Planiranje putanje mobilnog robota unutarnjim GPS sustavom

Hartl, Krešimir

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Krešimir Hartl

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Prof. dr. sc. Mladen Crneković, dipl. ing.

Student:

Krešimir Hartl

Zagreb, 2024.

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

Zahvaljujem se svojem mentoru profesoru dr. sc. Mladenu Crnekoviću na zadanom zadatku i na stručnom usmjeravanjem tijekom rješavanja zadatka. Također se zahvaljujem asistentu Branimiru Ćaranau na dostupnosti i omogućavanju pristupa svoj potrebnoj opremi. Na kraju, zahvaljujem se svojoj obitelji, djevojci i prijateljima koji su mi bili podrška tijekom cijeloga studija.

Krešimir Hartl

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Krešimir Hartl** JMBAG: **0035239762**

Naslov rada na hrvatskom jeziku: **Planiranje putanje mobilnog robota unutarnjim GPS sustavom**

Naslov rada na engleskom jeziku: **Path planning of a mobile robot with indoor GPS system**

Opis zadatka:

Lociranje mobilnog robota u zatvorenom prostoru jedan je od najdugovječnijih problema robotike. Do sada se problem uglavnom rješavao SLAM postupkom koji je računalno vrlo zahtjevan. Dolaskom na tržište Marvelmind indoor GPS sustava rješenje ima novu dimenziju.
Potrebno je primijeniti navedeni GPS sustav na lociranje mobilnog robota u zadanoj okolini, te navođenjem robota na cilj.

U radu je potrebno:

- Definirati radni prostor mobilnog robota i pozicije GPS stanica.
- Očitavati položaj robota.
- Grafički prikazati položaj i gibanje robota u zadanoj okolini.
- Metodom potencijalnih polja navoditi robota do cilja.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
24. 4. 2024.	**2. rok (izvanredni):** 11. 7. 2024. **3. rok:** 19. i 20. 9. 2024.	**2. rok (izvanredni):** 15. 7. 2024. **3. rok:** 23. 9. – 27. 9. 2024.

Zadatak zadao: Predsjednik Povjerenstva:

prof. dr. sc. Mladen Crneković

izv. prof. dr. sc. Petar Ćurković

SADRŽAJ

POPIS SLIKA

SAŽETAK

Problem lociranja mobilnog robota jedan je od osnovnih problema robotike. Porastom korištenja mobilnih robota u zatvorenim prostorima poput tvornica, skladišta i domova, problem dobiva novu dimenziju. U ovom radu testirat će se mogućnost korištenja *Marvelmind Indoor GPS* sustava za lociranje u zatvorenom prostoru. Lokacija je potrebna mobilnim robotima kako bi se uspješno navodili do cilja. Testiranje sustava *Marvelmind* provodit će se navođenjem mobilnog robota metodom potencijalnih polja. Testiranja će biti obavljena u prostorima CRTA-e (*Regional Center of Excellence for Robotic Technologies*) korištenjem mobilnog robota ASTRO (*Autonomus System for Teaching RObotics*). Provjera uspješnosti rada sustava provodit će se korištenjem sustava OptiTrack. OptiTrack je provjereni sustav za lokalizaciju visoke točnosti. Kako bi se svi sustavi povezali koristit će se ROS2 sustav. Korištenjem ROS2 sustava ostvarit će se fleksibilnost i mogućnost jednostavne implementaciju sustava u kasnijim projektima. Cilj ovog rada je utvrditi može li se sustav Marvelmind koristiti za sigurno navođenje mobilnog robota u zatvorenom prostoru.

Ključne riječi: Marvelmind, OptiTrack, GPS, Lociranje, Navođenje, Metoda potencijalnih polja, ASTRO, Mobilni roboti, ROS2

SUMMARY

The problem of localizing mobile robots is one of the fundamental challenges in robotics. With the increasing utilization of mobile robots in enclosed environments such as factories, warehouses, and homes, this issue acquires a new dimension. This dissertation will evaluate the feasibility of employing the Marvelmind Indoor GPS system for indoor localization. Accurate localization is essential for mobile robots to navigate successfully to their destinations. The testing of the Marvelmind system will be conducted by guiding a mobile robot using the potential field method. The experiments will take place at CRTA (Regional Center of Excellence for Robotic Technologies) utilizing the ASTRO (Autonomous System for Teaching Robotics) mobile robot. The effectiveness of the system will be verified using the OptiTrack system, a proven high-accuracy localization solution. The ROS2 framework will be employed to integrate all systems. Utilizing ROS2 will provide flexibility and facilitate straightforward implementation in future projects. The objective of this study is to determine whether the Marvelmind system can be utilized for the safe navigation of mobile robots in indoor environments.

Keywords**:** Marvelmind, OptiTrack, GPS, Localization, Navigation, Potential Field Method, ASTRO, Mobile Robots, ROS2

1. UVOD

Mobilni roboti su automatizirani, programabilni višenamjenski mehatronički sustavi s mogućnošću samostalnog gibanja u prostoru s ciljem izvođenja zadanih radnji.[1] Kako bi se mobilni roboti uspješno gibali u prostoru potrebno je poznavati njihov položaj u radnom prostoru. Praćenje položaja mobilnog robota može se izvršavati praćenjem unutarnjih veličina stanja ili praćenjem vanjskih veličina stanja robota. Primjer praćenja unutarnjih veličina stanja mobilnog robota je inkrementalni enkoder. Praćenjem signala inkrementalnog enkodera može se aproksimirati položaj robota u prostoru. Mana takvog sustava se očituje pri gibanju mobilnog robota s relativnim klizanjem u odnosu na podlogu. Pri takvom gibanju, promjena unutarnjih stanja više nije u korelaciji s pomakom u radnom prostoru. Iz tog razloga često se koriste sustavi praćenja vanjskih veličina stanja poput GPS-a, vizualnih senzora, infracrvenih senzora ili ultrazvučnih senzora. Sustav poput GPS-a daje apsolutnu poziciju u radnom prostoru čime je izvršena zadaća praćenja položaja mobilnog robota. *Global Positioning System* (GPS) je američki sustav koji pruža korisnicima usluge pozicioniranja, navigacije i mjerenja vremena.[2] Mana GPS-a je slabo prodiranje u zatvorene prostore. Postoje rješenja apsolutnog pozicioniranja mobilnog robota i u zatvorenim prostorima, jedno takvo rješenje je *Marvelmind Indoor GPS*. Sustav *Marvelmind* koristi ultrazvučne signale za interpolaciju položaja mobilnog robota. Drugi pristup dobivanja apsolutnog položaja mobilnog robota u prostoru bi bila SLAM metoda. *Simultaneous localization and mapping* (SLAM) je metoda koja se koristi u robotici za istovremeno određivanje pozicije robota u nepoznatom okruženju i izradu karte istog okruženja.[1] SLAM metoda ima prednost što se mobilni robot može pozicionirati u nepoznatom radnom prostoru, ali kako bi se SLAM metodom dobio visoki stupanj sigurnosti u poziciju te visoka rezolucija pozicije potrebno je provoditi računalno zahtjevne operacije. Nakon što se uspješno prati pozicija mobilnog robota, ista se može koristiti za navigaciju mobilnog robota do njegovog cilja. Navigacija se odnosi na sposobnost robota da se kreće kroz okolinu i dostiže željene ciljeve.[1] Postoje različite metode navigacije, jedna od tih metoda je metoda potencijalnih polja. Metodom potencijalnih polja dobiva se putanja od mobilnog robota do cilja. Nakon što je izračunata putanja, kontrolom mobilnog robota prati se ista do zadanog cilja.

2. ROS2

Robot Operating System 2 (ROS2) predstavlja napredni skup računalnih biblioteka i alata koji su osmišljeni za razvoj robotskih aplikacija.[3] Iako se naziva „operativni sustav“, ROS2 nije operativni sustav u klasičnom smislu, već *middleware*. To znači da služi kao posrednik koji omogućava različitim sustavima i aplikacijama međusobnu komunikaciju i razmjenu podataka. Osnovni korišteni protokol je *Data Distribution Service* (DDS) koji omogućuje skalabilnost, pouzdanost i dobre performanse. ROS2 pruža *real-time* performanse što je ključno za robotske sustave.

Glavni elementi ROS2-a su:

- Paketi (engl. *Packages*)
- Čvorovi (engl. *Nodes*)
- Teme (engl. *Topics*)
- Poruke (engl. *Messages*)
- Servisi (engl. *Services*).

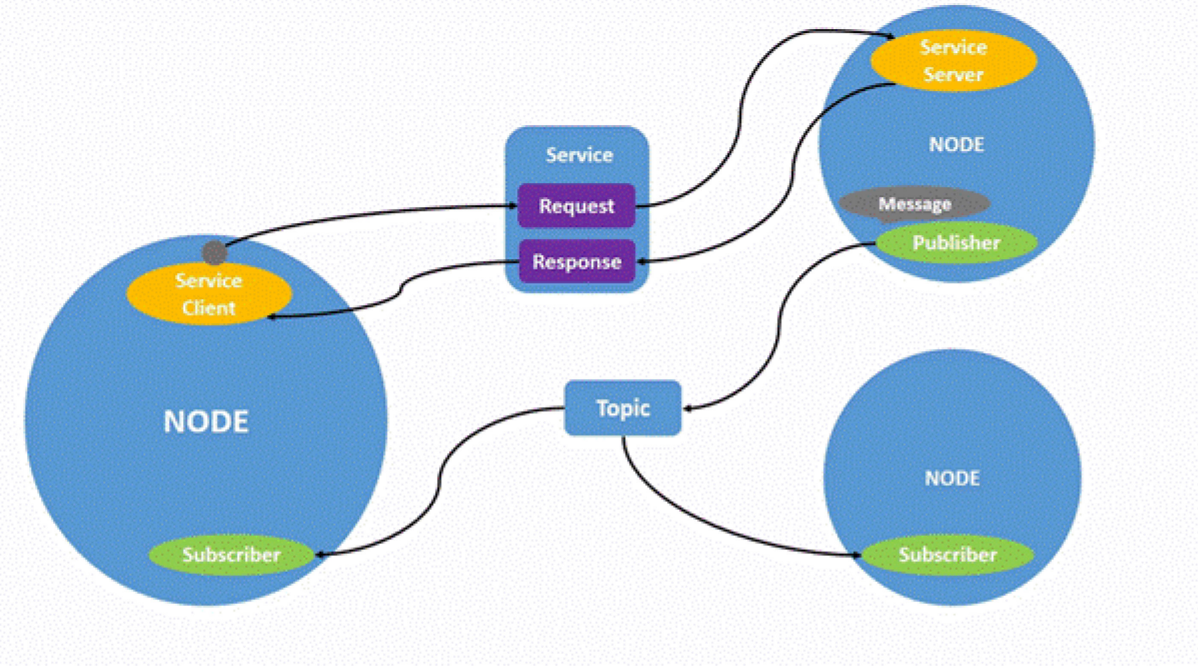
Paketi su osnovna jedinica ROS2-a koja objedinjuje čvorove, biblioteke, baze podataka i ostale potrebne datoteke. Paketi omogućuju organizaciju i ponovno korištenje koda.

Čvorovi su izvršne jedinice u ROS2. Čvorovi su programi napisani u Python ili C++ programskim jezicima korištenjem biblioteka *rospy* ili *roscpp.* Osnovna zadaća čvorova je izvršavanje računalnih operacija. Glavna karakteristika čvorova je mogućnost objavljivanja (*publish*) poruka na teme i mogućnost pretplaćivanja (*subscribe*) na teme i primanja poruke koje su objavljenje na iste. Parcijalizacijom zadataka preko različitih čvorova, jednostavno se mogu dodavati nove funkcionalnosti ili utvrđivati i otklanjati greške u programu.

Teme su kanali koji služe za omogućavanje komunikacije između čvorova. Čvorovi mogu objavljivati ili slušati poruke koje se objavljuju na određenu temu. Putem čvorova omogućena je asinkrona komunikacija.

Poruke su definirani oblici podatkovnih struktura. Definirane su datotekom unutar koje je zadan oblik podatka i naziv varijabli. Poruke mogu biti jednostavnog oblika poput jednog float64 broja ili kompleksnog oblika gdje jedna poruka u sebi sadrži drugu poruku koja unutra sebe sadrži više oblika podataka. Podržani su svi osnovni oblici podataka C++-a.

Servisi služe za interakciju s čvorovima na principu zahtjev-odgovor. Na zadani zahtjev jedan čvor pruža odgovor. Korisni su za izvršavanje jednokratnih radnji kod kojih je potrebna povratna informacija. Omogućuju sinkronu komunikaciju.



Slika 1. ROS2 struktura[3]

3. MOBILNI ROBOT ASTRO

Mobilni robot ASTRO (*Autonomus System for Teaching RObotics*) je mobilni robotski sustav razvijen u CRTA-i (*Regional Center of Excellence for Robotic Technologies*). ASTRO je razvijen u svrhu edukativnog i istraživačkog razvoja. Pruža dobru platformu za implementiranje i testiranje različitih sustava na fleksibilan način.

3.1. Arhitektura robota

Za upravljanje robotom, odnosno slanje upravljačkih signala elektromotorima, koristi se mikroračunalo *Teensy 4.0*. Glavne karakteristika *Teensya* su brzi ARM mikroprocesor i mogućnost programiranja korištenjem Arduino IDE-a ili C/C++ programskih jezika. Komunikacija između podređenog mikroračunala Teensy 4.0 i nadređenog upravljačkog računala ostvarena je putem UART *(Universal Asynchronous Receiver Transmitter*) komunikacije. Cijela arhitektura ASTRO-a može se vidjeti na Slika 3.

Funkciju nadređenog upravljačkog računala ima *Nvidia Jetson Nano*. *Jetson Nano* je malo računalo dizajnirano za učenje, razvoj alata i proizvoda te drugih primjena u granama umjetne inteligencije i robotike. *Jetson Nano* u ASTRO-u ima instaliran *Linux Ubuntu 18.04* operativni sustav. Za pokretanje potrebnih ROS2 čvorova koji omogućuju serijsku komunikaciju između komponenti koristi se *Docker.* Logika nadređenog sustava je implementirana u ROS2 prostoru. S pomoću pokrenutih čvorova primaju se informacije senzora, *Joysticka* ili informacije koje šalju dodatno spojeni uređaji. Također pokrenuti čvorovi šalju informacije podređenom računalu u obliku linearne i kutne brzine. Prikaz ROS2 strukture vidi se na Slika 4.

3.2. Senzori

ASTRO koristi sljedeće senzore:

- *Intel RealSense D435 Depth* kamera
- *Slamtec RPLIDAR A1* LIDAR
- *Adafruit Precision NXP 9-DOF* IMU

Intel RealSense D435 je stereo kamera što omogućava generiranje dubinske slike okoline. Dodatno je opremljena IMU senzorom koji uključuje 3-osni akcelerometar i 3-osni žiroskop. Kamera je povezana sa sustavom putem *RealSense ROS2 Wrappera* koji omogućuje daljnju integraciju s ROS2 okolinom.

LIDAR senzor koristi laserske zrake za mjerenje udaljenosti do objekata koji reflektiraju emitirane zrake. U robotici, LIDAR-i su ključni za prikupljanje podataka o trodimenzionalnom okruženju robota te se često koriste u zadacima mapiranja i lokalizacije.

ASTRO je opremljen enkoderima koji mjere broj rotacija elektromotora. Enkoderi služe za estimaciju brzine i stanja robota, omogućujući kontinuirano praćenje kretanja i pozicije unutar radnog prostora.

3.3. Kinematski model robota

ASTRO ima dva fiksna pogonska kotača i 4 pasivne slobodno rotirajuće kugle. Pogonski kotači su upravljani zadavanjem zasebnih kutnih brzina. Takva struktura robota se potpunosti podudara s kinematskim modelom diferencijalnog pogona.

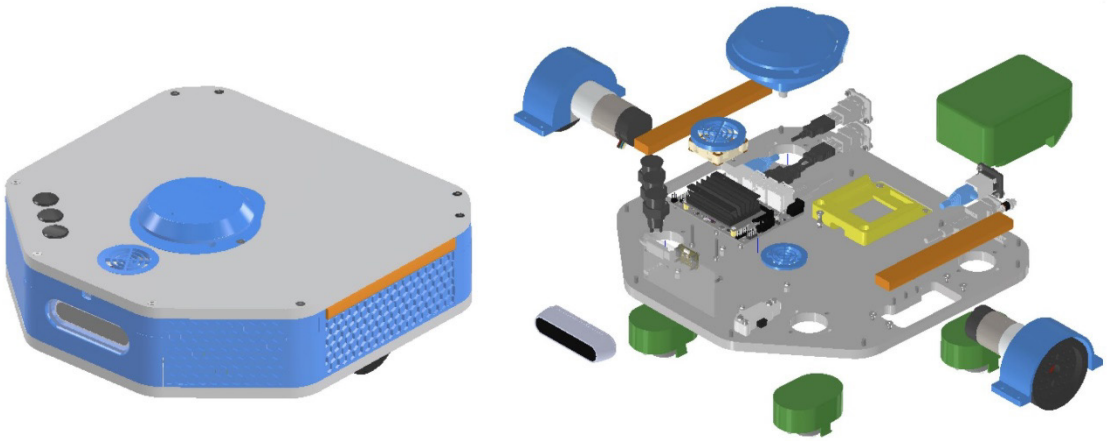
Pozicija i orijentacija robota su definirane sljedećim vektorom:

$$x = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \tag{3.1}$$

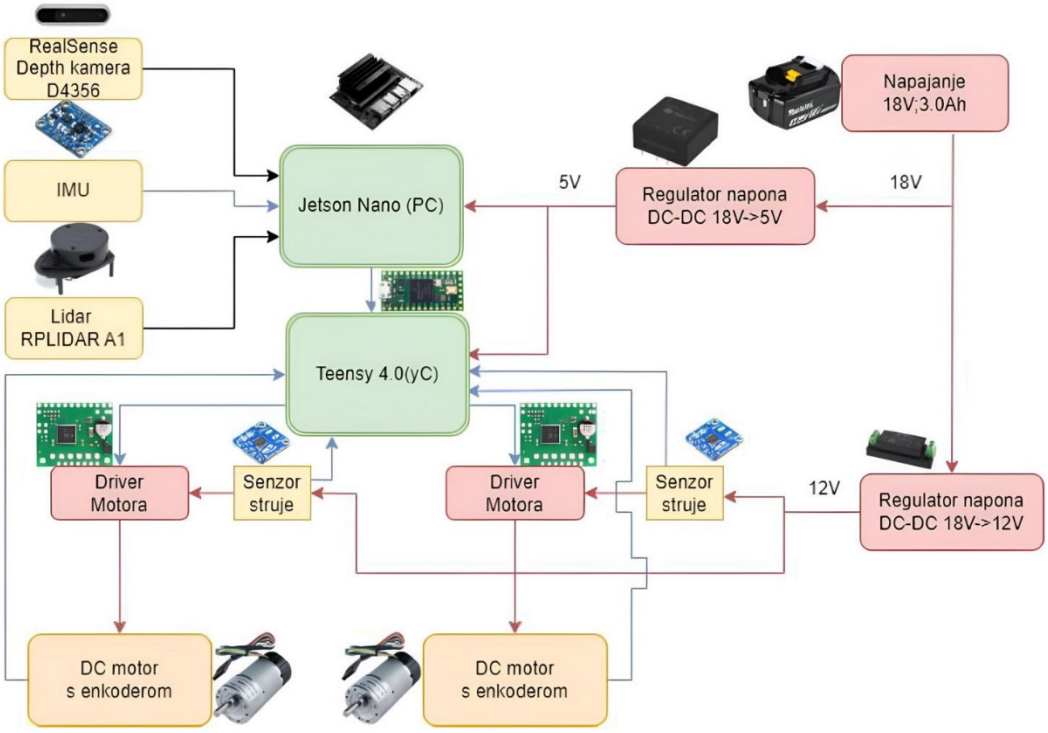
gdje su x i y koordinate u radnom prostoru, a kut θ rotaciju oko vertikalne osi.[4]

Brzina robota u globalnom koordinatnom sustavu može se zapisati kao:

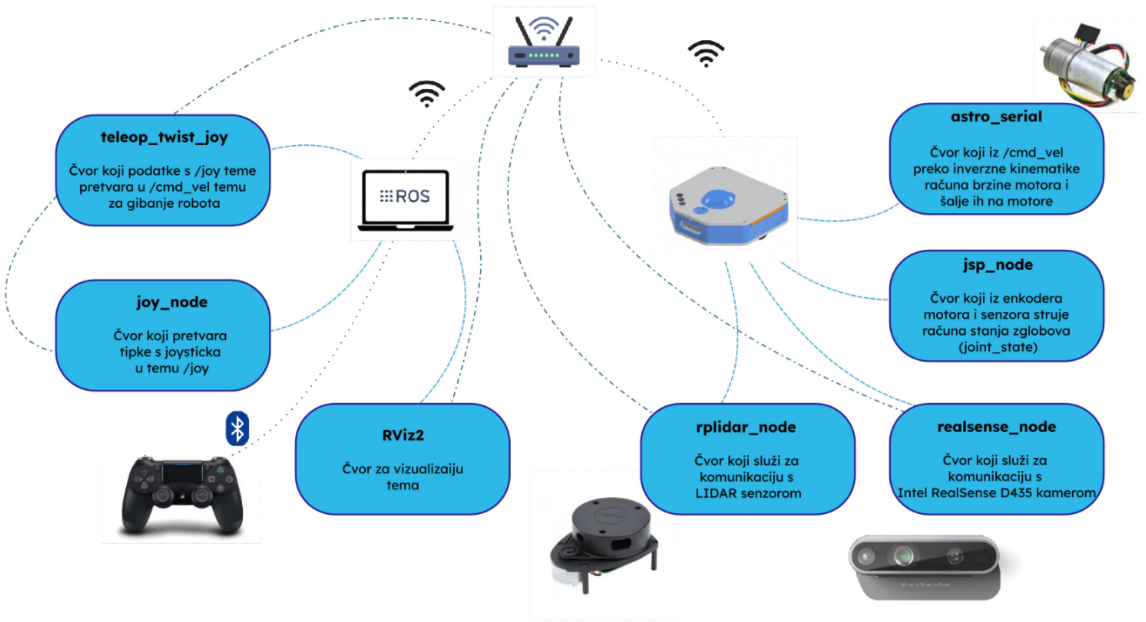
$$\dot{x} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}.[4] \tag{3.2}$$



Slika 2. ASTRO[9]



Slika 3. Arhitektura ASTRO-a[9]



Slika 4. ROS2 okruženje[10]

4. MARVELMIND INDOOR GPS

Marvelmind Indoor GPS je sustav koji koristi ultrazvučne odašiljače i prijemnike kako bi točno pozicionirao elemente u prostoru. Ovisno o karakteristikama prostora postavlja se mreža stacionarnih odašiljača pomoću kojih će se pratiti položaj elemenata s ugrađenim mobilnim odašiljačem. Preciznost sustav ovisi o uvjetima prostora. Točnost sustava u laboratorijskim uvjetima je $\pm 0{,}02\ m$ u prostoru površine do $1225\ m^2$.[5]

4.1. Instalacija sustava

Prvo je potrebno definirati koji će se tip sustava koristiti. U ovom zadatku potrebno je pratiti jedno vozilo u prostoru (x, y, θ). Marvelmind Indoor GPS sustav s 4 stacionarna „Super" odašiljača, dva mobilnim „Super" odašiljačem i „HW v5.1" modemom je optimalan za rješavanje problema. Sustav će biti postavljen u NIA (*Non-Inverse Architecture*) arhitekturi. Ovako postavljen sustav ima minimalan broj komponenti za praćenje pozicije i orijentacije vozila u radnom prostoru do površine 450 m^2[5]. Cijeli tok postavljanja sustava prikazan je na Slika 6.

4.1.1. Postavljanje odašiljača

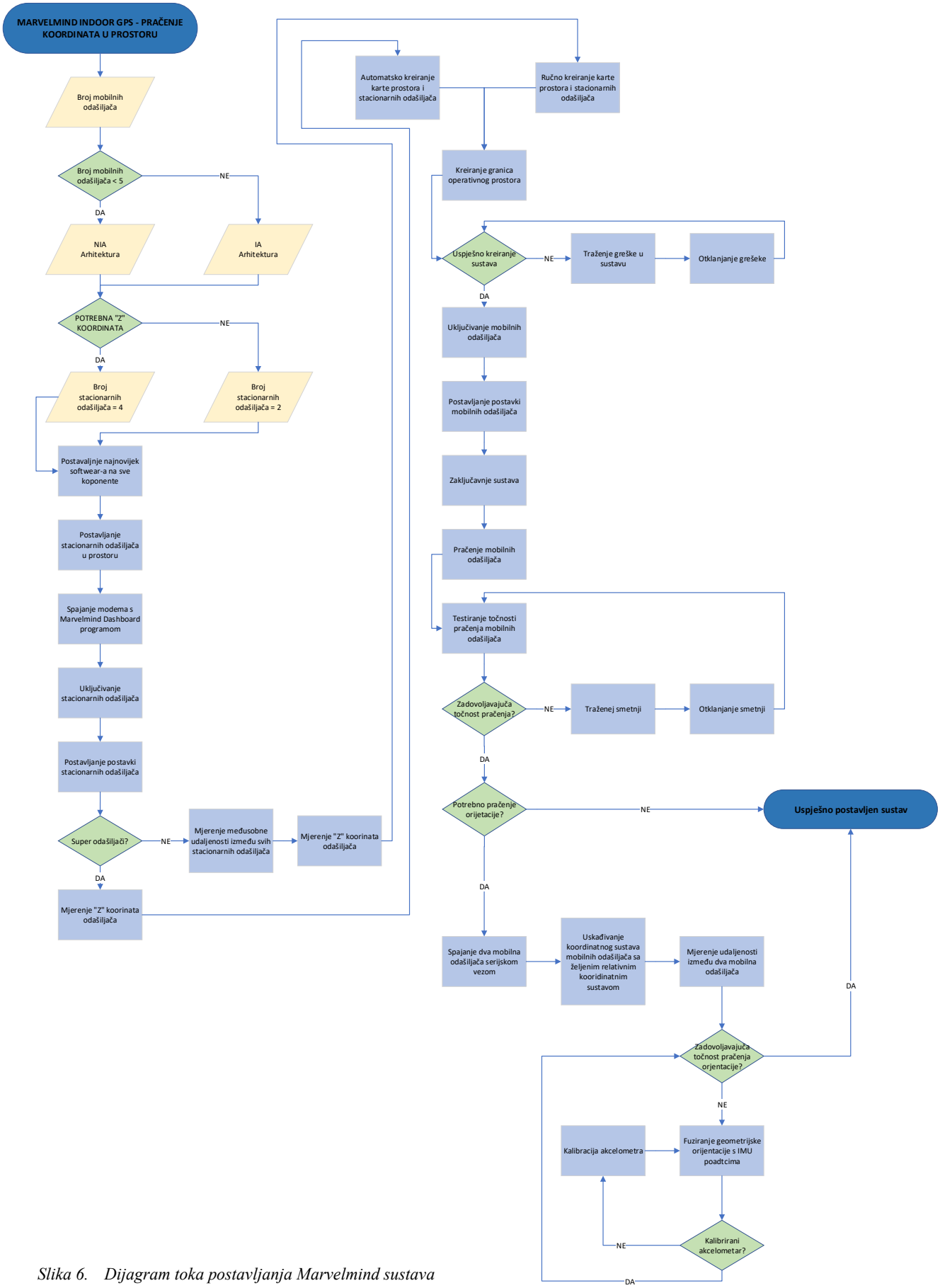
Marvelmind Robotics pruža uputstva za optimalno postavljanje stacionarnih odašiljača u prostoru i optimalno postavljanje mobilnih odašiljača u paru radi dobivanja orijentacije u globalnom koordinatnom sustavu. Sustav će se instalirati u središnjem prostoru CRTA-e. To je slobodni prostor površine $50\ m^2$.

Stacionarni odašiljači se postavljaju u radnom prostoru tako ta tvore zatvoreni krug. Sustav je optimalan i najstabilniji kada svi stacionarni odašiljači imaju različite orijentacije u radnom prostoru. Radni prostor potpuno je definiran stacionarnim odašiljačima. Svaki odašiljač predstavlja jednu graničnu koordinatu radnog prostora. Ishodište koordinatnog sustava radnog prostora definiran je oznakom u prostoru. Relativni položaj stacionarnih odašiljača u odnosu na ishodiše definiran je korištenjem dodatnog odašiljača koji je postavljen u ishodište. Prikaz postavljenih stacionarnih odašiljača u prostoru vidi se na Slika 7, a na Slika 8 vidi se prikaz radnog prostora u *Marvelmind Dashboard* programu.

Tandem mobilnih odašiljača koji će pratiti poziciju i orijentaciju ASTRO-a postavljaju se na ASTRO-a. Potrebno je simetralu dužine koja spaja dva odašiljača poravnati s $x - osi$ robota, a dužinu koja spaja dva odašiljač poravnati s $y - osi$ robota. Time je osigurano podudaranje

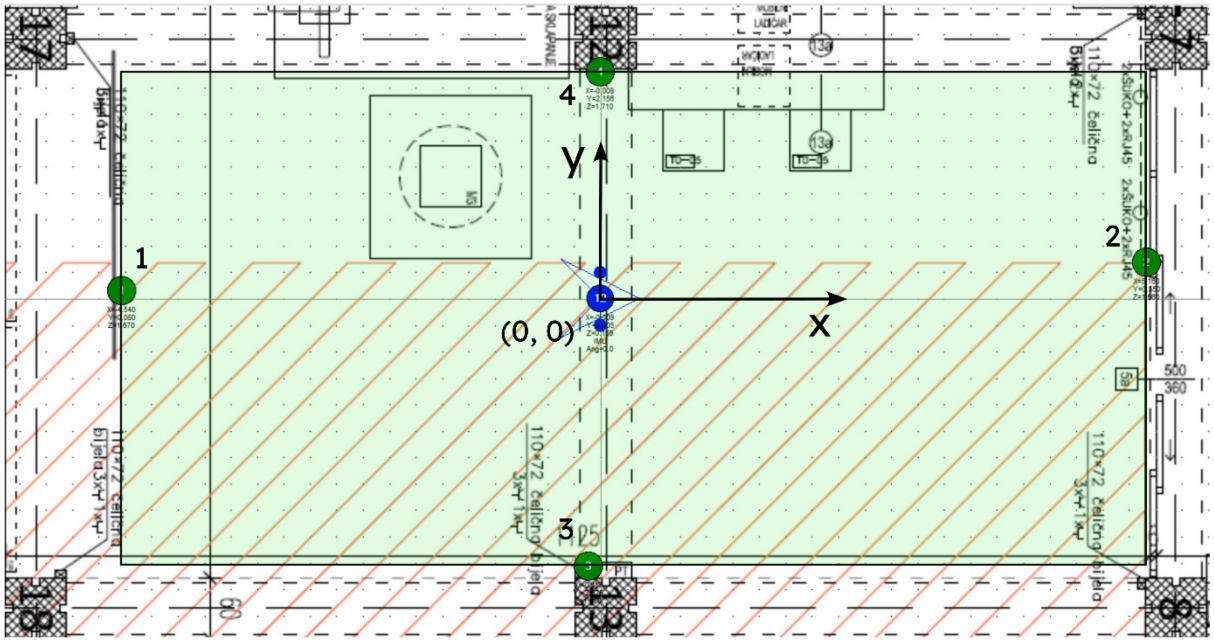
koordinatnog sustava ASTRO-a i *Marvelmind* tandema. Mobilni odašiljači su međusobno spojeni UART protokolom. Prikaz instaliranih mobilnih odašiljača vidi se na Slika 5. Nakon uspješnog postavljanja sustava uz pomoć *Marvelmind Dashboard* programa, karta prostora i stacionarnih odašiljača spremljena je u modemu i odašiljačima. U ovom trenutku *Marvelmind Indoor GPS* sustav je aktivan i prati se pozicija i orijentacija.

Slika 5. ASTRO s mobilnim odašiljačima



Slika 6. Dijagram toka postavljanja Marvelmind sustava

Slika 7. Stacionarni odašiljači u prostoru



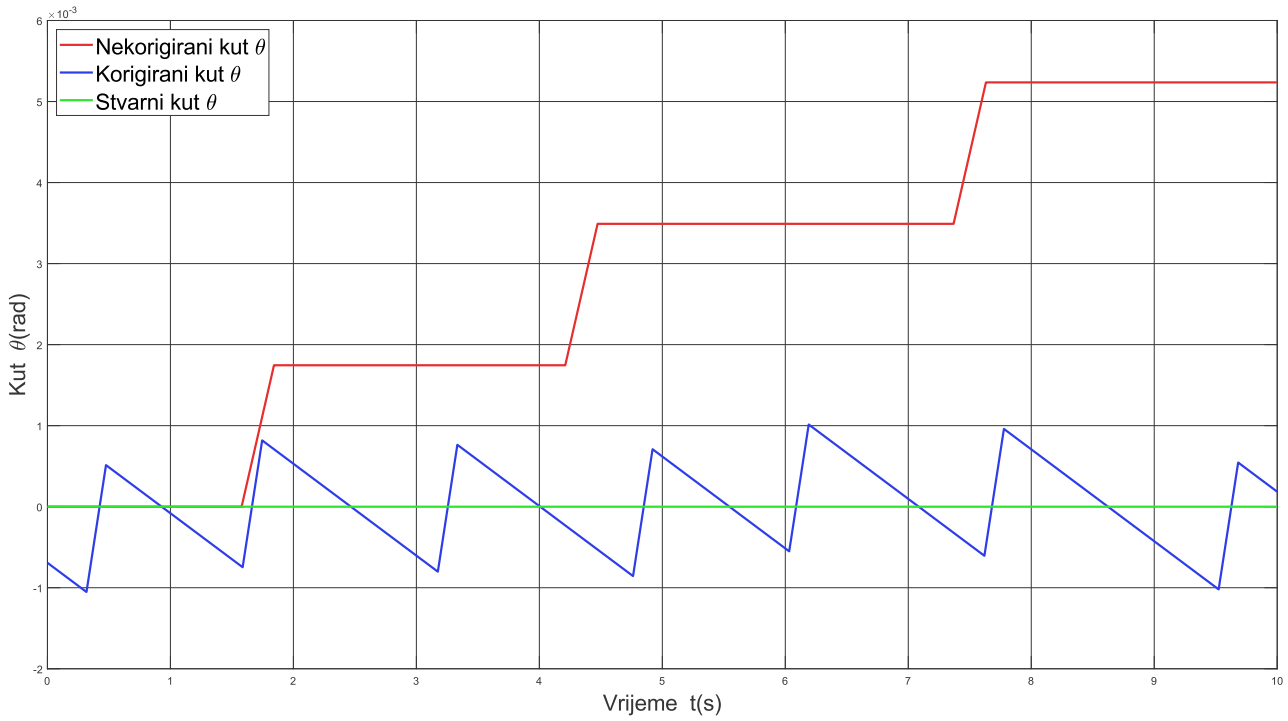
Slika 8. Prikaz radnog prostora

4.1.2. Kalibracija

Nakon postavljanja *Marvelmind Indoor GPS* sustava, uočena je kontinuirana rastuća greška. Sustav bi u stanju mirovanja periodično dodavao inkrementalno povećanje kuta. Greška nije prvobitno primjetna, ali pri dužem djelovanjem sustava inkrementalno mala povećanja će se akumulirati i tvorit će veliku sustavnu grešku. Nepoželjno periodičko povećanje kuta ispravlja se kontinuiranim smanjenjem kuta. Do optimalnog sustava dolazi se eksperimentalnim putem. Poruka o poziciji i orijentaciji objavljuje se brzinom od $12\ Hz$. Utvrđena srednja vrijednost pogreške $0{,}0033\ \frac{rad}{s}$. Sustav je optimalan kada se objavljeni kut smanji za

$$n_{poruke} * \frac{\omega_{pogreške}}{f_{sustava}} = n_{poruke} * \frac{0{,}0033\ rad*s^{-1}}{12\ Hz} = n_{poruke} * 0{,}000275\ rad, \qquad (4.1)$$

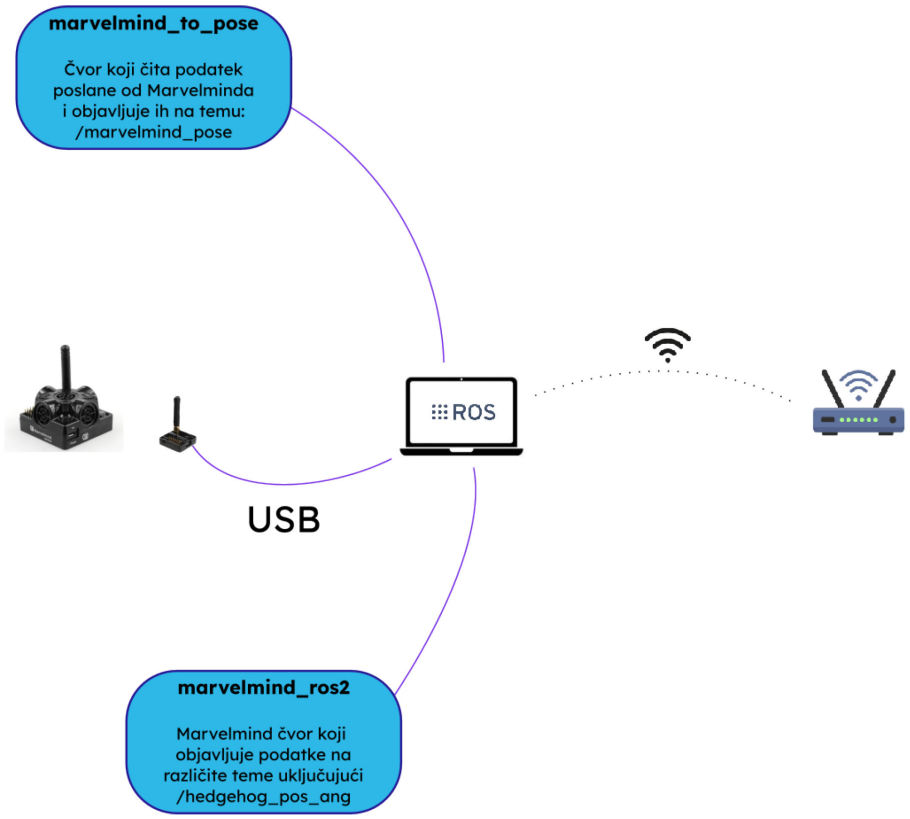
gdje je n_{poruke} broj objavljenih poruka nakon pokretanja sustava. Vizualizacija korekcije vidljiva je na Slika 9



Slika 9. Korekcija kuta

4.2. Marvelmind i ROS2

Komunikacija između *Marvelmind* sustava i ROS2 ostvaruje se ROS2 paketima *marvelmind_ros2* i *marvelmind_ros2_msg*. U paketu *marvelmind_ros2* nalazi se *.launch* datoteka koja pokreće serijsku komunikaciju između *Marvelmind* modema i računala. *Marvelmind* objavljuje poruke na više tema, jedna od tih tema je */hedgehog_pos_ang*. Na temu */hedgehog_pos_ang* objavljuju se poruke koje sadrže podatke pozicije i orijentacije. Popis svih dostupnih poruka vidljiv je u Prilog 2.



Slika 10. ROS2 okruženje Marvelmind

5. OPTITRACK

OptiTrack je sustav praćenja pokreta. Sustav prati zadane markere u prostoru te preko unaprijed definiranog međuodnosa markera računa gibanje praćenog objekta. *OptiTrack* nudi široku ponudu rješenja za razne primjene. Neke od primjena su robotika, interakcija s virtualnim okruženjima, medicina. U CRTA-i je instaliran *OptiTrack* sustav s 8 *OptiTrack Prime^{x}13* kamera. Sustav se odlikuje visokom točnošću praćenja objekta od $\pm 0{,}4\ mm$ i velikom brzinom praćenja do $240\ Hz$. Zbog svojih karakteristika *OptiTrack* sustav je odličan način praćenja referentnog položaja ASTRO-a. U narednim poglavljima točnost praćenja *Marvelmind* sustava će se uspoređivati u odnosu na *OptiTrack* sustav.

OptiTrack sustav koristi računalni program *Motive*. *Motive* služi za postavljanje, kalibraciju, grafički prikaz i obradu podataka *OptiTrack* sustava.

5.1. OptiTrack i ROS2

Komunikacija između sustava *OptiTrack* i ROS2 ostvaruje se putem VRPN paketa. Nakon instalacije VRPN-a, u radnom prostoru kreira se paket pod nazivom *„optitrack“*, unutar kojeg se generira *.launch* datoteka *„client.launch.yaml“* koja omogućava aktivaciju čvorova za komunikaciju između *OptiTracka* i ROS2-a. *OptiTrack* objavljuje složene poruke u kojima su sadržani podaci *headera,* pozicije i orijentacije.

Slika 11. OptiTrak kamere

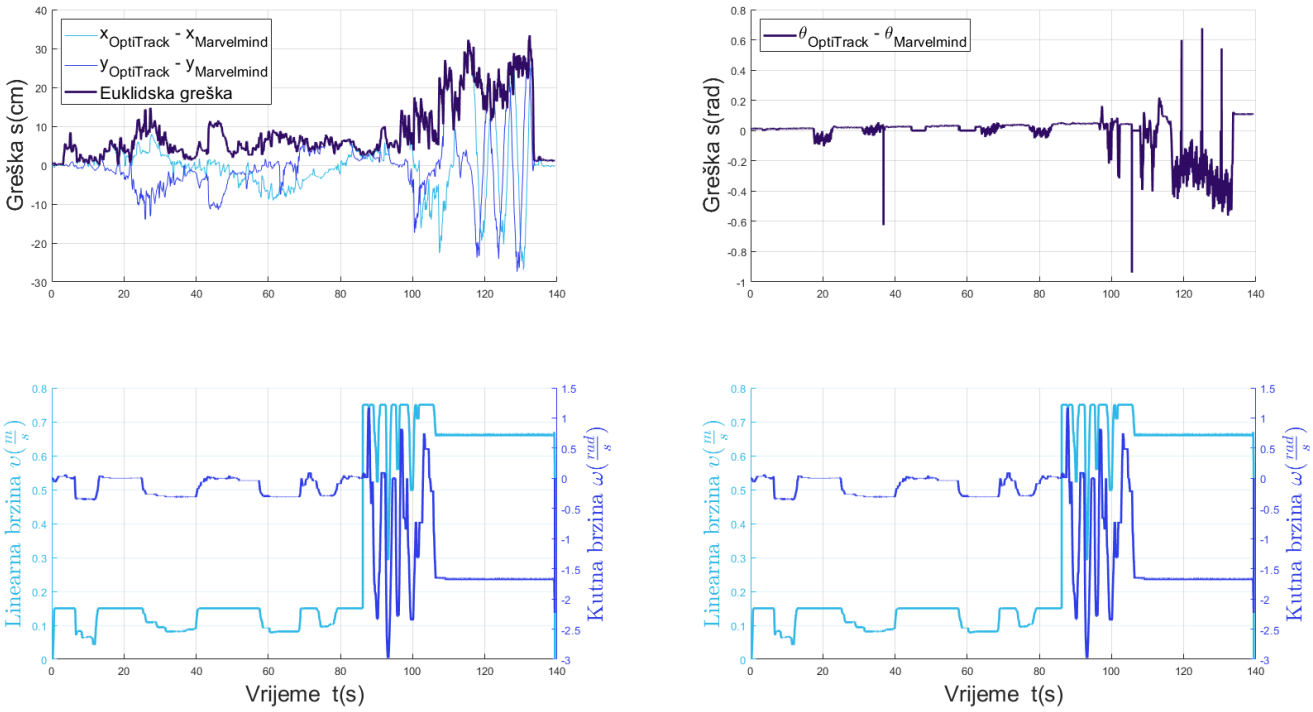
6. USPOREDBA PRAĆENJA POLOŽAJA I ORIJENTACIJE

Točnost *Marvelmind* sustava utvrdit će se eksperimentalno, paralelnim praćenjem *Marvelmind* i *OptiTrack* sustavima. *OptiTrack* sustav uzima se za referentnu vrijednost pozicije i orijentacije ASTRO-a. Za vrijeme praćenja podataka ASTRO je ručno upravljan *joystickom* po radnom prostoru. Snimanje je podijeljeno u dva dijela. U prvom dijelu snimanja brzine su male i malih promjena, a u drugom dijelu snimanja brzine su velike i velikih promjena.

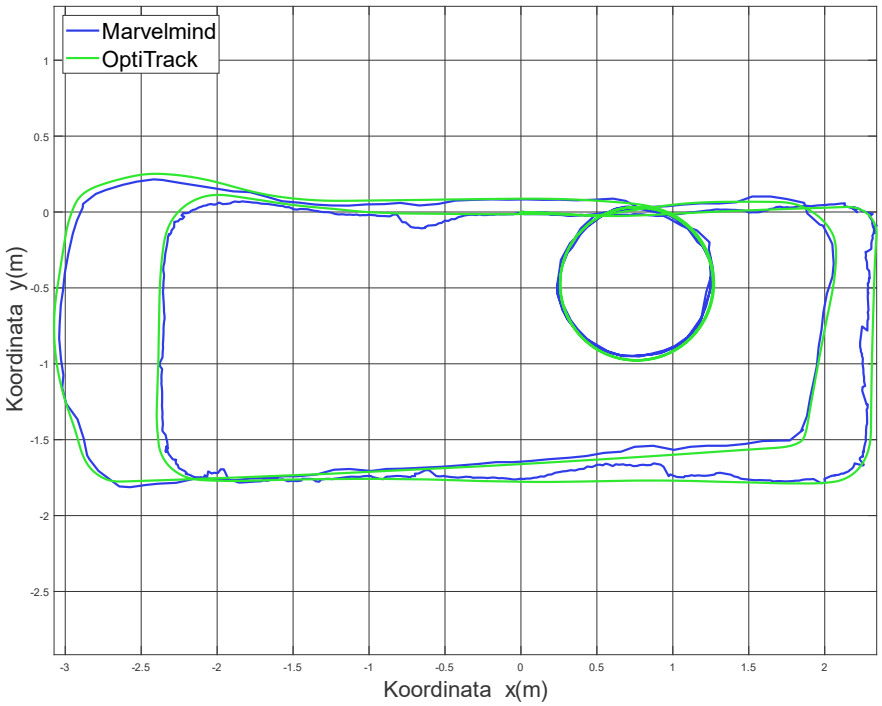
6.1. Interpretacija podataka

Pri malim brzinama prosječna greška pozicije dobivene *Marvelmind* sustavom iznosi $5\ cm$, a prosječna razlika orijentacije iznosi $0{,}013\ rad$.

Pri velikim brzinama prosječna greška pozicije dobivene *Marvelmind* sustavom iznosi $14{,}4\ cm$, a prosječna razlika orijentacije iznosi $0{,}076\ rad$. Korelacija greške s brzinom vidljiva je u Slika 12.



Slika 12. Greška u odnosu s brzinom



Slika 13. Putanja testiranja točnosti

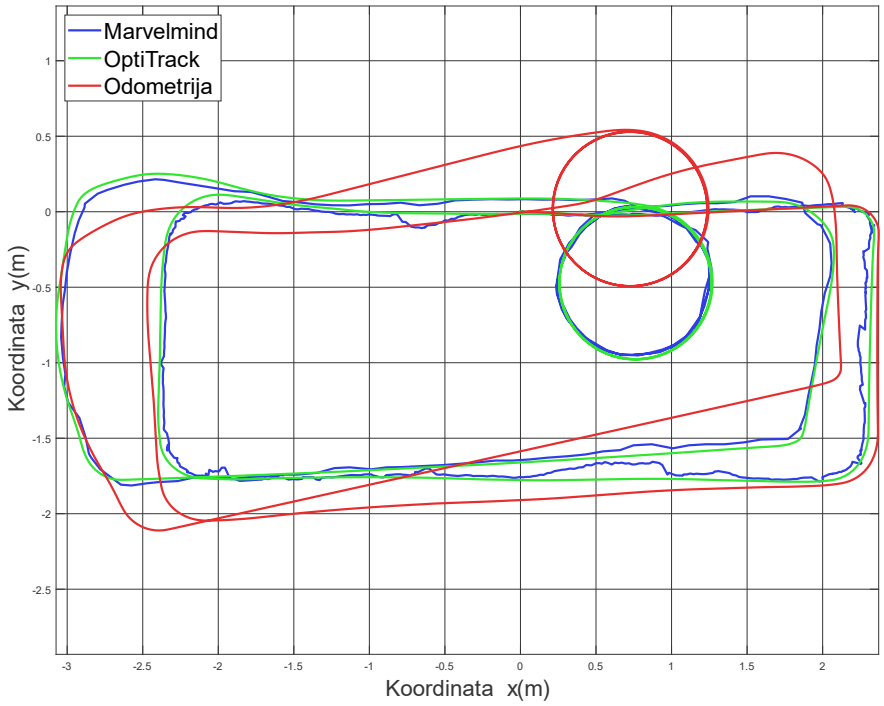
Točnost orijentacije *Marvelmind* sustava je jako dobra. Pri velikim brzinama i velikim oscilacijama greška ne prelazi 5°, što je približno $0.087266\ rad$. Točnost pozicije *Marvelmind* sustava je lošija od točnosti orijentacije. Pri malim brzinama točnost pozicije je prihvatljiva za sustave kojima nije potrebna visoka razina preciznosti. Pri veliki brzinama greška raste i počinje više oscilirati.

Iz perspektive navođenja robota do cilja *Marvelmind* sustav je prihvatljiv. Potrebno je uzeti u obzir mane sustava i prilagoditi navođenje robota. Brzine robota potrebno je ograničavati ovisno o potrebnoj preciznosti.

6.2. Usporedba s odometrijom

Jedna od najčešćih metoda praćenja pozicije i orijentacije robota je korištenjem odometrije. Mana takvih sustava je veliko povećanje greške s vremenom. U ASTRO-u odometrija se računa preko enkodera koji mjere broj rotacija motora. Kada postoji konstantna korelacija rotacije kotača i pomaka u radnom prostoru, sustav dobro radi. Pri velikim brzinama, pogotovo pri

velikim kutnim brzinama, rotacija kotača i pomak u radnom prostoru prestaju biti u korelaciji. Veliko odstupanje odometrije od stvarne puta vidljivo je na Slika 14.



Slika 14. Putanja testiranja s odometrijom

7. IZRAČUN PUTANJE DO CILJA

Određivanje putanje do cilja jedno je od područja robotike. Postoji više prepoznatih metoda za izračun putanje do cilja. Najčešće korištene metode:

- A*
- Dijkstraov algoritam
- RRT (*Rapidly-exploring Random Tree*)
- PRM (*Probabilistic Roadmap Method*)
- Genetski algoritmi
- Metoda potencijalnih polja

A* algoritam jedna je od najpopularnijih metoda za izračun putanje. Koristi heuristiku za pronalaženje najkraće putanje između dvije točke. Ovaj algoritam kombinira prednosti Dijkstraovog algoritma i heurističkog pretraživanja.

Dijkstraov algoritam je klasična metoda za pronalaženje najkraće putanje u grafu. Algoritam koristi pristup pohlepan (engl. *greedy*) i temelji se na postupnom istraživanju čvorova i dodjeljivanjem težina istima.

RRT metoda koristi slučajne uzorke za brzo istraživanje prostora i pronalaženje putanje.

PRM metoda prvo gradi probabilističku mapu slobodnog prostora, a zatim koristi tu mapu za pronalaženje putanja.

Genetski algoritmi koriste principe evolucije za optimizaciju putanja. Simuliraju procese prirodne selekcije kako bi pronašli optimalna rješenja.

Metoda potencijalnih polja gradi mapu umjetnih potencijala u kojoj cilj ima najniži potencijal, a prepreke imaju visoki potencijal.

7.1. Metoda potencijalnih polja

Metoda potencijalnih polja je jednostavna metoda za brz izračun putanje do cilja. Najčešće se koristi u unaprijed poznatom radnom prostoru, ali moguće su i varijante metode koje u stvarnom vremenu ažuriraju sustav. Metoda se zasniva na simuliranju interakcije potencijala:

$$F = -\frac{k*q_1*q_2}{r^2}.[6] \tag{7.1}$$

Ukupna sila koja će djelovati na česticu jednaka je zbroju svih sila koje djeluju na istu:

$$F_{cestica} = F_{privlacna} + F_{odbojna}. \tag{7.2}$$

Jednako tako može se graditi mapa umjetnih potencijala, gdje će se za svaku točku prostora izračunati privlačan potencijal i odbojan potencijal.

$$U(x) = U_{privlacni}(x) + U_{odbojni}(x) \tag{7.3}$$

$$U_{privlacni}(x) = \begin{cases} \frac{1}{2} * KP * \left\|x - x_{cilj}\right\|^2 & : \left\|x - x_{cilj}\right\| \leq s \\ s * KP * \left\|x - x_{cilj}\right\| - s^2 * \frac{1}{2} * KP & : \left\|x - x_{cilj}\right\| > s \end{cases} \tag{7.4}$$

$$U_{odbojni}(x) = \begin{cases} \frac{1}{2} * KO * \left(\frac{1}{\rho(x)} - \frac{1}{\rho_0}\right)^2 & : \rho(x) \leq \rho_0 \\ 0 & : \rho(x) > \rho_0 \end{cases} \tag{7.5}$$

$$\rho(x) = \left\|x - x_{prepreka}\right\| \tag{7.6}$$

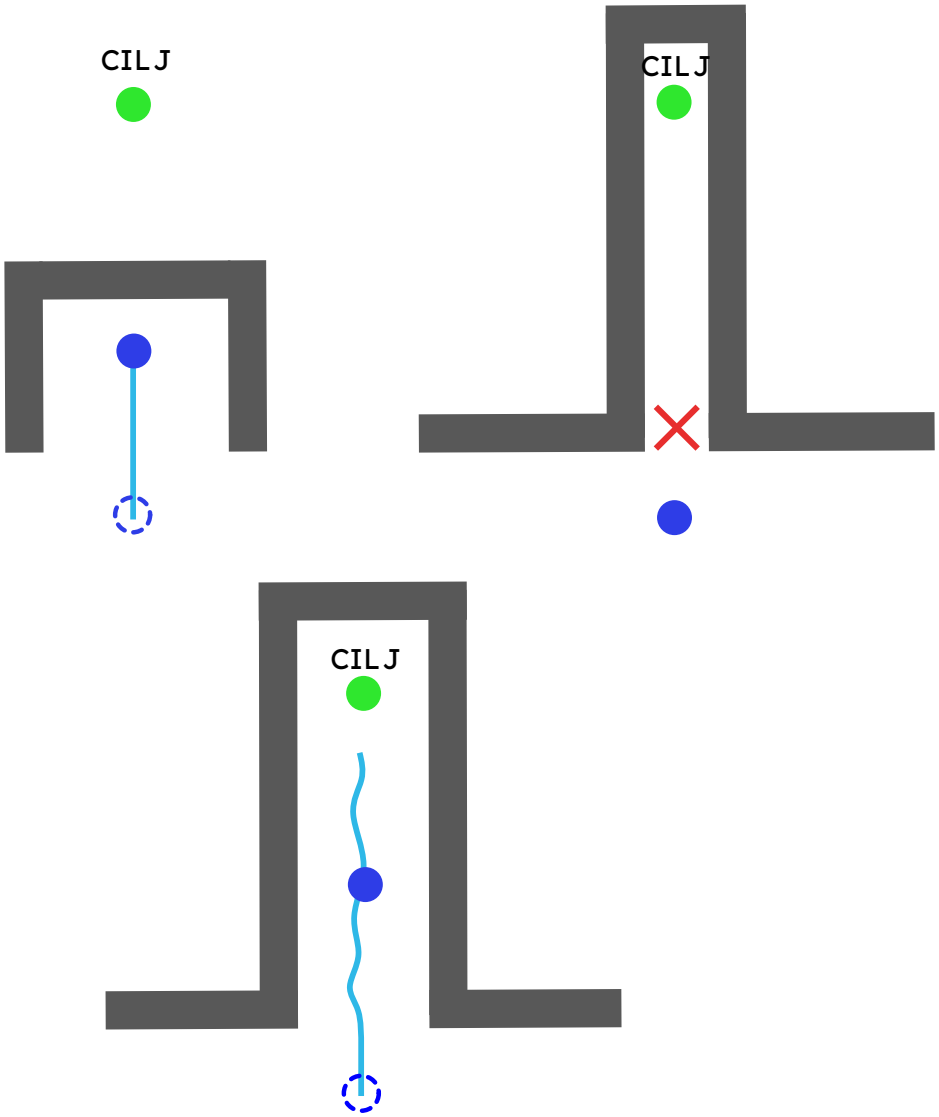
Privlačni potencijal $U_{privlacni}$ raste s porastom udaljenosti od cilja. Vidljivo na Slika 17 Funkcija je podijeljena na dva područja, linearni porast nakon granice s i porast s kvadratom unutra granice s. U funkciju su dodani faktori pojačanja KP i KO za mogućnost jednostavnog podešavanja sustava. Odbojni potencijal $U_{odbojni}$ definiran je kvadratom razlike inverza udaljenosti do prepreke $\rho(x)$ i inverza veličine ρ_0 koja predstavlja prostor djelovanja prepreke. Izvan prostora djelovanja ρ_0 odbojni potencijal $U_{odbojni}$ jednak je 0. Vidljivo na Slika 16. Zbrajanjem oba potencijala dobiva se ukupni potencijal točke $U(x)$. Prikaz polja vidljiv je na Slika 18

7.2. Određivanje putanje

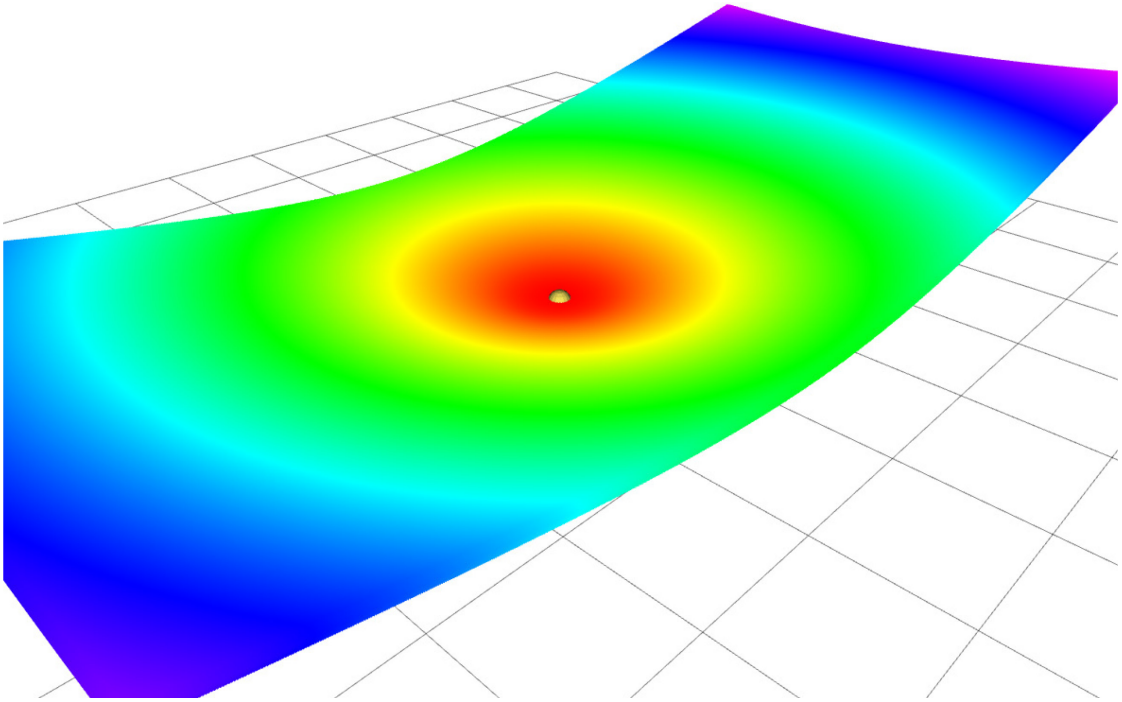
Putanja se određuje praćenjem negativnog gradijenta potencijalnog polja. Praćenje negativnog gradijenta očituje se kao pomicanje točke u smjeru najstrmijeg pada polja. Vidljivo na Slika 19. Takvo određivanje putanje predstavlja nekoliko problema:

- Putanja može završiti u lokalnom minimumu. U slučaju postojanja slijepih puteva, postoji rizik ulaska u njih i nemogućnosti dolaska u cilj.
- U vrlo uskim prolazima, odbojna sila generirana preprekama može zatvoriti prolaz do cilja.
- Metoda je karakterizirana oscilacijama što može dovesti do nepredviđenog ponašanja i neželjenih situacija poput kolizija s preprekama.

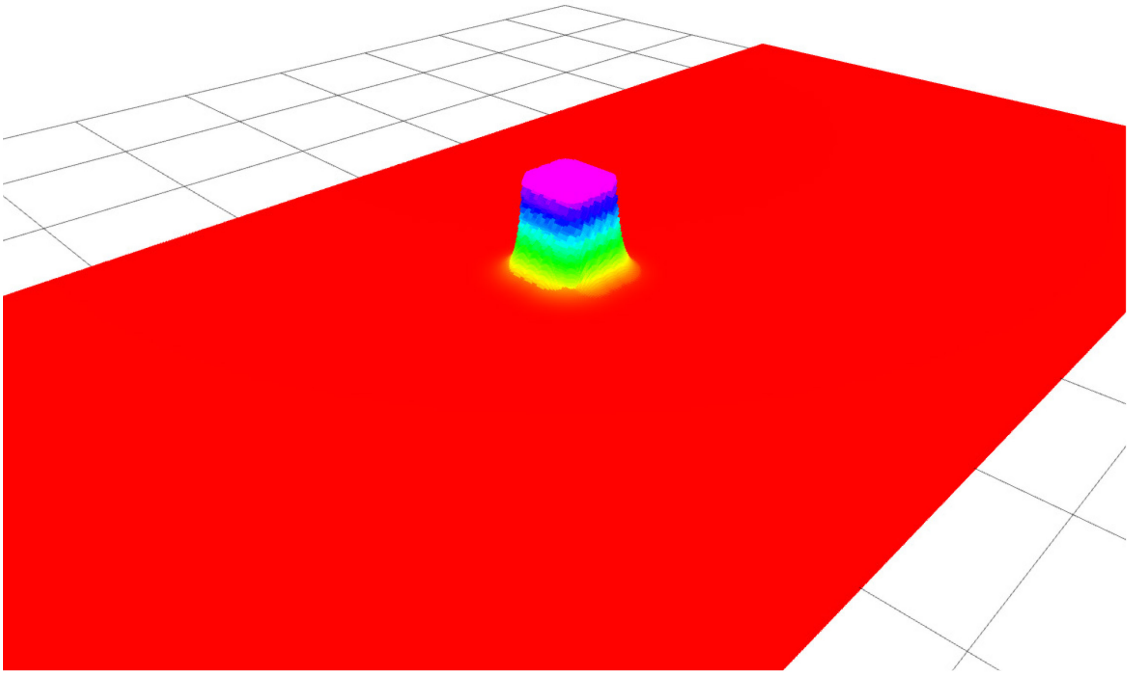
Postoje napredne metode kojima je cilj smanjiti negativne karakteristike metode potencijalnih polja. Također postoje napredne metode kojima je cilj optimizacija putanje na temelju vremena i utrošene energije.[6]



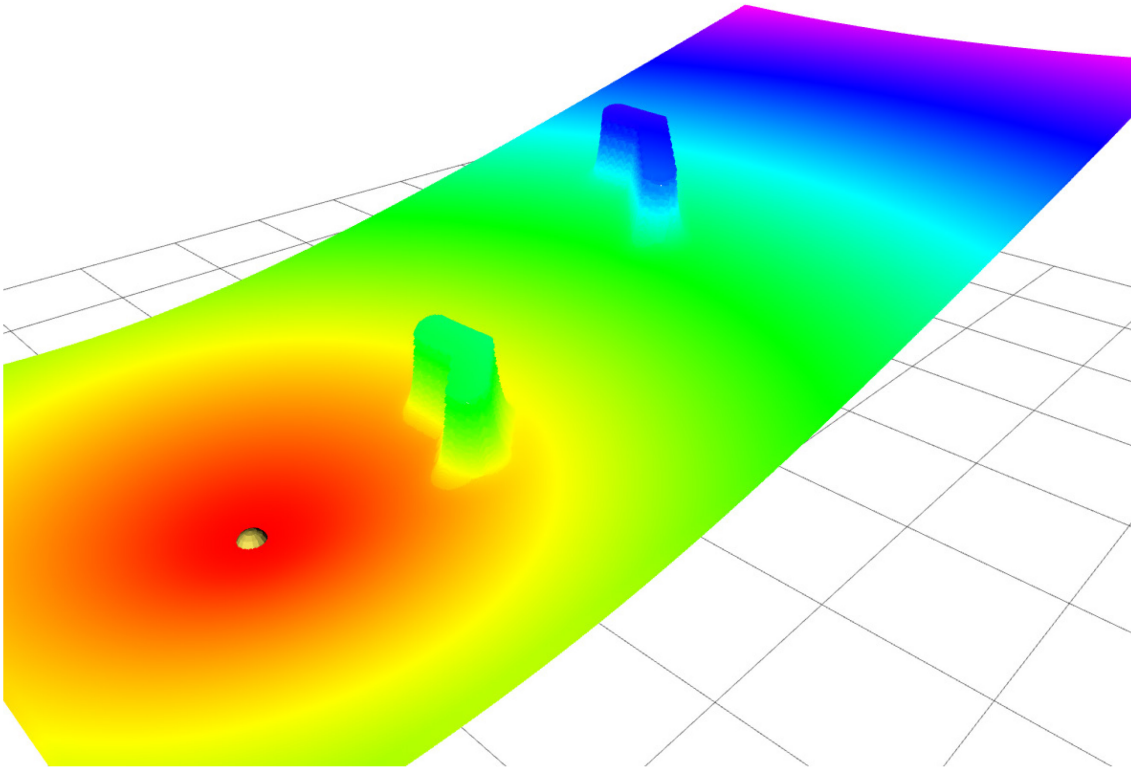
Slika 15. Problematične situacije[6]



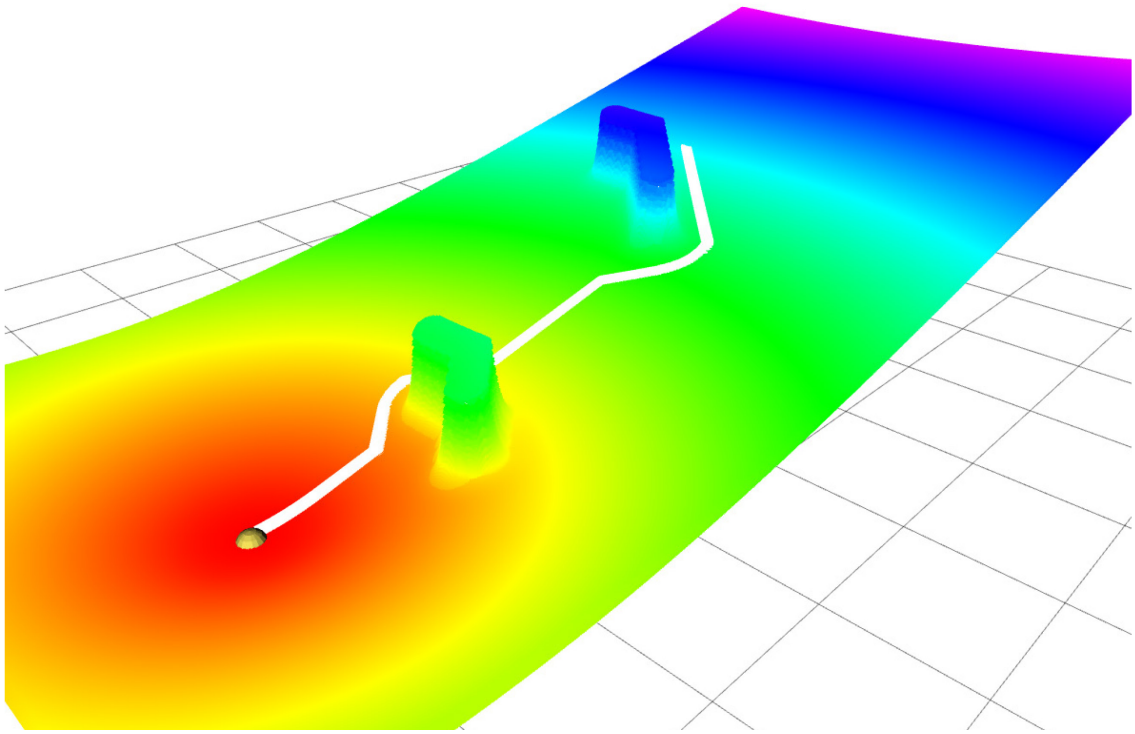
Slika 16. Privlačno potencijalno polje



Slika 17. Odbojno potencijalno polje



Slika 18. Potencijalno polje s preprekama



Slika 19. Putanja kroz potencijalno polje

8. PRAĆENJE PUTANJE DO CILJA

Nakon što je izračunata, poznata je putanja kojom će robot doći do cilja, potrebno je definirati kontroler kojim će se ista putanja pratiti. Postoji više modela kontrolera kojima je moguće pratiti zadanu putanju. Najčešće korišteni sustav je kombinacija *pursuit algorithma* i PID kontrolera.[7] Takav sustav navigira robota od točke do točke putanje. Konstantno se uzima nova točka koja je bliže cilju. Potrebnu linearnu brzinu računa PID kontroler koji za ulaz ima udaljenost do sljedeće točke. Kutna brzina računa se proporcionalnim kontrolerom koji djeluje na razliku trenutačne orijentacije i orijentacije prema točki. Orijentacija prema točki dobiva se korištenjem ATAN2 metode. Moguća su i druga rješenja sustava za praćenje putanje. Putanja dobivena metodom potencijalnih polja dobivena je praćenjem negativnog gradijenta. Isti gradijent može se koristiti za navigaciju robota.

8.1. Implementirani model

Implementirani model kombinacija je klasičnog *pursuit algorithma* s PID kontrolerom i metode potencijalnih polja. Kutna brzina dobiva se klasičnom metodom usporedbom orijentacija, a linearna brzina računa se preko gradijenta potencijalnog polja. Norma gradijenta obrađuje se proporcionalnim kontrolerom. Dobiveni sustav daje dobre karakteristike praćenja putanje. Reguliranjem linearne brzine korištenjem gradijenta potencijalnog polja dodaje se mogućnost optimiziranja. Moguće je dodatno optimizirati obilazak prepreka i podešavanje brzine prolaska kroz različita područja radnog prostora. Dobivene vrijednosti linearne i kutne brzine dalje se prosljeđuju ASTRO-u. ASTRO obrađuje ulaze i šalje odgovarajuće upravljačke signale elektromotorima.

8.2. Ulazni podaci

Kako bi kontroler davao kontinuirani stabilni izlaz potrebno je imati jednako takav ulaz. Prvobitni pokušaj ulaza su bili podaci dobiveni *Marvelmind* sustavom. Problem s tim podacima je što su diskretni i nije ih moguće pretvoriti u kontinuirani stabilni signal. Dolazi do prevelikih prebačaja koji rezultiraju velikim oscilacijama izlaznih brzina. Iz tog razloga *Marvelmind* sustav je potrebno ili dodatno stabilizirati primjenom IMU-a ili koristiti drugi izvor ulaznih podataka. U narednom testiranju ulazni podaci će biti uzeti iz odometrije ASTRO-a. Na početku gibanja odometrija se postavlja u nulti položaj s pomoću podataka *Marvelmind* sustava. Time je osigurana početna točnost podataka odometrije.

9. LOCIRANJE I NAVOĐENJE MOBILNOG ROBOTA

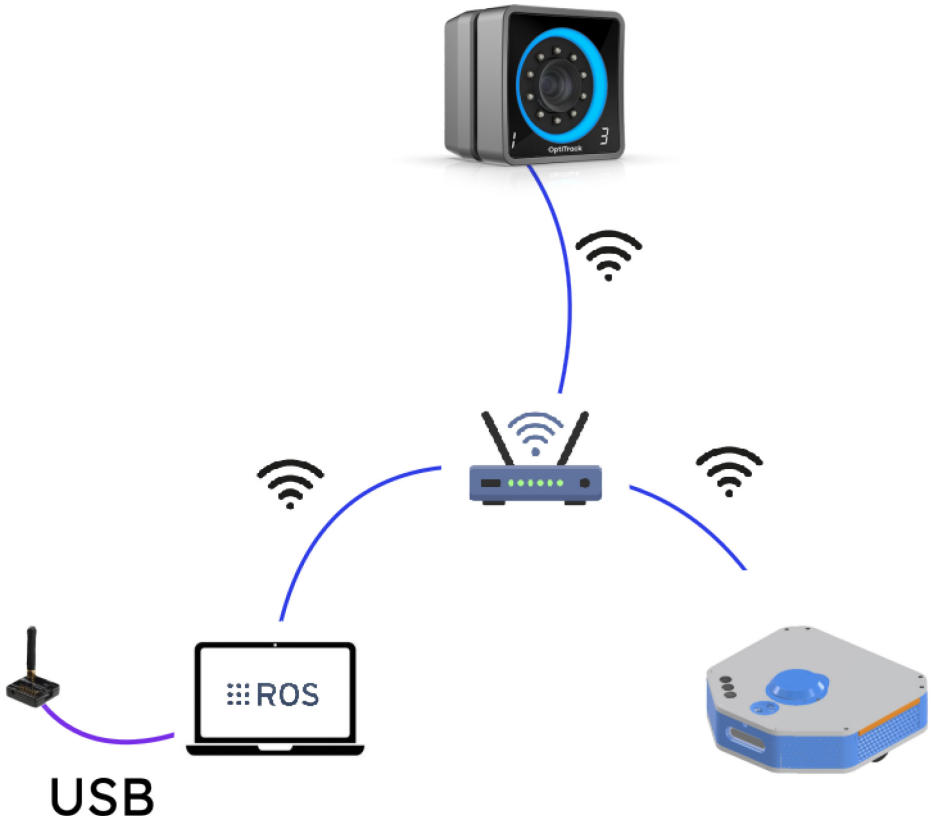
Lociranje i navođenje mobilnog robota do cilja metodom potencijalnih polja je složeni zadatak koji koristi više podsustava. Svi podsustavi prethodno su detaljno objašnjeni, a u ovom poglavlju dati će se opis njihove interakcije.

Sustav se sastoji od 3 podsustava:

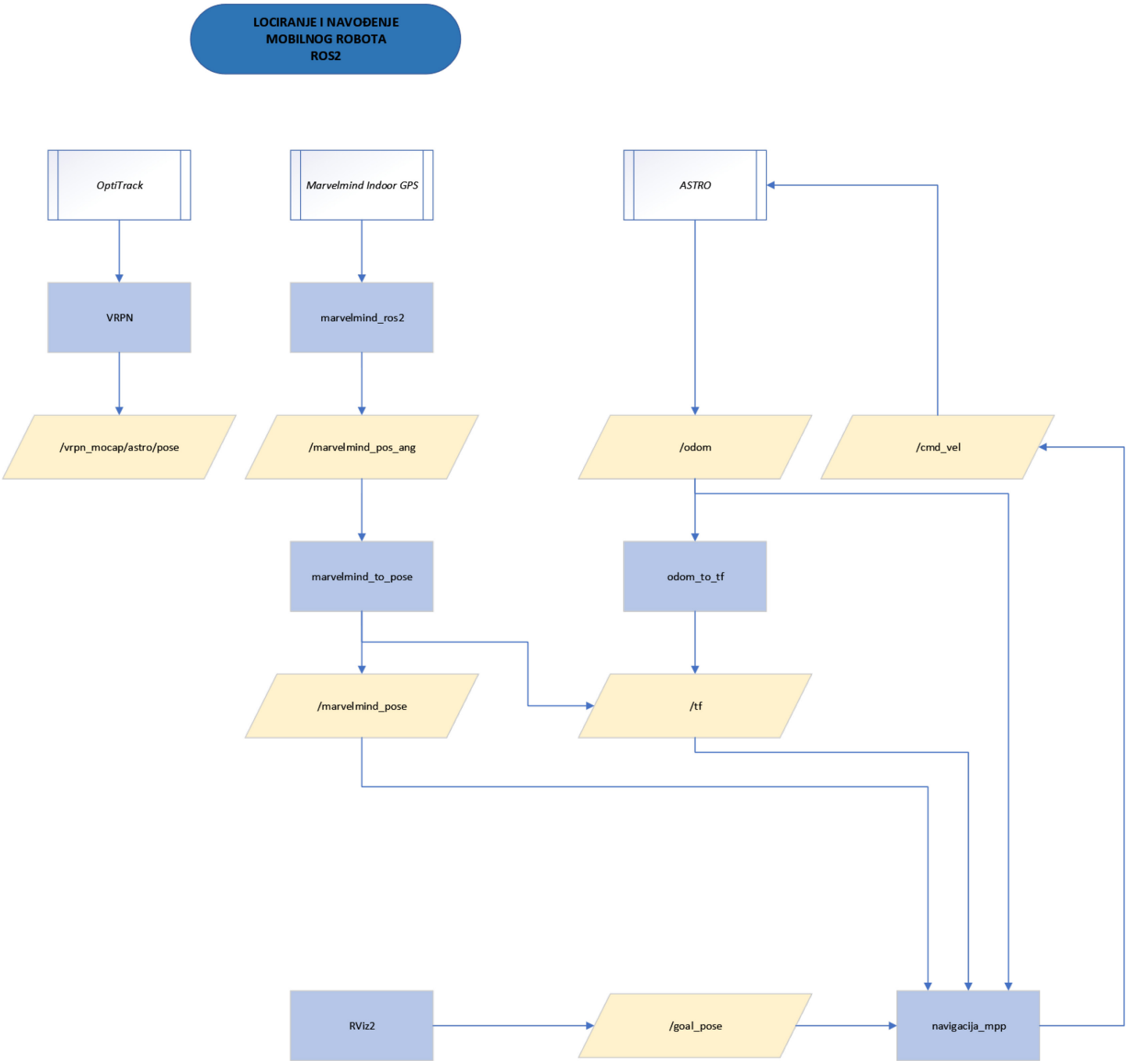
- *Marvelmind*
- *OptiTrack*
- *ASTRO*

Podsustavi međusobno su povezani čvorovima kreiranog paketa: *lociranje_i_navodenje.* U isto paketu se nalazi glavni izvršni čvor: *navigacija_mpp.* ASTRO, *OptiTrack* i glavno računalo koje pokreće *lociranje_i_navodenje* su spojeni na istu WLAN mrežu koja omogućuje prijenos podataka. *Marvelmind* i glavno računalo spojeni su USB protokolom.

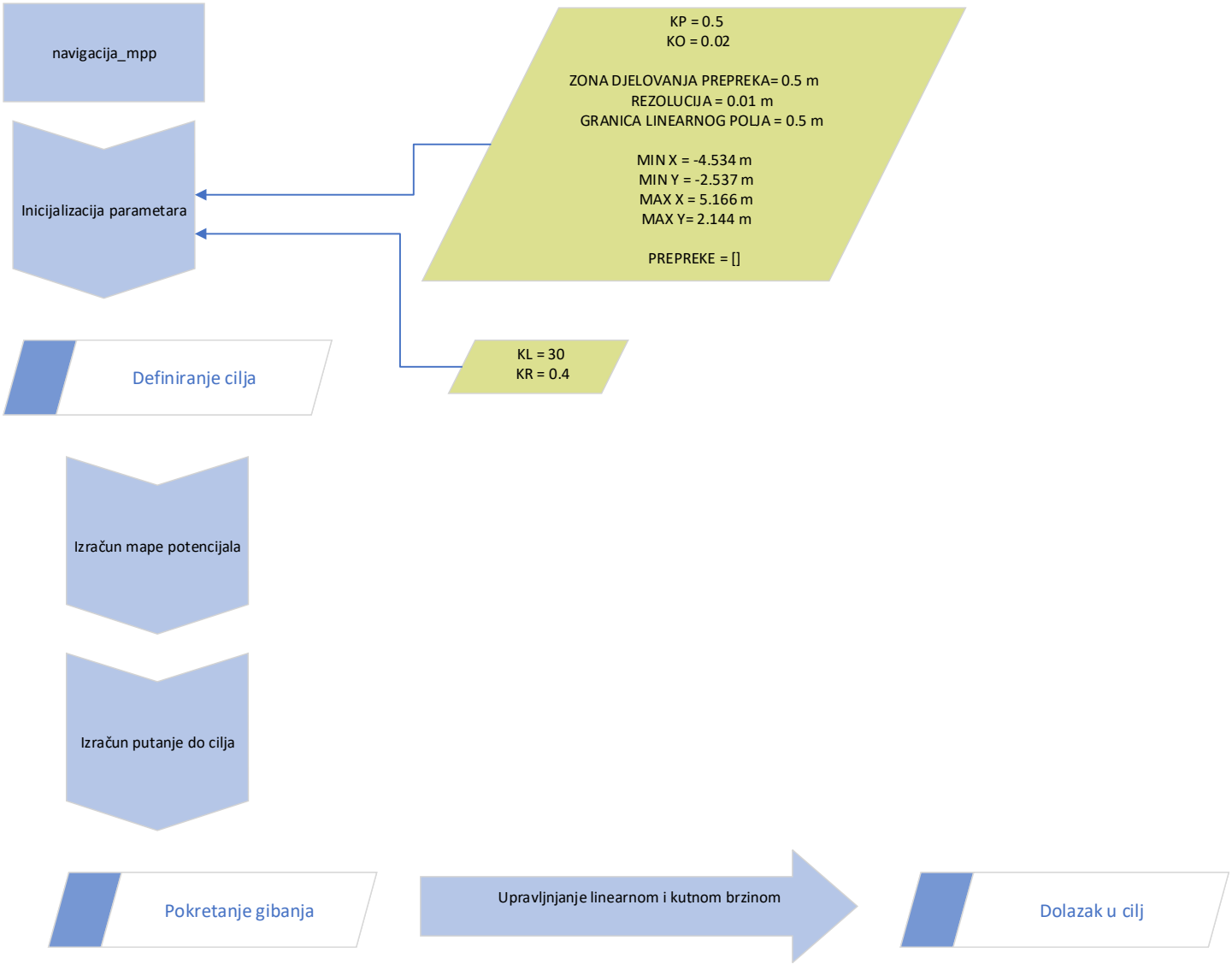
Međusobna integracija u ROS2 okruženju vidljiva je na Slika 21. Logika glavnog izvršnog čvora vidljiva je na Slika 22.



Slika 20. Komunikacija sustava



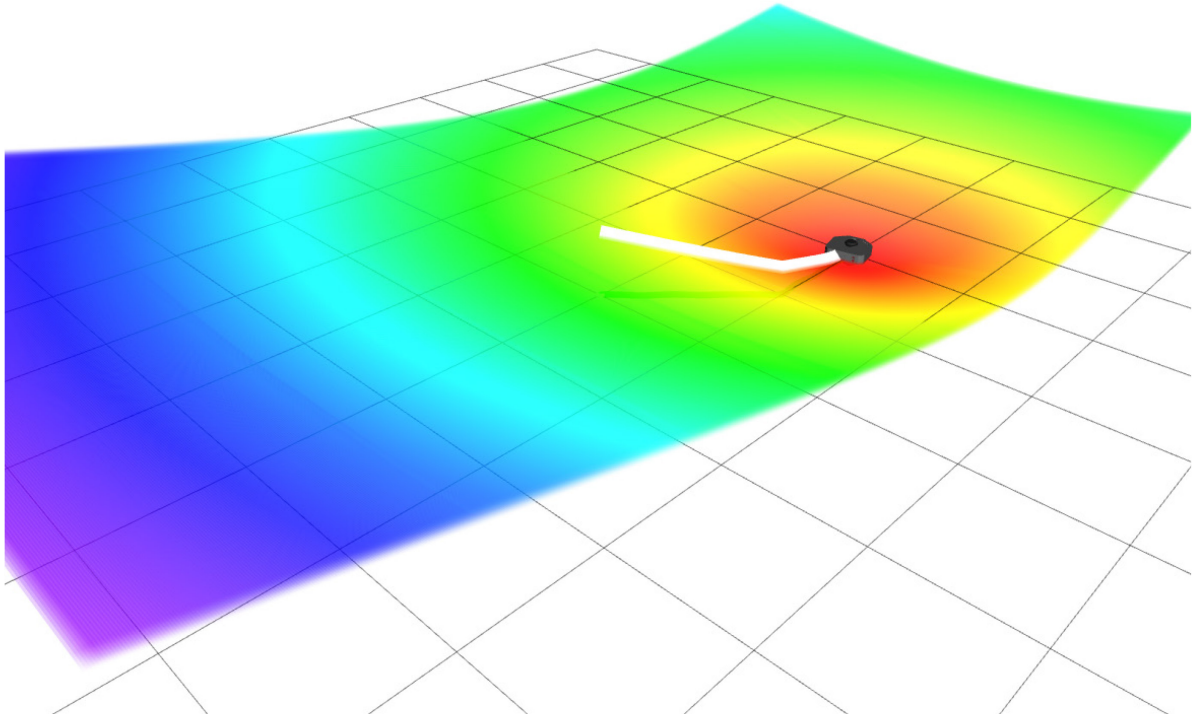
Slika 21. Lociranje i navođenje ROS2



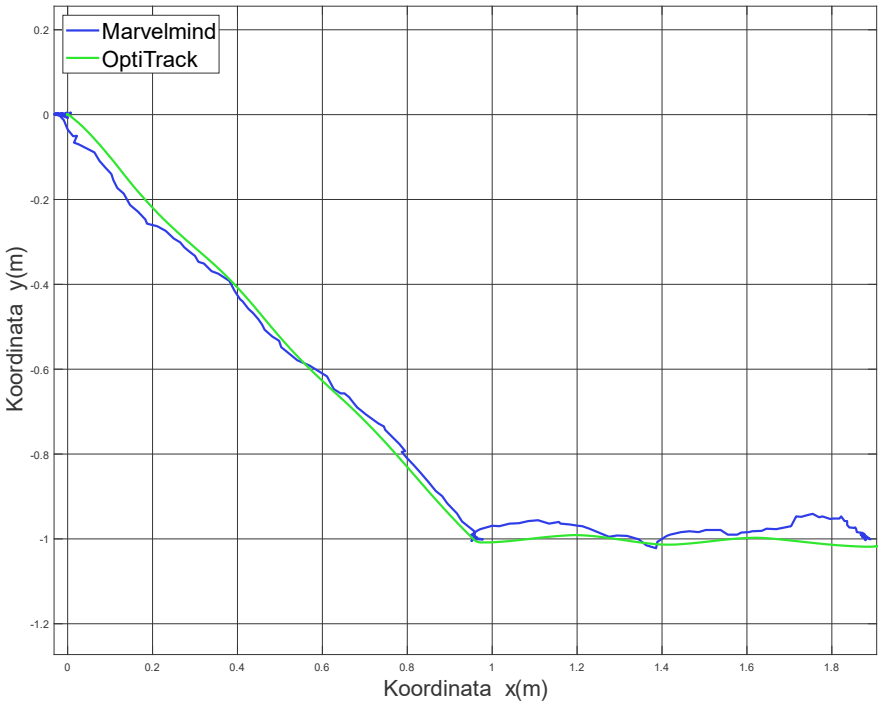
Slika 22. Dijagram toka navigacija_mpp

10. REZULTATI NAVOĐENJA ROBOTA DO CILJA

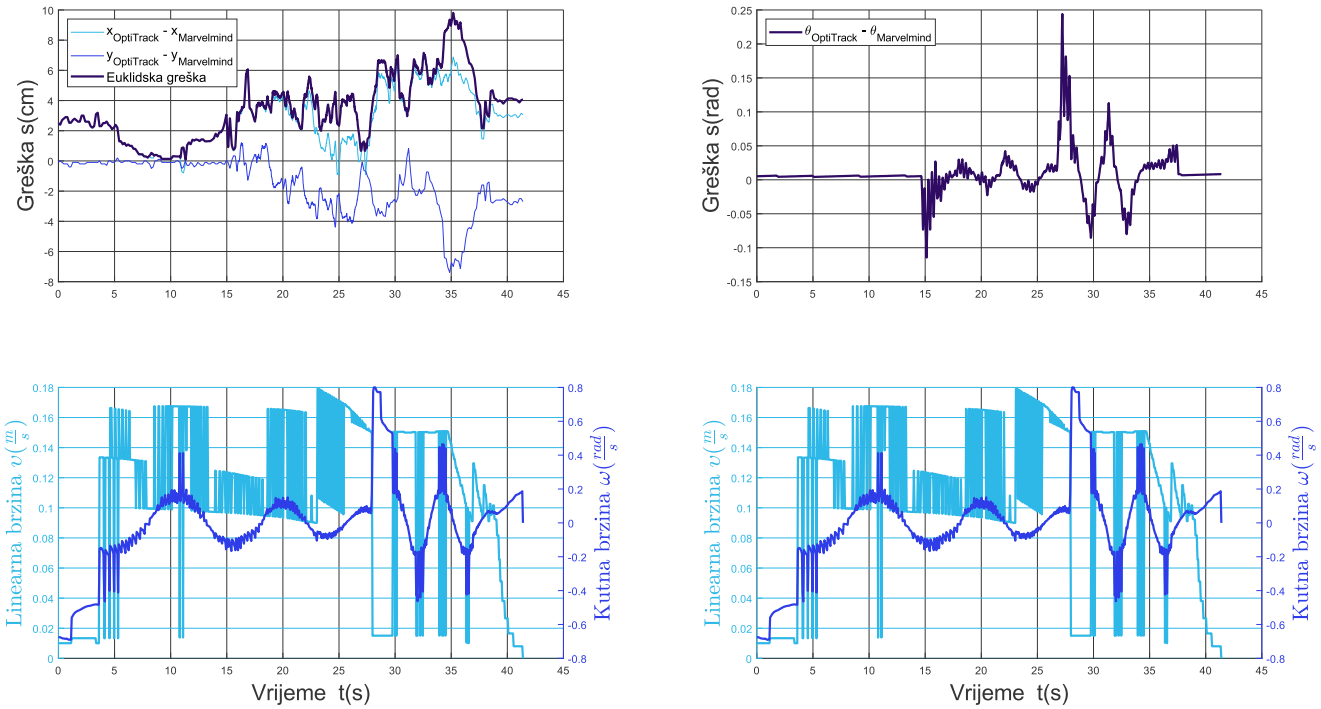
ASTRO uspješno dolazi do cilja. Kontrola s *OptiTrack* sustavom pokazuje kako ASTRO dolazi u cilj s odstupanjem do $\pm 2\ cm$. Vidljivo na Slika 25 i Slika 28. Zbog korištenja podataka odometrije, brzina gibanja limitirana je na $0{,}2\ m * s^{-1}$ kako bi se minimalizirale greške odometrije.



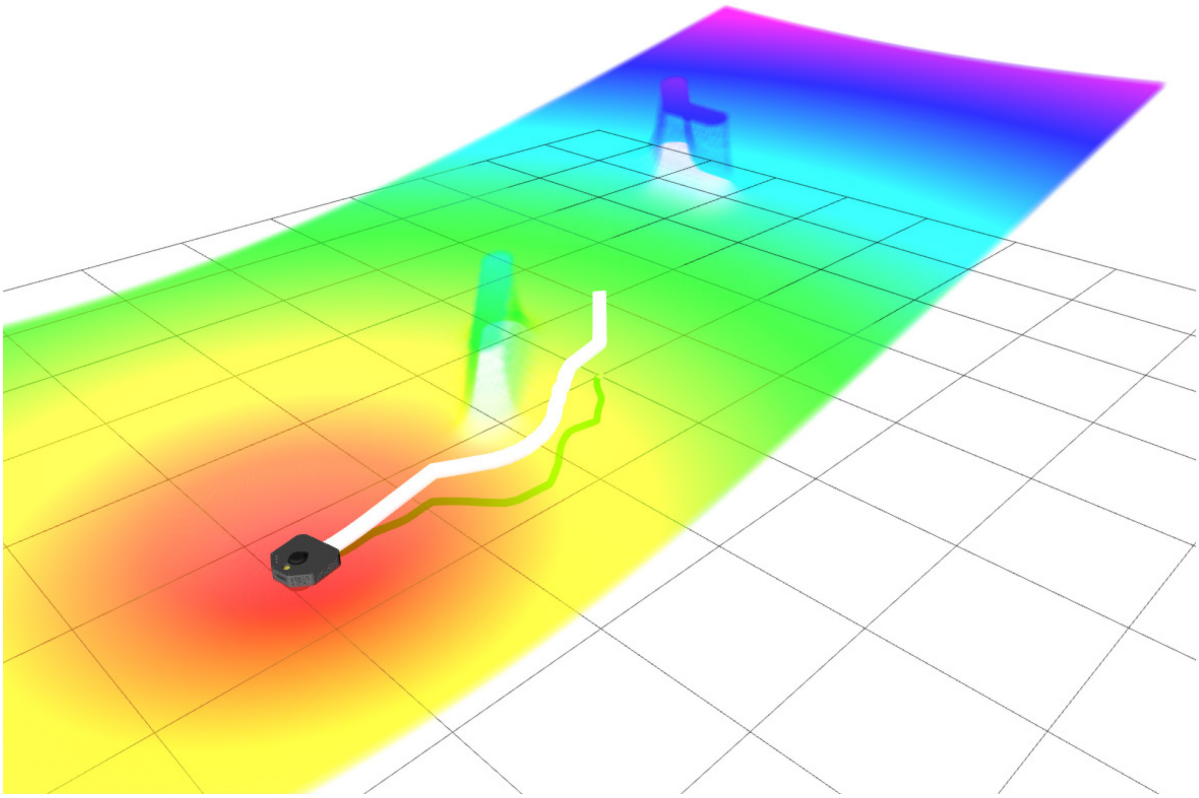
Slika 23. Potencijalno polje navođenja bez prepreka



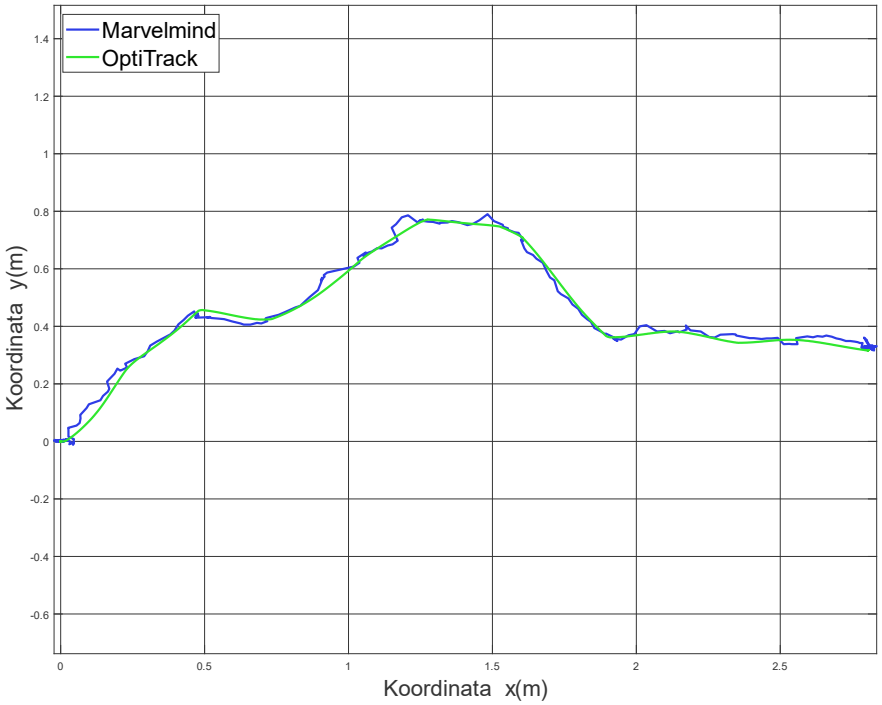
Slika 24. Putanja navođenja bez prepreka



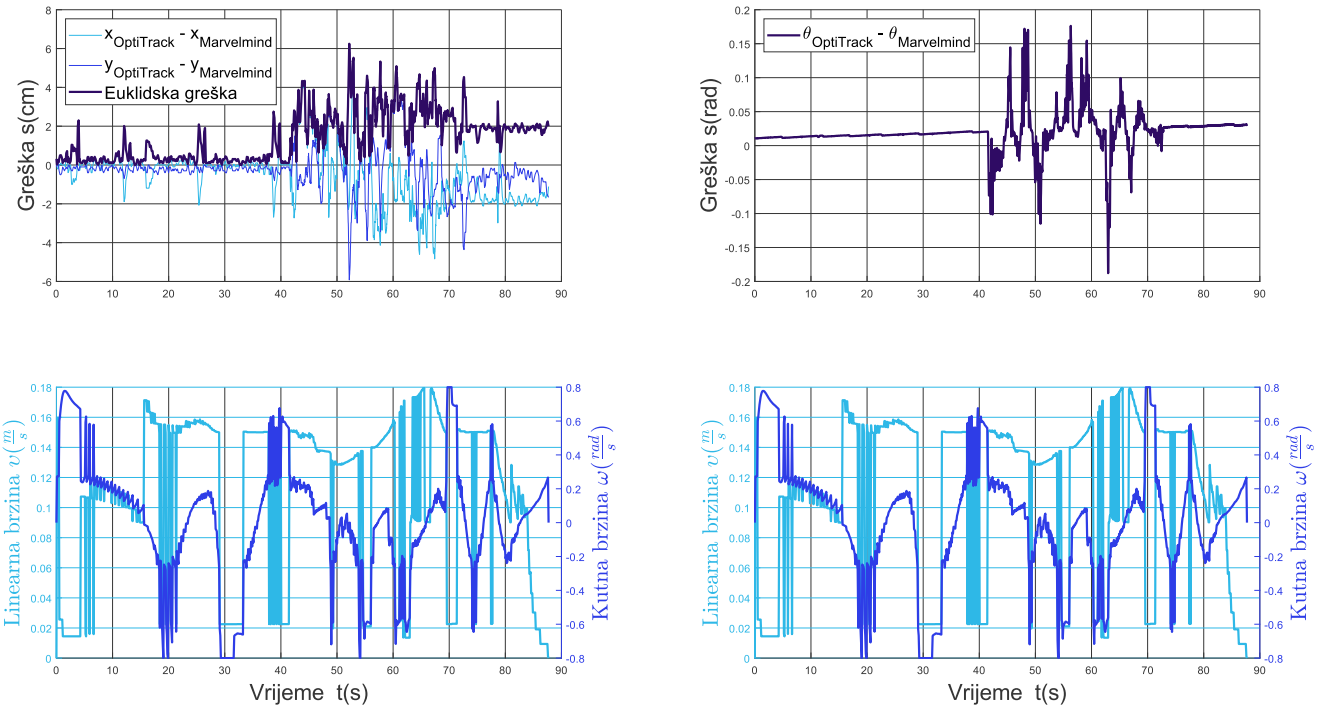
Slika 25. Greška navođenja bez prepreka



Slika 26. Potencijalno polje navođenja s preprekama



Slika 27. Putanja navođenja s preprekama



Slika 28. Greška navođenja s preprekama

11. KLJUČNI DIJELOVI PROGRAMA

U ovom poglavlju prikazat će se glavni programi koji su potrebni za rad sustava. Cijele verzije i ostatak potrebnih datoteka nalazi se na *GitHub* repozitoriju *KxHartl/zavrsni_rad* kojem se može pristupiti preko poveznice Prilog 1

11.1. hedgehog_to_pose.py

Python program *hedgehog_to_pose.py* je Python program u ulozi čvora koji preuzima podatke koje objavljuje *Marvelmind* sustav, obrađuje ih i objavljuje ih u standardnim ROS2 oblicima. Također u ovom Python programu sadržana je kalibracija orijentacije.

```
...
        self.angles = []
        self.angle_diff = 0.0
        self.angle_corected = False

        self.count = 0
        self.last_angle = 0.0
...

...
    def handle_hedgehog_pos_ang(self, msg):

        # Nuliranje kuta pri pokretanju sustava
        if not self.angle_corected:
            self.angles.append(msg.angle)
            if len(self.angles) > 20:
                self.angles.sort
                angles_med = self.angles[9]
                self.angle_diff = -angles_med
                self.angle_corected = True
        x_new = msg.x_m
        y_new = msg.y_m
```

```
        # Ispravljanje sustavne pogreske
        self.count = self.count + 1
        angle_drift = self.count * 0.00165/21

        q = quaternion_from_euler(0, 0, math.radians(msg.angle +
self.angle_diff) - angle_drift)

        t = TransformStamped()
        t.header.stamp = self.get_clock().now().to_msg()
        t.header.frame_id = 'world'
        t.child_frame_id = 'hedgehog'
        t.transform.translation.x = x_new
        t.transform.translation.y = y_new
        t.transform.translation.z = msg.z_m
        t.transform.rotation.x = q[0]
        t.transform.rotation.y = q[1]
        t.transform.rotation.z = q[2]
        t.transform.rotation.w = q[3]

        p = PoseStamped()
        p.header.stamp = self.get_clock().now().to_msg()
        p.header.frame_id = 'world'
        p.pose.position.x = x_new
        p.pose.position.y = y_new
        p.pose.position.z = msg.z_m
        p.pose.orientation.x = q[0]
        p.pose.orientation.y = q[1]
        p.pose.orientation.z = q[2]
        p.pose.orientation.w = q[3]

        self.tf_broadcaster.sendTransform(t)
        self.publisherHedgePose.publish(p)
```

```
    def handle_beacons_pos_addressed(self, msg):
        t = TransformStamped()
        t.header.stamp = self.get_clock().now().to_msg()
        t.header.frame_id = 'world'
        t.child_frame_id = str(msg.address)

        t.transform.translation.x = msg.x_m
        t.transform.translation.y = msg.y_m
        t.transform.translation.z = msg.z_m

        q = quaternion_from_euler(0, 0, 0)
        t.transform.rotation.x = q[0]
        t.transform.rotation.y = q[1]
        t.transform.rotation.z = q[2]
        t.transform.rotation.w = q[3]

        self.tf_broadcaster.sendTransform(t)
...
```

11.2. navigacija_mpp.py

Python program *navigacija_mpp.py* je *Python* program u ulozi čvora koji preuzima sve podatke i obrađuje ih. Za novi zadani cilj računa mapu umjetnih potencijala te kreira putanju do cilja. Nakon kreiranja putanje program prelazi u ulogu kontrolera koji zadaje linearnu i kutnu brzinu ASTRO-u.

```
...
        # kontrola treba li robot voziti
        self.drive = False
        # kontrola je li robot na pocetku
        self.is_start = True

        # radijus opisane kruznice robota                    [m]
        self.ROBOT_RADIUS = 0.5
        # faktor pojacanja privlacnog polja                  [/]
        self.KP = 0.5
        # faktor pojacanja odbojnog polja                    [/]
        self.KO = 0.02
        # rezolucija prosotra                                [m]
        self.RESOLUTION = 0.01
        # granica izvan koje je privlacna sila linearna      [m]
        self.LINEAR_BORDER = 0.5
        # rezolucija dolaska u cilj                          [m]
        self.RC = 0.003
        # faktor pojacanja kutne brzine                      [/]
        self.KR = 0.4
        # faktor pojacanja linerarne brzine                  [/]
        self.KL = 30

        # minimalna x-koordianta radnog prostora             [m]
        self.map_minimum_x = -4.533999919891357
        # minimalna y-koordianta radnog prostora             [m]
        self.map_minimum_y = -2.5369999408721924
```

```
        # maksimalna x-koordianta radnog prostora           [m]
        self.map_maximum_x = 5.165999889373779
        # maksimalna y-koordianta radnog prostora           [m]
        self.map_maximum_y = 2.1440000534057617

        self.start_x = 0.5          # start x lokacija  [m]
        self.start_y = 0.2          # start y lokacija  [m]
        self.goal_x = 2.0           # cilj x lokacija   [m]
        self.goal_y = 2.0           # cilj y lokacija   [m]

        # x-koorindate tocki prepreka     [m]
        self.obstacle_x = [0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, -1.5, -1.5, -
1.5, -1.5, -1.5, -1.6, -1.7, -1.8]
        # y-koorindate tocki prepreka     [m]
        self.obstacle_y = [-0.2, -0.1, 0.0, 0.1, 0.2, 0.3, 0.4, -0.2, -0.3,
-0.4, -0.5, -0.6, -0.7, -0.8, -0.9]

        # mapa iznosa potencijala za svaku tocku radnog prostora
        self.potential_field_map = []
        # koorinate izracunate putanje od pocetne poicije do cilja
        self.calculated_path = []

        self.robot_x = 0.0          # trenutna x -koordinata robota
        self.robot_y = 0.0          # trenutna y -koordinata robota
        self.robot_theta = 0.0      # trenutacna orijentacija robota
        self.point_index = 1
        self.robot_path = []        # prijedena putanja robota
...

...
    def handle_goal_pose(self, msg):
        self.calculate_speed(False)         # zaustavlanje robota
```

```
        # spremanje nove x -koordiante cilja
        self.goal_x = msg.pose.position.x
        # spremanje nove y -koordinate cilja
        self.goal_y = msg.pose.position.y
        self.is_start = True
        self.robot_path = []

        # izracun novog potencijalnog polja
        self.potential_field_map = self.calculate_potential_field()
        # izracun nove putanje do novog cilja
        self.calculated_path = self.calculate_path()



        cloud_points = []
        for i, row in enumerate(self.potential_field_map):
            for j, value in enumerate(row):
                i_n = i * self.RESOLUTION - abs(self.map_minimum_x)
                j_n = j * self.RESOLUTION - abs(self.map_minimum_y)

                cloud_points.append((i_n, j_n, value))
        header = std_msgs.msg.Header()
        header.stamp = self.get_clock().now().to_msg()
        header.frame_id = "world"
        cloud_msg = point_cloud2.create_cloud_xyz32(header, cloud_points)
        self.publisher_potential_field_map.publish(cloud_msg)

        path_msg = Path()
        path_msg.header.stamp = self.get_clock().now().to_msg()
        path_msg.header.frame_id = "world"
        for point in self.calculated_path:

            pose = PoseStamped()
            pose.header = path_msg.header
            pose.pose.position.x = point[0]
```

```
            pose.pose.position.y = point[1]
            pose.pose.position.z = point[2]
            path_msg.poses.append(pose)
        self.publisher_calculated_path.publish(path_msg)

        point_msg = PointStamped()
        point_msg.header = path_msg.header
        point_msg.point.x = self.goal_x
        point_msg.point.y = self.goal_y
        point_msg.point.z = 0.0
        self.publisher_goal_point.publish(point_msg)

    def calculate_potential_field(self):
            # broj potentiala u x-dimenziji
            n_x = int(round((self.map_maximum_x - self.map_minimum_x) /
self.RESOLUTION))
            # broj potentiala u y-dimenziji
            n_y = int(round((self.map_maximum_y - self.map_minimum_y) /
self.RESOLUTION))
            # x,y mapa potentiala
            potential_field_map = [[0.0 for i in range(n_y)]
for i in range(n_x)]

            for ix in range(n_x):
                x = ix * self.RESOLUTION - abs(self.map_minimum_x)

                for iy in range(n_y):
                    y = iy * self.RESOLUTION - abs(self.map_minimum_y)

                    # odredivanje iznosa privlacnog potentiala
                    p_attractive = self.calculate_attractive_potential(x, y)
                    # odredivanje odbojnog potentiala
                    p_repulsive = self.calculate_repulsive_potential(x, y)
                    # odredivanje ukupnog potentiala u tocki porsotora
                    potential_of_point = p_attractive + p_repulsive
```

```
                    # spremanje iznosa potencijala u mapu
                    potential_field_map[ix][iy] = potential_of_point

            return potential_field_map

    def calculate_path(self):
        distance_to_goal = np.hypot(self.start_x - self.goal_x, self.start_y
- self.goal_y)
        ix = int(round((self.start_x + abs(self.map_minimum_x)) /
self.RESOLUTION))
        iy = int(round((self.start_y + abs(self.map_minimum_y)) /
self.RESOLUTION))
        self.calculated_path = [[self.start_x, self.start_y,
self.potential_field_map[ix][iy]]]

        moving_opt = self.moving_options()

        while distance_to_goal >= self.RESOLUTION * 5:
            minimal_potential = float('inf')
            min_i_x, min_i_y = ix, iy

            for i, _ in enumerate(moving_opt):
                inx = int(ix + moving_opt[i][0])
                iny = int(iy + moving_opt[i][1])
                potential = self.potential_field_map[inx][iny]

                if potential < minimal_potential:
                    minimal_potential = potential
                    min_i_x = inx
                    min_i_y = iny

            ix = min_i_x
            iy = min_i_y
            point_x = ix * self.RESOLUTION - abs(self.map_minimum_x)
            point_y = iy * self.RESOLUTION - abs(self.map_minimum_y)
```

```
            distance_to_goal = np.hypot(self.goal_x - point_x, self.goal_y -
point_y)

            self.calculated_path.append([point_x, point_y,
minimal_potential])

        self.drive = True
        return self.calculated_path

    def calculate_attractive_potential(self, x, y,):
        point_to_goal = np.hypot(x - self.goal_x, y - self.goal_y)

        # izracun privlacnog potentiala u linearnom podrucju
        if point_to_goal > self.LINEAR_BORDER:
            ap = self.LINEAR_BORDER * self.KP * point_to_goal - 0.5 *
self.KP * self.LINEAR_BORDER ** 2
            return ap
       # izracun privlacnog potentiala u kvadratnom podrucju
        else:
            ap = 0.5 * self.KP * point_to_goal ** 2
            return ap

    def calculate_repulsive_potential(self, x, y):
        # najbliza prepreka u beskonacnosti
        minimum_distance_to_obstacle = float("inf")

        for i, _ in enumerate(self.obstacle_x):
            distance_to_obstacle = np.hypot(x - self.obstacle_x[i], y -
self.obstacle_y[i])

            if distance_to_obstacle <= minimum_distance_to_obstacle:
                minimum_distance_to_obstacle = distance_to_obstacle

        if minimum_distance_to_obstacle <= self.ROBOT_RADIUS:
```

```
            if minimum_distance_to_obstacle <= 0.1:
                minimum_distance_to_obstacle = 0.1
            # izracun iznos obojnog potentiala za zonu djelovanja
            return  0.5 * self.KO * (1.0 / minimum_distance_to_obstacle -
1.0 / self.ROBOT_RADIUS) ** 2
        else:
            # odbojni potential = 0 jer je van zone djelovanja
            return 0.0

    def moving_options(self):
        # moguca gibanja od celije do celije, trenutna celija [0, 0]
        moving_opt = [[0, 1],
                [0, -1],
                [1, 0],
                [1, 1],
                [1, -1],
                [-1, 0],
                [-1, -1],
                [-1, 1]]

        return moving_opt

    def calculate_speed(self, drive_or_stop):
        msg = Twist()
        # ako robot vozi
        if drive_or_stop:
            # udaljenost robota do cilja
            distance_robot_to_goal = np.hypot(self.robot_x - self.goal_x,
self.robot_y - self.goal_y)
            # odredivanje pozicije robota u x -osi polja
            ix = int(round((self.robot_x + abs(self.map_minimum_x)) /
self.RESOLUTION))
            # odredivanje pozicjie robota u u -soi polja
            iy = int(round((self.robot_y + abs(self.map_minimum_y)) /
self.RESOLUTION))
```

```
            # oredivanje potencijala robota
            robot_potential = self.potential_field_map[ix][iy]

            l = len(self.calculated_path) - 1
            is_end = False

            # trazenje nove tocke putanje
            while (robot_potential -
self.calculated_path[self.point_index][2]) < self.RC and (not is_end):
                if distance_robot_to_goal <= self.RESOLUTION * 10:
                    break
                elif self.point_index == l:
                    self.point_index = l
                    is_end = True
                else:
                    self.point_index += 1

            # razlika potencijala
            robot_to_point_p = robot_potential -
self.calculated_path[self.point_index][2]

            # potrebna orijentacija
            angle_new = math.atan2(self.calculated_path[self.point_index][1]
- self.robot_y, self.calculated_path[self.point_index][0] - self.robot_x)
            # razlika orijentacija
            angele_diff = angle_new - self.robot_theta

            # normalizacija kutna
            if angele_diff > math.pi:
                angele_diff = -2*math.pi + angele_diff
            elif angele_diff < -math.pi:
                angele_diff = 2*math.pi + angele_diff
```

```
            # definirai omjeri brzina za razlicine orijentacijske razlike
            if abs(angele_diff) > 0.7:
                msg.angular.z = self.KR*2 * angele_diff
                msg.linear.x = self.KL*0.15 * robot_to_point_p
            elif abs(angele_diff) > 1.5:
                msg.angular.z = self.KR*4 * angele_diff
                msg.linear.x = self.KL*0 * robot_to_point_p
            else:
                msg.angular.z = self.KR * angele_diff
                msg.linear.x = self.KL * robot_to_point_p

            # limitiranje brzina
            if msg.angular.z > 0.8:
                msg.angular.z = 0.8
            elif msg.angular.z < -0.8:
                msg.angular.z = -0.8
            else:
                pass
            if msg.linear.x > 0.3:
                msg.linear.x= 0.3
            elif msg.linear.x < -0.3:
                msg.linear.x = -0.3
            else:
                pass

            # zavrsetak gibanja robota
            if distance_robot_to_goal <= self.RESOLUTION * 5.5:
                msg.angular.z = 0.0
                msg.linear.x = 0.0
                self.drive = False
                self.is_start = True
                self.point_index = 1
                self.publisher_velocity.publish(msg)
                return
```

```
        else:
            msg.angular.z = 0.0
            msg.linear.x = 0.0
            self.drive = False
            self.is_start = True
            self.point_index = 1

            self.publisher_velocity.publish(msg)
            return

       # objavljivanje nove brzine
        self.publisher_velocity.publish(msg)

    def handle_marvelmind_pose(self, msg):
        if self.is_start:
            self.start_x = msg.pose.position.x
            self.start_y = msg.pose.position.y

            self.is_start = False
        else:
            pass

    # nova pozicija odometrije
    def handle_odom(self, msg):
        # spremanje nove x -koordinate robota
        self.robot_x = msg.pose.pose.position.x
        # spremanje nove y -koordinate robota
        self.robot_y = msg.pose.pose.position.y
        self.robot_path.append([self.robot_x, self.robot_y])

        angles = euler_from_quaternion([msg.pose.pose.orientation.x,
msg.pose.pose.orientation.y, msg.pose.pose.orientation.z,
msg.pose.pose.orientation.w])
        self.robot_theta = angles[2]
```

```
        if self.drive:
            # izracunavanje nove brzine
            self.calculate_speed(True)
            robot_path_msg = Path()
            robot_path_msg.header.stamp = self.get_clock().now().to_msg()
            robot_path_msg.header.frame_id = "world"
            for point in self.robot_path:
                pose = PoseStamped()
                pose.header = robot_path_msg.header
                pose.pose.position.x = point[0]
                pose.pose.position.y = point[1]
                pose.pose.position.z = 0.0
                robot_path_msg.poses.append(pose)
            self.publisher_robot_path.publish(robot_path_msg)
...
```

11.3. planiranje_putanje.launch.py

Python program *planiranje_putanje.launch.py* je program koji pokreće sve potrebne čvorove i potrebne druge *launch* programe.

```
from launch import LaunchDescription
from launch_ros.actions import Node
from launch_ros.substitutions import FindPackageShare
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import
PythonLaunchDescriptionSource
from launch.substitutions import PathJoinSubstitution
import os
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():
    rviz2_base =
os.path.join(get_package_share_directory('planiranje_putanje'), 'rviz')
    rviz2_full_config = os.path.join(rviz2_base,
'planiranje_putanje.rviz')

    return LaunchDescription([
        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([
                PathJoinSubstitution([
                    FindPackageShare('marvelmind_ros2'),
                    'launch',
                    'marvelmind_ros2.launch.py'
                ])
            ]),

            launch_arguments={}.items()
        ),

        IncludeLaunchDescription(
```

```
            PythonLaunchDescriptionSource([
                PathJoinSubstitution([
                    FindPackageShare('astro'),
                    'launch',
                    'rsp.launch.py'
                ])
            ]),

            launch_arguments={}.items()
        ),

        Node(
            package='planiranje_putanje',
            executable='hedgehog_to_pose',
            name='hedgehog_to_pose',
            arguments=[]
        ),

        Node(
            package='planiranje_putanje',
            executable='odom_to_tf',
            name='odom_to_tf',
            arguments=[]
        ),



        Node(
            package='rviz2',
            executable='rviz2',
            name='rviz2',
            arguments=['-d', rviz2_full_config]
        )

    ])
```

12. ZAKLJUČAK

Zadaci lociranja i navođenja mogu se riješiti na više načina. U ovom radu prikazane su 3 metode za praćenje pozicije i orijentacije: *Marvelmind Indoor GPS, OptiTrack* i odometrija. *OptiTrackom* je utvrđena točnost *Marvelmind* sustava. U stanju mirovanja pozicija dobivena *Marvelmind* sustavom ima točnost $\pm 1\ cm$, pri sporom gibanju ima točnost $\pm 5\ cm$, dok pri brzom gibanju točnost pozicije iznosi $\pm 15\ cm$. Drugi aspekt sustava je brzina osvježavanja pozicije koja iznosi $12\ Hz$. Uzimanjem svih parametara u obzir moguće je zaključiti kako *Marvelmind* sustav samostalno nije dovoljan za navođenje mobilnog robota. Dani podaci i zaključak odnose se na *Marvelmind* sustav koji ne koristi podatke IMU jedinica, koje fizički postoje u odašiljačima. Potrebno je daljnje istraživanje u kojem će se podaci pozicije dodatno usklađivati podacima IMU jedinica. Pretpostavka je kako će takav sustav zadovoljavati potrebnu točnost i stabilnost za uspješno navođenje mobilnog robota. Testirani sustav koristio je podatke pozicije i orijentacije dobivene *Marvelmind* sustavom za kalibriranje podataka odometrije. Tako definirani sustav uspješno navodi mobilnog robota do cilja s točnošću $\pm 2\ cm$. Za određivanje putanje do cilja korištena je metoda potencijalnih polja. Implementirana metoda je jednostavna za implementaciju i nije računalno zahtjeva. Postoje metode za optimizaciju metode potencijalnih polja kojima je moguće povećati efikasnost i brzinu dolaska u cilj. Cjelokupni testirani sustav je zadovoljavajući, ali nije optimalan te postoji mogućnost za unapređenjem kompletnog sustava.

LITERATURA

[1] Siciliano, B.; Khatib, O. Springer handbook of robotics. 2016
[2] U.S. government. GPS. https://www.gps.gov/systems/gps/. [Online]. Available at: https://www.gps.gov/systems/gps/. Accessed: September 10, 2024
[3] Macenski, S.; Foote, T.; Gerkey, B.; Lalancette, C.; Woodall, W. Robot Operating System 2: Design, architecture, and uses in the wild. Science Robotics. 2022;7(66), doi: 10.1126/scirobotics.abm6074.
[4] Mihelj, M.; Bajd, T.; Ude, A. et al. Robotics: Second edition. 2018
[5] Marvelmind. Marvelmind Indoor Navigation System Operating manual. 2024; [Online]. Available at: www.marvelmind.com.
[6] Szczepanski, Rafal; Tarczewski, Tomasz; Erwinski, Krystian. Energy Efficient Local Path Planning Algorithm Based on Predictive Artificial Potential Field. IEEE Access. 2022;10pp. 39729–39742. doi: 10.1109/ACCESS.2022.3166632.
[7] Corke, Peter. Robotics, Vision and Control. Cham: Springer International Publishing; 2017
[8] Marvelmind Robotics. Communication with Marvelmind devices using ROS 2 (Robot Operating System). 2024; [Online]. Available at: https://docs.ros.org/en/humble/Installation.html.
[9] Branimir Ćaran. Autonomni sustavi - Laboratorijske vježbe 1 - Branimir Ćaran. 2024;
[10] Branimir Ćaran. Autonomni sustavi - Vježbe 3 - Branimir Ćaran. 2024;

PRILOZI

https://github.com/KxHartl/zavrsni_rad.git

Topic	Message field	Type	Description
hedge_pos_ang	address	uint8	Address of mobile beacon
	timestamp_ms	uint32	Timestamp of location, milliseconds
	x_m	float64	X coordinate, meters
	y_m	float64	Y coordinate, meters
	z_m	float64	Z coordinate, meters
	flags	uint8	flags of location
	angle	float64	Orientation angle of paired beacons, degrees
beacon_pos_a	address	uint8	Address of stationary beacon
	x_m	float64	X coordinate, meters
	y_m	float64	Y coordinate, meters
	z_m	float64	Z coordinate, meters
beacon_distance	address_hedge	uint8	Address of mobile beacon
	address_beacon	uint8	Address of stationary beacon
	distance_m	float64	Raw distance from mobile to stationary beacon, meters
hedge_imu_fusion	timestamp_ms	int64	Timestamp of IMU fusion data, milliseconds
	x_m	float64	(X,Y,Z) coordinates of mobile beacon by IMU fusion. meters.
	y_m	float64	
	z_m	float64	
	qw	float64	Orientation quaternion of mobile beacon (qw,qx,qy,qz). Normalized ($qw^2+qx^2+qy^2+qz^2=1$)
	qx	float64	
	qy	float64	
	qz	float64	
	vx	float64	(vx, vy, vz) – speed vector of mobile beacon calculated by IMU fusion, meters/s
	vy	float64	
	vz	float64	
	ax	float64	(ax, ay, az) – acceleration of mobile beacon meters/s^2
	ay	float64	
	az	float64	
hedge_imu_raw	timestamp_ms	int64	Timestamp of raw IMU data, milliseconds
	acc_x	int16	(acc_x, acc_y, acc_z) – raw accelerometer data, 1 mg/LSB
	acc_y	int16	
	acc_z	int16	
	gyro_x	int16	(gyro_x, gyro_y, gyro_z) – raw gyroscope data, 0.0175 dps/LSB
	gyro_y	int16	
	gyro_z	int16	
	compass_x	int16	
	compass_y	int16	(compass_x, compass_y, compass_z) – raw compass data (only for HW4.9 beacons). X,Y: 1100 LSB/Gauss Z: 980 LSB/Gauss
	compass_z	int16	

hedge_quality	address	uint8	Address of the mobile beacon beacon
	quality_percents	uint8	Quality of location, percents
hedge_telemetry	battery_voltage	float64	Battery voltage of the mobile beacon, volts
	rssi_dbm	int8	RSSI (radio signal strength), dBm
marvelmind_waypoint	total_items	uint8	Total number of waypoint program items (N)
	item_index	uint8	Index of this waypoint item (0…N-1)
	movement_type	uint8	Type of action (6 = move to specified point)
	param1	int16	Parameter 1 (depends from movement_type) X coordinate of waypoint, cm if type= 6
	param2	int16	Parameter 2 (depends from movement_type) Y coordinate of waypoint, cm if type= 6
	param3	int16	Parameter 3 (depends from movement_type) Z coordinate of waypoint, cm if type= 6