visokih performansi. Zbog

povećanja trenda proizvodnje dijelova po narudžbi, vjeruje se kako će ovakav tip robota sve više postati primjenjivan u industrijama proizvodnih djelatnosti.

Sve navedene primjene mobilnih robota su teško vidljive svima, odnosno najčešće se s njima susreću inženjeri. Najbolji primjer svakodnevne primjene mobilnih robota je robotski usisivač kojeg se sve više može pronaći u kućanstvima. Na slici 3. [5] je dan izgled robota usisavača *iRobot Roomba 980*.

Slika 3. *iRobot Roomba 980*

Robotski usisavači su opremljeni kamerama i radarima kako bi u prvih nekoliko usisavanja mapirali prostor, a potom bi tu istu mapu kasnije koristili za lokalizaciju. Nakon što je robot u mogućnosti dobiti cijelu sliku površine koju treba usisati i lokalizirati se u njoj, robot može puno bolje i efektivnije usisavati.

Osim gore navedenih mobilnih robota, pod pojmom mobilne platforme nalaze se i autonomne podmornice kao i dronovi. Na slici 4. se nalazi primjer autonomne podmornice.

Slika 4. Autonomna podmornica [6]

Autonomne podmornice su opremljene raznim senzorima za navigaciju kao što su GPS, laserski skeneri, radari i ostali senzori koji omogućuju autonomno kretanje pod vodom. Takvi sustavi se koriste za ispitivanja podvodne flore i faune, istraživanje novih dijelova oceana ili za provjeru i instalaciju dalekovoda i naftnih platformi koje se postavljaju pod vodu.

Još jedan primjer mobilnih platformi su dronovi, dronovi imaju mogućnost kretanja po zraku na principu potisne sile nastale vrtnjom propelera. Na slici 5. je prikazan dron marke *DJI*.

Slika 5. ***DJI Phantom 4*** **[7]**

Dronovi su danas postali iznimno popularni iz razloga što omogućuju 6 stupnjeva slobode gibanja, relativno su jednostavni za upravljanje i mogu doći u teško dostupna područja. Dronovi na sebi imaju najčešće četiri aktuatora za potisnu silu, kamere, lasere, žiroskop i ultrazvučni senzor. Fuzijom tih senzora dron se može autonomno snalaziti i voziti po prostoru te izbjegavati prepreke.

2.2. Sustavi mobilnih robota

Mobilni roboti se sastoje od podsustava koji zajedno komuniciraju i omogućuju robotu kretanje. Osnovni dio je mehanička konstrukcija robota na koju se dodaju ostali sustavi u određenoj konfiguraciji kako bi robot mogao raditi i odrađivati zadani zadatak.

2.2.1. Upravljački sustav

Upravljački sustav je osnovni sustav nužan za autonomnost mobilnog robota. Mikrokontroler koji prikuplja podatke sa senzora, odlučuje što dalje napraviti te potom šalje upravljačke naredbe aktuatorima je osnovna jedinica cijelog upravljačkog sustava. Uz glavni mikrokontroler, na robotu se mogu nalaziti i drugi mikrokontroleri koji upravljaju senzorima, kamerama, aktuatorima ili bilo kojim drugim sustavom robota te potom šalju ili primaju podatke od glavnog mikrokontrolera. U glavnom mikrokontroleru se nalaze algoritmi za mapiranje, lokalizaciju, planiranje trajektorija ili bilo koji drugi algoritam potreban za obavljanje zadataka mobilnog robota. Upravljački sustavi mobilnog robota najčešće se programiraju u nekim od nižih jezika kao što su *C* ili *C++*.

2.2.2. Senzorski sustav

Senzorski sustav robota je zaslužan za prikupljanje informacija i podataka iz okoline te tako omogućuje robotu da dobije što realniju sliku o svome okruženju. Kvalitetno odabran senzorski sustav iznimno je važan dio robota jer mu omogućuje percepciju svoje okoline, a samim time i mogućnost snalaženja u prostoru i izbjegavanja prepreka. Izbor senzora koji se mogu koristiti na robotu iznimno je velik, a najčešće korišteni su senzori za mjerenje udaljenosti, senzori za mjerenje zakreta motora i senzori za mjerenje kutne brzine i akceleracije. Razlikujemo

propriocepcijske i eksteroceptivne senzore. Propriocepcijski senzori mjere unutarnja stanja robota kao što su kutovi zakreta motora, unutarnja temperatura robota, napon baterije i brzina robota, dok su eksteroceptivni senzori različite vrste kamera, ultrazvučni senzori, laserski senzori i IR senzori. Osnovni zadatak svih senzora je stvarne veličine iz okoline pretvoriti u naponske ili strujne signale te ih poslati glavnom upravljačkom sustavu koji na temelju istih odlučuje što i kako dalje napraviti kako bi robot obavio traženi zadatak.

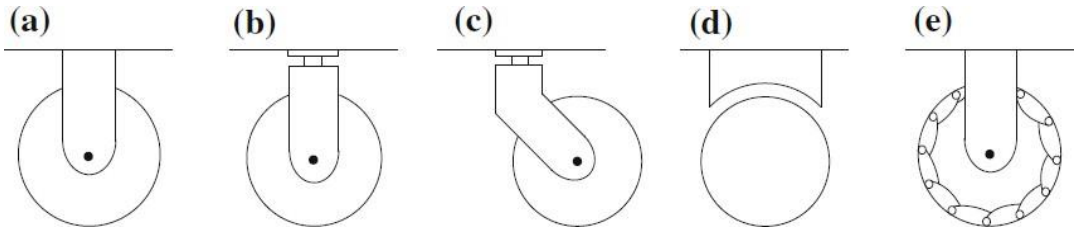
2.2.3. Aktuatorski sustav

Aktuatorski sustav zaslužan je za kretanje robota i manipulaciju predmetima. Kod mobilnih robota najvažniji su aktuatori za lokomociju, odnosno gibanje robota, jer je osnovna prednost i zadaća mobilnih robota gibanje u prostoru. Aktuatori koji su instalirani na mobilne robote najčešće su elektromotori istosmjerne struje, koji se izvode s ili bez četkica. Osim takvih motora, mogu se pronaći roboti s koračnim motorima ali rjeđe. Motori koji se nalaze na mobilnim robotima opremljeni su enkoderima – senzori za mjerenje kuta zakreta motora, koji omogućuju estimaciju pozicije, brzine i akceleracije mobilnog robota.

Upravljački sustav prima podatke sa senzora te sukladno algoritmu koji je programiran u mikrokontroler određuje kako se robot dalje treba ponašati, te preko matematičkog modela računa potrebne brzine motora te ih šalje aktuatorskom sustavu.

2.3. Vrste pogona mobilnih robota

Prijenos gibanja s aktuatorskog sustava na podlogu omogućuje kretanje robota, a to se ostvaruje preko kotača. Minimalan broj kotača potreban da bi mobilni robot bio stabilan je tri. Kotači mogu biti konstruirani na razne načine, na slici 6. je dan prikaz mogućih konstrukcija kotača.



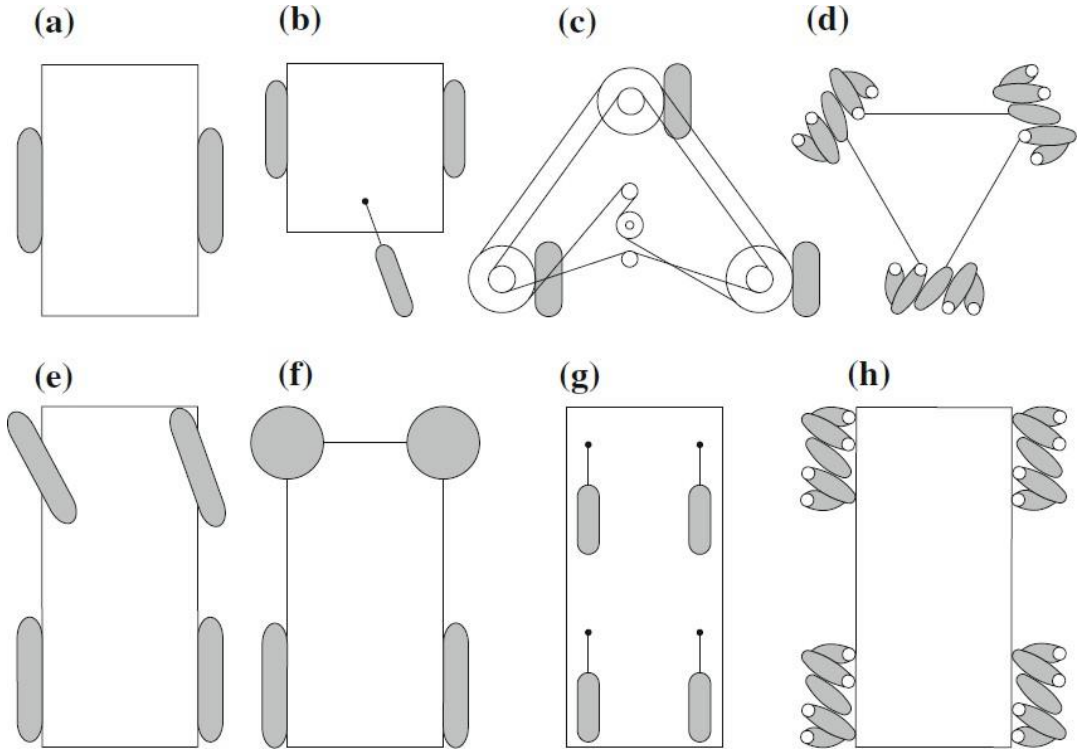
Slika 6. Konstrukcija kotača: a) fiksni kotač, b) rotirajući kotač, c) kutni rotirajući kotač, d) sferni kotač i e) švedski kotač [1]

Kao što je prethodno navedeno, minimalan broj kotača za mobilnog robota je tri, najčešća konfiguracija robota je s dva aktivna fiksna kotača koji su pogonjeni elektromotorom i jednim pasivnim rotirajućim kotačem.

Također, postoje i ostale vrste konfiguracija mobilnih robota sukladno potrebama za rješavanje zadataka. Broj kotača na robotu se može proizvoljno povećavati, međutim preporučuje se da

broj kotača ne bude veći od četiri. Povećanjem broja kotača dolazi do povećanja kompleksnosti kinematičkog modela mobilnog robota, što otežava sintezu regulatora i planera trajektorija gibanja.

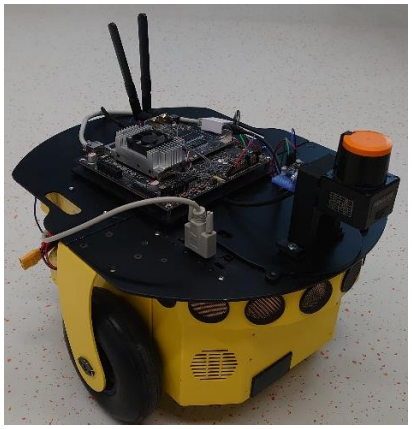
Na slici 7. su prikazane moguće konfiguracije mobilnih robota sukladno konstrukcijskim rješenjima i postavljanjima kotača.



Slika 7. Prikaz mogućih konfiguracija: a) robot s dva fiksna kotača, b) robot s dva fiksna i jednim rotirajućim kotačem, c) robot s tri sinkronizirana kotača, d) robot u obliku trokuta sa švedskim kotačima, e) robot s Ackermannovim mehanizmom skretanja, f) robot s dva fiksna i dva sferna kotača, g) robot s četiri pogonska kutna kotača i h) robot s četiri švedska višesmjerna kotača [1]

3. PIONEER 2-DX MOBILNI ROBOT

Mobilni robot korišten u ovom radu je robot tvrtke *ActivMedia, Pioneer 2-DX* na kojeg je dodano računalno s *Linux* operativnim sustavom kako bi se mogao koristiti ROS i napredni senzorski sustavi. Odabrani robot ima konfiguraciju diferencijalnog robota s dva fiksna kotača i jednim pasivno rotirajućim. Na slici 8. prikazan je izgled odabranog robota.

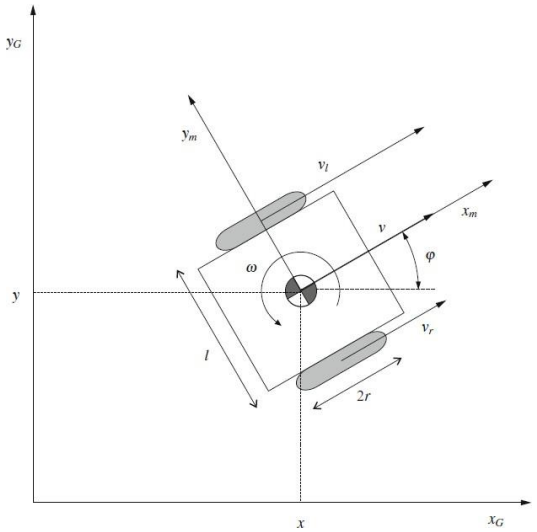


Slika 8. ***Greyp Pioneer 2-DX robot***

Na robota su dodani senzorski i upravljački sustavi koji omogućuju mapiranje, lokalizaciju i navigaciju mobilnog robota. U narednim poglavljima bit će detaljnije objašnjeni svi podsustavi robota, a poglavito oni potrebni za mapiranje i lokalizaciju.

3.1. Kinematički model diferencijalnog mobilnog robota

Kinematički model robota služi kako bi se matematički opisalo ponašanje mobilnog robota, ovdje će biti dan model diferencijalnog mobilnog robota jer je u radu korišten takav robot. Diferencijalni mobilni robot ima fiksna kotača koji su pričvršćeni na aktuatore i jedan pasivni rotirajući kotač. Na slici 9. je prikazan diferencijalni robot u X-Y ravnini.



Slika 9. Prikaz mobilnog robota u X-Y ravnini [1]

Kako bi kinematički model bio ispravno postavljen, potrebno je uvesti neke pretpostavke:

- Gibanje robota je planarno, odnosno u horizontalnoj ravnini
- Nema proklizavanja kotača
- Mobilni robot se promatra kao kruto tijelo na kotačima

Pozicija robota se može opisati s tri koordinate u prostoru, prve dvije predstavljaju poziciju u ravnini (X i Y koordinata), dok treća opisuje orijentaciju oko vertikalne Z osi. Kao što je vidljivo na slici 9. globalni koordinatni sustav označen je s x_G i y_G, dok su lokalne koordinate mobilnog robota dane s x_m i y_m, gdje je x_m definiran u smjeru kretanja robota. Izraz (1) daje vektor stanja mobilnog robota

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \tag{1}$$

gdje su *x* i *y* relativne koordinate lokalnog koordinatnog sustava u odnosu na globalni koordinatni sustav.

Kao što je rečeno, na robotu se nalaze dva aktuatora na koje su pričvršćeni kotači. Ukoliko se kotači okreću u istom smjeru i istom brzinom, robot će se gibati pravocrtno, dok će u slučaju okretanja kotača u suprotnim smjerovima, robot vršiti rotaciju oko centra mase. Centar rotacije mobilnog robota može biti u bilo kojoj točki koja se nalazi na pravcu koji prolazi kroz oba kotača. Izraz (2) prikazuje kako brzine lijevog i desnog kotača utječu na linearnu brzinu mobilnog robota u smjeru osi x_m

$$v = \frac{v_r + v_l}{2}, \tag{2}$$

gdje su v_r brzina desnog kotača, a v_l brzina lijevog kotača.

Izrazom (3) definirana je kutna brzina mobilnog robota oko osi z:

$$\omega = \frac{\omega_r + \omega_l}{2}, \tag{3}$$

gdje su ω_r kutna brzina desnog kotača, a ω_l kutna brzina lijevog kotača. Kutne brzine se estimiraju iz pozicije koju daje enkoder pričvršćen na motor.

Relacija između v_r i ω_r te v_l i ω_l dana preko polumjera kotača prikazana je izrazom (4)

$$v_r = \omega_r r, \qquad v_l = \omega_l r \tag{4}$$

Izrazi (2) i (3) daju relativnu brzinu mobilnog robota u lokalnom koordinatnom sustavu x_m i y_m, a kako bi se dobila brzina mobilnog robota u globalnom koordinatnom sustavu x_g i y_g potrebno je pronaći matricu transformacija T koja je dana izrazom (5):

$$T = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Primjenom matrice transformacije T, može se dobiti transformacija između lokalnog i globalnog koordinatnog sustava, za translacijsku brzinu v dan je izraz (6):

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ 0 \end{bmatrix}, \tag{6}$$

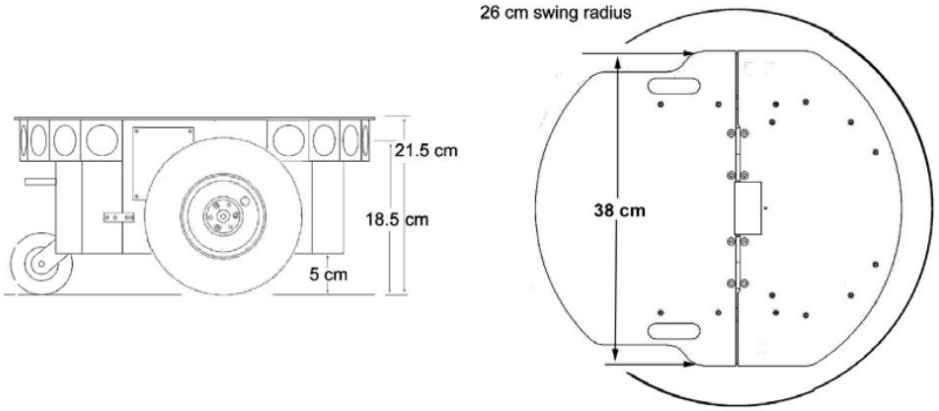
dok za rotacijski dio ne treba koristiti matricu transformacije jer je kutna brzina u lokalnom i globalnom koordinatnom sustavu jednaka. Imajući to u vidu, kao i izraz (6), dobije se da su translacijska i rotacijska brzina u globalnom koordinatnom sustavu definirane kao:

$$\dot{x} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}. \tag{7}$$

Iz gore izvedenog kinematičkog model mobilnog robota se vidi kako je riječ o pod-upravljanom sustavu. Na robotu se nalaze dvije upravljačke varijable, to su kutne brzine ω_r i ω_l, a njima se utječe na tri varijable stanja x, y i θ, što govori kako algoritmi upravljanja mobilnog robota nisu jednostavni, nego zahtijevaju kompliciraniju sintezu planera trajektorija.

3.2. Mehaničke komponente robota

Na robotu se nalaze dva istosmjerna motora s povratnom vezom. Povratna veza je ostvarena preko inkrementalnih optičkih enkodera koji daju relativnu poziciju i brzinu motora. Na motore su pričvršćeni kotači s ispunjenim gumama. Gabaritne dimenzije robota su 36x26x21.5cm. Kućište robota je izrađeno od aluminijskog lima debljine 1.6mm. Na slici 10. su dane detaljnije dimenzije robota.



Slika 10. Dimenzije mobilnog robota [8]

Na prednjoj strani robota se nalazi kućište za sonarne senzore koji mjere udaljenost i omogućuju robotu da vidi ispred sebe ukoliko nailazi na prepreku. Osim čistih mehaničkih parametara danih na slici 10., važno navesti i parametre motora i kotača jer su oni važni za performanse robota i kinematički i dinamički model robota. U tablici 1. su dane karakteristike pogonskog sustava robota.

Broj kotača	2
Vrsta kotača	ispunjena guma
Promjer kotača [mm]	165
Širina kotača [mm]	37
Širina pomoćnog kotača [mm]	75
Prijenosni omjer	19,7:1
Maksimalna translacijska brzina [mm/s]	1600
Maksimalna rotacijska brzina [º/s]	300
Broj enkodera	2

Tablica 1. Parametri kotača i motora [8]

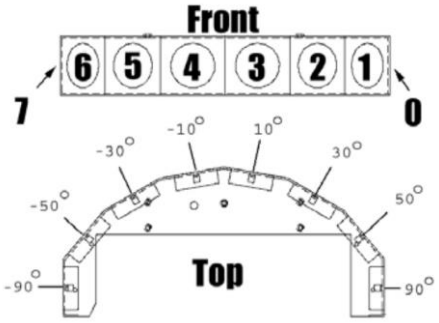
Važno je reći kako ukupna masa robota iznosi 9kg, dok je nosivost robota 17kg. Osim gore navedenih parametara robota, javljaju se i ostali parametri koji su važni za rad na ovakvom robota, kao što su maksimalna translacijska i rotacijska brzina, tablica 2. prikazuje preostale parametre mobilnog robota.

Masa	9 kg
Nosivost	17 kg
Radijus skretanja	0 cm
Radijus rotacije	26 cm
Maksimalna translacijska brzina	1.2 m/s (vrh 1.6 m/s)
Maksimalna rotacijska brzina	300°/s
Maksimalna translacijska brzina [mm/s]	1600
Maksimalna rotacijska brzina [º/s]	300

Tablica 2. Ostali parametri robota [8]

3.3. Senzorski sustav robota

Pioneer 2-DX na sebi ima instalirane ultrazvučne senzore na prednjem dijelu konstrukcije koji mu služe da ne bi došao u koliziju s preprekama koje mu se nađu na putu. Na slici 11. se nalazi prikaz i raspored sonarnih senzora robota.



Slika 11. Raspored sonarnih senzora [8]

Zbog zahtjeva za mapiranjem prostora, a kasnije i lokalizaciji u istom, na robota je dodan lidar senzor. Lidar je vrsta senzora koja mjeri udaljenost na principu lasera. Laser je postavljen na elektromotor te zakretanjem motora skenira predmete u svom video krugu.

Na robota je postavljen senzor *Hokuyo UST-10LX*, 2D laserski skener. Senzor ima zakret od 270° te je postavljen na prednji dio robota kako bi mogao skenirati prostor ispred sebe i tako mapirati ga. Na slici 12. je prikazan senzora.

Slika 12. *Hokuyo UST-10LX* [9]

U tablici 3. su dani svi važni parametri laserskog skenera [9].

Kut skeniranja	270°
Broj koraka	1080
Napajanje	10-30V
Potrošnja struje	150 mA (450 mA prilikom paljenja)
Mogućnost mjerenja	0.06m to 10m
Valna duljina lasera	905nm, laser klase 1
Preciznost	-+40mm
Kutna rezolucija	0,25°
Komunikacija	Ethernet 100BASE-TX
Dimenzije	50x50x70 mm
Masa	130g

Tablica 3. Karakteristike *Hokuyo UST-10LX* senzora

Posljednji senzor instaliran na robota je IMU (*engl. inertial measurement unit) Bosch BNO055*. Odabrani senzor sastoji se od akcelerometra, žiroskopa i magnetometra. Fuzijom tri manja senzora, senzor ima mogućnost vraćanja podataka koji se koriste za upravljanje i pozicioniranje mobilnog robota. Za rad senzora, odnosno fuziju i slanje svih podataka, zaslužan je ARM Cortex-M0 koji se nalazi na samom senzoru. Na slici 13. je prikazan senzor, dok su u tablici 4. parametri istoga.

Slika 13. Prikaz IMU-a *Bosch BNO055* [10]

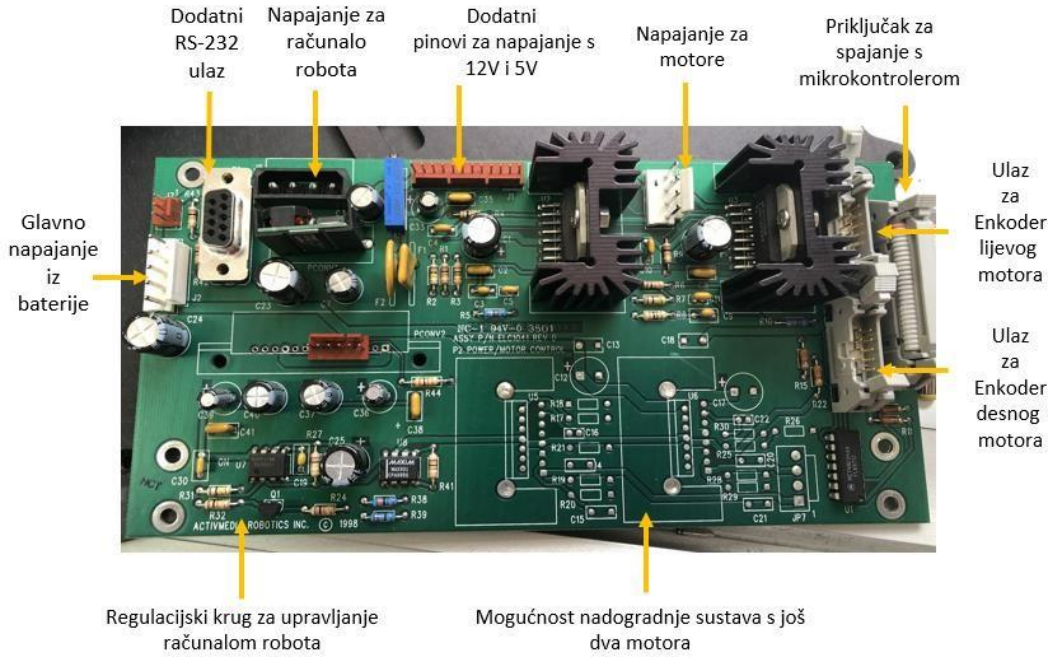
Napajanje	3.3-5V
Komunikacija	I2C
Apsolutna orijentacija	Eulerov vektor ili kvaternioni
Kutna brzina	Za sve 3 osi u rad/s
Linearna akceleracija	Gravitacija + linearno gibanje u sve 3 osi u m/s
Magnetsko polje	Jačina polja u sve 3 osi u uT
Gravitacija	Djelovanje gravitacije u sve 3 osi
Temperatura	Da, u °C

Tablica 4. Karakteristike senzora *Bosch BNO055* [10]

3.4. Upravljački sustav robota

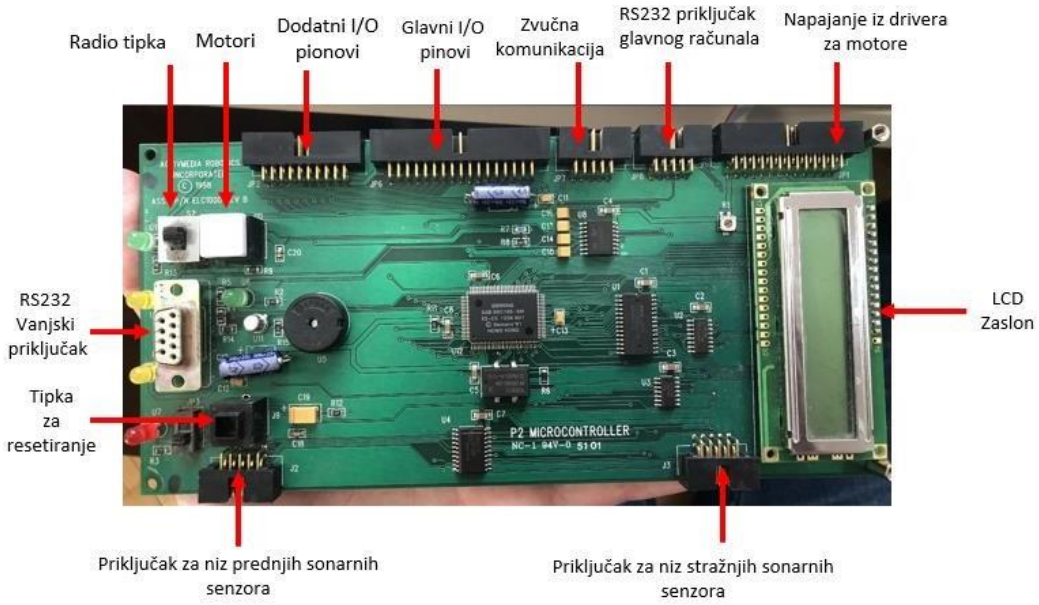
Upravljački sustav robota se sastoji od tri dijela, najniži dio je upravljačka pločica motora, potom mikrokontroler robota te na kraju računalo s *Linux* operativnim sustavom.

Na pločici za upravljanje motora nalazi se i dio za energetsku elektroniku, odnosno pretvarači koji napon baterije pretvaraju u 5V i 12V. *Driveri* koji se koriste za upravljanje motorima se nalaze na pločici, ali postoji mogućnost za nadogradnju, odnosno dodavanje još dva motora. Osim gore navedenih segmenata upravljačke pločice motora, na istoj pločici se nalazi i dodatni konektor za RS-232 komunikacijski protok. Na slici 14. je dan prikaz pločice za napajanje i upravljanje motorima.



Slika 14. Pločica za upravljanje motorima i napajanje [11]

Idući važni segment upravljačkog sustava robota je mikrokontrolerska pločica koja je zadužena za upravljanje robotom na najnižoj razini. Pločica na sebi sadrži mikrokontroler *Siemens 88C166* s integriranom memorijom od 32K Flash-ROM te dinamičkim RAM-om od 32K. Na pločici se također nalazi i LCD ekran s mogućnosti prikaza 32 znaka. Njegova glavna zadaća je prikaz stanja baterije i komunikacije s računalom. Serijska komunikacija između računala(klijenta) i robota se ostvaruje preko RS-232 priključka. Na slici 15. je prikazan mikorkontroler.



Slika 15. Mikorkontrolerska pločica robota [11]

Posljednja i najvažnija upravljačka komponenta je računalo na kojem se nalazi *Linux* operativni sustav. Osnovna na tom računalu je instaliran robotski operativni sustav (ROS) te su spojeni lidar i IMU senzor. Algoritmi za mapiranje i lokalizaciju se izvode na tom računalo, ono radi sve izračune te šalje naredbe za brzine motora mikrokontrolerskoj pločici robota. Odabrano računalo je *Nvidia Jetson TX2 Developer Kit*. Na slici 16. je prikazano računalo.

Slika 16. je *Nvidia Jetson TX2 Developer Kit* [12]

Računalo je odabrano zbog svojih performansi, mogućnosti Ethernet i I2C komunikacije koja je potrebna za spajanje senzorskih sustava. Na računalu se također nalaze WiFi i Bluetooth bežične komunikacije koje omogućuju dodatno spajanje na internet i komunikaciju s ostalim uređajima. Detaljne karakteristike računala su dane u tablici 5.

Procesor	2 Denver 64-bit CPU + Quad-Core A57 Complex
RAM memorija	8GB L128 DDR4
Memorija	32FB eMMC 5.1 Flash Storage
Bežična komunikacija	802.11a WiFi i Bluetooth
Ethernet komunikacija	10/100/1000BASE-T Ethernet
USB	USB 3.0 Type A i USB 2.0 Micro AB
Komunikacijski protokoli	I2C, I2S, SPI, CAN, TTL UART
Kamera	5MP MIPI CSI kamera
Napajanje	19V
HDMI	Da

Tablica 5. Karakteristike *Nvidia Jetson TX2 Developer Kit*-a [12]

3.5. Programski sustavi robota

Pioneer 2DX mobilni robot u sebi ima programski sustav koji omogućuje rad na principu server-klijent. Robot je u ovom slučaju server na kojeg se spaja klijent, koji može biti bilo koje računalo. Računalo šalje glavnom mikrokontroleru robota naredbe za željenim brzinama robota, te mikrokontroler potom na nižoj razini radi proračune koje potom šalje upravljačkoj pločici motora. Važno je reći kako postoji mogućnost povezivanja više robota s jednim klijentom, a to omogućuje razvoj i primjenu algoritama za upravljanje flotama robota.

Operativni sustav koji se koristi u mikrokontroleru robota je *P2OS*. Njegova je zadaća da vrši kontrolu svih sustava robota na najnižoj razini. Zadaće koje on obavlja su prikupljanje podataka sa sonarnih senzora i enkodera motora te upravljanje motorima. Naredbe za prikupljanje svih podataka i slanje naredbi se vrši preko aplikacije *ARIA* koja je instalirana na računalu klijenta. Takav način rada omogućava razvojnim programerima da ne moraju ulaziti u srž robota, već se o tome brine operativni sustav *P2OS* instaliran na mikrokontroleru.

ARIA (*engl. AcrivMedia Robotics Interface for Applications)* je okruženje namijenjeno razvojnim programerima za upravljanje *Pioneer* robotima. Biblioteka *ARIA* je napisana u C++ programskom jeziku i omogućuje kvalitetnu server-klijent komunikaciju s robotom. Razvojni

programeri rade na programu u okruženju *ARIA*, koje potom sve podatke šalje *P2OS* operativnom sustavu koje potom upravlja robotom. Glavna klasa u *ARIA-i* je *ArRobot* koja služi za glavnu komunikaciju i izmjenu podataka između robota i klijenta. Njezina zadaća je prikupljanje podataka kao što su translacijska i rotacijska brzina robota, smjer gibanja robota i pozicija u X-Y ravnini.

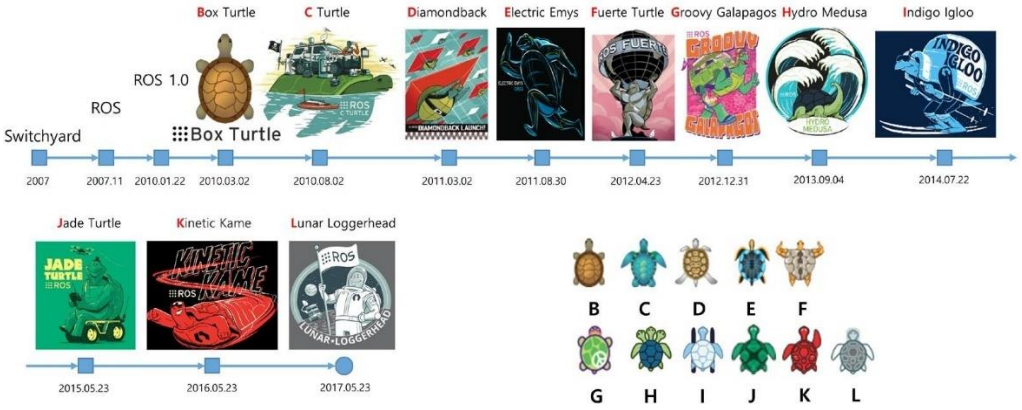
4. ROBOTSKI OPERATIVNI SUSTAV – ROS

ROS je jedan od najvećih pothvata u segmentu robotike na kojem je sudjelovalo puno ljudi i suradnika iz područja robotike. Sve većim razvojem i primjenom robotike u industriji, javila se potreba za razvojem sustava koji je otvoren i svima dostupan te ga je lako za koristiti. Početak ROS-a seže još u 2007. kada je Morgan Quigley, student sveučilišta Stanford, pokrenuo projekt robotskog sustava, kojeg je nazvao *Switchyard*, dostupnog svima kao podrška projektu STAIR (*engl. Stanford Artificial Intelligence Robot).* ROS je od samog početka razvijan u više različitih laboratorija za robotiku što je isprva izgledalo loše i nije se vjerovalo da će ROS zaživjeti. Kasnije se ispostavilo da je glavna značajka ROS-a u tome što svaka skupina istraživača može razvijati vlastiti repozitorij, podijeliti ga na serverima s ostalim korisnicima, ali i dalje zadržati vlasništvo i potpunu kontrolu nad istim.

ROS je operativni sustav za robote, kojem je glavna zadaća pojednostaviti razvoj softvera za robote. ROS objedinjuje veliku zbirku alata, knjižnica i konvencija koje omogućuju to pojednostavljeno stvaranje iznimno složenih sustava. Razvoj određenih algoritama je iznimno kompleksan i teško je očekivati da će pojedinac ili skupina istraživača uspjeti sama istražiti i razviti sve algoritme sama. U tom slučaju, prednost ROS-a je u tome što jedan laboratorij, specijaliziran za mapiranje i lokalizaciju, može raditi isključivo na razvoju tih algoritama, drugi laboratorij, specijaliziran za navigaciju, može raditi na razvoju planera trajektorija, a treći laboratorij može razvijati napredne vizijske sustave. Svaki laboratorij ili institucija može svoje klase spremiti na server te ih potom ostali korisnici ROS-a mogu koristiti. Važno je da se prilikom razvoja softvera za ROS, poštuju određena pravila kako bi svi mogli koristiti razvijene klase.

ROS je operativni sustav, međutim za razliku od standardnih operativnih sustava, ROS se može koristiti za različite kombinacije hardvera. ROS pruža razna okruženja koja su usko specijalizirana za određene robotske primjene.

Prva službena verzija ROS-a, bila je *ROS 0.4 Mango Tango* koja je izašla 01.01.2009. Kroz godine se ROS nadograđivao i postao sve bolji, a posljednja verzija je izašla 23.05.2017. pod nazivom *ROS Lunar Loggerhead*. Trenutno je aktualan ROS2 na kojem se još radi, a vjeruje se kako će biti još bolji od ROS-a. Slika 17. prikazuje vremensku crtu razvoja ROS-a.



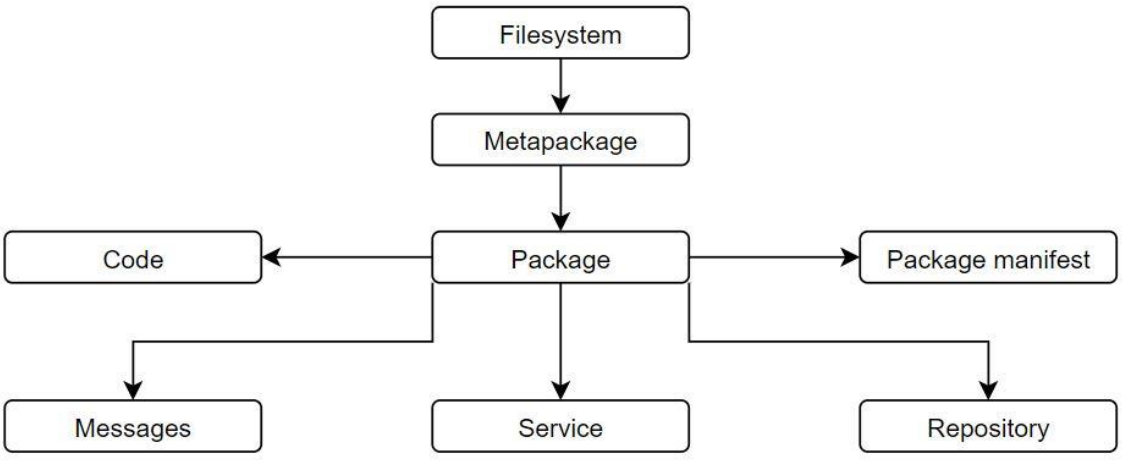
Slika 17. Vremenski prikaz razvoja ROS-a [13].

4.1. Arhitektura ROS-a

Arhitektura ROS-a je iznimno bitna jer omogućuje jednostavan razvoj softvera za robote. Arhitektura se sastoji od: *filesystem* razine, *computation* razine i *community* razine.

4.1.1 Filesystem razina

Glavni cilj ove razine je centralizacija procesa izgradnje i podizanja projekta, ali i pružanje fleksibilnosti za decentralizaciju njegovih podsustava. Ovakav način razvoja softvera omogućuje strukturirano i jednostavno snalaženje u programu samog robota te jednostavno dodavanje novih dijelova programa potrebnih za obavljanje zadataka robota. Slika 18. prikazuje *filesystem* razinu.



Slika 18. *Filesystem* razina

Metapackages (*hrv. meta paketi)* je naziv za skup više paketa koji služe istoj svrsi. Jedan od primjera je navigacija, u kojoj se nalaze paketi za lokalizaciju, planiranje trajektorija, estimaciju pozicije i zaobilaženje prepreka.

Packages (hrv. paketi) su najvažniji dio ROS okruženja iz razloga što se u njima nalaze *Nodes(hrv. čvorovi)* koji služe za izvođenje svih procesa. U paketima se također nalaze i ostale datoteke i biblioteke koje služe za konfiguraciju i rad s čvorovima.

Packages manifest (*hrv. manifest paketa)* se očituje u vidu datoteke *package.xml* u kojem se nalaze svi važni segmenti potrebni za izgradnju softvera u ROS-u. Oni služe za modifikaciju ukoliko je to potrebno, a u sebi sadrže i informacije poput autora, licenci i zavisnosti o kompajlerima.

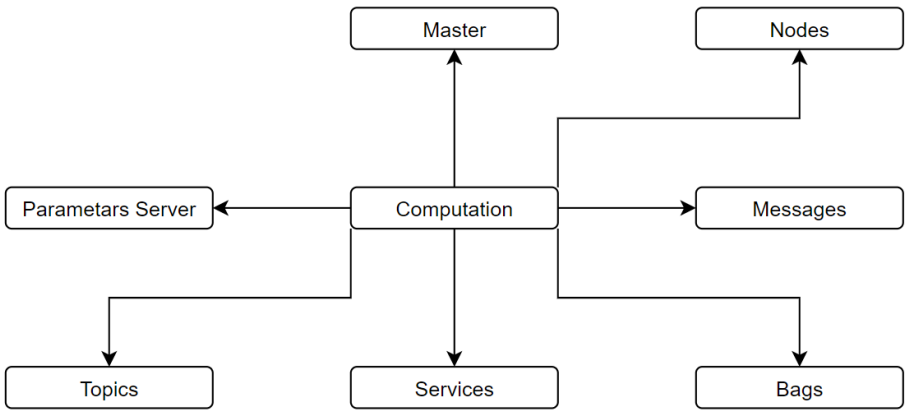
Services (hrv. servisi) su segmenti paketa koji rade na principu pitanje-odgovor, a služe za interakciju među procesa unutar ROS-a. Servisi se nalaze u *srv* datoteci radnog prostora čvora.

Repository (hrv. repozitorij) služi za redovno osvježavanje novijih verzija unutar ROS-a. Najpoznatiji softver za održavanje je *Git* kojeg se koristi i za ROS pakete.

Messages su vrste datoteka koje služe za razmjenu podataka između više procesa unutar ROS-a, poruke se definiraju i spremaju u mapu *msg* unutar paketa s nastavkom *.msg*.

4.1.2. Computation razina

Ova razina ROS sustava se bavi procesima koji se nazivaju čvorovima, odnosno mrežom istih. Svaki čvor u mreži pristupiti istoj toj mreži te imati interakciju s ostalim čvorovima ukoliko razvojni programer to omogući. Osim interakcije s čvorovima, moguće se ostvariti slanje raznih podataka na mrežu gdje čvorovi imaju mogućnost pristupa istoj. Struktura *Computation(hrv. razvojne)* razine prikazana je na slici 19.



Slika 19. Prikaz *Computation* razine

Master je središnji dio ROS sustava koji je nužan za rad sustava. On nadzire rad svih čvorova, poruka i servisa te komunikaciju među njima. Na računalu može biti pokrenut samo jedan ROS master. *Master* radi na principu DNS servera, gdje se prilikom pokretanja čvorova isti spaja na *Master* i započinje rad s ostalim čvorovima. Takav sustav radi na principu TCP/IP protokola gdje je preko varijable *ROS_MASTER_URI* definirana IP adresa ROS *Mastera* preko koje se čvorovi spajaju na isti.

Nodes(hrv. čvorovi) su osnovna jedinica ROS sustava. U njima se nalaze proces koji izvode radnje. Čvorovi su pisani u programskim jezicima C++ i Python pomoću knjižica *roscpp i rospy*. Procesi koji se izvode u čvorovima su srž ROS sustava te imaju mogućnost međusobne komunikacije. Primjer korištenja čvorova je da se jedan čvor brine za proračun potrebnih brzina motora te iste te brzine šalje na kontroler motora, dok drugi čvor radi planiranje trajektorije robota. Glavna prednost korištenja čvorova je u tome što ukoliko jedan od segmenata rada robota ne radi, ne mora se analizirati i popravljati cijeli programski kod, nego je dovoljno vidjeti koji segment robota ne radi i samo čvor zadužen za isti popraviti.

Parametars server(hrv. server parametara) je server koji omogućuje spremanje varijabli na principu ključ brava što omogućuje čvorovima da pristupaju serveru te brišu, mijenjaju ili čitaju parametre sa servera parametara.

Topics (hrv. teme) služe kako bi čvorovi lakše izmjenjivali poruke. Čvor može objaviti poruku na temu te je u tom slučaju *publisher(hrv. objavljivač)* ili ukoliko se pretplati na temu te prima poruke koju objavljene na temi postaje *subscriber(hrv. pretplatnik)*.

4.1.3. Community razina

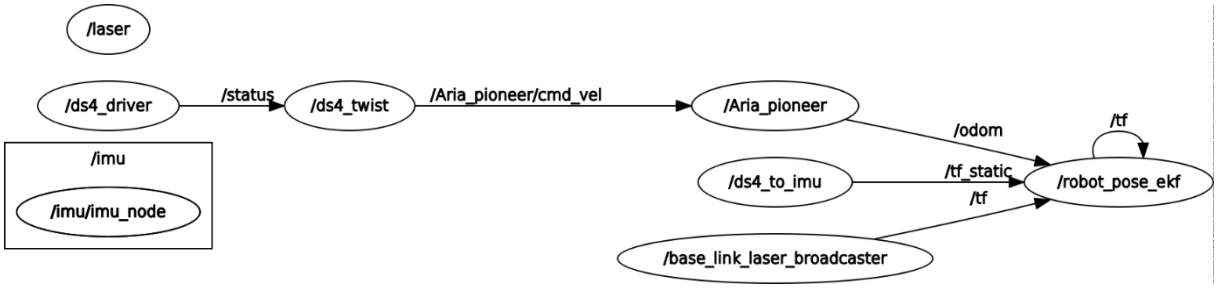
Ova razina omogućuje razmjenu paketa između pojedinaca, istraživača i ostalih zajednica koje koriste ROS okruženje. Također ova zajednica omogućuje novim korisnicima da se lakše snađu u ROS-u koji je za početnike jako kompliciran. Osim razmjene iskustava i pomoći, ROS zajednica služi za razmjenu ROS paketa, opis istih i način korištenja. Na stranicama ROS-a [13] mogu se pronaći odgovori na sva važna pitanja, kao i e-mail adrese razvojnih programera za pojedine pakete što omogućuje novim korisnicima pojedinih paketa direktan komunikacijski kanal s razvojnim programerom tog paketa.

5. IMPLEMENTACIJA ROS-A NA MOBILNOG ROBOTA

ROS je danas moguće instalirati na sve operativne sustave, Linux, Windows i macOS. U samim početcima ROS je bilo moguće instalirati isključivo na Linux, no do danas su ga programeri prilagodili za sve operativne sustave. Prilikom odabira verzije ROS-a, potrebno je obratiti pozornost na vrstu i verziju instaliranog operativnog sustava kako bi se mogla odabrati adekvatna verzija ROS-a.

Kao što je prije navedeno, na mobilnog robota *Pioneer 2DX* postavljeno je računalo *Nvidia Jetson TX2* na koje je instaliran operativni sustav *Linux Ubuntu 16.04.* za kojeg je predviđeno korištenje *ROS Kinetic Kame* okruženja. Razlog tome je taj što je odabrana verzija ROS-a najstarija stabilna verzija koja se još uvijek nadograđuje i nudi mogućnost instalacije paketa *RosAria*. Osim toga, odabrana verzija je trenutno najkorištenija verzija stoga je otklanjanje programskih pogrešaka jednostavno jer postoji zajednica koja nailazi na slične ili iste probleme.

Slika 20. prikazuje graf nužnih čvorova koji se koriste za rad s mobilnim robotom. Slika je generirana čvorom *rqt_graph* koji služi za grafički prikaz čvorova koji su pokrenuti i tema na koje objavljuju ili se pretplaćuju.



Slika 20. Prikaz čvorova za rad mobilnog robota

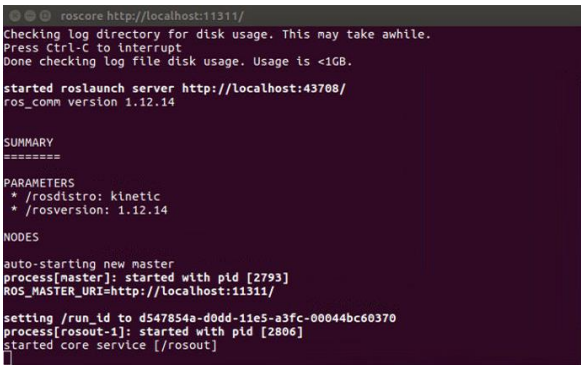
5.1. *ROSARIA*

ROSARIA je paket u kojem se nalazi čvor *RosAria* koji omogućava povezivanja ROS okruženja s *Pioneer 2DX* robotom. Osim tog robota, *RosAria* ima mogućnost spajanja sa svim robotima koji koriste okruženje *ARIA* kako bi se njima upravljalo. *RosAria* omogućuje primanje vrijednosti sonarnih senzora i enkodera s mobilnog robota, isto kao i slanje komandi za brzine motora. *RosAria* je najvažniji čvor jer omogućuje gore navedene radnje, a ostali paketi nude mogućnosti mapiranja, lokalizacije i navigacije čije se informacije šalju robotu.

Prvo je potrebno pokrenuti *Master* čvor. Čvorovi se pokreću iz Terminala upisivanjem sljedeće naredbe:

```
$ roscore
```

Na slici 21. je prikazano pokretanje *Master* čvora s kojeg se vidi IP adresa Master čvora.

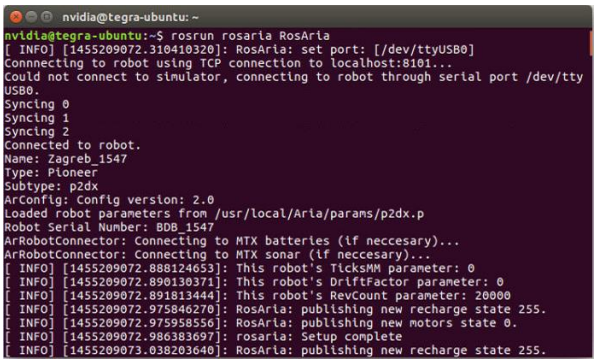


Slika 21. Pokretanje Master čvora

Nakon što je pokrenut *Master* čvor možemo pokrenuti *RosAria* čvor primjenom sljedeće naredbe:

```
$ rosrun rosaria RosAria
```

Na slici 22. je prikazano pokretanje *RosAria* čvora koji inicijalizira sve podsustave robota kao što su motori, enkoderi, i sonarni senzori.



Slika 22. Pokretanje RosAria čvora

Nakon pokretanja, *RosAria* čvor se pretplaćuje na temu:

- *cmd_vel* – navedena tema služi za razmjenu informacija o brzinama koje robot mora ostvariti. Čvorovi kao što su *ds4_driver* i *teleop_twist_keyboard* objavljuju na tu temu brzine koje korisnik zadaje preko tipkovnice ili drugog upravljača, dok čvor *move_base* na tu temu objavljuje izračunate brzine koje robot mora postići kako bi odradio određeni zadatak.

Teme na koje čvor *RosAria* objavljuje su:

- *pose* – na ovu temu *RosAria* objavljuje informacije o odometriji robota. Odometrija mobilnog robota omogućuje određivanje položaja samog robota

- *sonar* – služi za objavljivanje vrijednosti prednjih ultrazvučnih senzora ukoliko postoji pretplatnik na iste
- *battery_state_of_charge* – objavljuje stanje napunjenosti baterije, gdje je 0.0 vrijednost skroz prazne baterije, a 1.0 maksimalno napunjena baterija
- ostale teme koje *RosAria* čvor objavljuje, a nisu nužne za rad robota prilikom mapiranja, lokalizacije i navigacije su: *sonar_pointcloud2*, *battery_voltage*, *battery_recharge_state* i *motor_state*

Parametri koji se učitavaju pokretanjem čvora *RosAria* su:

- *odom_frame* (zadano: *odom*) – parametar kojim se određuje u odnosu na koji koordinatni sustav će čvor objavljivati informacije o odometriji robota
- *base_link_frame* (zadano: *base_link*) – ime na koje će se objavljivati koordinatni sustav robota
- ostali parametri koji se koriste, ali nemaju *trans_accel*, *trans_decel*, *rot_accel*, *rot_decel*, *lat_accel*, *lat_decel*, *TicksMM*, *DriftFactor*, *RevCount*, *cmd_vel_timeout*, *bumpers_frame*, *sonar_frame*.

Servisi koje je moguće koristiti u sklopu čvora su: *enable_motors* i *disable_motors*

5.2. *ds4_driver*

Kako bi se robotom lakše upravljalo prilikom procesa mapiranja i lokalizacije koristit će se daljinski upravljač konzole *Playstation 4*. Odabrani upravljač ima mogućnost spajanja putem *bluetooth* komunikacije što je pogodno za odabrano računalo koje na sebi ima integriranu navedenu komunikaciju. Osim navedene komunikacije, upravljač je odabran iz razloga što je za njega napisan paket *ds4_driver* koji omogućuje spajanje upravljača na ROS kao i komunikaciju s istim. Prije pokretanja čvora za upravljanje robotom preko upravljača, potrebno je pokrenuti čvor *ds4_driver_node* koji služi za pokretanje *bluetooth* komunikacije i ostalih segmenata potrebnih za uspješan rad upravljača. *Ds4_driver_node* se pokreće na sljedeći način:

```
$ rosrun ds4_driver ds4_driver_node.py
```

Nakon što je uspješno pokrenut *ds4_driver_node*, potrebno je pokrenuti čvor koji je zaslužan za rad daljinskog upravljača – *ds4_twist_node*. Njegova zadaća je da prima ulazne podatke daljinskog upravljača, obradi ih i pošalje na definiranu temu (npr. *cmd_vel)*. Čvor za daljinski upravljač se pokreće na sljedeći način:

```
$ rosrun ds4_driver ds4_twist_node.py
```

Na slici 23. je prikazan odabrani upravljač kao i način pokretanja *ds4_driver* čvora.

Slika 23. Prikaz *Playstation 4* upravljača-lijevo i pokretanje *ds4_driver* čvora-desno

Nakon pokretanja gore navedenih čvorova za daljinski upravljač, potrebno je pritiskom *PS* i *Share* prekidača na upravljaču spojiti isti s računalom kako bi se ostvarila komunikacija i započeo rad.

Čvor *ds4_driver_node* se pretplaćuje na temu *set_feedback* koja služi za davanje povratnih informacija korisniku preko vizualnog LED segmenta i vibracija. Tema koja se objavljuje je *status.* Preko navedene teme se objavljuju informacije vezane za stanje napunjenosti baterije, stanje pritisnutih tipki ili zakrenutih gljiva upravljača te rotacije samog upravljača oko sve tri osi. Parametri koji se koriste prilikom pokretanja *ds4_driver_node* čvora su: *device_addr*, *backend, use_standard_msgs*, *deadzone*, *frame_id* i *imu_frame_id*. Navedeni parametri nemaju velik značaj za proces upravljanja robotom prilikom mapiranja prostora stoga se za sve navedene parametre koriste predefinirane vrijednosti koji se mogu pronaći na [14].

Ds4_twist_node se pretplaćuje na temu *status* na koju *ds4_driver_node* objavljuje informacije o stanju upravljača. Nakon prikupljanja tih informacija, čvor ih obrađuje sukladno parametrima koji se definiraju. Nakon obrade, čvor objavljuje nove informacije o potrebnim brzinama motora na temu *cmd_vel* na koju je robot preko čvora *RosAria* pretplaćen te se potom robot kreće sukladno naredbama koje je dobio od korisnika preko upravljača. Parametri koje se koriste za *ds4_twist_node* su *stamped*, *inputs* i *scales*. Vrijednosti koje se koriste za parametre *stamped* i *inputs* su predefinirane i mogu se pronaći na [14], dok parametar *scales*, koji se koristi kao faktori za skaliranje, je podešen kao što je navedeno niže:

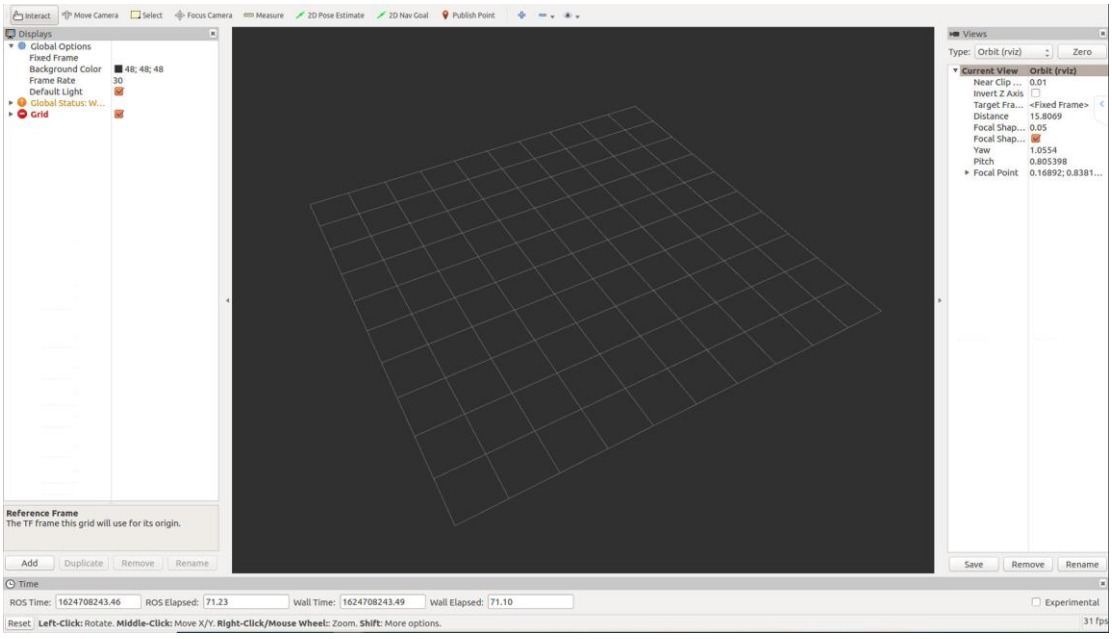
```
scales:
  linear:
    x: 0.2
    y: 0.2
    z: 1.0
  angular:
    x: 1.0
    y: 1.0
    z: 0.3
```

Ovi parametri su bitni za proces mapiranja jer se preko njih direktno utječe na brzinu robota korištenjem upravljača, a kvaliteta procesa mapiranja značajno ovisi o brzini robota. Kako se mobilni robot može kretati samo linearno u smjeru osi x i vršiti rotaciju oko osi z, ti su parametri bili podešavani u ovom radu dok se nisu dobili zadovoljavajući rezultati mapiranja.

5.3. *rviz*

Rviz je čvor koji služi za grafičku vizualizaciju raznih segmenata koji se dobivaju s robota. Također, u istom se mogu vizualizirati razni procesi poput mapiranja, lokalizacije i navigacije. *Rviz* omogućuje da korisnik učita postojeću mapu prostora, lokalizira se u istoj te postavi željeni cilj u kojeg želi da robot stigne. Osim navedenih mogućnosti, postoje i ostale koje će biti objašnjene u nastavku. Naredba niže pokreće *rviz* dok slika 24. prikazuje sučelje *rviz*-a.

```
$ rosrun rviz
```



Slika 24. Prikaz *rviz* sučelja

S lijeve strane sučelja se nalaze opcije koje omogućuju *rviz* čvoru da sluša poruke koje čvorovi izmjenjuju te ih ukoliko postoji mogućnost vizualizira. Globalne postavke koje su vezane na samo sučelje su *Fixed Frame* kojim se definira glavni koordinatni sustav u odnosu na kojeg se

primjenom matrica transformacije, odnosno čvorom *tf* prikazuju svi ostali koordinatni sustavi koji se koriste prilikom rada robota. *Background Color*, *Frame Rate* i *Default Light* su postavke koje definiraju pozadinsku boju, broj slika u sekundi i osvjetljenje samog sučelja. Paljenje opcije *Grid* omogućuje prikaz mreže unutar sučelja.

Segmenti koji se mogu dodati u *rviz*, a važni su za procese mapiranja, lokalizacije i navigacije (koji su detaljnije objašnjeni u naknadnim poglavljima) su:

- *LaserScan* – nudi opciju pretplaćivanja na teme vezane uz lidar senzore koji objavljuju informacije o kutu zakreta i udaljenosti predmeta te vizualizaciju dobivenih podataka
- *Map* – služi za vizualizaciju mape prilikom procesa mapiranja, učitavanje već snimljene mape ili prikaz mape troškova koja služi za detekciju dinamičkih prepreka prilikom kretanja robota po referentnoj trajektoriji.
- *Odometry* – je segment *rviz*-a koji je namijenjen za vizualizaciju odometrije, odnosno pozicije i orijentacije mobilnog robota u prostoru
- *Path* – nudi mogućnost pretplaćivanja i vizualizacije trajektorija prilikom navigacije mobilnog robota u prostoru. Postoje mogućnosti pretplate na globalnu trajektoriju na razini cijele mape ili na lokalnu trajektoriju koja se generira kako bi robot pratio globalnu.
- *Polygon* – u sklopu ROS-a se mogu definirati razni poligoni koji se vizualiziraju u *rviz*-u, a u slučaju mobilne robotike to su najčešće gabaritne mjere robota koje služe za proces navigacije(*robot_footprint*).
- *Pose* – služi za vizualizaciju poruka *geometry_msgs::PoseStamped*, a u ovom radu se koristi za prikaz željenih ciljeva prilikom procesa navigacije.
- *PoseArray* – vizualizira poruke *geometry_msgs::PoseStamped* u vidu oblaka strelica, a koristi se prilikom procesa lokalizacije kako bi se prikazale moguće pozicije robota. U početku je oblak pozicija raštrkan, a s vremenom vožnje konvergira i smanjuje se.
- *TF* – vizualizira transformacije između koordinatnih sustava

Ostali segmenti koji se mogu vizualizirati su: *Axes*, *Camera*, *DeepthCloud*, *Effort*, *FluidPressure*, *Grid*, *GridCells*, *Illuminance*, *Image*, *InteractiveMarkers*, *Marker*, *MarkerArray*, *PointCloud*, *PointCloud2*, *PointStamped*, *PoseWithCovariance*, *Range*, *RelativeHumidity*, *RobotModel*, *Temperature*, *WrenchStamped*.

Alatna traka *rviz*-a sadrži alate za upravljanje kamerom unutar sučelja. Osim navedenih alata, na alatnoj traci se nalaze i alati:

- *2D Pose Estimate* – je alat koji služi za estimaciju pozicije robota prilikom učitavanja mape, a koristi se kao pomoć algoritmu za lokalizaciju
- *2D Nav Goal* – služi za određivanje cilja kojeg želimo poslati robotu kako bi izvršio navigaciju do istog
- *Publish Point* – je namijenjen za objavljivanje proizvoljne točke na mapi

S desne strane sučelja se nalaze alati koji služe za namještanje pogleda na ono što vizualiziramo, dok su na dnu sučelja informacije o simulaciji poput vremena i broja slika u sekundi.

5.4. *imu_bno055*

Ovaj paket omogućuje komunikaciju između IMU senzora i računala na kojem je pokrenut ROS. IMU šalje podatke za rotaciju oko svih osi kao i linearne i kutne akceleracije robota. Primjena IMU senzora je važna iz razloga što zajedno s enkoderima robota uz primjenu proširenog Kalman filtera nudi mogućnost bolje estimacije odometrije mobilnog robota.

Komunikacija između IMU senzora i robota ostvarena je putem I2C protokola. I2C komunikacija može stvarati probleme ukoliko dolazi do velikih vibracija, međutim kako se robot kreće po ravnoj plohi taj problem je izbjegnut. IMU senzor je potrebno smjestiti dovoljno visoko u odnosu na motore, jer elektromagnetna zračenja nastala rotacijom motora mogu stvarati smetnje prilikom korištenja IMU senzora. Pokretanje čvora nadležnog za IMU senzor se vrši preko sljedeće naredbe:

```
$ rosrun imu_bno055 bno055_i2c_node
```

Pokrenuti čvor se ne pretplaćuje na teme, nego samo služi za objavljivanje. Teme na koje objavljuje su:

- *data* – fuzija svih podataka s IMU senzora
- *raw* – sirovi podaci s akcelerometar
- *mag* – sirovi podaci s magnetometra
- *temp* – podaci o temperaturi
- *status* –služi za informacije o kalibraciji, prekidima i stanju IMU senzora.

Parametri koje je potrebno pozvati prilikom pokretanja čvora su *device* i *address* koji služe za definiranje putanje(*/dev/i2c-1)* do uređaja i I2C adresu(*0x28*.) uređaja. Servisi koje je moguće pozvati prilikom rada čvora su:

- *reset* – servis koji služi za resetiranje samog uređaja
- *calibrate* – servis koji pokreće automatsku kalibraciju senzora

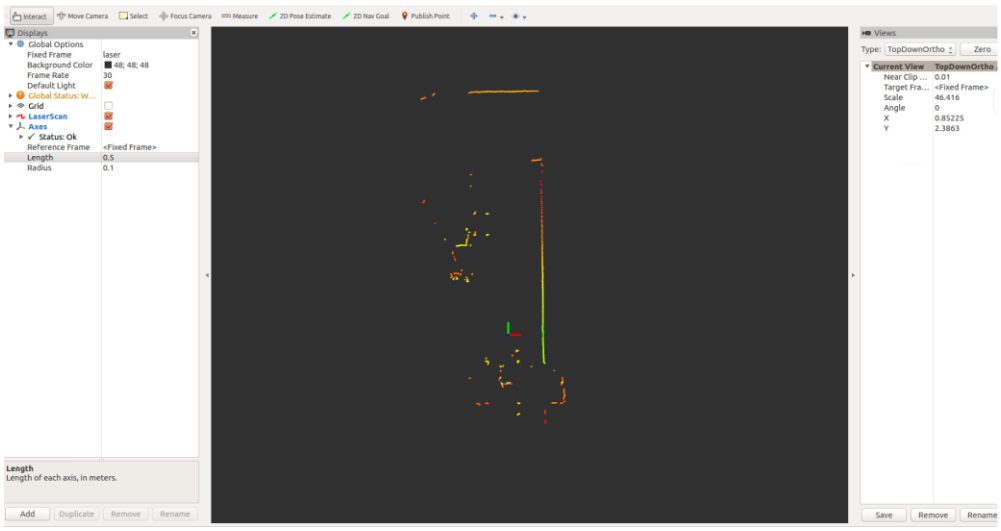
5.5. *urg_node*

Urg_node paket sadrži istoimeni čvor koji služi za pokretanje i čitanje podataka s lidar senzora tvrtke *Hokuyo*. Kako je lidar spojen s računalom preko *ethernet* komunikacijskog protokola, prilikom pokretanja čvora potrebno je definirati parametar adrese lidar senzora kako bi se komunikacija ostvarila. Pokretanje *urg_node* čvora se vrši na sljedeći način:

```
$ rosrun urg_node urg_node _ip_address:=192.168.0.10
```

Prilikom pokretanja *urg_node* čvora, isti počinje sa objavljivanjem na temu *scan* u kojoj se nalaze informacije vezane za kut zakreta i udaljenost predmeta. Na tu temu se pretplaćuju ostali čvorovi poput *gmapping* ili *amcl* čvora koji služe za mapiranje i lokalizaciju mobilnog robota preko lidar senzora.

Kao što je navedeno gore, potrebno je definirati parametar *ip_address*, a ostali parametri koji se mogu postaviti su: *angle_max*, *angle_min*, *cluster*, *frame_id*, *skip*, *tf_prefix* i *time_offset*. Za navedene parametre se koriste predefinirane vrijednosti koje se mogu pronaći na [15]. Na slici 25. je prikazano očitavanje lidar senzora u *rviz* sučelju.



Slika 25. 2D prikaz laserskog očitanja

Koordinatni sustav vidljiv na slici (x – crvene boje, y – zelene boje) predstavlja položaj lidara, a točke predstavljaju laserska čitanja sa senzora.

5.6. *robot_pose_ekf*

Čvor kojem je glavna svrha da primjenom proširenog Kalman filtra što točnije odredi odometriju, odnosno poziciju robota. Čvor se pokreće na sljedeći način:

```
$ rosrun robot_pose_ekf robot_pose_ekf
```

Prilikom pokretanja čvora, isti se pretplaćuje na sljedeće teme:

- *odom* – 2D pozicija i orijentacija robota na tlu koja se dobije iz odometrije kotača robota
- *imu_data* – 3D orijentacija koja sadrži informacije o Eulerovim kutovima, a dobiva se s IMU senzora
- *vo* – vizualna odometrija koja prezentira potpunu poziciju i orijentaciju robota kao i kovarijance pozicije

Tema na koju čvor objavljuje je *odom_combined*, a sadrži podatke koje je prošireni Kalman filter dobio iz ulaznih podataka. Osim toga, čvor daje transformaciju iz koordinatnog sustava *odom_combined* u *base_footprint*.

Kako bi prošireni Kalman filter uspješno radio potrebno je definirati sljedeće parametre:

- *output_frame* – koordinatni sustav u odnosu na kojeg prošireni Kalman filter daje rezultate
- *odom_used* – parametar kojim se određuje hoće li se za algoritam koristiti odometrija kotača robota
- *imu_used* – parametar kojim definiramo hoće li se koristiti IMU senzor prilikom korištenja algoritma
- *vo_used* – parametra kojim definiramo koristi li algoritam vizualnu odometriju

Osim navedenih parametara koriste se i sljedeći parametri: *freq*, *sensor_timeout*, *debug*, *self_diagnose*. Niže je prikazano definiranje svih parametara potrebnih za rad *robot_pose_ekf* čvora.

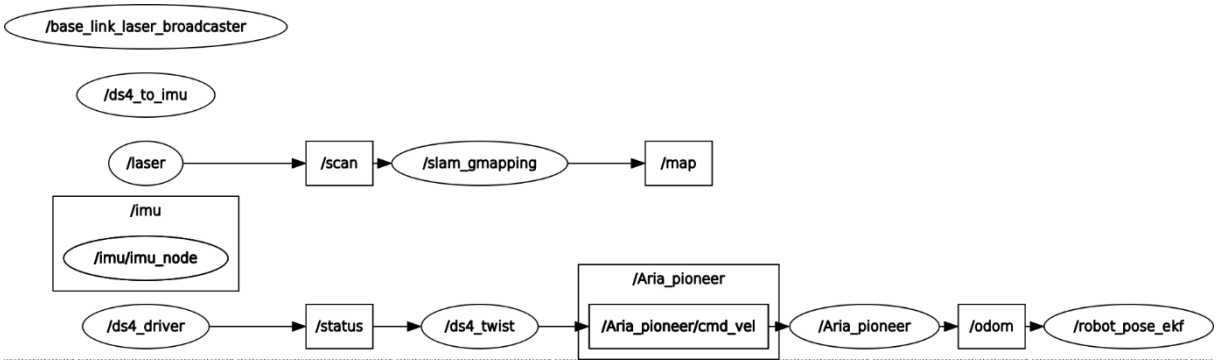
```
output_frame: odom
freq: 30.0
sensor_timeout: 1.0
odom_used: true
imu_used: true
vo_used: false
debug: false
self_diagnose: false
```

6. MAPIRANJE PROSTORA PRIMJENOM ROS-a

U računalnoj geometriji i robotici, *simultaneous localization and mapping – SLAM (hrv. simultana lokalizacija i mapiranje)* postoji računski problem stvaranja ili nadogradnje mape nepoznatog prostora. Tijekom tog procesa potrebno je simultano pratiti lokaciju mobilnog robota unutar kreirane mape. Postoji nekoliko algoritama koji rješavaju ovaj problem za određene okoline. Popularna rješenja uključuju filter čestica, prošireni Kalman filtar, presijecanje kovarijanci i *GraphSLAM*. Algoritmi za *SLAM* se koriste u navigaciji, mapiranju i lokalizaciji mobilnih robota, također se koriste i za proračun odometrije robota, ali i za virtualnu i proširenu stvarnost.

SLAM algoritmi su prilagođeni raspoloživim računalnim resursima, stoga nisu usmjereni na savršenstvo već na postizanje zadovoljavajućeg rješenja. Navedeni proces mapiranja se koristi u autonomnim automobilima, bespilotnim letjelicama, podmornicama, svemirskim roverima, mobilnim robotima, a u posljednje vrijeme se koriste za mapiranje ljudskog tijela.

Slika 26. prikazuje čvorove koji se koriste pokretanjem *mapping.launch* datoteke kako bi se pokrenuo proces mapiranja.



Slika 26. Prikaz čvorova koji sudjeluju u procesu mapiranja prostora

6.1. *gmapping*

ROS paket odgovoran za mapiranje prostora je *gmapping*. Taj paket koristi *GMapping* algoritam baziran na iznimno efikasnom *Rao-Blackwellized* filteru čestica pomoću kojeg se stvara mapa prostora bazirana na očitanjima lidar senzora [16].

Algoritam su razvili G. Grisetii, C. Stanchniss i W. Burgard. *Rao-Blackwellized* filtri su uvedeni kao učinkovito sredstvo rješavanja problema istodobne lokalizacije mapiranja. Ovakav pristup koristi filtar čestica u kojem svaka čestica nosi kartu okoline. Sukladno tome, ključno pitanje je kako smanjiti broj čestica, a da mapa što bolje prezentira snimljenu okolinu. Predložen pristup izračunava točnu raspodjelu prijedloga točaka uzimajući u obzir ne samo kretanje robota

već i najnovije promatranje okoline. Osim navedenog, primjenjuje se i pristup za selektivnog uzimanja uzoraka što drastično smanjuje problem velikog broja čestica.

Gmapping čvor se pokreće sljedećom naredbom:

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

Pokretanje *gmapping* čvora, isti se pretplaćuje na sljedeće teme:

- *tf* – potrebne transformacije vezane za koordinatne sustave lasera, baze robota i odometrije
- *scan* – podaci vezani za skeniranja lidar senzora

Teme koje objavljuje čvor su:

- *map_metadata* – opći podaci vezani za kreiranu mapu poput: vremena nastanka mape, rezolucije, visine, širine i ishodišta same mape
- *map* – sadrži podatke koji reprezentiraju 2D mapu u kojoj svaka ćelija predstavlja vjerojatnost da se objekt tamo nalazi. Vrijednosti ćelija mogu varirati od 0 do 100
- *entropy* – procjena entropije raspodjele s obzirom na poziciju robota (veća vrijednost ukazuje na veću nesigurnost)

Gmapping nudi mogućnost pozivanja servisa *dynamic_map* koji omogućuje pozivanje podataka o mapi u bilo kojem trenutku.

Kako bi proces mapiranja prostora bio što kvalitetniji, potrebno je postaviti određene parametre:

- *odom_frame* – koordinatni sustav odometrije mobilnog robota
- *base_frame* – koordinatni sustav mobilnog robota
- *map_frame* – koordinatni sustav kreirane mape
- *map_update_interval* – učestalost osvježavanja mape, smanjivanje ovog broja povećava frekvenciju osvježavanja na račun većeg opterećenja procesora
- *maxUrange* – maksimalna iskoristiva veličina zrake lasera, preporučuje se da bude malo manja od maksimalne udaljenosti koju laser može očitati
- *maxRange* – maksimalna udaljenost koju senzor može detektirati, ukoliko se ispod te udaljenosti ne nalazi neki objekt, algoritam to shvaća kao prazan prostor
- *particles* – broj čestica unutar filtera koji se koristi za proces mapiranja
- *linearUpdate* – parametar koji definira svakih koliko metara u linearnom smjeru robota će se procesuirati podaci dobiveni s lidar senzora

- *angularUpdate* – parametar koji definira svakih koliko radijana rotacije oko z osi robota će se procesuirati podaci dobiveni s lidar senzora
- *temporalUpdate* – vrijeme koje je potrebno proći da robot stoji na mjestu da bi se procesuirali podaci dobiveni sa senzora
- *delta* – rezolucija mape (metri po ćeliji zauzetosti mape)
- *sigma* – parametar koji se koristi kako bi se odredilo podudaranje krajnjih točaka mape
- *kernelSize* – kernel u kojem se traže korespondencija prilikom skeniranja
- *ogain* – pojačanje za ocjenjivanje vjerojatnosti, služi za zaglađivanje
- ostali parametri koji se učitavaju prilikom korištenja *gmapping* čvora su: *throttle_scans, lstep, astep, iterations, lsigma, lskip, minimumScore, srr, srt, stt, resampleThreshold, xmin, ymin, ymax, ymax, llsamplerange, llsamplestep, lasamplerange, lasamplestep, transform_publish_period, occ_thresh*

Kako bi *gmapping* kvalitetno radio, potrebno je s obzirom na vrstu zadatka, odnosno prostorije podesiti parametre. Većina parametara se iskustveno postavlja, te se po potrebi mijenja kako bi se dobila što kvalitetnija mapa prostora. Niže su dane vrijednosti parametara koji su postavljeni za svrhu ovog rada.

```
odom_frame: odom
base_frame: base_footprint
map_frame: map
map_update_interval: 0.5
maxUrange: 9.5
maxRange: 10.5
particles: 100
linearUpdate: 0.1
angularUpdate: 0.1
temporalUpdate: 2.0
delta: 0.05
sigma: 0.05
kernelSize: 1
ogain: 3.0
```

Ostali parametri su korišteni kao predefinirani od strane kreatora *gmapping* čvora, a njihove se vrijednosti mogu pronaći na [16].

Gmapping čvor osim parametara, zahtjeva i transformacije među koordinatnim sustavima, tako je za uspješan proces mapiranja, potrebno osigurati sljedeće transformacije:

- *laser* → *base_footprint* – ova transformacija osigurava poznavanje pozicije lasera relativno u odnosu na središte samog robota. Poznavanje ove transformacije osigurava kvalitetno skeniranje prostora u odnosu na središte samog robota.

- *base_footprint* → *odom* – transformacija potrebna kako bi se mogla znati odometrija robota, odnosno točna pozicija robota.

Transformacija koju osigurava *gmapping* čvor je:

- *map* → *odom* – potrebna transformacija kako bi se mogla odrediti pozicija robota unutar kreirane mape.

6.2. *mapping.launch*

U sklopu ROS-a se svi čvorovi moraju pokretati iz terminala. Osim čvorova iz terminala se postavljaju i parametri potrebni za adekvatan rad čvorova.

Kako bi se postiglo kvalitetno pokretanja procesa mapiranja (osiguravanje da se svi potrebni parametri postave i čvorovi pokrenu) potrebno je kreirati *.launch* datoteku u kojoj su sadržane naredbe za pokretanje svih čvorova i postavljanje svih parametara potrebnih za rad robota. Takve datoteke su pisane u *XML* programskom jeziku, a njihovo pokretanje se vrši iz terminala primjenom naredbe *roslaunch*. Niže je dan primjer pokretanja *.launch* datoteke iz terminala.

```
roslaunch pioneer_robot mapping.launch
```

Pioneer_robot je ručno napravljen paket u kojem se nalaze *.launch* datoteke, konfiguracijske datoteke koje sadrže definirane parametre te datoteke u kojima su spremljene snimljene mape prostora.

```
<?xml version="1.0"?>
<launch>

<!-- POKRENI ROSARIU -->
  <node pkg="rosaria" type="RosAria" name="Aria_pioneer">
    <remap from="Aria_pioneer/pose" to="odom"/>
  </node>
```

U početku *.launch* datoteke se definira verzija *XML* programskog jezika, te s naredbom *<launch>* je označen početak *.launch* datoteke.

Potom se pokreće *RosAria* čvor iz *rosaria* paketa pod nazivom *Aria_pioneer*. Nakon pokretanja čvora, tema pod nazivom *Aria_pioneer/pose* se mijenja u *odom* kako bi se ostali čvorovi koji koriste odometriju robota mogli pretplatiti na tu temu.

```
<!-- POKRENI LIDAR -->
  <node pkg="urg_node" type="urg_node" name="laser">
    <rosparam>
      ip_address: 192.168.0.10
    </rosparam>
  </node>

<!-- TRANSFORMACIJA TF-A LIDAR-A -->
  <node pkg="tf" type="static_transform_publisher"
name="base_link_laser_broadcaster" args="0.16492 0 0 0 0 0 base_footprint
laser 100" />
```

Potom se pokreće čvor *urg_node* za lidar senzor s definiranim parametrom *ip_address*. Nakon pokretanja čvora za lidar, potrebno je pokrenuti čvor *tf* zaslužan za transformaciju koordinatnih sustava. Koordinatni sustav lidar senzora *laser* se postavlja na udaljenosti od 0.16492m u smjeru osi X u odnosu na koordinatni sustav baze robota *base_footprint*. Iznos transformacije u smjeru osi X je dobiven na način koji će kasnije biti objašnjen.

```
<!-- POKRENI IMU -->
  <include file="$(find imu_bno055)/launch/imu.launch" />

<!-- POKRENI KALMAN FILTER -->
  <include file="$(find pioneer_robot)/src/ekf.launch" />
```

Nakon pokretanja lidar senzora, pokreće se čvor *imu_bno055* na način da se u sklopu *mapping.launch* datoteke poziva druga *.launch* datoteka pod nazivom *imu.launch*. *Imu.launch* datoteka je dana u prilogu ovog rada, a u njoj se nalaze naredbe za pokretanje *imu_bno055* čvora i parametri koje je potrebno postaviti kako bi čvor radio.

Robot_pose_ekf čvor se pokreće također preko druge *.launch* datoteke pod nazivom *ekf.launch* koja također pokreće čvor zajedno sa svim parametrima potrebnim za adekvatan rad samog čvora.

Kako je za proces mapiranja važno voziti robota po nepoznatom prostoru kako bi robot skenirao i memorirao mapu prostora potrebno je pokrenuti daljinski upravljač za robota.

```
<!-- POKRENI JOYSTICK -->
  <arg name="addr" default="" />
  <arg name="dof" default="6" />
  <arg name="stamped" default="false" />
  <include file="$(find ds4_driver)/launch/ds4_driver.launch" >
    <arg name="addr" value="$(arg addr)" />
    <arg name="use_standard_msgs" value="false" />
  </include>
  <node pkg="ds4_driver" type="ds4_twist_node.py" name="ds4_twist"
output="screen" >
    <rosparam command="load" file="$(find ds4_driver)/config/twist_$(arg
dof)dof.yaml" />
    <param name="stamped" value="$(arg stamped)" />
    <remap from="cmd_vel" to="Aria_pioneer/cmd_vel"/>
  </node>
```

Prvo se pokreće čvor *ds4_driver* nužan za rad upravljača preko *ds4_driver.launch* datoteke u kojoj se nalaze definirani svi parametri. Nakon toga pokrenut je čvor *ds4_twist_node* zajedno sa svim potrebnim parametrima za primanje informacija s upravljača i slanja istih na temu *cmd_vel*, gdje je potom tema *cmd_vel* preimenovana u *Aria_pionneer/cmd_vel* kako bi kontroler robota, pretplaćen na istu, mogao čitati informacije i sukladno njima upravljati motorima robota.

```
<!-- POKRENI MAPIRANJE -->
  <arg name="scan_topic" default="scan" />
  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping">
    <rosparam>
      odom_frame: odom
      base_frame: base_footprint
      map_frame: map

      map_update_interval: 0.5 # Publish new map

      maxUrange: 9.5 # Should be just less than sensor range
      maxRange: 10.5 # Should be just greater than sensor range
      particles: 100 # Increased from 80

      # Update frequencies
      linearUpdate: 0.1
      angularUpdate: 0.1
      temporalUpdate: 2.0
      resampleThreshold: 0.5

      # Initial Map Siz
      xmin: -10.0 # was -100
      ymin: -10.0 # was -100
      xmax: 10.0 # was 100
      ymax: 10.0 # was 100
      delta: 0.05

      # All default
      sigma: 0.05
      kernelSize: 1
      lstep: 0.05
      astep: 0.05
      iterations: 5
      lsigma: 0.075
      ogain: 3.0
      lskip: 0
      llsamplerange: 0.01
      llsamplestep: 0.01
      lasamplerange: 0.005
      lasamplestep: 0.005

    </rosparam>
    <remap from="scan" to="$(arg scan_topic)"/>
  </node>
```

Pokretanje čvora za mapiranje *slam_gmapping* se ostvaruje također preko *.launch* datoteke gdje se učitavaju svi važni parametri objašnjeni u prethodnom poglavlju. Za parametre koji nisu definirani u *.launch* se uzimaju predefinirane vrijednosti.

```
<!-- POKRENI RVIZ -->
<node pkg="rviz" type="rviz" name="rviz">
</node>

</launch>
```

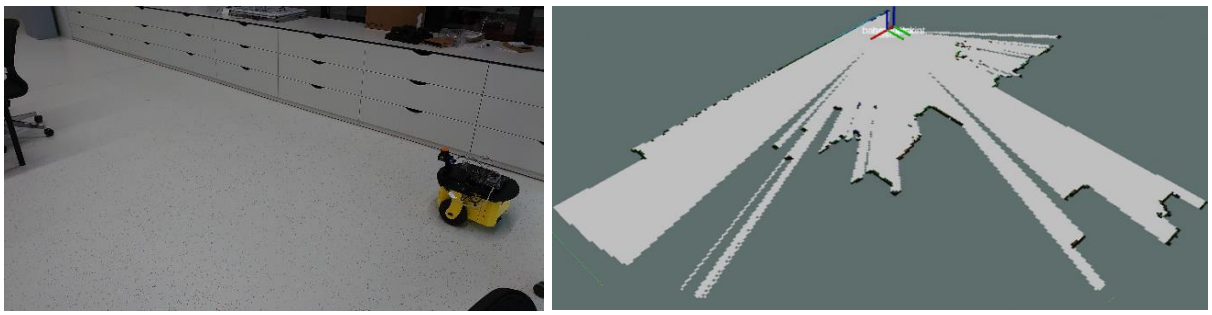
Na kraju se pokreće *rviz* čvor u kojem se može vizualno nadgledati proces mapiranja prostora te se završava *.launch* datoteka s naredbom *</launch>*.

Na kraju je važno napomenuti kako se prilikom pokretanja *.launch* datoteke, automatski pokreće i *rosmaster* te ga nije potrebno ručno pokretati.

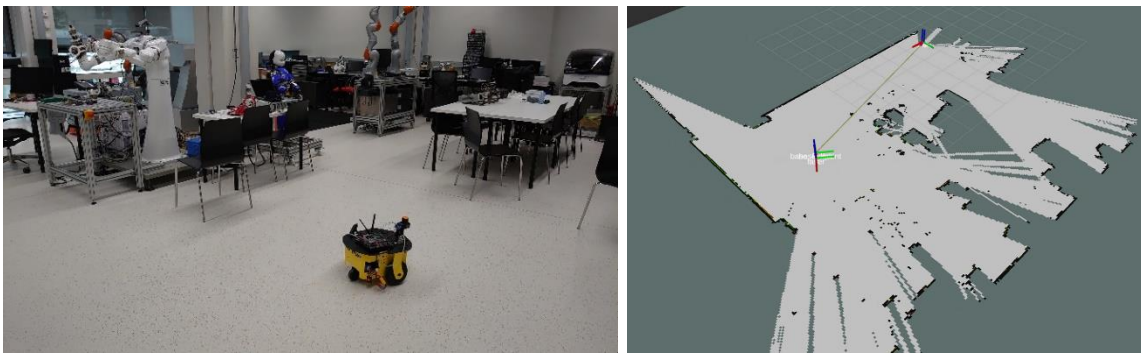
6.3. Proces mapiranja prostora

Proces mapiranja prostora započinje na način da se robota postavi na određenu poziciju u prostoru te pokretanjem *mapping.launch* datoteke počinje proces mapiranja. Nakon pokretanja mapiranja, potrebno je povezati daljinski upravljač *bluetooth* komunikacijom s robotom, kako bi ga se moglo upravljati po prostoru.

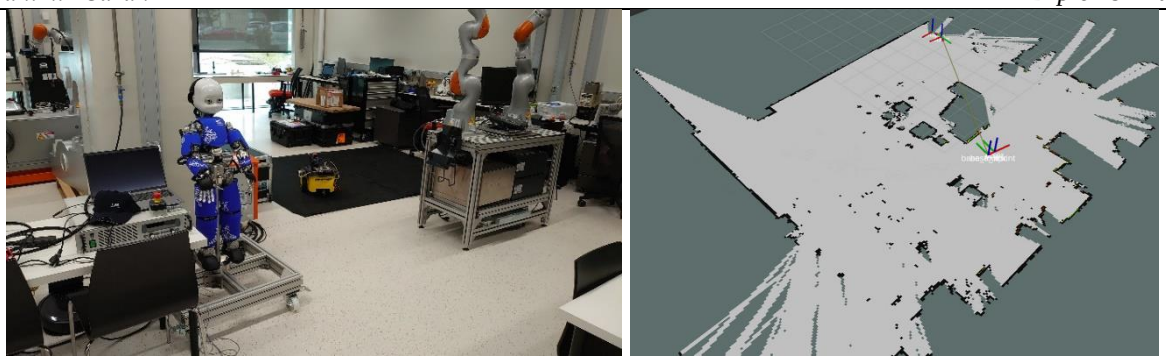
Za potrebe ovog rada izvršeno je snimanje *Regionalnog centra izvrsnosti za robotske tehnologije (CRTA)*. Na slikama 27., 28., 29., i 30. je dan proces snimanja *Laboratorija za računalnu inteligenciju – laboratorij L2.*



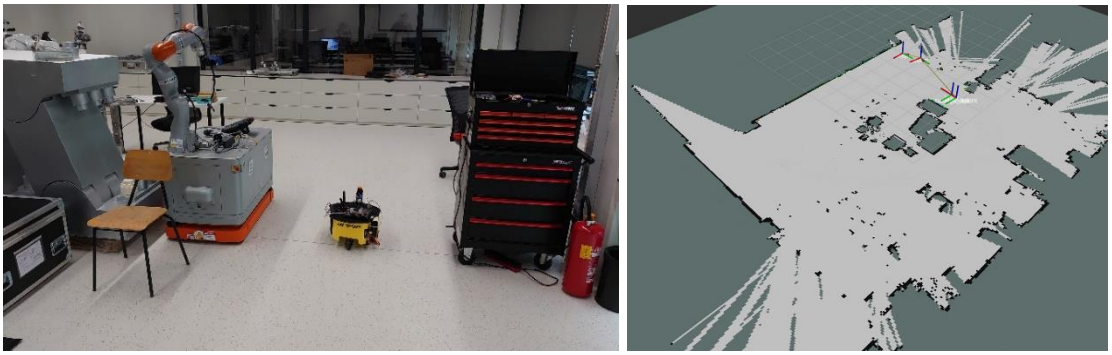
Slika 27. Početak mapiranja *Laboratorija L2*



Slika 28. ½ mapiranja *Laboratorija L2*



Slika 29. ¾ mapiranja *Laboratorija L2*



Slika 30. Kraj mapiranja *Laboratorija L2*

S gore prikazanih slika je vidljivo kako proces mapiranja zahtjeva prolazak kroz što više dijelova prostorije kako bi se dobila što vjernija mapa. Također, sa slika je vidljivo kako zrake lidar senzora prolaze kroz prozirne plohe poput stakla što uvelike može imati utjecaj. Stoga je potrebno obratiti pozornost da se prilikom mapiranja prostora koristi prostor koji nije okružen staklenim površinama. Na slikama je prikazan proces mapiranja samo jedne prostorije, međutim mapiranje prostora se obavilo na području cijelog laboratorija, te slika 30. prikazuje snimljeni tlocrt cijele *CRTA*-e. Po završetku mapiranja, mapa je spremljena na računalo kako bi se kasnije mogla pozvati i da se u njoj izvrši lokalizacija robota. Spremanje mape je izvršeno na način da se prvo otišlo u direktorij gdje se spremaju mape.

```
cd catkin_ws/src/pioneer_robot/maps
```

Te se pokrenuo čvor *map_saver* u sklopu paketa *map_server* kako bi se spremila mapa prostora.

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

Prilikom spremanja mape prostora, stvaraju se dvije datoteke, jedna s nastavkom *.yaml* i druga s nastavkom *.pgm*. U *.yaml* datoteci su sadržani parametri vezani za mapu, dok *.pgm* sadrži sami izgled mape. Parametri u *.yaml* datoteci su:

- *image* – sadrži putanju do *.pgm* datoteke koja prikazuje informacije o okupiranosti snimljenog prostora
- *resolution* – rezolucija mape (metar/piksel)
- *origin* – 2D pozicija ishodišnog koordinatnog sustava mape
- *occupied_thresh* – pikseli s vjerojatnošću okupiranosti većom od ovog parametra se smatraju kao okupiran prostor
- *free_thresh* – pikseli s vjerojatnošću okupiranosti manjom od ovog parametra se smatraju kao slobodan prostor
- *negate* - parametar za zamjenu bijelo/crnih piksela i slobodno/okupiranog prostora mape. Parametri *free_thresh* i *occupied_thresh* ne mijenjaju svoje značenje

Nakon spremanja mape pod nazivom *CRTA*, unutar *.yaml* datoteke se nalaze sljedeći parametri:

```
image: CRTA.pgm
resolution: 0.05
origin: [-10.0, -45.0, 0.0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

A *CRTA.pgm* je pretvorena u *CRTA.bmp* kako bi mogla biti prikazana u ovom radu. Na slici 31. je dan prikaz tlocrta *Regionalnog centra izvrsnosti za robotske tehnologije* snimljenog mobilnim robotom. Na slici je vidljiv problem kojeg uzrokuju stakleni zidovi zbog kojih zrake lasera prolaze kroz iste te je to rasipanje zraka vidljivo na slici.

Postoji mogućnost naknadne obrade slike tako da se u nekom od alata za grafičko uređivanje fotografija, ručno povuku crne linije na mjestima gdje su stakleni zidovi kako robot ne bi kasnije, prilikom navigacije zaključio da se tamo nalazi prazan prostor.

Slika 31. Tlocrt *CRTA*-e snimljen mobilnim robotom

7. LOKALIZACIJA ROBOTA U PROSTORU PRIMJENOM ROS-a

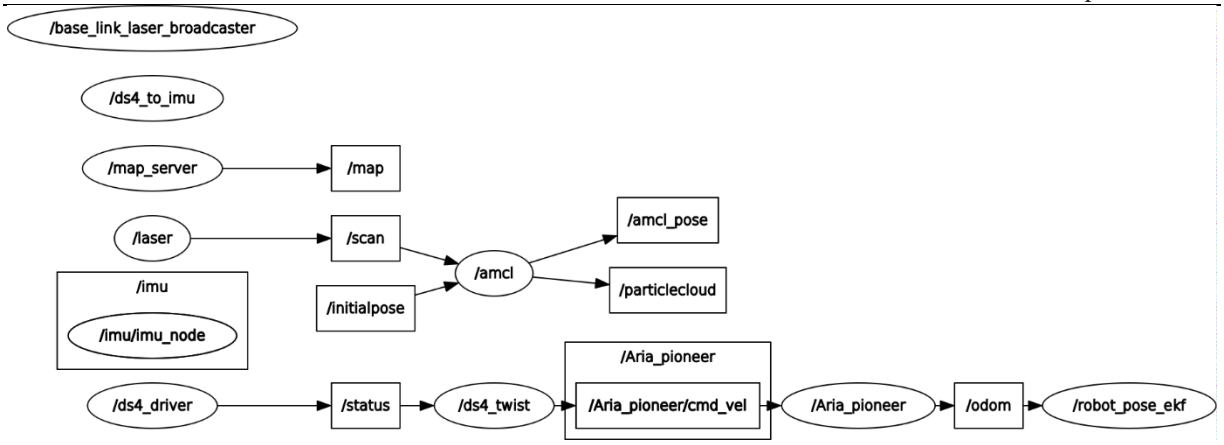
Kako bi se robot uspješno kretao po prostoru, prije toga je potrebno provesti postupak lokalizacije. Lokalizacija robota se vrši na način da se robota smjesti bilo gdje u prostor i učita mu se već spremljena mapa prostora. Potom je potrebno robotu otprilike odrediti gdje se nalazi na mapi te započeti s vožnjom robota kako bi robot pomoću stacionarnih značajki prostora što bolje odredio svoju poziciju u prostoru.

7.1. *Adaptive Monte Carlo Localization – AMCL*

AMCL radi lokalizaciju pomoću filtera čestica. S obzirom na mapu okoline, algoritam procjenjuje položaj i orijentaciju robota dok se robot kreće i skenira okolinu. Algoritam koristi filtar čestica za predstavljanje raspodjele vjerojatnih pozicija i orijentacija, pri čemu svaka čestica predstavlja moguću poziciju i orijentaciju robota, odnosno pretpostavku di bi robot mogao biti. Algoritam započinje rad s jednoličnom i slučajnom raspodjelom mogućih pozicija po prostoru, što znači da robot ne zna gdje se nalazi i pretpostavlja da bi mogao biti u bilo kojoj točki na mapi. Svakim pomicanjem robota, čestice se premještaju kako bi se bolje predvidjelo novo stanje nakon pomicanja. Kad god robot odradi novi pregled mape pomoću lidar senzora, čestice se uzorkuju na temelju rekurzivne Bayesove estimacije, tj koliko dobro skenirani podaci koreliraju s predviđenim pozicijama i orijentacijama. U konačnici, čestice bi trebale konvergirati prema stvarnom položaju robota.

Najvažniju ulogu u određivanju pozicije robota primjenom *amcl* algoritma ima laser, stoga je potrebno odrediti točnu poziciju lidar senzora u odnosu na bazu robota. U poglavlju o kalibraciji robota će biti više riječ o pronalasku pozicije lidar senzora.

Pokretanje *localization.launch* datoteke, pokreće čvorove koji omogućuju robotu da se lokalizira u prostoru za kojeg ima spremljenu mapu. Čvorovi koji se pokreću i njihove teme na koje objavljuju i pretplaćuju se su prikazani su na slici 32.



Slika 32. Čvorovi koji omogućuju lokalizaciju mobilnog robota u prostoru

7.2. *amcl*

U sklopu ROS-a postoji paket koji omogućuje lokalizaciju robota primjenom *AMCL* algoritma. Za pokretanje *amcl* čvora potrebno je u terminalu upisati sljedeću naredbu:

```
rosrun amcl amcl scan:=scan
```

Čvor *amcl* se pretplaćuje na potrebne teme kako bi mogao započeti i izvršavati proces lokalizacije mobilnog robota u prostoru:

- *scan* – tema koja osigurava informacije s lidar senzora
- *tf* – tema u kojoj su sadržane sve informacije vezane za transformacije između koordinatnih sustava mobilnog robota
- *initalpose* – tema koja sadrži informacije o srednjoj vrijednosti i kovarijancama o poziciji s obzirom na koju je potrebno inicijalizirati filter čestica
- *map* – sadrži mapu skeniranog prostora

Nakon pokretanja čvora i pretplate na gore navedene teme, *amcl* u ROS okruženje kreće objavljivati sljedeće teme:

- *amcl_pose* – estimirana pozicija robota s kovarijancama
- *particlecloud* – skup estimirani pozicija koje daje filter
- *tf* – objavljuje transformaciju između *odom* koordinatnog sustava i koordinatnog sustava mape *map*

Servisi koje je moguće pozivati prilikom korištenja *amcl* čvora su:

- *global_localization* – inicijalizira globalnu lokalizaciju gdje su sve čestice koje sadrže potencijalnu poziciju i orijentaciju robota nasumično generirane

- *request_nomotion_update* – servis zadužen da se ručno izvrši osvježavanje novih čestica s potencijalnim pozicijama robota
- *set_map* – omogućuje ručno učitavanje nove mape
- *static_map* – servis kojeg *amcl* samostalno poziva kako bi učitao mapu koja se koristi za lokalizaciju

Kako je algoritam za lokalizaciju iznimno kompleksan, postoji velik broj parametara na koje je potrebno postaviti kako bi algoritam uspješno radio:

- *min_particles* – minimalni broj čestica koje sadrže poziciju i orijentaciju robota
- *max_particles* – maksimalni broj čestica koje sadrže poziciju i orijentaciju robota
- *kld_err* – maksimalna greška između stvarne distribucije čestica i estimirane
- *kld_z* – gornji standardni normalni kvantil za (1-*p*), gdje je *p* vjerojatnost da će pogreška na estimiranom poremećaju biti manja od *kld_err*
- *update_min_d* – translacijska udaljenost koju je potrebno proći da bi se ažurirao filter čestica pozicije i orijentacije
- *update_min_a* – rotacijsko gibanje koje je potrebno napraviti da bi se ažurirao filter čestica pozicije i orijentacije
- *resample_interval* – broj ažuriranja filtera prije ponovnog uzorkovanja
- *transform_tolerance* – vrijeme za koje je potrebno odgoditi transformaciju koja se objavljuje kako bi se pokazalo da je transformacija valjana u budućnosti
- *recovery_alpha_slow* – eksponencijalna stopa propadanja za spori filter s prosječnom težinom, koji se koristi pri odlučivanju kada će se filter popraviti/obnoviti s dodavanjem nasumičnih pozicija
- *recovery_alpha_fast* – eksponencijalna stopa propadanja za brzi filter s prosječnom težinom, koji se koristi pri odlučivanju kada će se filter popraviti/obnoviti s dodavanjem nasumičnih pozicija
- *initial_pose_x* – početna pozicija u smjeru osi X, koristi se za inicijalizaciju filtera s Gausovom distribucijom
- *initial_pose_y* – početna pozicija u smjeru osi Y, koristi se za inicijalizaciju filtera s Gausovom distribucijom
- *initial_pose_a* – početna rotacija oko osi Z, koristi se za inicijalizaciju filtera s Gausovom distribucijom

- *initial_cov_xx* – inicijalna kovarijanca X*X korištena za inicijalizaciju Kalman filtera
- *initial_cov_yy* – inicijalna kovarijanca Y*Y korištena za inicijalizaciju Kalman filtera
- *initial_cov_aa*– inicijalna kovarijanca za rotaciju oko osi Z korištena za inicijalizaciju Kalman filtera
- *gui_publish_rate* – maksimalna frekvencija na kojoj se objavljuju skenirani podaci i putanje za vizualizaciju
- *save_pose_rate* – maksimalna frekvencija na kojoj će se spremati posljednje estimirana pozicija i kovarijance na server parametara
- *use_map_topic* – ako je postavljeno kao *true* čvor se odmah pretplaćuje na temu *map* i ne poziva servis *set_map*
- *first_map_only* – kada se postavi u *true* čvor koristi prvu mapu na koju se pretplati te ne osvježava svaki put temu s novom mapom
- *selective_resampling* – služi za smanjivanje vremena između dva uzorkovanja i pomaže u izbjegavanju nedostataka čestica

Osim parametara vezanih za sam algoritam, potrebno je definirati parametre lidar senzora kako bi lokalizacija bila uspješna. Savjet je da se koriste što sličniji parametri onima koji su korišteni prilikom mapiranja prostora.

- *laser_min_rang* – minimalna udaljenost laserske zrake lidar senzora
- *laser_max_range* – maksimalna udaljenost laserske zrake lidar senzora
- *laser_max_beams* – broj jednoliko udaljenih laserskih zraka koje se koriste prilikom osvježavanja filtera
- *laser_z_hit* – težina za *z_hit* varijablu modela
- *laser_z_short* – težina za *z_short* varijablu modela
- *laser_z_max* – težina za *z_max* varijablu modela
- *laser_z_rand* – težina za *z_rand* varijablu modela
- *laser_sigma_hit* – standardna devijacija za Gausov model korišten u *z_hit* dijelu modela
- *laser_lambda_short* – eksponencijalna raspodjela parametra za *z_short* dio modela
- *laser_likelihood_max_dist* – maksimalna udaljenost do „napuhane“ prepreke na mapi, koristi se u *likelihood_field* modelu

- *laser_model_type* – parametar za određivanje modela, može se birati između *beam, likelihood_field* i *likelihood_field_prob*

Na kraju je potrebno točno postaviti model mobilnog robota kako bi proces lokalizacije što točnije mogao estimirati poziciju i orijentaciju robota u mapi. Za to je potrebno odrediti vrijednosti sljedećih parametara:

- *odom_model_type* – parametar kojim se određuje tip pogona mobilnog robota, može se birati između *diff* (diferencijalni pogon), *omni* (svesmjerni pogon), *diff-corrected* ili *omni-corrected*
- *odom_alpha1* – određuje očekivani šum u procjeni rotacije odometrije iz rotacijske komponente kretanja robota.
- *odom_alpha2* – određuje očekivani šum u procjeni rotacijske odometrije iz translacijske komponente kretanja robota.
- *odom_alpha3* – određuje očekivani šum u procjeni translacije odometrije iz rotacijske komponente kretanja robota.
- *odom_alpha4* – određuje očekivani šum u procjeni translacije odometrije iz translacijske komponente kretanja robota.
- *odom_alpha5* – parametar šuma povezan s translacijskim gibanjem
- *odom_frame_id* – naziv koordinatnog sustava odometrije
- *base_frame_id* – naziv koordinatnog sustava baze mobilnog robota
- *global_frame_id* – naziv globalnog koordinatnog sustava
- *tf_broadcast* – parametar kojim se određuje hoće li *amcl* čvor objavljivati transformaciju između *odom* i *map* koordinatnih sustava

Niže je dana definicija svih parametara koji su određeni eksperimentalno, dok su vrijednosti parametara koji nisu definirani prilikom pokretanja *amcl* čvora definirane prema [17].

```
odom_model_type: diff
odom_alpha5: 0.1
transform_tolerance: 0.2
gui_publish_rate: 10.0
laser_max_beams: 30
min_particles: 500
max_particles: 5000
kld_err: 0.05
kld_z: 0.99
odom_alpha1: 0.2
odom_alpha2: 0.2
odom_alpha3: 0.8
odom_alpha4: 0.2
laser_z_hit: 0.5
laser_z_short: 0.05
laser_z_max: 0.05
laser_z_rand: 0.5
laser_sigma_hit: 0.2
laser_lambda_short: 0.1
laser_lambda_short: 0.1
laser_model_type: likelihood_field
laser_likelihood_max_dist: 2.0
update_min_d: 0.2
update_min_a: 0.5
odom_frame_id: odom
resample_interval: 1
transform_tolerance: 0.1
recovery_alpha_slow: 0.0
recovery_alpha_fast: 0.0
initial_pose_xvalue: 0.0
initial_pose_yvalue: 0.0
initial_pose_zvalue: 0.0
```

Kako bi se adekvatno pokrenuo proces lokalizacije potrebno je napraviti *.launch* datoteku u kojoj se postavljaju svi parametri i pokreću svi potrebni čvorovi. Kreirana je *localization.launch* datoteka koja je slična *mapping.launch* datoteci, ali se umjesto *slam_gmapping* čvora za mapiranje, pokreće *amcl* čvor za lokalizaciju mobilnog robota.

Niže je dan način pokretanja *amcl* i *map_server* čvorova u *.launch* datoteci dok je cijela *mapping.launch* datoteka dana u prilogu rada.

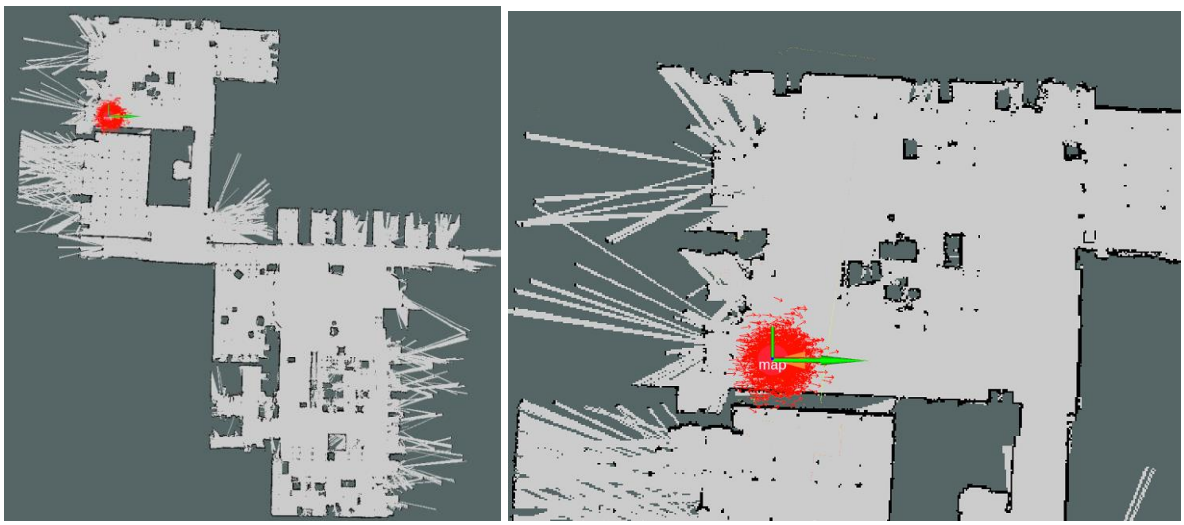
```
<!-- UCITAJ MAPU -->
<arg name="map_file" default="$(find pioneer_robot)/maps/CRTA.yaml"/>
<node name="map_server" pkg="map_server" type="map_server" args="$(arg
map_file)" />

<!-- POKRENI AMCL -->
<node pkg="amcl" type="amcl" name="amcl" output="screen">
   <rosparam file="$(find pioneer_robot)/config/amcl_config.yaml"
command="load"/>
</node>
```

7.3. Proces lokalizacije

Lokalizacija mobilnog robota započinje tako što se iz terminala pokreće *mapping.launch* datoteke sljedećom naredbom:

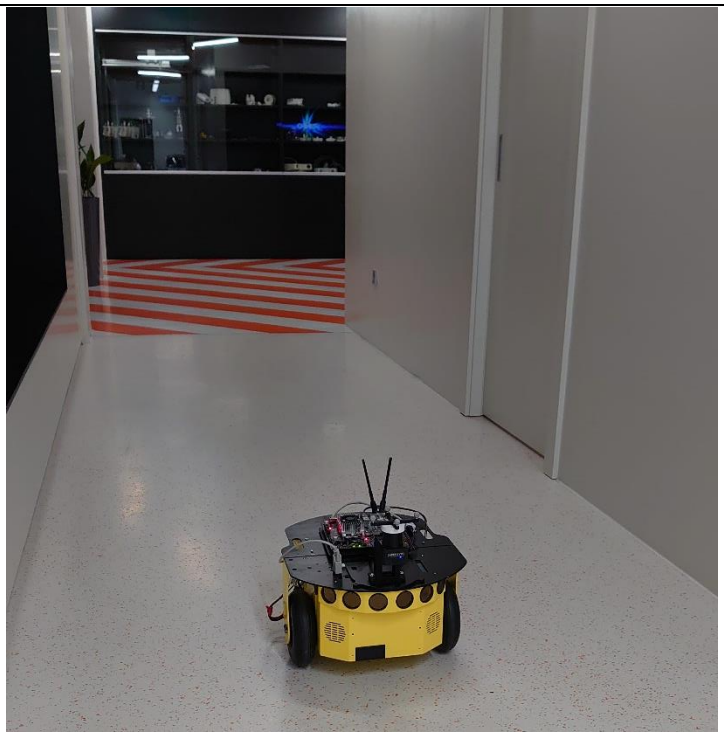
```
roslaunch pioneer_robot localization.launch
```

Pokretanjem se otvara *rviz* sučelje u koje se učitava mapa prostora te se predviđena pozicija robota stavlja u ishodište mape (postavljeno u konfiguracijskoj datoteci) – mjesto od kud je započeto mapiranje prostora. Slika 33. prikazuje predviđenu početnu poziciju mobilnog robota u mapi (zelena strelica) kao i sve moguće pozicije koje filter *amcl* algoritma daje kao potencijalne pozicije i orijentacije (crvene strelice).



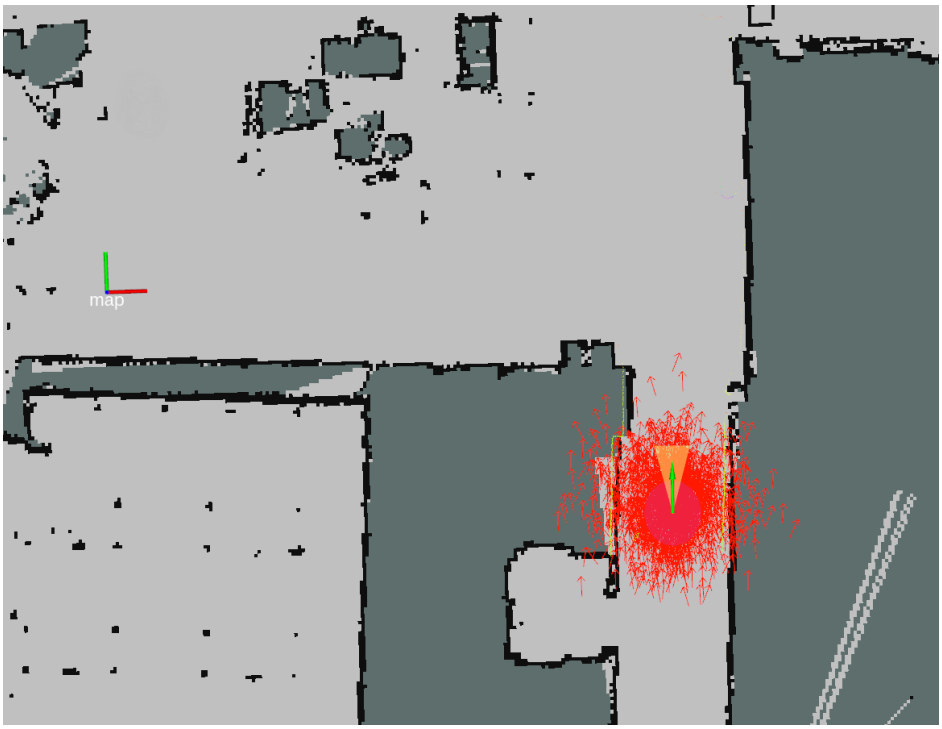
Slika 33. Predviđena pozicija robota u *CRTA*-i – lijevo, predviđena pozicija robota u *Laboratoriju za računalnu inteligenciju L2* – desno

Ukoliko se robot ne nalazi u blizini početne pozicije potrebno je s alatom *2D Pose Estimate* odrediti okvirnu poziciju mobilnog robota u prostoru. Kako bi se demonstrirao proces, robot je postavljen u hodnik koji spaja glavni ulaz u *CRTA*-u i *Laboratorij za računalnu inteligenciju L2*. Slika 34. prikazuje stvarnu poziciju robota u hodniku.



Slika 34. Pozicija mobilnog robota u hodniku

Korištenjem alata *2D Pose Estimate*, otprilike je određeno gdje se robot nalazi u hodniku te se estimirana pozicija i polje čestica koje sadrže poziciju i orijentaciju promijenilo. Slika 35. prikazuje novi početni položaj robota prije procesa lokalizacije.

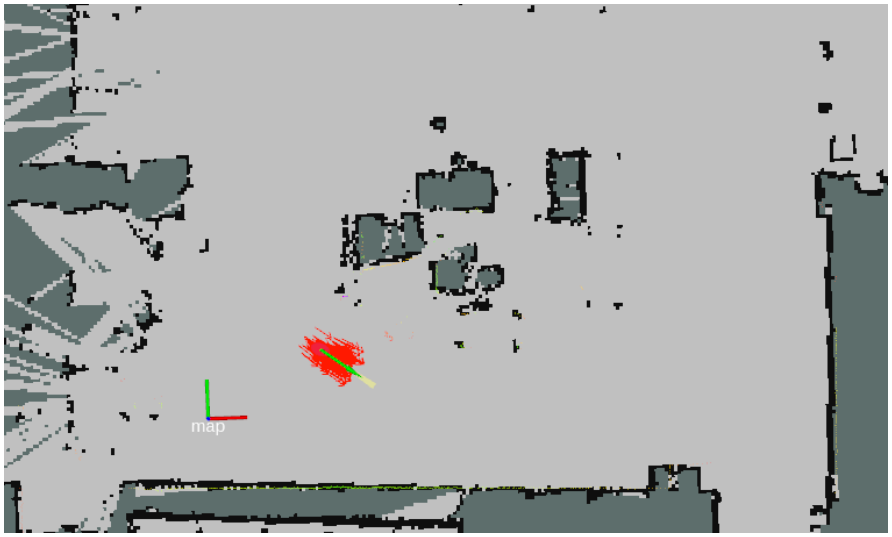


Slika 35. Novi početni položaj robota prije početka procesa lokalizacije

Upravljanjem robota daljinskim upravljačem, robot prepoznaje značajke prostora lidar senzorom te sukladno tome polje potencijalnih pozicija i orijentacija sve više konvergira te robot postaje sigurniji gdje se točno nalazi u prostoru. Slika 36. prikazuje kako je oblak čestica pozicije i orijentacije krenuo konvergirati u smjeru osi X (u odnosu na koordinatni sustav mape – crvena os). Razlog tome je što je robot bio smješten u hodniku stoga je *amcl* mogao mjeriti udaljenost među dva zida i tako točnije odrediti poziciju u smjeru osi X nego Y.

Slika 36. Početak konvergencije oblaka čestica pozicija i orijentacija

Kako bi se dobila što bolja lokalizacija i konvergencija oblaka čestica, potrebno je robota voziti dovoljno dugo i truditi se da robot prođe kraj što više značajki prostora poput zidova, uskih prolaza i stupova kako bi pomoću tih značajki što točnije proveo proces lokalizacije. Slika 37. pokazuje poziciju robota i oblak čestica nakon što je robot napravio puni krug oko *Laboratorija za računalnu inteligenciju L2.* Sa slike je vidljivo da se robot uspješno lokalizirao.

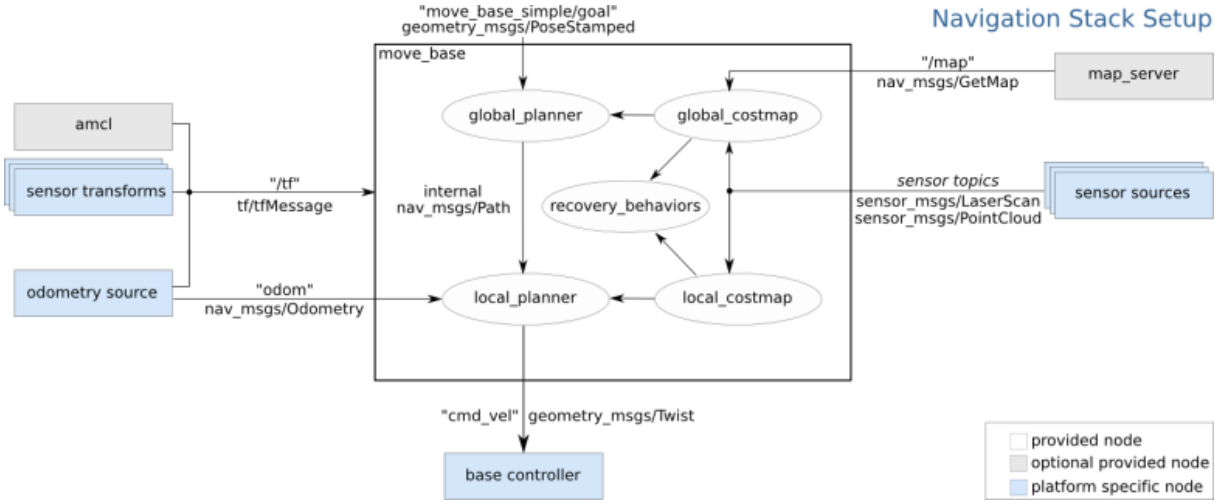


Slika 37. Prikaz zadovoljavajuće konvergencije pozicije i orijentacije robota

8. NAVIGACIJA MOBILNOG ROBOTA PRIMJENOM ROS-a

Navigacija je proces vođenja robota iz jedne pozicije u drugu uz zadovoljavanje određenih uvjeta i izbjegavanje prepreka.

U ROS okruženju postoji *navigation stack* u kojem se nalaze paketi i čvorovi koji omogućuju sigurnu navigaciju mobilnog robota između dviju točaka u mapi. Navigacija u ROS okruženju uzima informacije o mapi, odometriji, signalima senzora i krajnjoj točci te ih preračunava u brzine koje se šalju kontrolerima motora kako bi mobilni robot mogao realizirati željene ciljeve u mapi. U sklopu *navigation stack-a* se nalaze sljedeći paketi koje je moguće koristiti prilikom navigacije mobilnog robota: *amcl*, *base_local_planner*, *carrot_planner*, *clear_costmap_recovery*, *costmap_2d*, *dwa_local_planner*, *fake_localization*, *global_planner*, *map_server*, *move_base*, *move_base_msgs*, *move_slove_and_clear*, *nav_core*, *navfn*, *rotate_recovery*, *voxel_grid*. Slika 38 daje generalni pregled načina rada navigacije u sklopu ROS-a.



Slika 38. Generalni pregled *navigation stack-a* unutar ROS-a [26]

Kućice ispunjene sivom bojom su opcijski čvorovi bez kojih navigacija u ROS-u može raditi, moguće je realizirati navigaciju u ROS-u bez mape prostora i procesa lokalizacije primjenom *amcl* čvora.

Svaki robot ima svoj način upravljanja motorima, svoje senzore i pozicije istih te drugačiji izvor odometrije, te segmente je potrebno prilagoditi kako bi navigacija bila uspješna. Ti segmenti su označeni plavom bojom na slici 38.

Proces navigacije izvršavaju ostali segmenti označeni bijelom bojom na slici 38. te se njih može pronaći u *navigation stack*-u.

8.1. *move_base*

Čvor *move_base* omogućuje implementaciju navigacije na mobilnom robotu. Uzimajući odometriju i ciljanu poziciju i orijentaciju, *move_base* izračunava potrebne brzine te ih šalje robotu. Čvor se pokreće na iz terminala:

```
rosrun move_base move_base
```

Nakon pokretanja, *move_base* se pretplaćuje na sljedeće teme:

- *move_base/goal* – tema osigurava na kojeg mobilni robot mora stići
- *move_base/cancle* – zahtjev za otkazivanjem određenog cilja
- *move_base_simple/goal* – osigurava sučelje bez radnje za *move_base* za korisnike kojima nije bitno praćenje statusa o izvršavanju cilja

Teme na koje *move_base* objavljuje nakon izračuna potrebnih brzina su:

- *move_base/feedback* – tema sadrži informacije o trenutnoj poziciji robota u prostoru
- *move_base/status* – sadrži informacije o ciljevima na koje je *move_base* čvor pretplaćen
- *move_base/result* – sadržava informacije o uspješnosti dosezanja cilja mobilnog robota

Kako je tijekom izvođena navigacije potrebno napraviti određene izmjene ili saznati informacije, moguće je pozvati neke od navedenih servisa:

- *make_plan* – omogućuje vanjskom korisniku da dobije plan do cilja bez da izvrši plan
- *clear_unknown_space* – omogućuje da se očisti nepoznat prostor u neposrednoj blizini robota
- *clear_costmap* – omogućuje čišćenje prepreka u *costmap*-i. Ovaj servis može stvarati probleme jer robot onda gubi pregled nad preprekama i postoji mogućnost da se zabije u iste

Kao u svim čvorovima, ovdje također postoje parametri koje je potrebno definirati kako bi navigacija dovoljno dobro radila.

- *base_global_planner* – naziv dodatka za globalni planer o kojem će biti više riječ u nastavku
- *base_local_planner* – naziv dodatka za lokalni planer o kojem će biti više riječ u nastavku
- *recovery_behaviors* – lista ponašanja u slučaju da robot zapne i ne može pronaći izlaz

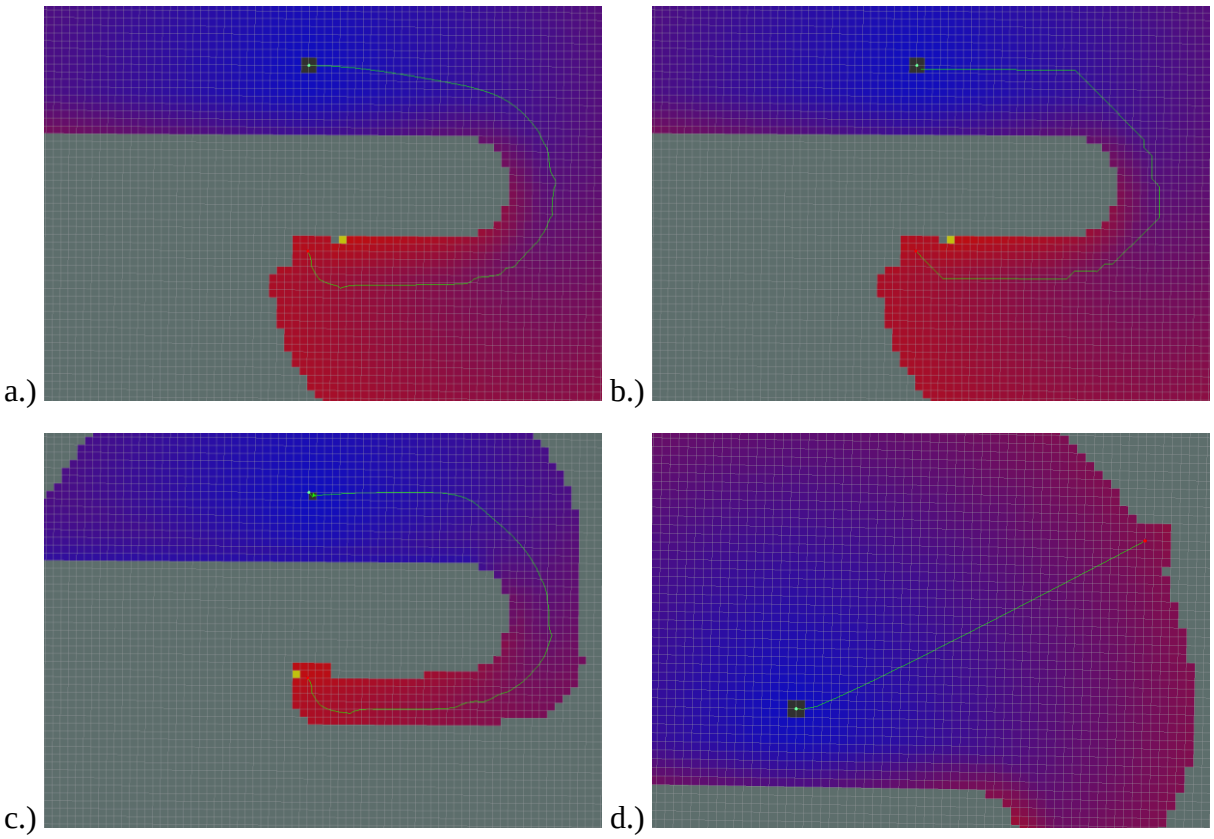
- *controller_frequency* – frekvencija kontrolera za izvršavanje upravljačke petlje i slanje brzina kontroleru robota
- *planner_patience* – vrijeme koliko dugo će planer pokušavati pronaći adekvatnu trajektoriju prije nego što krene izvoditi manevre u slučaju zastoja
- *controller_patience* – vrijeme koliko dugo će kontroler čekati bez primanja adekvatne informacije da bi pokrenuo manevre u slučaju zastoja
- *conservative_reset_distance* – udaljenost od robota iznad koje će prepreke biti maknute s *costmap*-e
- *recovery_behavior_enabled* - parametar koji odlučuje hoće li se koristiti manevri u slučaju zastoja
- *clearing_rotation_allowed* – parametar za odlučivanje hoće li robot provoditi rotacijsko gibanje kako bi očistio prostor oko sebe
- *shutdown_costmap* – parametar koji odlučuje hoće li se ugasiti *costmap*-a kada je *move_base* neaktivan
- *oscillation_timeout* – koliko dugo će se robotu dopustiti oscilacije prije nego li krene izvršavati manevre u slučaju zastoja
- *oscillation_distance* – koliko se metara robot mora kretati kako bi se smatralo da robot ne oscilira na mjestu
- *planner_frequency* – frekvencija na kojoj robot provodi globalnu trajektoriju
- *max_planning*_retries – broj pokušaja za planiranje trajektorije prije nego što robot krene provoditi manevre u slučaju zastoja

8.1.1. global_planner

Kada korisnik čvoru *move_base* zada ciljanu lokaciju, potrebno je sukladno trenutnoj poziciji robota, zadanom cilju i rasporedu slobodnog i okupiranog prostora na mapi odrediti adekvatnu trajektoriju koja će izbjeći sve prepreke i uspješno dovesti robota na željeni cilj. Dodatak čvoru *move_base* koji proračunava potrebnu trajektoriju se naziva *global_planner*, a u sebi sadrži više različitih planera. Tema na koju *global_planner* objavljuje je *plan*, a u njoj su sadržane informacije o proračunatoj trajektoriji koja se šalje lokalnom planeru o kojem će kasnije biti više riječ. Parametri koji se postavljaju za globalni planer nisu toliko bitni jer se njima većinski određuje vrsta globalnog planera. Parametri koji imaju utjecaj na planiranje trajektorije su:

- *allow_unknown* – parametar kojim se određuje hoće li se robot kretati po neistraženim dijelovima mape
- *default_toleracnce* – tolerancija krajnjeg cilja planera

Globalni planeri koji se mogu koristiti unutar dodatka *global_planner* su: *navfn*, *djikstra*, *quadratic*, *grid_path* i *old_navfn*, *A** i *dijkstra*. Slika 39 prikazuje izračun globalnih trajektorija s obzirom na odabrani planer.



Slika 39. a.) *navfn*, b.) *grid_path*, c.) *A i d.) *dijkstra* [18]**

Parametri koji su postavljeni za globalni planer u obliku *.yaml* datoteke su:

```
controller_frequency: 5.0
recovery_behaviour_enabled: true

NavfnROS:
  allow_unknown: true
  default_tolerance: 0.01
```

Globalni planer koji se koristi u ovom radu je *navfn*, dok su ostali parametri za globalni planer trajektorija automatski postavljeni na predefinirane vrijednosti koje se mogu pronaći na [18].

8.1.2. base_local_planner

Kako bi se navigacija uspješno provela potrebno je da robot što bolje prati referentnu trajektoriju koju je generirao globalni planer. Pri slijeđenju trajektorije, potrebno je da robot izbjegava prepreke koje mu se mogu pronaći na putu, a nisu vidljive u mapi prostora. Kako bi se prethodno navedeni zadaci uspješno izvršili potrebno je odabrati adekvatan lokalni planer.

U sklopu *base_local_planner* paketa se nalaze tri lokalna planera: *TrajectoryPlannerROS*, *dwa_local_planner* i *teb_local_planner*.

Lokalni planeri se pretplaćuju na temu *odom* iz koje dobivaju informacije o odometriji robota, dok objavljuju na teme *global_plan*, *local_plan* i *cost_cloud* koje se koriste isključivo za vizualizaciju trajektorija. Osnovni parametri lokalnih planera koji su zajednički svim planerima se mogu podijeliti na parametre vezane za robota, toleranciju dosezanja cilja i točnost praćenja trajektorije. Osnovni parametri svih lokalnih planera su:

- *acc_lim_x* – maksimalna akceleracija robota u smjeru osi X
- *acc_lim_y* – maksimalna akceleracija robota u smjeru osi Y
- *acc_lim_theta* – maksimalna akceleracija robota oko osi Z
- *max_vel_x* – maksimalna brzina robota u smjeru osi X
- *min_vel_x* – minimalna brzina robota u smjeru osi X
- *max_vel_theta*– maksimalna brzina rotacije oko osi Z
- *min_vel_theta* – minimalna brzina rotacije oko osi Z
- *holonomic_robot* – parametar kojim se definira da li je robot holonomski ili ne-holonomski
- *yaw_goal_tolerance* – tolerancija rotacije prilikom dostizanja cilja
- *xy_goal_tolerance* – tolerancija dostizanja cilja u X i Y smjeru

Detaljniji opis lokalnih parametara bit će dan u poglavlju gdje će se provjeravati točnost lokalnih planera, dok su za implementaciju navigacije odabrani *dwa_local_planner* i sljedeći parametri.

```
DWAPlannerROS:
  acc_lim_x: 2.5
  acc_lim_y: 0
  acc_lim_th: 3.2

  max_vel_x: 0.1
  min_vel_x: 0.0
  max_vel_y: 0
  min_vel_y: 0

  max_trans_vel: 0.1
  min_trans_vel: 0.1
  max_rot_vel: 1.0
  min_rot_vel: 0.0

  yaw_goal_tolerance: 0.02
  xy_goal_tolerance: 0.05
  latch_xy_goal_tolerance: false

  sim_time: 2.0
  sim_granularity: 0.02
  vx_samples: 6
  vy_samples: 0
  vtheta_samples: 20
  penalize_negative_x: true


  path_distance_bias: 32.0
  goal_distance_bias: 24.
  occdist_scale: 0.01
  forward_point_distance: 0.325
  stop_time_buffer: 0.2
  scaling_speed: 0.25
  max_scaling_factor: 0.2
  oscillation_reset_dist: 0.25
```

Parametri koji nisu definirani u gore navedenoj *.yaml* datoteci su automatski inicijalizirani na predefinirane vrijednosti koje se mogu pronaći na [19].

8.1.3. costmap_2d

Ovaj paket pruža mogućnost implementacije 2D mape težinskih faktora koja uzima podatke senzora iz okoline robota i gradi 2D ili 3D mapu okupiranosti prostora. Osim navedenog, ovaj paket ima mogućnost povezivanja s *map_server* paketom i stvaranjem mape težinskih faktora iz već snimljene mape prostora te kreiranjem mape težinskih faktora temeljem trenutnih očitanja senzora.

Dva su osnovna tipa *costmap*-e, prvi je globalna mapa koja se inicijalizira prilikom učitavanja mape prostora, dok je druga mapa ona koja se trenutno nalazi oko robota i služi za detekciju prepreka koje se nalaze u bliskoj okolini robota.

Pokretanjem ovog paketa, isti se pretplaćuje na temu *footprint* u kojoj se nalaze informacije o gabaritnim dimenzijama robota. *Costmap_2d* paket objavljuje podatke na sljedeće teme:

- *costmap* – vrijednosti težinskih faktora unutar mape
- *costmap_updates* – vrijednosti ažuriranih težinskih faktora unutar mape
- *voxel_grid* – sadrži informacije o 3D prostoru ukoliko postoji senzorska oprema koja te informacije može osigurati

Kako su globalna i lokalna *costmap*-a iznimno važne za realizaciju procesa navigacije, potrebno je adekvatno postaviti parametre. Parametri koje je potrebno postaviti su:

- *global_frame* – globalni koordinatni sustav
- *robot_base_frame* – koordinatni sustav mobilnog robota
- *transform_tolerance* - vrijeme za koje je potrebno odgoditi transformaciju koja se objavljuje kako bi se pokazalo da je transformacija valjana u budućnosti
- *update_frequency* – frekvencija osvježavanja mape
- *publish_frequency* – frekvencija objavljivanja mape
- *rolling_window* – parametar kojim se definira hoće li se mapa kretati zajedno s robotom
- *static_map* – parametar kojim se definira hoće li mapa bit statična ili ne
- *always_send_full_costmap* – parametar kojim se određuje hoće li se uvijek slati cijela mapa ili samo promjene koje su nastale

Ostali parametri koje je potrebno postaviti, a nemaju veliku ulogu u procesu navigacije su: *width, height, resolution, origin_x, origin_y*.

Osim parametara, za adekvatan rad paketa potrebno je osigurati transformaciju iz globalnog koordinatnog sustava (*global_frame)* u koordinatni sustav robota (*robot_base_frame*). Transformacija se može dobiti iz odometrije robota ili primjenom *amcl* čvora za lokalizaciju.

Kao što je ranije objašnjeno, potrebno je koristiti dvije *costmap*-e, globalnu i lokalnu. Zajednički parametri za obje su dani niže.

```
footprint: [[-0.2, -0.2], [-0.2, 0.2], [0.2, 0.2], [0.2, -0.2]]
footprint_padding: 0.01

robot_base_frame: base_footprint
update_frequency: 4.0
publish_frequency: 3.0
transform_tolerance: 0.5

resolution: 0.05

obstacle_range: 5.5
raytrace_range: 6.0

footprint_clearing_enabled: true

#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:
    inflation_radius: 0.35
    cost_scaling_factor: 1
```

Osim zajedničkih, postoje parametri koje je potrebno definirati za svaku *costmap*-u zasebno. Parametri globalne *costmap*-e su dani niže.

```
global_costmap: map
global_frame: map
rolling_window: false
track_unknown_space: true
update_frequency: 5.0
static_map: true

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

Parametri lokalne *costmap*-e su:

```
global_frame: map
rolling_window: true

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

Ostali parametri koji nisu definirani za *costmap*-e su automatski inicijalizirani s predefiniram vrijednostima koje se mogu pronaći na [20].

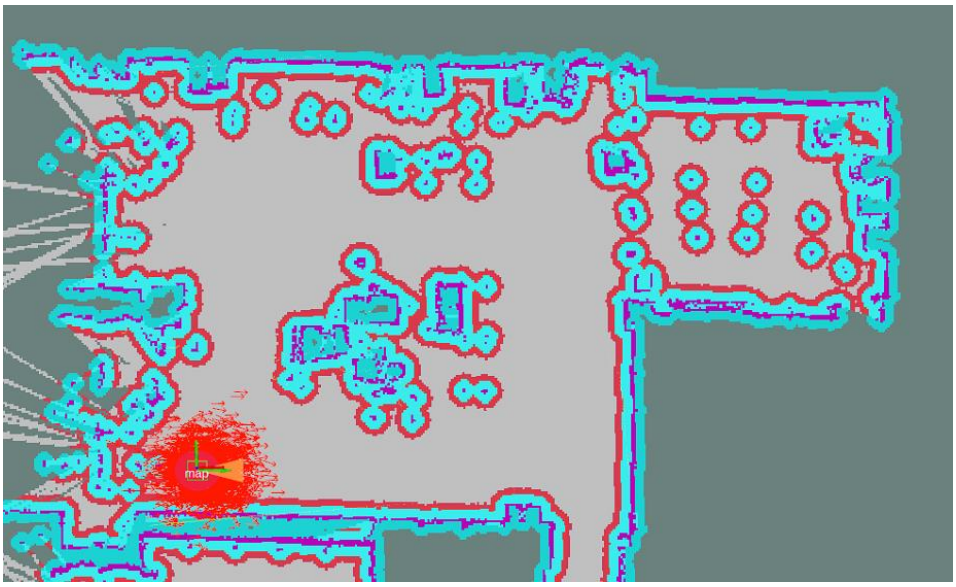
8.2. Proces navigacije robota

Kako bi se proces navigacije adekvatno proveo potrebno je lokalizirati robota s *amcl* čvorom. Kako *amcl* i *navigation_stack* paketi iziskuju učitavanje mnogih parametara, kreiran je *.launch* datoteka s kojom se pokreću svi potrebni čvorovi i učitavaju parametri. *Navigation.launch* datoteka je identična *localization.launch* datoteci, osim što su izbačeni čvorovi za *ds4_driver* i *ds4_twist_node* zaduženi za upravljanjem robotom pomoću upravljača dodan je čvor za navigaciju *move_base*. Niže je dan dodatak za pokretanje *move_base* čvora i učitavanja parametara za globalni i lokalni planer te *costmap*-e.

```
<launch>

  <arg name="no_static_map" default="false"/>
  <arg name="base_global_planner" default="navfn/NavfnROS"/>
  <arg name="base_local_planner" value="base_local_planner/DWAPlannerROS"/>

  <node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">

    <param name="base_global_planner" value="$(arg base_global_planner)"/>
    <param name="base_local_planner" value="$(arg base_local_planner)"/>
    <rosparam file="$(find pioneer_robot)/config/planner.yaml"
command="load"/>

    <rosparam file="$(find pioneer_robot)/config/costmap_common.yaml"
command="load" ns="global_costmap" />
    <rosparam file="$(find pioneer_robot)/config/costmap_common.yaml"
command="load" ns="local_costmap" />

    <rosparam file="$(find pioneer_robot)/config/costmap_local.yaml"
command="load" ns="local_costmap" />
    <rosparam file="$(find
pioneer_robot)/config/costmap_global_static.yaml" command="load"
ns="global_costmap" />

    <rosparam file="$(find pioneer_robot)/config/costmap_global_laser.yaml"
command="load" ns="global_costmap" if="$(arg no_static_map)"/>
    <param name="global_costmap/width" value="10.0" if="$(arg
no_static_map)"/>
    <param name="global_costmap/height" value="10.0" if="$(arg
no_static_map)"/>

    <remap from="cmd_vel" to="Aria_pioneer/cmd_vel"/>

  </node>

</launch>
```

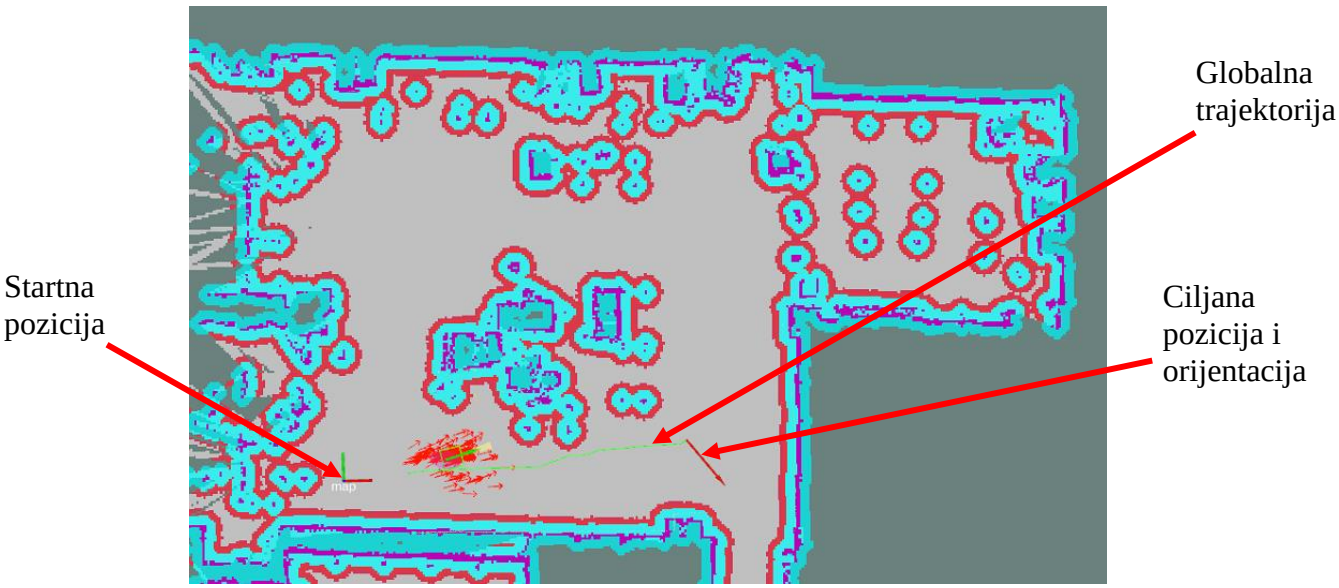
Pokretanje *navigation.launch* datoteke pokreće se *rviz* sučelje, učitava se mapa prostora na koju se dodaje *costmap*-a s radijusima zakrivljenosti kako bi robot na dovoljnoj udaljenosti od zidova pratio željenu trajektoriju. Slika 40. pokazuje stanje robota u mapi prilikom pokretanja, a niže se nalazi naredba za pokretanja navigacije.

```
roslaunch pioneer_robot navigation.launch
```



Slika 40. Prikaz mape za navigaciju

Korištenjem alata *2D Nav Goal* se određuje željena pozicija i orijentacija koju robot treba dostići, a objavljuje se na temu *move_base_simple/goal*. Slika 41. prikazuje ciljanu poziciju i orijentaciju robota i generiranu globalnu trajektoriju koju robot uz pomoć lokalnog planera pokušava što točnije pratiti.



Slika 41. Prikaz željenog cilja i generirane trajektorije

9. PROVJERA TOČNOSTI LOKALNIH PLANERA

Navigacija mobilnog robota u prostoru je iznimno važan segment jer omogućuju sigurno kretanje robota po prostoru. Osim sigurnog kretanja, potrebno je zadovoljiti praćenje referentne trajektorije koju generira globalni planer i točnost pozicioniranja na krajnjoj destinaciji robota. Kako je za točnost pozicioniranja zadužen lokalni planer, u ovom poglavlju će biti riječ o provjeri točnosti lokalnih planera. Planeri koji će se provjeravati su *TrajectoryPlannerROS, DWALocalPlanner* i *TEBLocalPlanner*, a za provjeru planera će se koristiti *UMBmark – A Method for Measuring, Comparing and Correcting Dead-reckoning Errors in Mobile Robots*, jedan od glavnih načina ispitivanja točnosti mobilnih robota [21].

Mobilni roboti odometriju estimiraju na način da koriste informaciju o trenutnoj poziciji te integracijom brzine određuju položaj. Kako se integracija odvija u diskretnom vremenskom periodu s vremenom se akumuliraju greške koje se očituju u pogrešnom pozicioniranju.

9.1. *UMBmark*

Metoda je nazvana prema *University of Michigan Benchmark test* je posebno dizajnirana metoda za otkrivanje i popravljanje sustavnih grešaka mobilnih robota. U ovom poglavlju će ova metoda biti korištena za provjeru točnosti pozicioniranja mobilnog robota, dok će u idućem biti objašnjena njezina kalibracija mobilnog robota.

Metoda provjere je zamišljena da se provede na dva načina, statička i dinamička provjera. U sklopu statičke provjere, robotu se zadaje referentna trajektorija kvadratnog oblika gdje se pamti početna pozicija robota te krajnja i računa se koliko je robot pogriješio.

Dinamička provjera se izvršava na način da se robotu svaki put u odnosu na njegovu poziciju zada zadatak da ide naprijed i okrene se lijevo u odnosu na njegov trenutni položaj. Dinamička provjera bi trebala dati lošije rezultate jer se greška pozicioniranja nakuplja svakim izvršavanjem.

Test je zamišljen da se izvede na način da se robota pusti da odradi kvadratnu trajektoriju dimenzija 4m x 4m, međutim zbog ne mogućnosti provedbe tako velike trajektorije za potrebe rada se test proveo na kvadratnoj trajektoriji 1m x 1m.

9.2. *TrajectoryPlannerROS*

Osnovni lokalni planer koji se nalazi u sklopu ROS-a, a osnovni koraci njegovog algoritma su:

1.) Uzimanje diskretne pozicije (*dx, dy* i *dtheta*) u lokalnom koordinatnom sustavu robota.

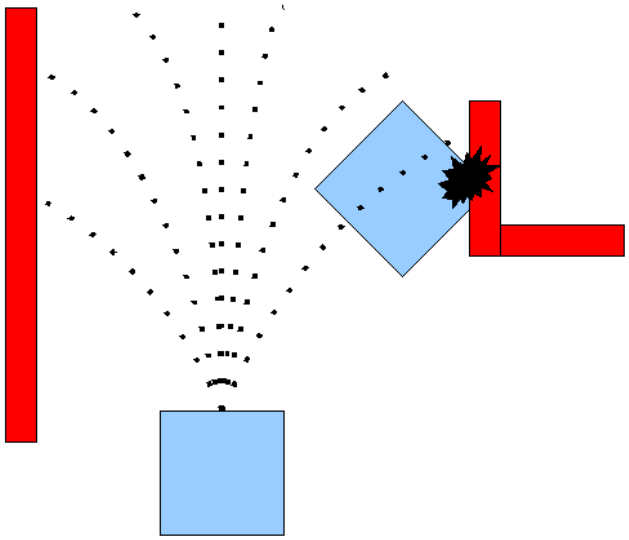
2.) Za svaku uzorkovanu brzinu algoritam izvodi estimaciju buduće pozicije robota te predviđa što će se dogoditi s robotom ukoliko se ta brzina primijeni u kratkom vremenu.

3.) Algoritam ocjenjuje svaku dobivenu putanju iz točke 2. koristeći karakteristike kao što su: blizina prepreka, blizina cilja, blizina globalne trajektorije i brzina, a ukoliko neka od navedenih trajektorija dovodi robota u koliziju s preprekom, algoritam ju odbacuje.

4.) Algoritam odabire trajektoriju s najvećom ocjenom i šalje brzine koje su dobivene proračunom simulacije.

5.) Algoritam ponavlja korake 1.-4. sve dok ne dostigne željeni cilj.

Slika 42. prikazuje način rada *TrajectoryPlannerROS* lokalnog planera.



Slika 42. Ilustracija rada *TrajectoryPlannerROS* lokalnog planera

Funkcija troška po kojoj navedeni planer odlučuje koja trajektorija najbolje zadovoljava je:

$$cost = pdist\_scale * s_{ep} + gdist\_scale * s_{locg} * occdis\_scale * s_{obst} \qquad \textbf{(8)}$$

gdje su parametri *pdist_scale*, *gdist_scale* i *occdist_scale* definirani u *.yaml* datoteci s parametrima koja se poziva prilikom pokretanja čvora *move_base*. Dok su varijable s_{ep} – udaljenost od krajnjeg cilja trajektorije, s_{locg} – udaljenost trenutnog položaja robota do željene trajektorije, s_{obst} – maksimalni težinski faktor prepreke (0-254).

9.3. *DWALocalPlanner*

Planer radi na sličan način kao i *TrajectoryPlannerROS*, ali razlika je u tome kako se uzorkuje okruženje oko robota. Kao što je ranije objašnjeno, *TrajectoryPlannerROS* uzorkuje cijeli spektar dostupnih brzina kroz cijelu simulaciju s obzirom na ograničenja u akceleracijama robota, dok *DWALocalPlanner* uzorkuje sve dostupne brzine ali u samo jednom koraku simulacije.

Zbog gore navedenog *DWALocalPlanner* je puno efikasniji od algoritam jer uzorkuje i proračunava puno manje mogućih slučajeva u budućnosti, ali ga *TrajectoryPlannerROS* može nadmašiti za robote s malim ograničenjima ubrzanja jer *DWALocalPlanner* ne simulira unaprijed konstantna ubrzanja.

9.4. TEBLocalPlanner

Lokalni planer, koji je proširenje *EBandLocalPlaner*-a o kojem se više može pronaći na [22], uzima u obzir kratku udaljenost ispred globalne trajektorije i s obzirom na nju kreira lokalnu trajektoriju koja sadrži set srednjih poza robota. Ovaj planer je računski iznimno efikasan jer koristi već generiranu globalnu trajektoriju i na temelju nje generira lokalnu trajektoriju s ciljem smanjenja vremena izvršavanja globalne trajektoriju.

Ovaj planer zahtjeva informacije o kinematici i dinamici mobilnog robota, geometrijskom obliku, akceleracijama robota i sigurnu udaljenost kako bi izbjegao prepreke. Planer nudi mogućnost generiranja trajektorija za holonimčne i neholonomične robote te pruža mogućnost generiranja više trajektorija od jednom ali zahtjeva više procesorske snage kako bi to proveo.

9.5. Oprema korištena za provjeru točnosti lokalnih planera

9.5.1. Polaris Vicra

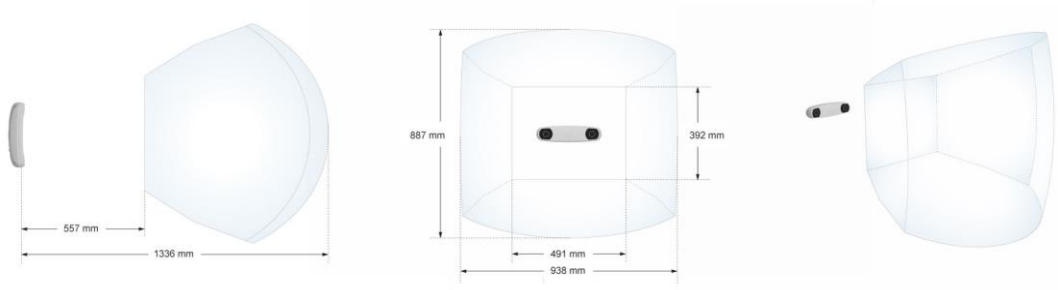
Polaris Vicra je uređaj tvrtke *NDI* koji služi za precizna mjerenja pozicije i orijentacije markera u prostoru. Uređaj prati 3D pozicije markera sa sub-milimetarskom točnosti i ponovljivosti. Markeri koji se prate su precizno praćeni i lokalizirani u realnom vremenu s volumnom točnosti od 0.25mm i 95% sigurnosnim intervalom od 0.5mm. *Polaris Vicra* je iznimno kompaktan uređaj te nudi mogućnost postavljanja na različita mjesta što omogućuje fleksibilnost mjerenja različitim okolinama. Slika 43. prikazuje uređaj *Polaris Vicra*.

Slika 43. *Polaris Vicra* [23]

Tablica 5. prikazuje karakteristike uređaja [23].

Volumna točnost	0.25 mm
Maksimalna frekvencija slika	20 Hz
Volumen mjerenja	Prikazan na slici 42.
Vrsta alata	Pasivni i aktivni
Maksimalan broj alata	15 alata, moguće simultano pratiti 6 alata
Broj markera po alatu	6 jednostranih i 20 višestranih
Dimenzije	273 x 69 x 69 mm
Masa	0.8 kg

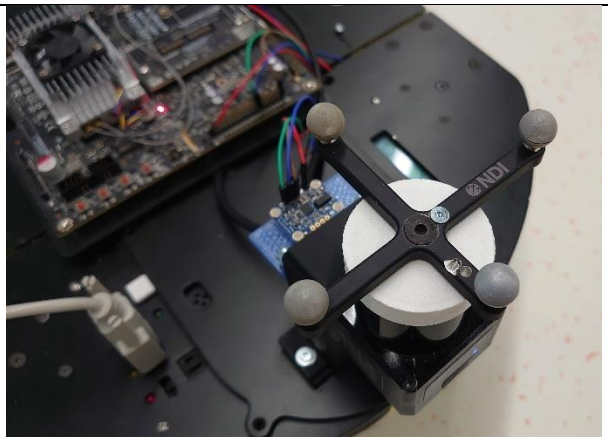
Tablica 6. Karakteristike *Polaris Vicra*-e [23]



Slika 44. Volumen mjerenja *Polaris Vicra*-e [23]

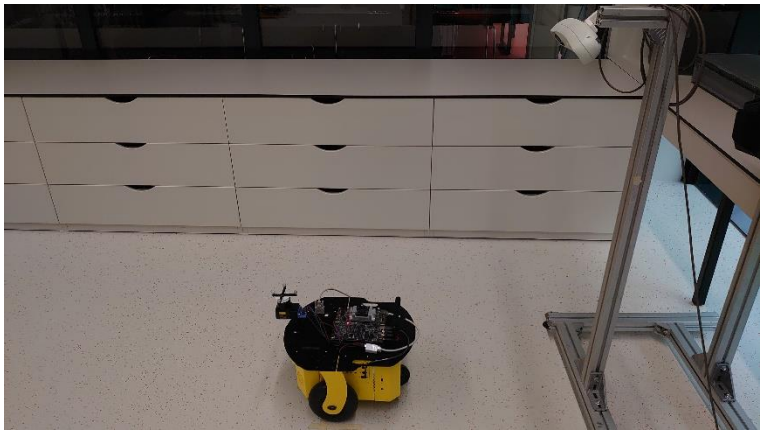
9.5.2. Marker

Kako bi *Polaris* mogao uspješno pratiti kretanja robota, potrebno je na robota postaviti marker kojeg uređaj može pratiti. Primjenom programa *NDI 6D Architect* se generira *.ROM* datoteka koja se učitava u *Polaris* kako bi se marker mogao pratiti. Važno je reći kako *Polaris* mjeri relativnu poziciju markera u odnosu na koordinatni sustav kamere. Slika 45. prikazuje marker i poziciju markera na robotu.



Slika 45. Pozicija markera na robotu

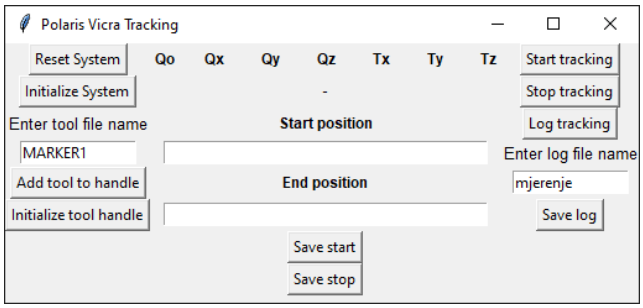
Nakon odabira adekvatnog uređaja za mjerenje i postavljanja markera za praćenje na robota, definirane su pozicije stalka za kameru i početnog položaja robota s kojeg će se puštati robot da prati trajektoriju. Slika 46. prikazuje sliku postava kamere i mobilnog robota.



Slika 46. Postav za mjerenje točnosti lokalnih planera

9.5.3. PolarisVicraGUI

Kako bi se kamerom moglo upravljati i snimiti početnu i krajnju točku robota napravljen je GUI koji služi za interakciju s kamerom. Između računala i kamere je uspostavljena USB serijska komunikacija. Slika 47. prikazuje GUI, dok su ispod slike objašnjene funkcije tipki unutar programa.



Slika 47. *PolarisVicraGUI*

- *Reset System* – ponovno pokretanje kamere (naredba: *RESET*)
- *Initialize System* – inicijalizacija kamere (naredba: *INIT*)
- *Enter Tool File Name* – naziv *.ROM* datoteke
- *Add tool handle* – dodavanje markera alatu (naredba: *PHRQ*)
- *Initialize tool handle* – inicijalizacija alata (naredba: *PINIT*)
- *Start Tracking* – početak praćenja alata (naredba: *TSTART*)
- *Stop Tracking* – zaustavljanje praćenja alata (naredba: *TSTOP*)
- *Log Tracking* – sprema pozicije markera u matricu
- *Save Log* – sprema vrijednosti matrice u *.csv* datoteku pod nazivom upisanim u *Enter Log File Name*

Sredina GUI programa služi za prikaz koordinata u realnom vremenu i spremanje početne i krajnje pozicije.

9.6. Rezultati provjere točnosti

U ovom poglavlju će se prezentirati rezultati provjere točnosti sva tri planera, dok će na kraju biti dana zajednička usporedba sva tri planera na istom grafu. Kako bi se točnost što bolje provjerila, za sva tri planera su ključni parametri postavljeni na vrijednosti prikazane u tablici 7. Podešavanje parametara i planera je napravljeno prema [24].

PARAMETAR	VRIJEDNOST
xy_goal_tolerance	0.05
yaw_goal_tolerance	0.13
sim_time	1.0
vx_samples	20
vtheta_samples	20
controller_frequency	10
max_vel_x	2.4
acc_lim_x	2.5
acc_lim_theta	3.2

Tablica 7. Parametri lokalnih planera

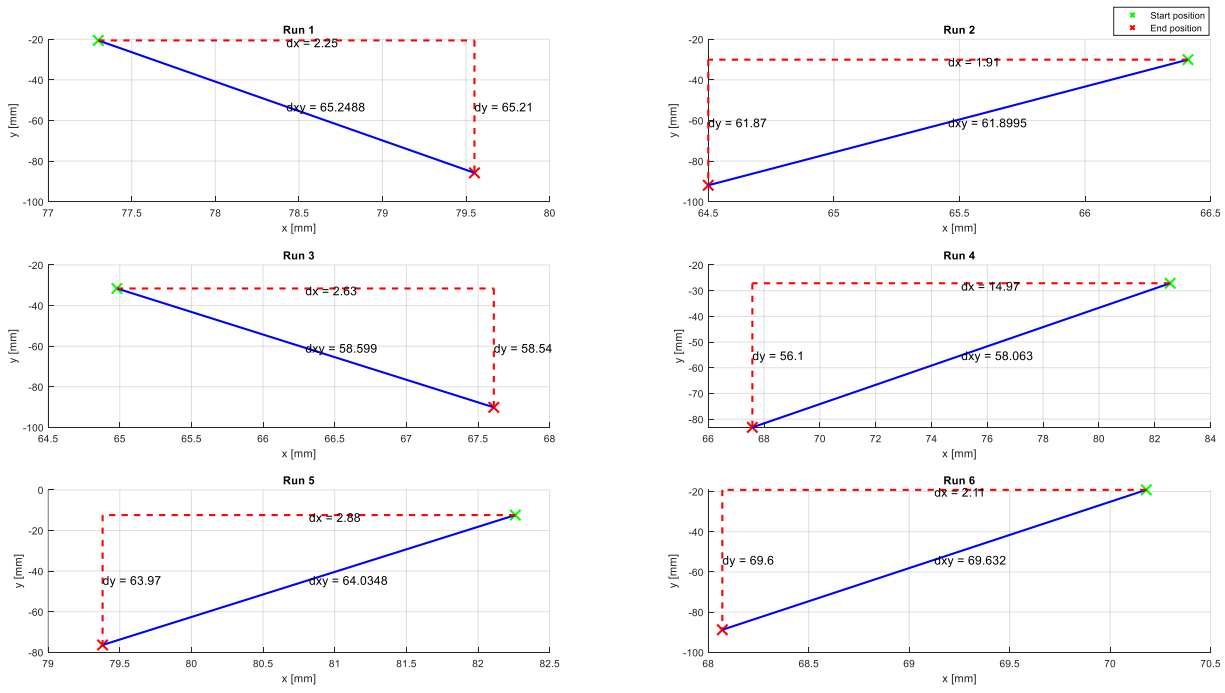
Osim definiranja tih parametara, potrebno je napraviti dva čvora koja šalju robotu potrebne trajektorije. U prilogu rada se nalaze kodovi za oba čvora.

9.6.1. Točnost TrajectoryPlannerROS planera

Tablica 8. prikazuje grešku koju planer napravi prateći statičku trajektoriju.

Br. snimanja	Greška u smjeru osi X (*dx*)	Greška u smjeru osi Y (*dy*)	Greška u smjeru osi X i Y (*dxy*)
1.	2.25 mm	65.21 mm	65.25 mm
2	1.91 mm	61.87 mm	61.9 mm
3.	2.63 mm	58.54 mm	58.6 mm
4.	14.97 mm	56.1 mm	58.06 mm
5.	2.88 mm	63.97 mm	64.03 mm
6.	2.11 mm	69.6 mm	69.63 mm
Srednja vrijednost	4.4583 mm	62.5483 mm	62.9128 mm
Standardna devijacija	5.1617 mm	4.8361 mm	4.3593 mm

Tablica 8. Pogreške nastale primjenom *TrajectoryPlannerROS* planera za statičnu kvadratnu trajektoriju

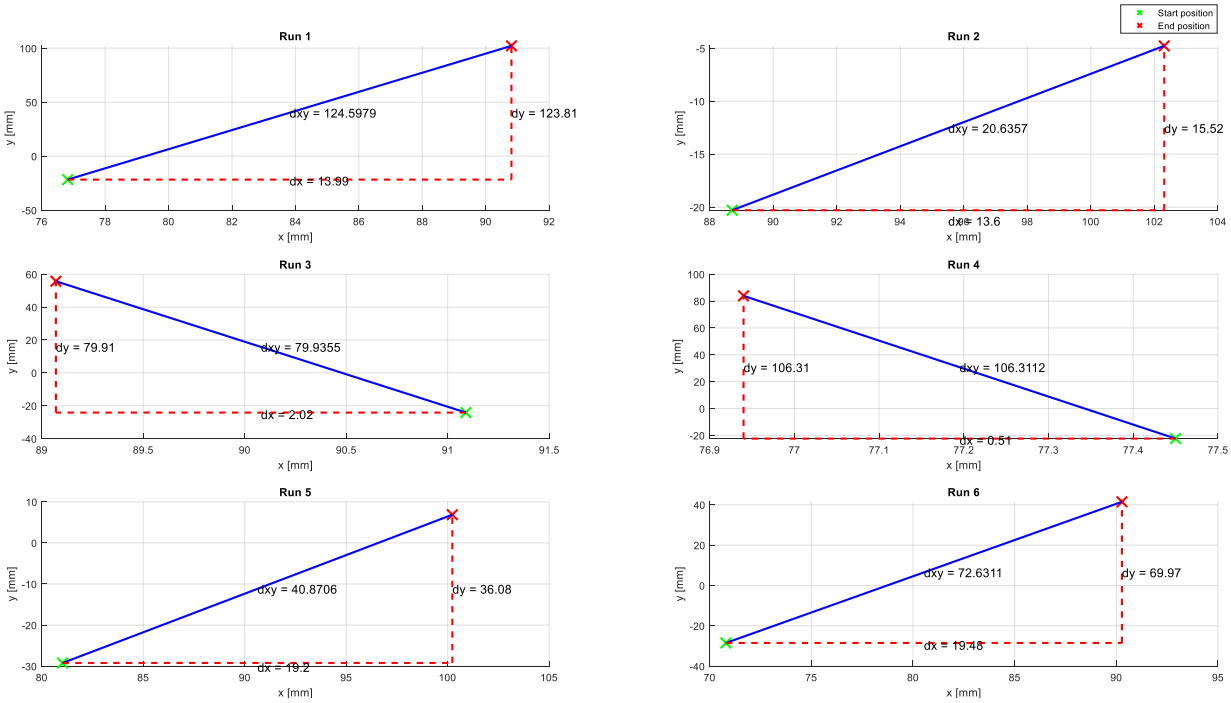


Slika 48. Grafički prikaz podataka iz tablice 7.

Tablica 9. prikazuje grešku koju planer napravi prateći dinamičku grešku.

Br. snimanja	Greška u smjeru osi X (*dx*)	Greška u smjeru osi Y (*dy*)	Greška u smjeru osi X i Y (*dxy*)
1.	13.99 mm	123.81 mm	124.6 mm
2	13.6 mm	15.52 mm	20.64 mm
3.	2.02 mm	79.91 mm	79.94 mm
4.	0.51 mm	106.31 mm	106.31 mm
5.	19.2 mm	36.08 mm	40.87 mm
6.	19.48 mm	69.97 mm	72.63 mm
Srednja vrijednost	11.4667 mm	71.9333 mm	74.1637 mm
Standardna devijacija	8.2973 mm	41.0082 mm	38.9523 mm

Tablica 9. Pogreške nastale primjenom *TrajectoryPlannerROS* planera za dinamičnu kvadratnu trajektoriju



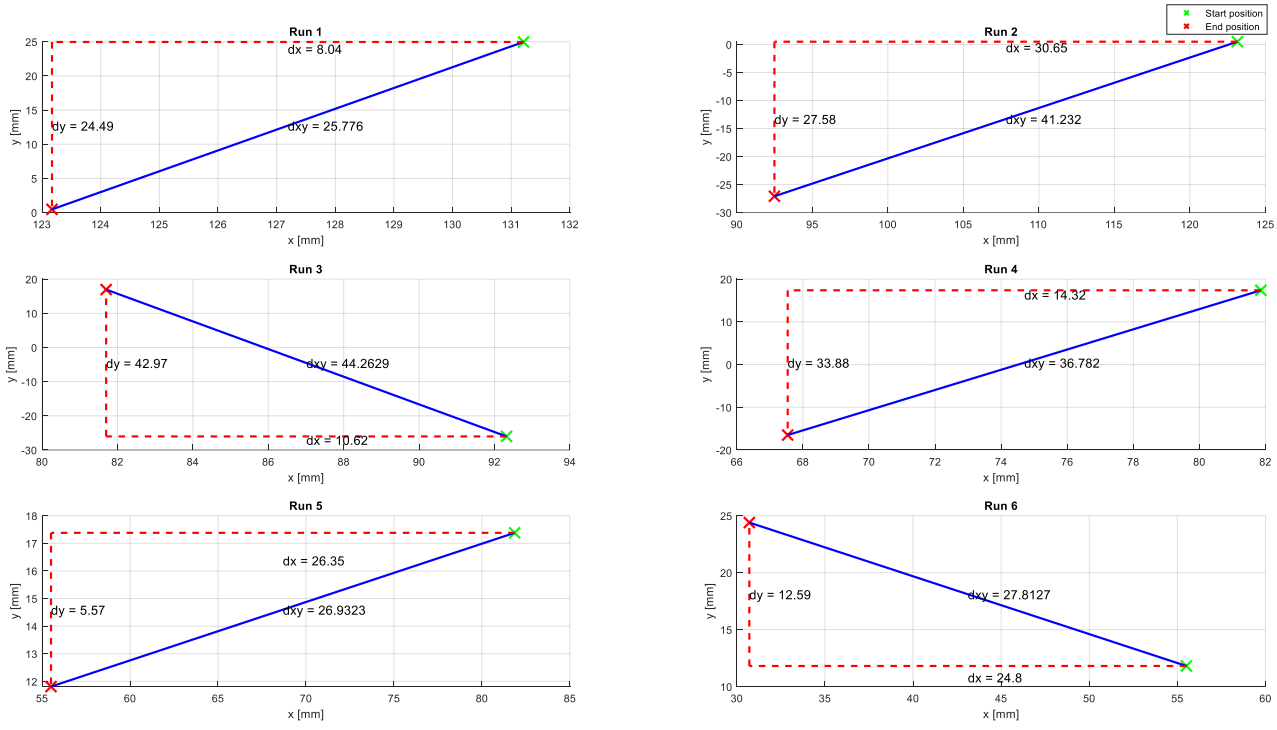
Slika 49. Grafički prikaz podataka iz tablice 8.

9.6.2. Točnost DWALocalPlanner planera

Tablica 10. pokazuje grešku koju planer napravi prateći statičku kvadratnu trajektoriju.

Br. snimanja	Greška u smjeru osi X (*dx*)	Greška u smjeru osi Y (*dy*)	Greška u smjeru osi X i Y (*dxy*)
1.	8.04 mm	24.49 mm	25.78 mm
2	30.65 mm	27.58 mm	41.23 mm
3.	10.62 mm	42.97 mm	44.27 mm
4.	14.32 mm	33.88 mm	36.78 mm
5.	26.35 mm	5.57 mm	26.93 mm
6.	24.8 mm	12.59 mm	27.81 mm
Srednja vrijednost	19.13 mm	24.5133 mm	33.7997 mm
Standardna devijacija	9.3331 mm	13.7026 mm	8.0125 mm

Tablica 10. Pogreške nastale primjenom *DWALocalPlanner* planera za statičku kvadratnu trajektoriju

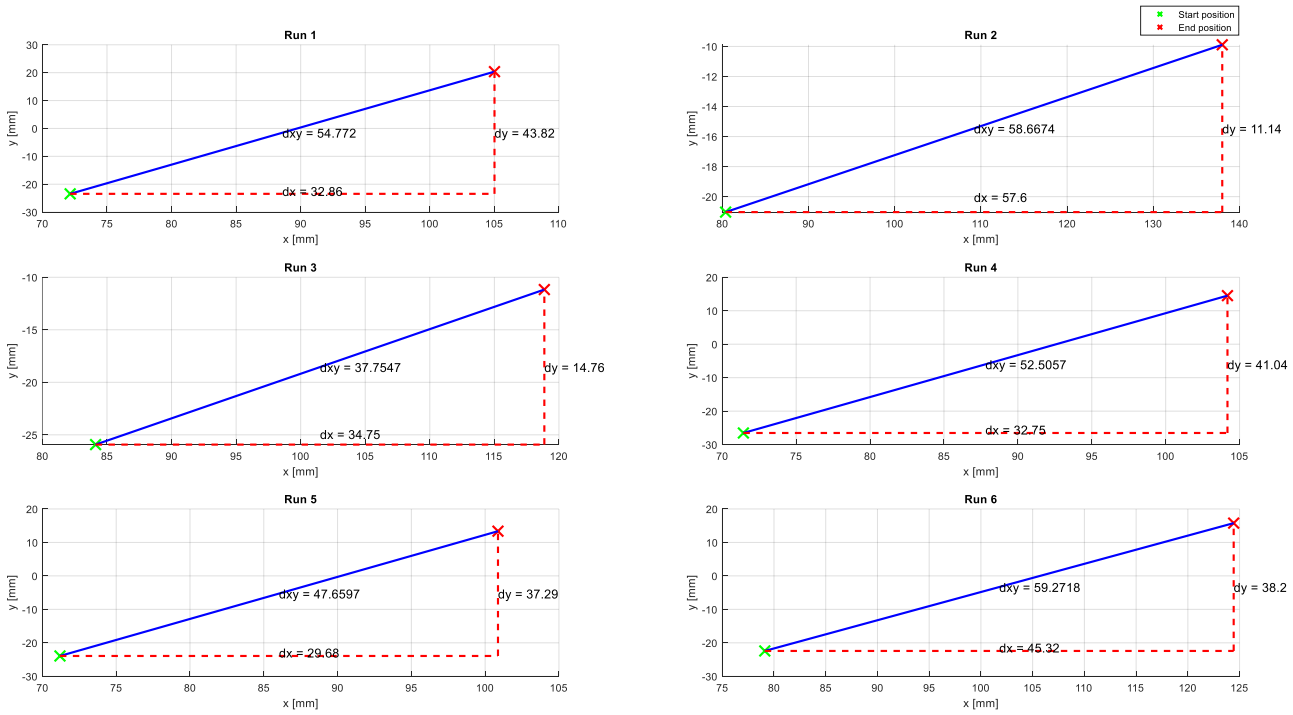


Slika 50. Grafički prikaz podataka iz tablice 9.

Tablica 11. pokazuje grešku koju planer napravi prateći dinamičnu kvadratnu trajektoriju.

Br. snimanja	Greška u smjeru osi X (*dx*)	Greška u smjeru osi Y (*dy*)	Greška u smjeru osi X i Y (*dxy*)
1.	32.86 mm	43.82 mm	54.77 mm
2	57.6 mm	11.14 mm	58.67 mm
3.	34.75 mm	14.76 mm	37.75 mm
4.	32.75 mm	41.04 mm	52.51 mm
5.	29.68 mm	37.29 mm	47.66 mm
6.	45.32 mm	38.2 mm	59.27 mm
Srednja vrijednost	38.8267 mm	31.0417 mm	51.7719 mm
Standardna devijacija	10.6528 mm	14.2457 mm	8.0836 mm

Tablica 11. Pogreške nastale primjenom *DWALocalPlanner* planera za dinamičku kvadratnu trajektoriju



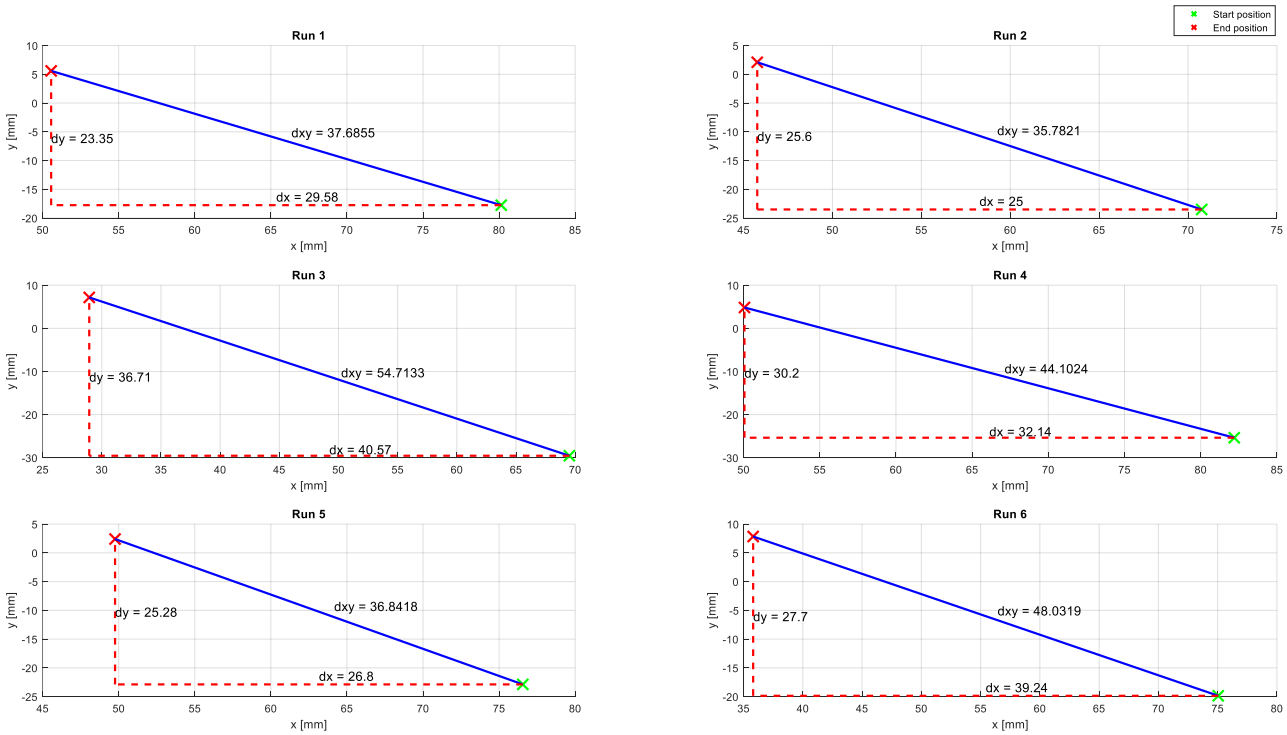
Slika 51. Grafički prikaz podataka iz tablice 10.

9.6.3. Točnost TEBLocalPlanner planera

Tablica 12. pokazuje grešku koju planer napravi prateći statičku kvadratnu trajektoriju.

Br. snimanja	Greška u smjeru osi X (*dx*)	Greška u smjeru osi Y (*dy*)	Greška u smjeru osi X i Y (*dxy*)
1.	29.58 mm	23.35 mm	37.69 mm
2	25.00 mm	25.6 mm	35.78 mm
3.	40.57 mm	36.71 mm	54.71 mm
4.	32.14 mm	30.2 mm	44.1 mm
5.	26.8 mm	25.28 mm	36.84 mm
6.	39.24 mm	27.7 mm	48.03 mm
Srednja vrijednost	32.2217 mm	28.14 mm	42.8595 mm
Standardna devijacija	6.4418 mm	4.8054 mm	7.5085 mm

Tablica 12. Pogreške nastale primjenom *TEBLocalPlanner* planera za statičku kvadratnu trajektoriju

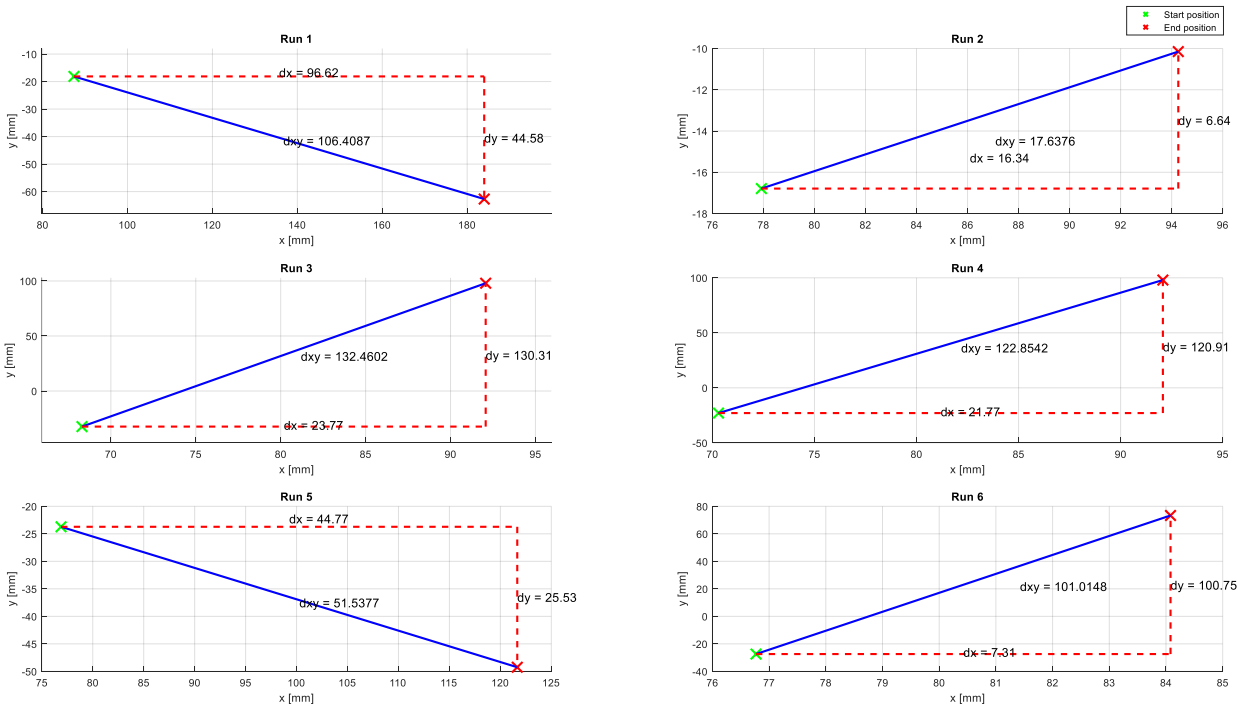


Slika 52. Grafički prikaz podataka iz tablice 11.

Tablica 13 pokazuje grešku koju planer napravi prateći dinamičku kvadratnu trajektoriju.

Br. snimanja	Greška u smjeru osi X (*dx*)	Greška u smjeru osi Y (*dy*)	Greška u smjeru osi X i Y (*dxy*)
1.	96.62 mm	44.58 mm	106.41 mm
2	16.34 mm	6.64 mm	17.64 mm
3.	23.77 mm	130.31 mm	132.46 mm
4.	21.77 mm	120.91 mm	122.85 mm
5.	44.77 mm	25.53 mm	51.54 mm
6.	7.31 mm	100.75 mm	101.01 mm
Prosječna vrijednost	35.0967 mm	71.4533 mm	88.6522 mm
Standardna devijacija	32.5837 mm	52.5362 mm	44.6713 mm

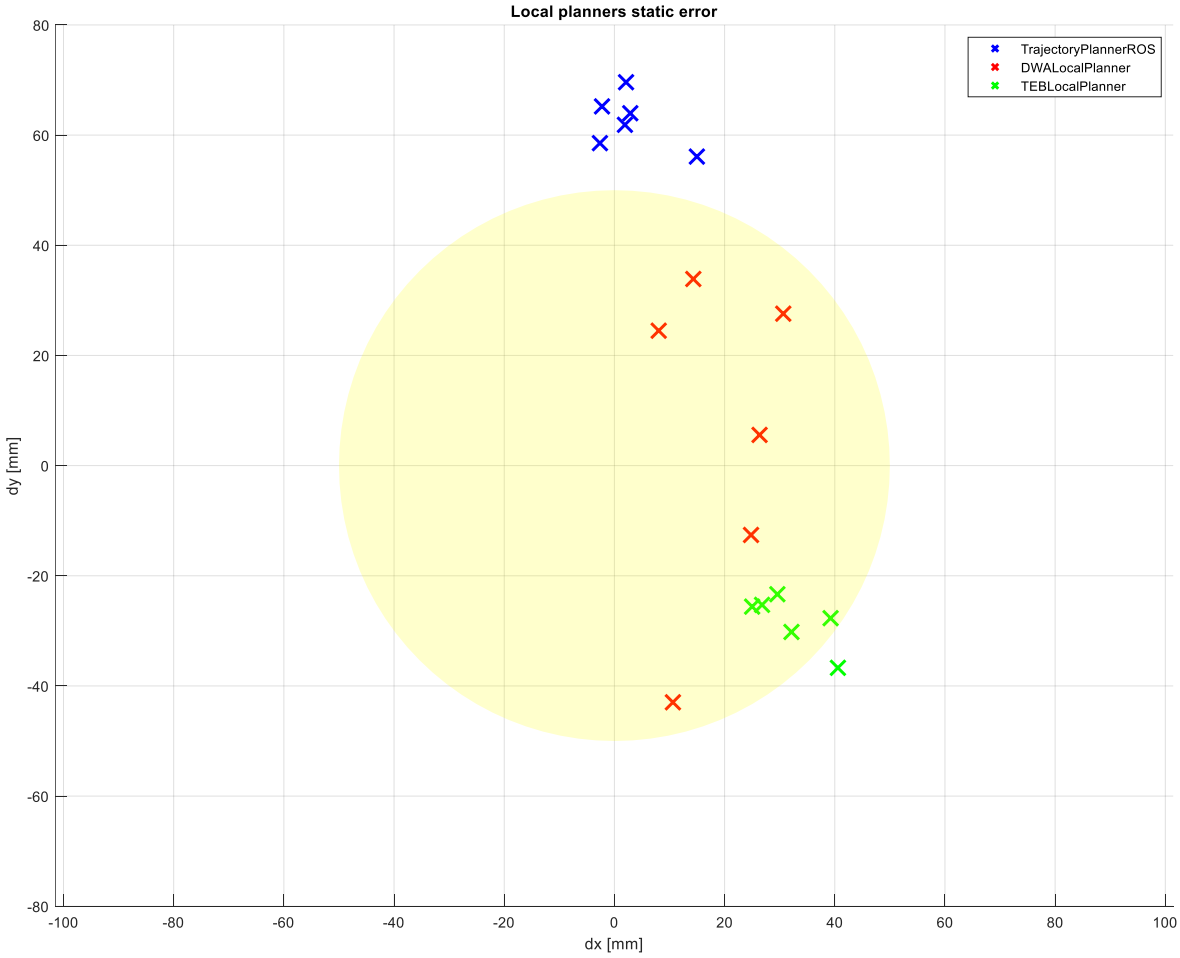
Tablica 13. Pogreške nastale primjenom *TEBLocalPlanner* planera za dinamičku kvadratnu trajektoriju



Slika 53. Grafički prikaz podataka iz tablice 12

9.6.4. Objedinjeni prikaz rezultata i zaključak mjerenja

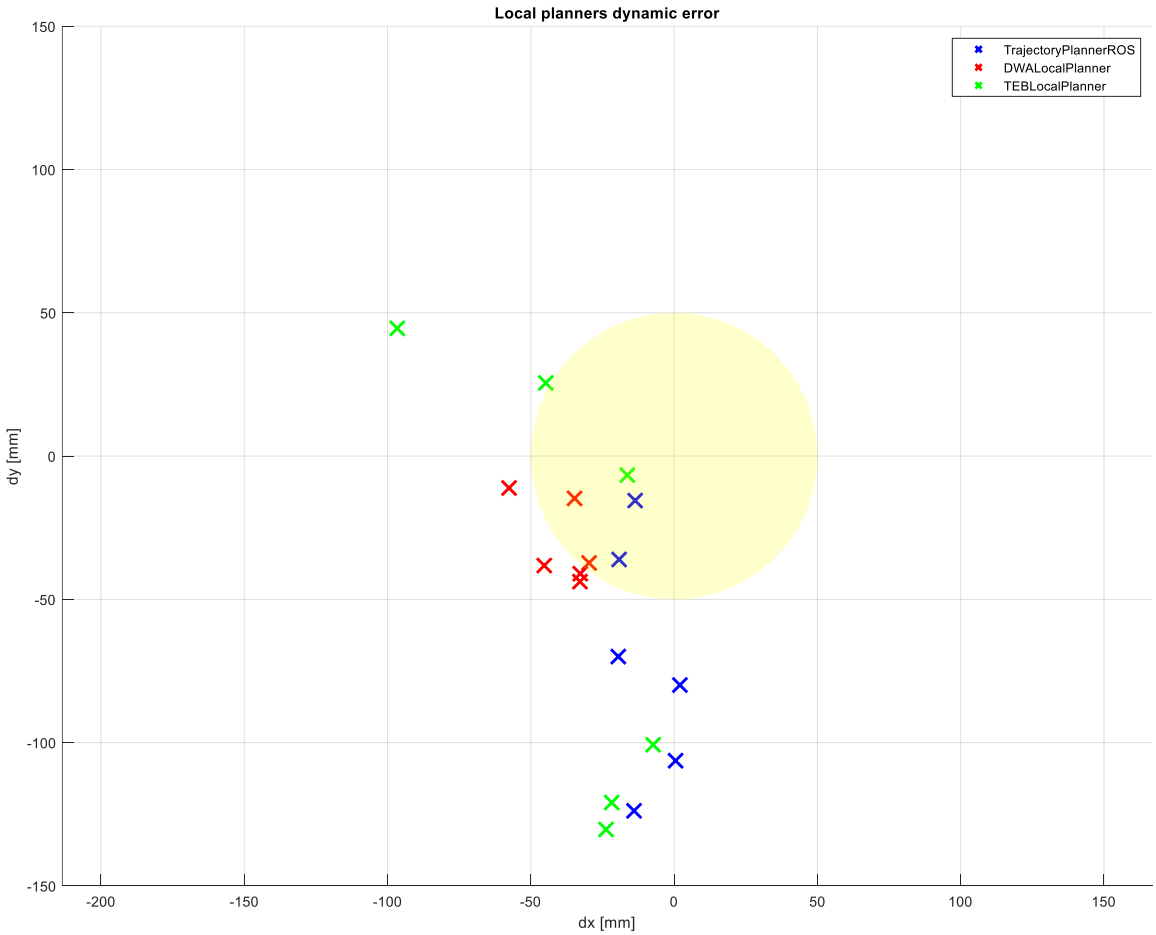
Na slici 54. su prikazani objedinjeni rezultati grešaka koje robot napravi prateći statičku kvadratnu trajektoriju.



Slika 54. Objedinjeni prikaz grešaka lokalnih planera za statičku trajektoriju

Sa slike je vidljivo kako greške *DWALocalPlanner* i *TEBLocalPlanner* planera ne izlaze iz zone postavljene tolerancije (žuti krug na slici). Unatoč ispadanju van granica tolerancije, *TrajectoryPlannerROS* se pokazao kao planer s iznimno dobrom ponovljivosti kao i *TEBLocalPlanner*. Zaključak na temelju grafičkog prikaza je da je *TEBLocalPlanner* najbolji odabir ukoliko se mobilnom robotu šalju ciljevi u vidu koordinata koje se postavljaju u odnosu na globalni koordinatni sustav jer greške ne ispadaju iz tolerancijske kao i *DWALocalPlanner* lokalnom planeru, ali ima bolju ponovljivost u odnosu na *DWALocalPlanner*.

Slika 55. prikazuje greške koje nastaju u sva tri lokalna planera.



Slika 55. Objedinjeni prikaz grešaka lokalnih planera za dinamičku trajektoriju

Sa slike 55. je vidljivo kako prilikom praćenja dinamičke trajektorije dolazi do puno većih grešaka na kraju. Glavni razlog tome je nakupljanje grešaka jer se robotu svaki put daje pozicija relativno na njegovu trenutnu poziciju te se greška prethodne pozicije nastavlja na svaku sljedeću.

Lokalni planer *DWALocalPlanner* se pokazao kaj najbolji planer za praćenje dinamičkih točaka, jer mu je ponovljivost zadovoljavajuća i greške se nalaze relativno blizu krugu tolerancije.

Temeljem rezultata prikazanih na slikama 54 i 55 se može zaključiti kako je optimalni lokalni planer za navigaciju po prostoru *TEBLocalPlanner*. Za točnije pozicioniranje mobilnih robota se ne preporučuje korištenje gotovih planera, jer se smanjenjem vrijednosti parametra *xy_goal_tolerance* ispod 0.05m drastično povećava vrijeme praćenja trajektorije te robot počinje oscilirati oko krajnje točke što unosi problem sporosti realiziranja navigacije po prostoru. Ukoliko postoje zahtjevi za preciznim pozicioniranjem, idealna strategija je da se

robot po mapi navodi lokalnim planerom, uz uvjet da mu se zadaju koordinate u odnosu na globalni koordinatni sustav jer su ispitivanja pokazala kako su točnosti značajno bolje. Nakon što robot dođe na krajnju poziciju u mapi, kako bi što bolje pozicionirao potrebno je koristeći vizijski sustav pronaći karakterističnu značajku (npr. natpis, objekt itd.) te se pozicionirati relativno u odnosu na njega.

10. KALIBRACIJA MOBILNOG ROBOTA

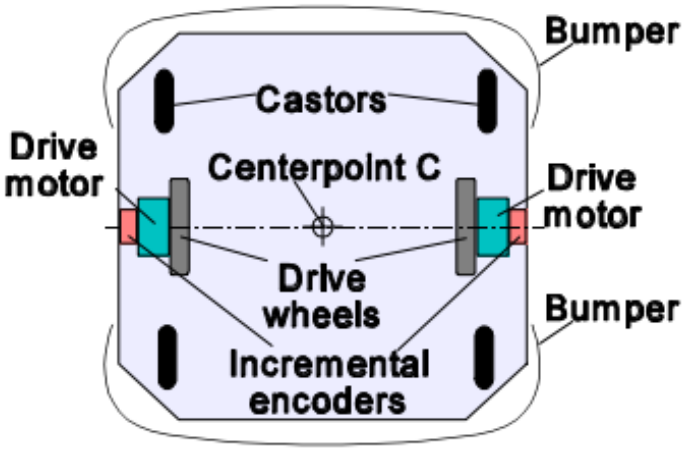
Osnovna karakteristika mobilnih robota je sposobnost kretanja po prostoru. Kako bi kretanje po prostoru bilo što kvalitetnije, potrebno je adekvatno estimirati poziciju robota. Za estimaciju pozicije mobilnog robota postoje jednostavni matematički modeli koji koriste enkodere motora kako bi estimirali relativnu poziciju robota u odnosu na početni koordinatni sustav. Kako ni jedan matematički model ne može idealno opisati realni sustav, potrebno je određene parametre modela izmjeriti na samom robotu da bi se dobili što točniji estimirani podaci. Osim klasične estimacije pozicije, za kvalitetnije mapiranje, lokalizaciju i navigaciju mobilnog robota, potrebno je, što je to moguće točnije, odrediti poziciju lidar senzora smještenog na robota.

U ovom poglavlju je detaljno razrađena metoda kalibracije odometrijskog sustava robota, dok je kalibracija pozicije lidar senzora detaljno opisana i provedena eksperimentalno.

10.1. Svojstva *dead-reckoning* grešaka

U većini primjena mobilnih robota, javljaju se dva osnovna načina estimacije pozicije: apsolutno i relativno pozicioniranje. Relativno pozicioniranje robota obično se temelji na mrtvom računanju (*engl. dead-reckoning)*, točnije praćenju okretaja enkodera te izračunom pomaka s poznatog početnog položaja. *Dead-reckoning* je jednostavno, jeftino i primjenjivo u realnom vremenu.

Slika 56. prikazuje mobilnog robota s diferencijalnim pogonom. Na konstrukciji robota se nalaze inkrementalni enkoderi montirani na pogonske motore kako bi se brojali okretaji motora, a sukladno tome i kotača. Korištenjem jednostavnih matematičkih formula moguće je izračunati poziciju robota relativno u odnosu na već poznatu startnu poziciju.



Slika 56. Mobilni robot s osnovnim segmentima: bumper (*hrv. branik*), castor (*hrv. pomoćni kotačić*), drive motor (*hrv. pogonski motor*), drive wheels (*hrv. kotači*), incremental encoder (*hrv. inkrementalni enkoder*), centerpoint C (*hrv. središnja točka C*)

Pretpostavljeno je da se uzorkuju impulsi koje daju lijevi i desni enkoder te je porast inkremenata imenovan N_L za lijevi motor i N_R za desni motor. Uvodi se oznaka:

$$c_m = \frac{\pi D_n}{n C_e}, \tag{9}$$

gdje je:

- c_m – koeficijent pretvorbe impulsa enkodera u linearni pomak kotača
- D_n – nazivni promjer kotača [mm]
- C_e – rezolucija enkodera [impulsi/okretaju]
- n – omjer redukcije zupčanika između motora i kotača.

S uvedenim koeficijentom u izrazu (9) se mogu izračunati inkrementalni pomaci lijevog kotača ΔU_L i desnog kotača ΔU_R prema izrazu (10):

$$\Delta U_{L/R,i} = c_m N_{L/R,i} \tag{10}$$

Primjenom izraza (10) za pomak lijevog i desnog kotača se može izračunati inkrementalni linearni pomak središnje točke robota C (11) i inkrementalna rotacija robota (12):

$$\Delta U_i = \frac{\Delta U_{L,i} + \Delta U_{R,i}}{2} \tag{11}$$

$$\Delta \theta_i = \frac{\Delta U_{R,i} - \Delta U_{L,i}}{b} \tag{12}$$

gdje je b nazivna udaljenost između idealnih dodirnih točaka kotača i površine.

Nova pozicija robota, relativna na prethodnu se može izračunati koristeći izraz:

$$x_i = x_{i-1} + \Delta U_i \cos \theta_i \tag{13}$$

$$y_i = y_{i-1} + \Delta U_i \sin \theta_i \tag{14}$$

gdje su x_i i y_i relativne pozicije središnje točke robota C u inkrementu i.

Relativna orijentacija robota θ_i se može izračunati primjenom izraza:

$$\theta_i = \theta_{i-1} + \Delta \theta_i \tag{15}$$

Jednadžbe (9) do (15) pokazuju da je implementacija estimatora pozicije mobilnog robota jednostavna pod uvjetom da se rotacija kotača idealno pretvori u translacijsko gibanje. Ekstreman primjer nemogućnosti te pretvorbe je glatka i klizava površina tla koja onemogućuje tu pretvorbu. Nedostatak estimacije pozicije primjenom gore navedenih izraza je nakupljanje grešaka uzrokovanih sustavnim i nesustavnim anomalijama.

10.1.1 Nesustavne greške

Nesustavne greške su uzrokovane interakcijom robota s nepredvidivim segmentima okoline u kojoj se robot nalazi. Ispupčenja, pukotine, krhotine i prašina su nepravilnosti površine koje mogu uzrokovati da se kotač robota okreće više nego što je predviđeno te uzrokuje grešku mjerenja pozicije robota. Nesustavne greške su značajan problem prilikom pozicioniranja mobilnog robota jer se na njih ne može jednostavno utjecati.

10.1.2. Sustavne greške

Sustavne greške mobilnog robota su najčešće uzrokovane nesavršenostima u konstrukciji robota i greškama prilikom izrade samog robota. Najčešće greške su:

- nejednakost promjera kotača – glavni uzrok nejednakosti promjera je materijal od kojeg su proizvedeni kotači. Kako bi se izbjeglo proklizavanje kotača, potrebno je kotače izraditi od gume koja s većinom podloga ima zadovoljavajuće trenje. Gumene kotače je iznimno zahtjevno za proizvesti da budu jednakih promjera. Osim proizvodnih grešaka, problem nastaje i zbog same strukture gume. Guma ima elastična svojstva što zajedno s nesimetričnim opterećenjem može uzrokovati razlike u promjerima kotača. Oba od navedenih problema mogu uzrokovati znatno nakupljanje greške prilikom rada robota.
- nesigurnost u razmak među kotačima – udaljenost među kotačima se definira kao udaljenost između dodirnih točaka lijevog i desno kotača te podloge. Poznavanje udaljenosti među kotačima je nužna kako bi se točno mogla odrediti rotacija mobilnog robota. Nesigurnost u točan razmak među kotačima je uzrokovana činjenicom da dodir između kotača i podloge nije u točki nego u dodirnoj površini. Razlika između konstrukcijske i stvarne udaljenosti među kotačima varira oko 1% u komercijalnim mobilnim platformama.
- Ostale sustavne greške mobilnih robota su: prosječan promjer kotača robota se razlikuje od nominalne vrijednosti, neusklađenost kotača, limitirana rezolucija enkodera i limitirana frekvencija uzorkovanja enkodera.

Greška na koju se može utjecati je nejednakost promjera kotača, stoga se definira izraz za grešku:

$$E_d = \frac{D_R}{D_L} \tag{16}$$

gdje su D_R promjer desnog kotača i D_L promjer lijevog kotača, gdje E_d govori u kojem se omjeru razlikuju promjeri lijevog i desnog kotača. Idealan omjer je 1.

Osim nejednakosti promjera kotača, moguće je utjecati na razmak između kotača. Omjer stvarne i nazivne udaljenosti između kotača je:

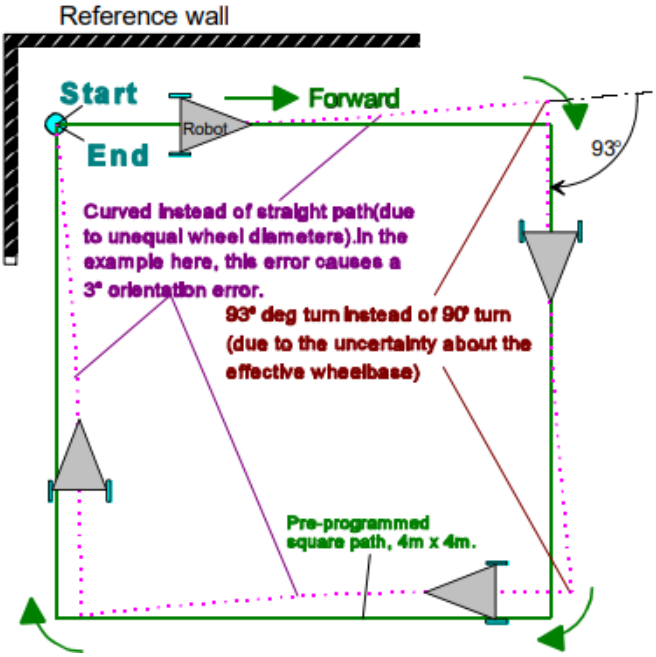
$$E_b = \frac{b_a}{b_n} \tag{17}$$

gdje je b_a stvarni, a b_n nazivni razmak između kotača.

10.2. Mjerenje sustavnih grešaka

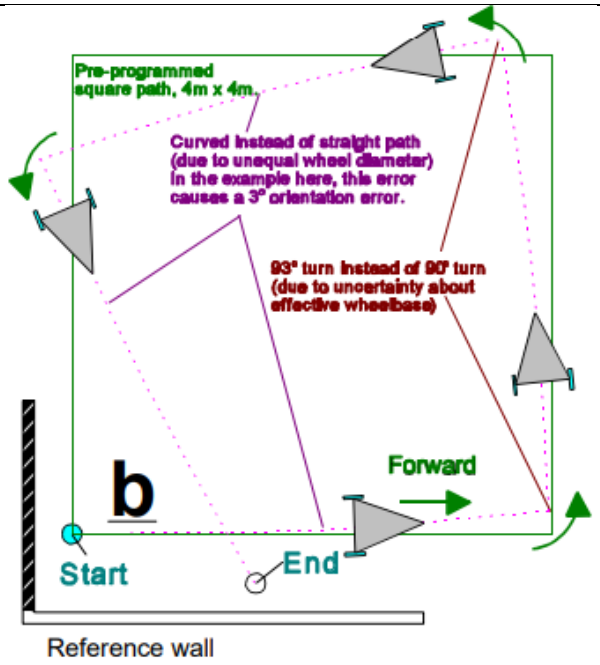
Kako bi se sustavne greške ispravile, potrebno ih je mjeriti. Standardizirani način mjerenja je predstavljen u [21] pod nazivom The bi-directional square path experiment: „UMBMark“ (*hrv. eksperiment dvosmjerne kvadratne trajektorije:"UMBMark"*).

Eksperiment se provodi na kvadratnoj trajektoriji dimenzija 4m x 4m u smjeru kazaljke i suprotno od kazaljke na satu. Slika 57. prikazuje eksperiment u samo jednom smjeru iz koje se može zaključiti da se robot vratio u istu poziciju iz koje je krenuo prilikom ručnog podešavanja parametara udaljenosti među kotačima i promjera kotača.



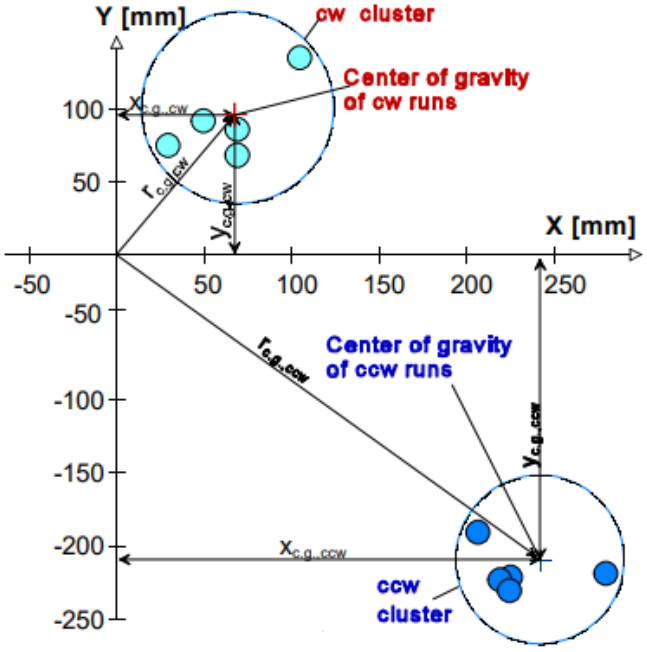
Slika 57. Utjecaj dominantnih sustavnih grešaka E_b i E_d. Primjetno je kako se obje anomalije međusobno mogu poništiti ukoliko se test provodi samo u jednom smjeru [21]

Slika 58. prikazuje da skrivena pogreška sa slike 57. postaje jasno vidljiva kada robot prođe kvadratnu trajektoriju u suprotnom smjeru. Razlog tome je što se dvije dominantne greške, koje kompenziraju jedna drugu kada robot prolazi trajektoriju u jednom smjeru, međusobno nadodaju i povećavaju krajnju grešku u suprotnom smjeru.



Slika 58. Prikaz nakupljanja greške uzrokovane dominantnim sustavnim greškama E_b i E_d u slučaju prolaska robota kvadratnom trajektorijom u suprotnom smjeru [21]

U sklopu eksperimenta robot je programiran da prati referentnu trajektoriju kvadratnog oblika, dimenzija 4m x 4m, počevši iz pozicije (0,0). Za primjer kako se eksperiment provodi korišteni su podaci iz [21]. Krajnje pozicije robota za pet vožnji po trajektoriji u smjeru kazaljke na satu i smjeru suprotnom od kazaljke na satu su dani na slici 59.



Slika 59. Tipičan prikaz rezultata nakon *UMBMark* eksperimenta s ne kalibriranim robotom u smjeru kazaljke na satu (*eng. cw direction*) i suprotno od kazaljke na satu (*eng. ccw direction*) [21]

Primjer rezultata sa slike 59. može biti interpretiran na dva načina:

- zaustavne pozicije robota nakon vožnje u oba smjera su grupirane u dva različita klastera
- distribucija krajnjih točaka unutar klastera je rezultat nesustavnih anomalija

Slika 59. prikazuje kako na nekalibriranog robota osim sustavnih grešaka, utječu i nesustavne greške, ali njihov utjecaj je vidno manji (međusobna udaljenost klastera je znatno veća od međusobnih udaljenosti krajnjih pozicija unutar samog klastera).

Nakon provedbe eksperimenta, potrebno je odrediti jednu numeričku vrijednost kojom će se opisati greška estimacije pozicije s obzirom na sustavne greške. Kako bi se minimizirao utjecaj nesustavnih grešaka, potrebno je pronaći težište svakog od klastera kao vrijednost koja će predstavljati svaki klaster.

Koordinate težišta klastera se računaju kao:

$$x_{c.g.,cw/ccw} = \frac{1}{n}\sum_{i=1}^{n} \in x_{i,cw/ccw} \quad \textbf{(18)}$$

$$y_{c.g.,cw/ccw} = \frac{1}{n}\sum_{i=1}^{n} \in y_{i,cw/ccw} \quad \textbf{(19)}$$

gdje je n broj vožnji robota u svakom smjeru, a ostali parametri su:

$$\in x = x_{abs} - x_{calc} \quad \textbf{(20)}$$

$$\in y = y_{abs} - y_{calc} \quad \textbf{(21)}$$

gdje su:

- $\in x$, $\in y$ – greške po X i Y osi nastale zbog loše odometrije
- x_{abs}, y_{abs} – apsolutna pozicija robota u X i Y ravnini
- x_{calc}, y_{calc} – pozicija robota u X i Y ravnini dobivena iz odometrije.

Apsolutna udaljenost težišta klastera od ishodišta se izračunava primjenom sljedećeg izraza:

$$r_{c.g.,cw/ccw} = \sqrt{(x_{c.g.,cw/ccw})^2 + (y_{c.g.,cw/ccw})^2} \quad \textbf{(22)}$$

Na kraju se veća vrijednost od oba klastera koristi kao točnost odometrije za sustavne greške.

$$E_{max,syst} = \max\left(r_{c.g.,cw}, r_{c.g.,ccw}\right) \quad \textbf{(23)}$$

Razlog zašto se ne koristi srednja vrijednost udaljenosti dvaju težišta klastera je prema [21] takav da je za praktičnu primjenu korisnije raditi s najvećom mogućom greškom koju robot napravi.

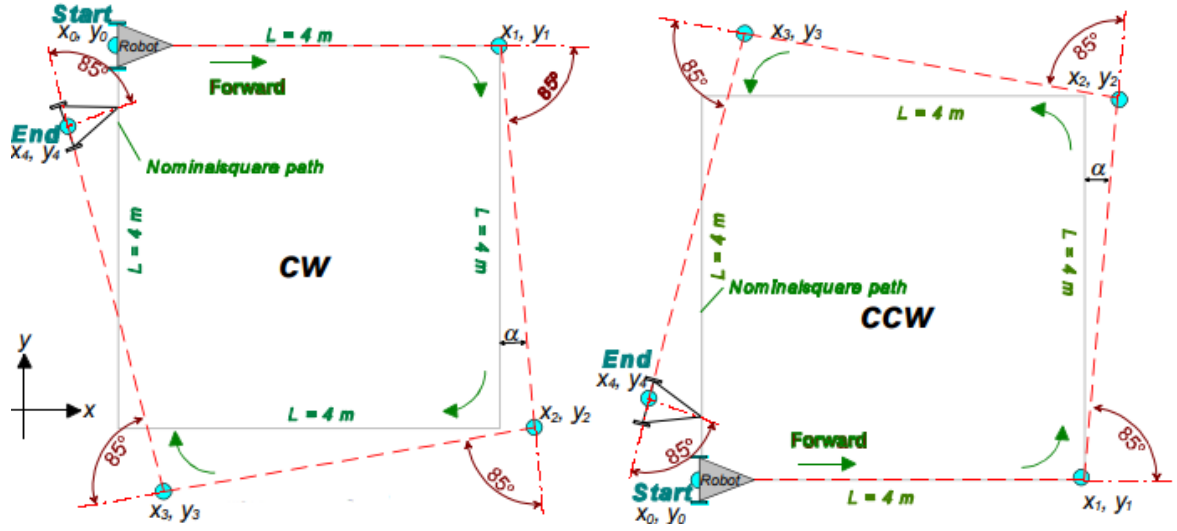
Iz izraza (23) je vidljivo kako krajnja orijentacija θ nije uzeta u obzir. Razlog tome je što zbog simetričnosti trajektorije s obje strane, sustavna greška orijentacije direktno utječe na grešku pozicije. Detaljnije će biti objašnjeno u nastavku rada.

10.3. Analiza sustavne greške

Osnovna prednost *UMBMark* eksperimenta je u tome što uz relativno jednostavna mjerenja i analitičke jednadžbe se može doći do drastičnih poboljšanja u odometriji robota. Prije raspisa matematičkog modela pogreške, potrebno je definirati dva tipa grešaka, tip A i tip B.

10.3.1. Greška tipa A

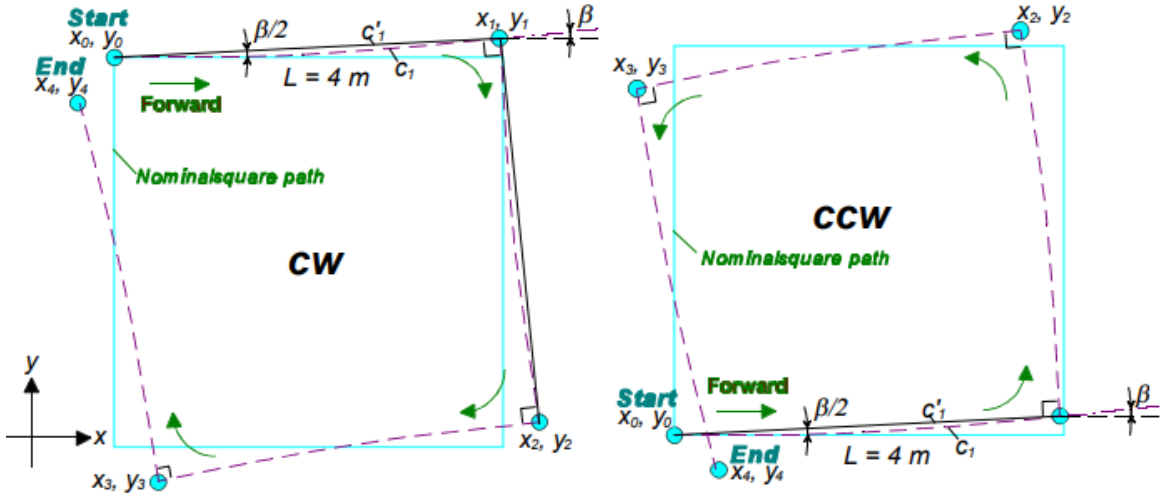
Greška tipa A je definirana kao greška orijentacije koja povećava ili smanjuje ukupnu količinu rotacije robota prilikom izvođenja kvadratne trajektoriju u smjeru kazaljke na satu i u suprotnom smjeru kazaljke na satu. Ukoliko se pretpostavi da je stvarni kut zakreta robota manji od nazivnog (90°) za oba smjera se može napisati: $|\theta_{total,cw}| < |\theta_{nominal,cw}|$ i $|\theta_{total,ccw}| < |\theta_{nominal,cw}|$. Greška tipa A nastaje najčešće zbog lošeg omjera stvarne i nazivne udaljenosti među kotačima robota E_b. Slika 60. prikazuje anomaliju tipa A u oba smjera prolaženja kvadratnom trajektorijom.



Slika 60. Anomalija tipa A u oba smjera praćenja kvadratne trajektorije nastala zbog omjera E_b [21]

10.3.2. Greška tipa B

Greška tipa B se može definirati kao orijentacijska greška koja povećava ili smanjuje orijentaciju u jednom smjeru te suprotno tome smanjuje ili povećava orijentaciju u drugom smjeru. Tako se za jedan smjer praćenja kvadratne trajektorije može zapisati $|\theta_{total,cw}| < |\theta_{nominal,cw}|$, dok će za drugi smjer biti $|\theta_{total,ccw}| > |\theta_{nominal,cw}|$ i obrnuto. Greška tipa B je najčešće uzrokovana lošim omjerom promjera kotača E_d. Slika 61. prikazuje anomaliju tipa B.



Slika 61. Anomalija tipa B u oba smjera praćenja kvadratne trajektorije nastala zbog omjera E_d [21]

10.3.3. Matematičke pretpostavke

Nakon što je definiran model potrebno je uvesti pretpostavke kako bi se pojednostavnio matematički model greške koja nastaje. Aproksimacije koje su uveden za male kutove grešaka nastale slijeđenjem kvadratne trajektorije su:

$$\begin{aligned} L\sin\gamma &\approx L\gamma \\ L\sin 2\gamma &\approx 2L\gamma \\ L\sin 3\gamma &\approx 3L\gamma \end{aligned} \qquad \textbf{(24)}$$

$$\begin{aligned} L\cos\gamma &\approx L \\ L\cos 2\gamma &\approx L \\ L\cos 3\gamma &\approx L \end{aligned} \qquad \textbf{(25)}$$

gdje su:

- L – duljina svake stranice kvadratne trajektorije
- γ – svaka inkrementalna greška uzrokovana E_d ili E_b

Još jedna pretpostavka je da je početna pozicija (x_0, y_0) mobilnog robota u $(0, 0)$.

10.3.4. Matematički model grešaka tipa A i B

Za grešku tipa A u smjeru suprotnom od kazaljke na satu prema slici 60. se mogu raspisati sljedeći izrazi:

$$\begin{aligned} x_1 &= x_0 + L \\ y_1 &= y_0 \end{aligned} \quad \textbf{(26)}$$

$$\begin{aligned} x_2 &= x_1 + L\sin\alpha \approx L + L\alpha \\ y_2 &= y_1 + L\cos\alpha \approx L \end{aligned} \quad \textbf{(27)}$$

$$\begin{aligned} x_3 &= x_2 - L\cos 2\alpha \approx L\alpha \\ y_3 &= y_2 + L\sin 2\alpha \approx L + 2L\alpha \end{aligned} \quad \textbf{(28)}$$

$$\begin{aligned} x_4 &= x_3 - L\sin 3\alpha \approx -2L\alpha \\ y_4 &= y_3 - L\cos 3\alpha \approx 2L\alpha \end{aligned} \quad \textbf{(29)}$$

Za grešku tipa A u smjeru kazaljke na satu prema slici 60. se mogu raspisati sljedeći izrazi:

$$\begin{aligned} x_1 &= x_0 + L \\ y_1 &= y_0 \end{aligned} \quad \textbf{(30)}$$

$$\begin{aligned} x_2 &= x_1 + L\sin\alpha \approx L + L\alpha \\ y_2 &= y_1 - L\cos\alpha \approx -L \end{aligned} \quad \textbf{(31)}$$

$$\begin{aligned} x_3 &= x_2 - L\cos 2\alpha \approx L\alpha \\ y_3 &= y_2 - L\sin 2\alpha \approx -L - 2L\alpha \end{aligned} \quad \textbf{(32)}$$

$$\begin{aligned} x_4 &= x_3 - L\sin 3\alpha \approx -2L\alpha \\ y_4 &= y_3 + L\cos 3\alpha \approx -2L\alpha \end{aligned} \quad \textbf{(33)}$$

Zbog krivudavog kretanja uzrokovanog E_d robot stvara grešku orijentacije označenu s β na slici 59. Važno je primijetiti kako linija c'_1, spaja stvarne točke robota i ima nagib od $\frac{\beta}{2}$ zato što je paralelna s tangentom u središtu luka c_1, Za grešku tipa B u smjeru suprotnom od kazaljke na satu prema slici 61. se mogu raspisati sljedeći izrazi:

$$\begin{aligned} x_1 &= x_0 + L\cos\frac{\beta}{2} \approx L \\ y_1 &= y_0 + L\sin\frac{\beta}{2} \approx L\frac{\beta}{2} \end{aligned} \quad \textbf{(34)}$$

$$\begin{aligned} x_2 &= x_1 - L\sin\frac{3\beta}{2} \approx L - \frac{3\beta}{2} \\ y_2 &= y_1 + L\cos\frac{3\beta}{2} \approx L\frac{\beta}{2} + L \end{aligned} \quad \textbf{(35)}$$

$$x_3 = x_2 - L\cos\frac{5\beta}{2} \approx -3L\frac{\beta}{2}$$
$$y_3 = y_2 - L\sin\frac{5\beta}{2} \approx -2L\beta + L \quad \textbf{(36)}$$

$$x_4 = x_3 + L\sin\frac{7\beta}{2} \approx 2L\beta$$
$$y_4 = y_3 - L\cos\frac{7\beta}{2} \approx -2L\beta \quad \textbf{(37)}$$

Za grešku tipa B u smjeru kazaljke na satu prema slici 61. se mogu raspisati sljedeći izrazi:

$$x_1 = x_0 + L\cos\frac{\beta}{2} \approx L$$
$$y_1 = y_0 + L\sin\frac{\beta}{2} \approx L\frac{\beta}{2} \quad \textbf{(38)}$$

$$x_2 = x_1 + L\sin\frac{3\beta}{2} \approx L + 3L\frac{\beta}{2}$$
$$y_2 = y_1 - L\cos\frac{3\beta}{2} \approx L\frac{\beta}{2} - L \quad \textbf{(39)}$$

$$x_3 = x_2 - L\,cos\frac{5\beta}{2} \approx 3L\frac{\beta}{2}$$
$$y_3 = y_2 - L\,sin\frac{5\beta}{2} \approx -L(2\beta + 1) \quad \textbf{(40)}$$

$$x_4 = x_3 - L\,sin\frac{7\beta}{2} \approx -2L\beta$$
$$y_4 = y_3 - L\,cos\frac{7\beta}{2} \approx -2L\beta \quad \textbf{(41)}$$

Superponiranjem grešaka tipa A i B za kretanje robota u smjeru kazaljke na sat u smjeru osi X vrijede izrazi:

$$x_{c.g.,cw} = -2L(\alpha + \beta) \quad \textbf{(42)}$$

$$x_{c.g.,ccw} = -2L(\alpha - \beta) \quad \textbf{(43)}$$

oduzimanjem (43) od (42)

$$-4L\beta = x_{c.g.,cw} - x_{c.g.,ccw} \quad \textbf{(44)}$$

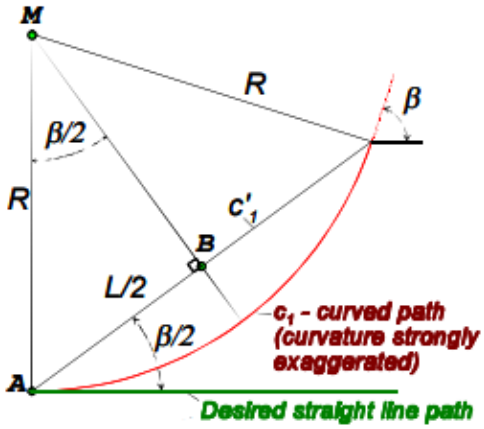
odnosno u stupnjevima β iznosi:

$$\beta = \frac{x_{c.g.,cw} - x_{c.g.,ccw}}{-4L} * \frac{180°}{\pi} \quad \textbf{(45)}$$

Za pomake po Y osi vrijede slični rezultati:

$$\beta = \frac{y_{c.g.,cw} - y_{c.g.,ccw}}{-4L} * \frac{180°}{\pi} \quad \textbf{(46)}$$

Nakon dobivenih izraza, koristeći jednostavne geometrijske odnose može se izračunati polumjer krivulje R sa slike 62. koristeći trokut ABM sa slike 60.



Slika 62. Geometrijski odnosi za izračun polumjera skretanja *R* [21]

$$R = \frac{L/2}{\sin(\frac{\beta}{2})} \quad \textbf{(47)}$$

Nakon što je izračunat polumjer, jednostavno je dobiti omjer promjera lijevog i desnog kotača E_d:

$$E_d = \frac{D_R}{D_L} = \frac{R + \frac{b}{2}}{R - \frac{b}{2}} \quad \textbf{(48)}$$

Slično kako je pronađen kut β, može se pronaći i kut α, zbrajanjem jednadžbi (42) i (43):

$$-4L\alpha = x_{c.g.,cw} + x_{c.g.,ccw} \quad \textbf{(49)}$$

odnosno kut α u stupnjevima iznosi:

$$\alpha = \frac{x_{c.g.,cw} + x_{c.g.,ccw}}{-4L} * \frac{180°}{\pi} \quad \textbf{(50)}$$

Za pomake u Y osi:

$$\alpha = \frac{y_{c.g.,cw} - y_{c.g.,ccw}}{-4L} * \frac{180°}{\pi} \quad \textbf{(51)}$$

S izračunatim kutom α, moguće je izračunati grešku u omjeru stvarnog i nazivnog razmaka između kotača E_b. Kako je razmak među kotačima b direktno proporcionalan stvarnoj rotaciji, koristeći jednadžbu (12) može se zapisati:

$$\frac{b_a}{90°} = \frac{b_n}{90° - \alpha} \tag{52}$$

Sređivanjem izraza (52) se može izračunati E_b:

$$E_b = \frac{b_a}{b_n} = \frac{90°}{90° - \alpha} \tag{53}$$

10.4. Kompenzacija sustavnih grešaka mobilnog robota

Nakon što su izračunate kvantitativne vrijednosti omjera E_b i E_d, jednostavno je programski kompenzirati sustavne pogreške. Korekcija udaljenosti između kotača je jednostavna, udaljenost b je promijenjena u diferencijalnom kontroleru mobilnog robota primjenom izraza (53).

Kompenzacija promjera kotača nije toliko jednostavna iz razloga što primjenom kompenzacijskog faktora treba obratiti pozornost da srednji promjer kotača D_a ostane konstantan jer bi se u protivnom i on trebao kalibrirati. Uvjet da srednji promjer ostane isti se može promatrati kao ograničenje

$$D_a = \frac{D_R + D_L}{2} \tag{54}$$

Promatranjem izraza (48) i (54) kao sustava dviju jednadžbi s dvije nepoznanice D_R i D_L, mogu se izračunati promjeri lijevog i desnog kotača:

$$D_L = \frac{2}{E_d + 1} D_a \tag{55}$$

$$D_R = \frac{2}{(1/E_d) + 1} D_a \tag{56}$$

Novo izračunati promjeri kotača se mogu koristiti kako bi se izrazio novi korekcijski faktor za lijevi i desni kotač:

$$c_L = \frac{2}{E_d + 1} \tag{57}$$

$$c_R = \frac{2}{(1/E_d) + 1} \tag{58}$$

Korekcijski faktori se mogu implementirati u *dead-reckoning* algoritam za estimaciju pozicije robota tako da se prepiše jednadžba:

$$\Delta U_{L/R,i} = c_{L/R} c_m N_{L/R,i} \quad \textbf{(59)}$$

Ovom implementacijom su korigirane obje dominantne sustavne greške mobilnog robota.

10.5. Izračun korekcijskih faktora za odometriju

Za određivanje korekcijskih faktora, robot je pušten da prati kvadratnu trajektoriju u smjeru kazaljke na satu i smjeru suprotnom od kazaljke na satu. Primjenom *Polaris Vicra* uređaja za praćenje i *PolarisVicraGUI* programa su snimljene početna i krajnja pozicija robota te je izračunata apsolutna greška na kraju trajektorije. U tablici 14 su prikazane greške za kretanje robota u smjeru kazaljke na satu zajedno sa srednjom vrijednosti i standardnom devijacijom.

Br. snimanja	Greška u smjeru osi X (*dx*)	Greška u smjeru osi Y (*dy*)	Greška u smjeru osi X i Y (*dxy*)
1.	111.5 mm	198.88 mm	228.0033 mm
2	106.08 mm	194.15 mm	221.2401 mm
3.	108.46 mm	206.03 mm	232.8346 mm
4.	111.71 mm	201.72 mm	230.5864
5.	110.81 mm	183.53 mm	214.3878 mm
Srednja vrijednost	109.712 mm	196.862 mm	225.4104 mm
Standardna devijacija	2.4059 mm	8.6143 mm	7.5419 mm

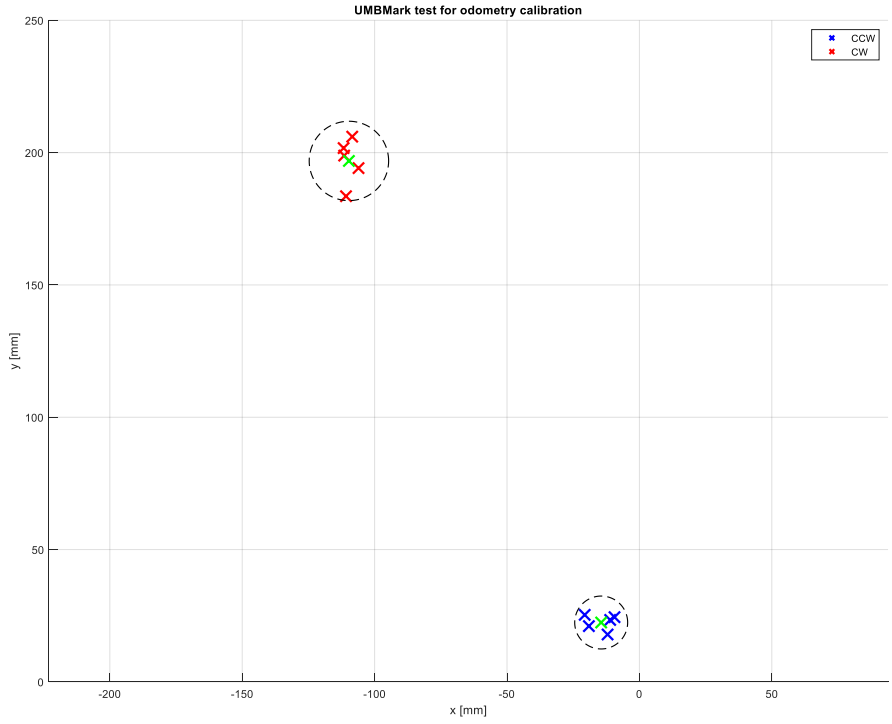
Tablica 14. Prikaz grešaka, srednjih vrijednosti i standardne devijacije grešaka nastalih praćenjem kvadratne trajektorije u smjeru kazaljke na satu

U tablici 15. su prikazane greške robota prilikom praćenja referentne trajektorije u smjeru suprotnom od kazaljke na satu, zajedno s prosječnom vrijednosti i standardnom devijacijom.

Br. snimanja	Greška u smjeru osi X (*dx*)	Greška u smjeru osi Y (*dy*)	Greška u smjeru osi X i Y (*dxy*)
1.	9.4 mm	24.46 mm	26.204 mm
2	12.02 mm	17.91 mm	21.5696 mm
3.	11.04 mm	23.41 mm	25.8826 mm
4.	20.66 mm	25.33 mm	32.6871 mm
5.	19.08 mm	21.07 mm	28.4252 mm
Srednja vrijednost	14.44 mm	22.436 mm	26.9537 mm
Standardna devijacija	5.0753 mm	2.9904 mm	4.0538 mm

Tablica 15. Prikaz grešaka, srednjih vrijednosti i standardne devijacije grešaka nastalih praćenjem kvadratne trajektorije u smjeru suprotnom od kazaljke na satu

Slika 63. grafički prikazuje podatke iz tablica 14 i 15, s ucrtanom srednjom vrijednosti za oba slučaja.



Slika 63. Grafički prikaz podataka iz tablica 13 i 14

Koordinate težišta klastera su ujedno i srednje vrijednosti, stoga su njihova vrijednosti:

$$\begin{aligned} x_{c.g.,cw} &= -109.712\ mm \\ y_{c.g.,cw} &= 22.436\ mm \\ x_{c.g.,ccw} &= -14.44\ mm \\ x_{c.g.,ccw} &= -22.436\ mm \end{aligned} \quad \textbf{(60)}$$

Primjenom izraza (45) i (50) te vrijednosti iz izraza (59) dobiveni su kutovi α i β:

$$\alpha = 0.8892° \quad \textbf{(60)}$$

$$\beta = 0.6823° \quad \textbf{(61)}$$

Korekcijski faktor za udaljenost među kotačima E_b, izračunat je primjenom (53) i (61):

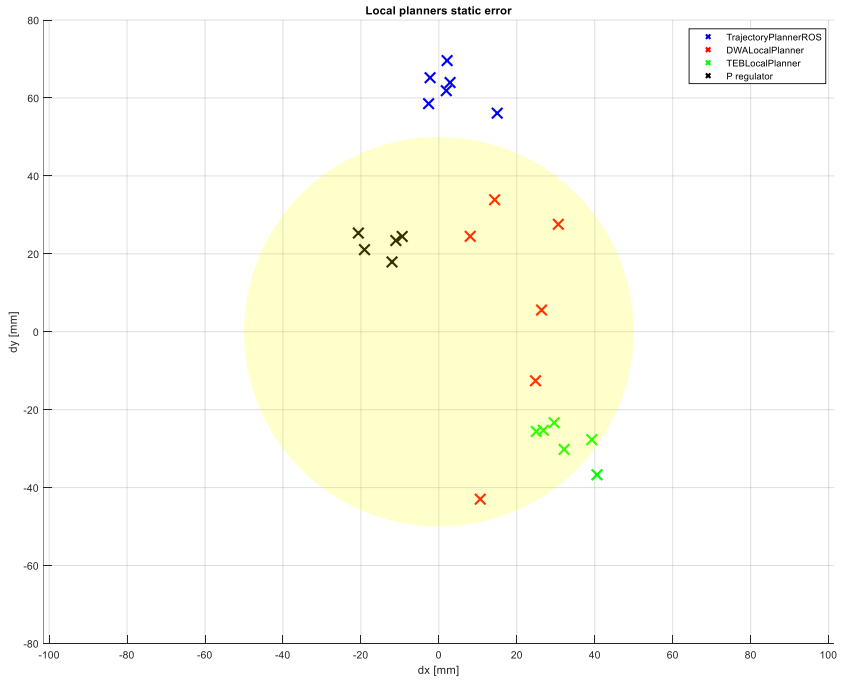
$$E_b = 1.01 \quad \textbf{(62)}$$

dok je faktor za korekciju promjera kotača izračunat primjenom (48) i (60):

$$E_d = 1.001 \quad \textbf{(63)}$$

Implementacija i provjera dobivenih faktora nije moguća iz razloga što diferencijalni kontroler, u kojega je potrebno unijeti parametre, ne dopušta pristup *firmware*-u od strane ROS-a.

Za provedbu *UMBMark* testa, korišteni su P regulatori po poziciji i kutu zakreta, stoga je njegova točnost uspoređena s točnosti ranije opisanih lokalnih planera. Kako su lokalni planeri izvodili kvadratnu trajektoriju u smjeru suprotnom od kazaljke na satu, iskorišteni su podaci iz tablice 15 za usporedbu s ostalim planerima.



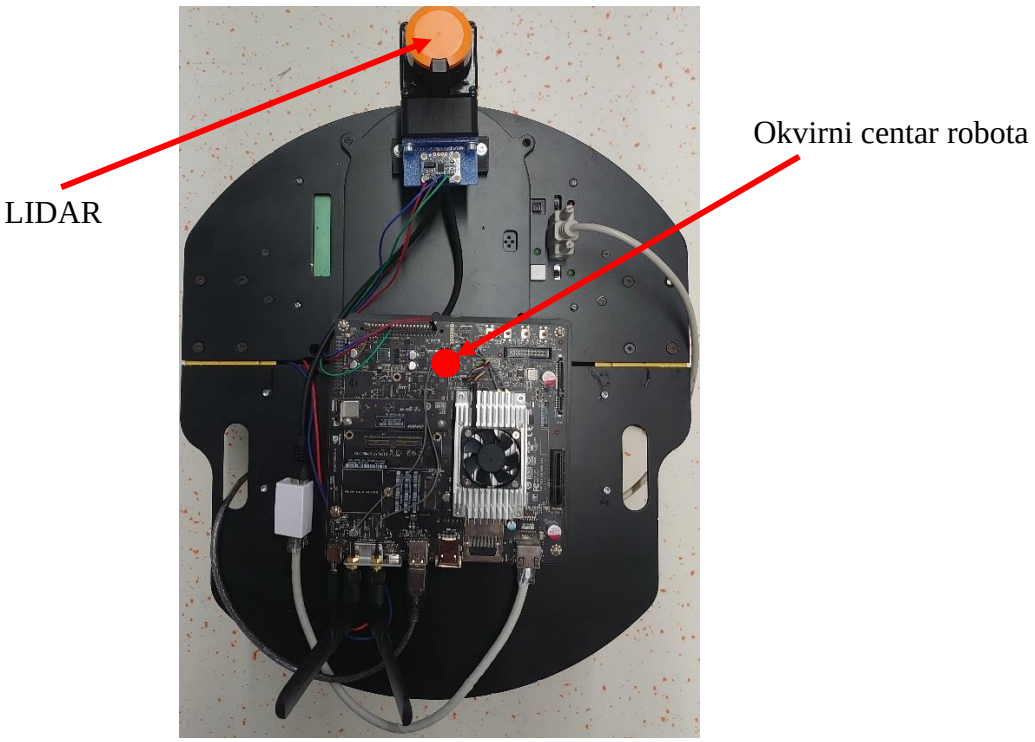
Slika 64. Usporedba tri lokalna planera iz ROS-a s jednostavnim P regulatorom pozicije

Sa slike 64. je vidljivo kako P regulator pozicije pokazuje najbolje karakteristike točnosti i ponovljivosti. Mana P regulatora pozicije je u tome što ne omogućuje adekvatno praćenje trajektorije s mogućnosti izbjegavanja prepreke.

Idealna strategija precizne navigacije je da se to određene točke odlazi s *TEBLocalPlanner* lokalnim planerom, te po završetku se koristi P regulator pozicije za precizno pozicioniranje.

10.6. Određivanje pozicije lidar senzora

Lidar je uz enkodere najvažniji senzor na mobilnom robotu za procese mapiranja i lokalizacije, dok se važnost lidar-a za proces navigacije očituje u detekciji dinamičkih prepreka koje se nisu nalazile u mapi za vrijeme procesa mapiranja i što točnije određivanje apsolutne pozicije robota. Kako bi navedeni procesi bili što bolji, potrebno je pronaći točnu poziciju lidar senzora u odnosu na centar rotacije mobilnog robota koji je ujedno i centar ishodišta koordinatnog sustava mobilnog robota. Slika 65. prikazuje poziciju lidar senzora na mobilnom robotu.



Slika 65. Pozicija lidar senzora na mobilnom robotu

Točan centar robota nije moguće odrediti jer roboti projektirani u CAD alatima nisu jednaki onim stvarnim proizvedenim robotima. Glavni uzroci tome su već navedeni u prethodnim poglavljima, a mogu biti: nejednaki promjeri kotača, netočna udaljenost između dva kotača, nejednaka raspodjela opterećenja na robotu, netočnost 3D ispisanog nosača senzora itd.

Kako bi se pronašla točna pozicija robota u odnosu na centar rotacije robota, iskorištena je ista oprema objašnjena u 9. poglavlju. Postav koji se koristio za pronalaženje točne lokacije lidar senzora je prikazan na slici 46. u poglavlju 9.

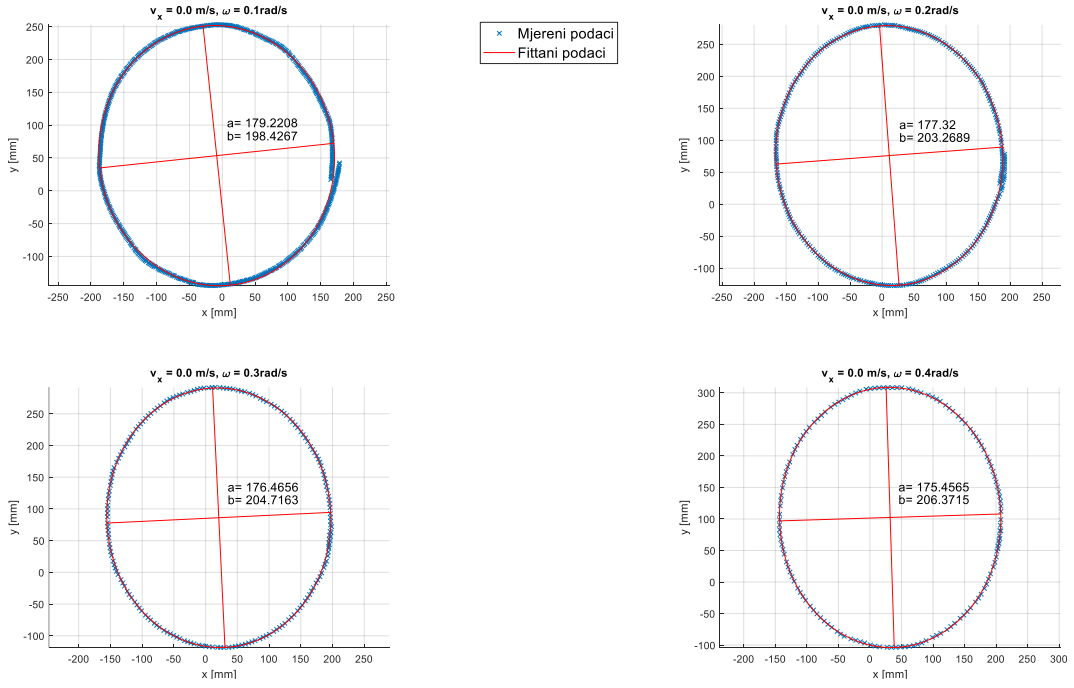
Postupak pronalaska pozicije lidar senzora:

- postaviti translacijsku brzinu robota $v_x = 0$ i rotacijsku brzinu $\omega \neq 0$
- pokrenuti snimanje podataka primjenom uređaja *Polaris Vicra* i programa *PolarisVicraGUI* te spremiti snimljene podatke u *.csv* format
- učitati spremljene podatke u *MatLab*, projicirati 3D kružnicu u 2D ravninu te pronaći elipsu koja najbolje odgovara dobivenim podacima

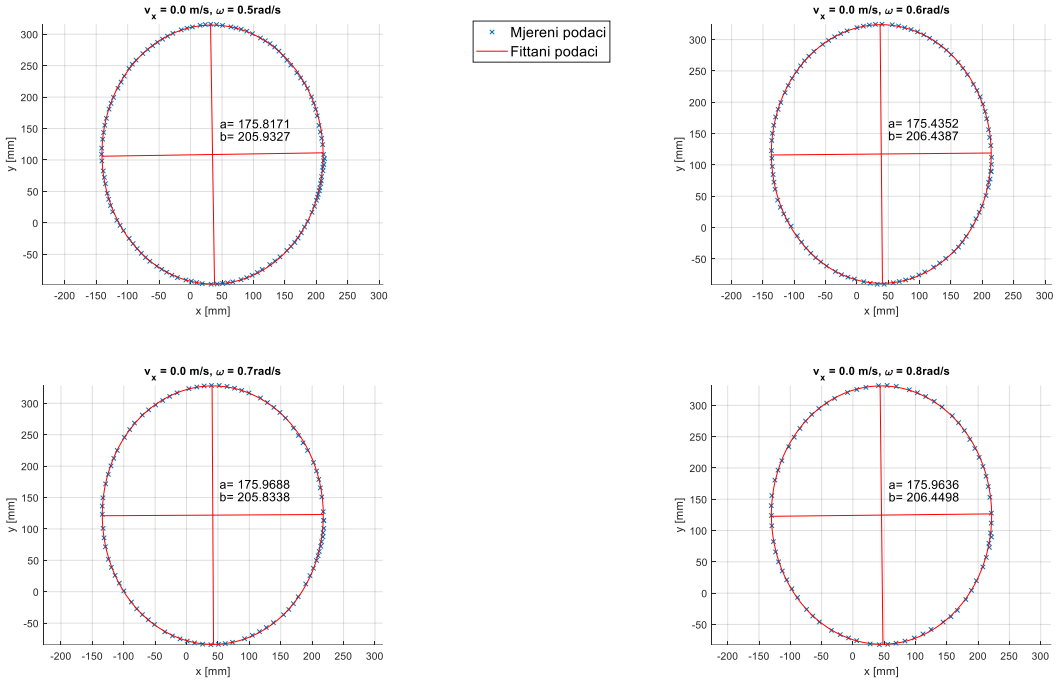
Kako bi pronalazak pozicije lidar senzora bio što točniji, robotu su dane rotacijske brzine od 0.1 $\frac{rad}{s}$ do 1.0 $\frac{rad}{s}$ i 1.5 $\frac{rad}{s}$ kako bi se izbjeglo klizanje robota u jednu stranu pri manjim brzinama.

10.1.1. Rezultati mjerenja

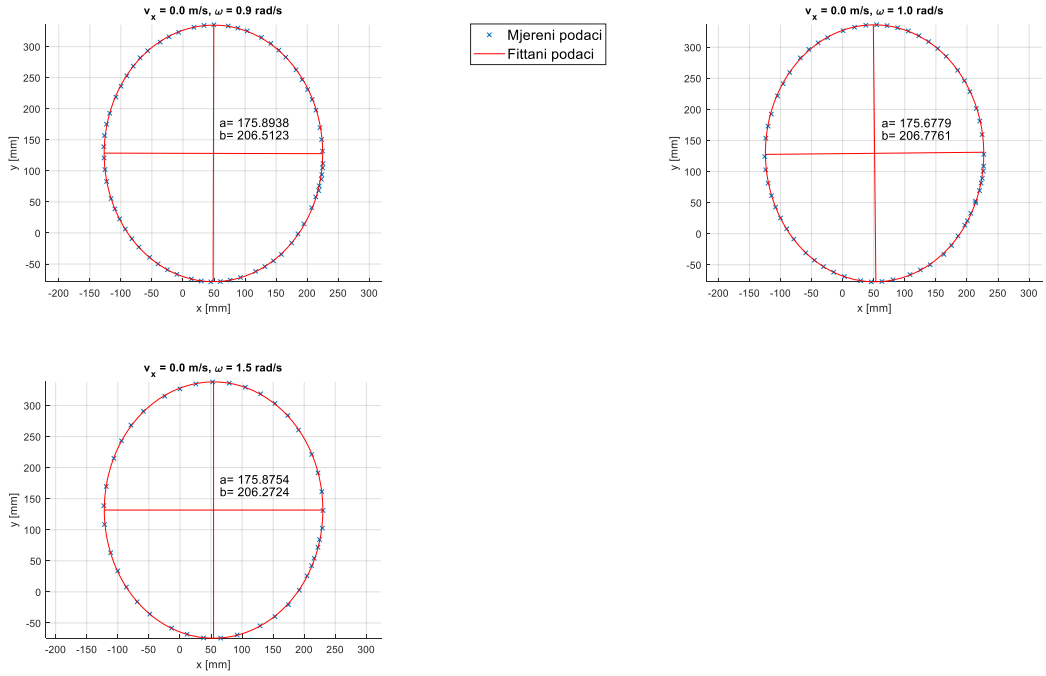
Na slikama 66., 67. i 68. su prikazani rezultati mjerenja dobiveni s *Polaris Vicra* kamere za koje je naknadno nađena jednadžba elipse koja ih najbolje opisu. Na slikama se također nalaze parametri elipse *a* i *b*. Parametar *b* projicirane elipse odgovara originalnoj kružnici u 3D prostoru.



Slika 66. Prikaz rezultata za kutne brzine robota od 0.1 do 0.4 rad/s

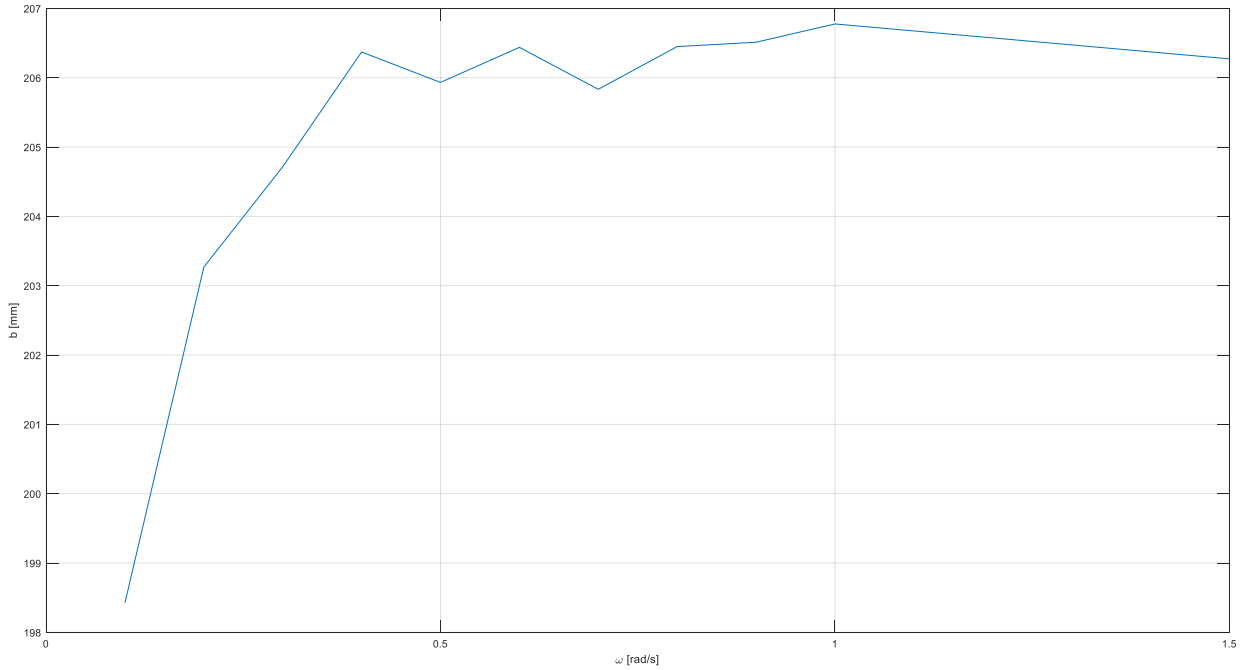


Slika 67. Prikaz rezultata za kutne brzine robota od 0.5 do 0.8 rad/s



Slika 68. Prikaz rezultata za brzine robot 0.9, 1.0 i 1.5 rad/s

S gore priloženih grafova je vidljivo kako je parametar elipse *b* koji prezentira polumjer rotacije mobilnog robota promjenjiv s obzirom na rotacijsku brzinu mobilnog robota. Na slici 69 je prikazano kretanje parametra *b* s obzirom na rotacijsku brzinu.

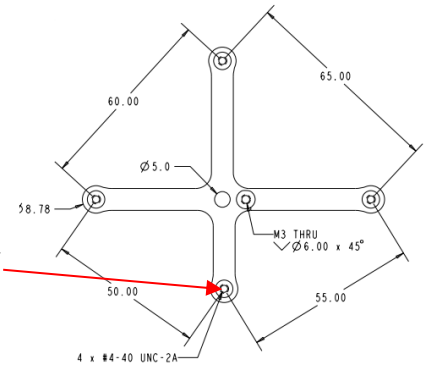


Slika 69. Promjena parametra *b* elipse s obzirom na rotacijsku brzinu robota

Sa slike 69. je vidljivo kako parametra *b* konvergira vrijednosti od 206 mm. Vrijednost koja je odabrana kao referentna za danji izračun pozicije lidar senzora je *b* = 206.27 mm, dobivena za rotacijsku brzinu od 1.5 $\frac{rad}{s}$.

Kako referentni koordinatni sustav markera nije smješten u središte markera nego na jednu od kuglica, potrebno je od dobivenog parametra *b* oduzeti njegovu udaljenost od centra kako bi se dobila točna pozicija markera čije središte je smješteno u središte lidar senzora. Slika 70. prikazuje izgled markera s označenom referentnom kuglicom.

Referentni koordinatni sustav



Slika 70. Dimenzije markera za *Polaris Vicra* kameru

Primjenom jednostavnih geometrijskih izraza za sličnost trokuta, izračunato je da je udaljenost referentnog koordinatnog sustava od središta markera $l_{ref} = 41.027\ mm$.

Točna udaljenost lidar senzora od središta rotacije robota je:

$$x_{LIDAR} = b - l_{ref} = 165.243\ mm \quad \textbf{(60)}$$

Primjena ovog podatka u ROS-u se očituje kroz čvor *tf* koji služi za transformaciju koordinatnih sustava. Dio iz *.launch* datoteke za transformaciju lidar senzora je dan niže, ali s upisanim metrima za transformaciju.

```
<!-- TRANSFORMACIJA TF-A LIDAR-A -->
  <node pkg="tf" type="static_transform_publisher"
name="base_link_laser_broadcaster" args="0.165243 0 0 0 0 0 base_footprint
laser 100" />
```

11. ZAKLJUČAK

Osnovna ideja ovog rada je bila pokazati kako u sklopu robotskog operativnog sustava koristiti iznimno kompleksne algoritme za procese mapiranja, navigacije i lokalizacije. ROS osigurava da se inženjeri bave isključivo svojim područjem, a uz pomoć ROS-a implementiraju algoritme koje nije jednostavno brzo i efikasno razviti na razinu na kojoj se već nalaze. Po završetku mapiranja prostora i lokalizacije mobilnog robota u istom, provedena je navigacija mobilnog robota po prostoru. Kao što je rečeno, navigacija iziskuje globalno planiranje trajektorija kako bi definirala put od početka do cilja, međutim kako bi se trajektorija uspješno pratila, potrebno je koristiti neki od lokalnih planera. Kako bi se adekvatno odabrao odgovarajući lokalni planer, provedena su ispitivanja točnosti pozicioniranja tri lokalna planera koji se nalaze u sklopu ROS-a (*TrajectoryPlannerROS, DWALocalPlanner* i *TEBLocalPlanner)*. Sva tri planera su testirana i međusobno uspoređena, a na kraju je napravljen P regulator pozicije kako bi se bolje mogli usporediti lokalni planeri.

Zbog potreba za što točnijim određivanjem pozicije mobilnog robota, detaljno je opisan i proveden proces kalibracije odometrije mobilnog robota, međutim zbog ne mogućnosti pristupa niskoj razini kontrolera, nije bilo moguće testirati dobivene korekcijske faktore. Osim odometrije, za procese mapiranja, lokalizacije i navigacije iznimno je važno poznavanje točne pozicije lidar senzora u odnosu na koordinatni sustav mobilnog robota. Stoga je proveden proces kalibracije te je dobivena udaljenost implementirana u rad robota.

LITERATURA

[1] Mihel M, Bajd T, Ude A, Lenarčič J, Stanovnik A, Munih M, Rejc J, Šlajpah S. Robotics. Secnod Edition. Ljubljana: Springer; 2019.

[2] https://www.gideonbros.ai/, 08.05.2021.

[3] https://www.kuka.com/, 08.05.2021.

[4] https://100.fsb.hr/hr/119/FSB+robot+RONNA+u+neurokirurgiji+u+Zagrebu, 08.05.2021.

[5] https://www.irobot.hr/ , 08.05.2021.

[6] https://www.edrmagazine.eu/eca-group-announces-a-new-sale-of-its-autonomous-underwater-vehicle-auv-a18, 08.05.2021.

[7] https://www.dji.com/hr, 08.05.2021.

[8] https://www.generationrobots.com/en/402395-robot-mobile-pioneer-3-dx.html, 24.05.2021.

[9] https://hokuyo-usa.com/, 24.05.2021.

[10] https://www.adafruit.com/, 24.05.2021.

[11] Šebalj, K. Upravljanje mobilnim robotom primjenom robotskog operativnog sustava (ROS)

[12] https://www.nvidia.com/en-us/, 24.05.2021.

[13] https://www.ros.org/, 24.06.2021.

[14] http://wiki.ros.org/ds4_driver, 24.06.2021.

[15] http://wiki.ros.org/urg_node, 24.06.2021.

[16] http://wiki.ros.org/gmapping, 25.06.2021.

[17] http://wiki.ros.org/amcl, 26.06.2021.

[18] http://wiki.ros.org/global_planner, 28.06.2021.

[19] http://wiki.ros.org/dwa_local_planner, 28.06.2021.

[20] http://wiki.ros.org/costmap_2d, 28.06.2021.

[21] Borenstein, J., Feng, L.: UMBmark – A Method for Measuring, Comparing and Correcting Dead-reckoning Errors in Mobile Robots.

[22] http://wiki.ros.org/eband_local_planner, 03.07.2021.

[23] https://www.ndigital.com/products/polaris-vicra/, 03.07.2021.

[24] M. Alajlan, A. Kouba (2016) Writing Global Path Planners Plugins in ROS: A Tutorial. In: A. Koubaa (eds) Robot Operating System (ROS). Studies in Computational Intelligence, vol 625. Springer, Cham

[25] http://wiki.ros.org/move_base?distro=kinetic, , 05.07.2021.

PRILOZI

I. Programski kodovi za čvorove i *.launch* datotke – https://github.com/BCaran/Pionner-P2DX-ROS.git

II. Programski kod za *Polaris Vicra GUI* - https://github.com/BCaran/Polaris-Vicra-GUI.git