Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Mobiliosios robotinės platformos navigacinės sistemos sukūrimas ir tyrimas

Baigiamasis magistro projektas

Edgaras Stasiukaitis

Projekto autorius

doc. dr. Gintaras Dervinis

Vadovas

Kaunas, 2024

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Mobiliosios robotinės platformos navigacinės sistemos sukūrimas ir tyrimas

Baigiamasis magistro projektas

Valdymo technologijos (6211EX014)

Edgaras Stasiukaitis

Projekto autorius

doc. dr. Gintaras Dervinis

Vadovas

doc. dr. Virginijus Baranauskas

Recenzentas

Kaunas, 2024

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Edgaras Stasiukaitis

Mobiliosios robotinės platformos navigacinės sistemos sukūrimas ir tyrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdama(s) kitų asmenų autoriaus ar kitų teisių, laikydamasi(s) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs (-usi);

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalinta(s) iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Edgaras Stasiukaitis

Patvirtinta elektroniniu būdu

Stasiukaitis, Edgaras. Mobiliosios robotinės platformos navigacinės sistemos sukūrimas ir tyrimas. Magistro baigiamasis projektas / vadovas doc. dr. Gintaras Dervinis; Kauno technologijos universitetas, Elektros ir elektronikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Elektronikos inžinerija, Inžinerijos mokslai.

Reikšminiai žodžiai: *Hector SLAM*, *Google Cartographer*, *GMapping*, *Karto SLAM*, mobilioji navigacinė sistema.

Kaunas, 2024. 81 p.

Santrauka

Darbe yra pristatomas mobiliosios robotinės platformos navigacinės sistemos sukūrimas ir tyrimas. Pastaruoju metu mobiliųjų robotų navigacijos problemoms buvo skirta nemažai dėmesio dėl vis jų augančio populiarumo automatizavimo srityse. Dažnai uždarose aplinkose nepakanka tradicinių priemonių, kaip roboto pozicijos nustatymo remiantis *GPS*. Tokių problemų sprendimui vis dažniau pasitelkiami *SLAM* metodai.

Šio darbo tikslas yra iš išanalizuotų lokalizacijos ir žemėlapio sudarymo metodų rasti tinkamiausią algoritmą atliekant eksperimentus su skirtingais lokalizacijos ir kartografavimo metodais bei parametrais, jog robotas gebėtų be klaidų atlikti navigaciją dinamiškai kintančioje aplinkoje ir transportuoti krovinius.

Analitinėje dalyje yra apžvelgiami šiuolaikiniai naudojami lokalizacijos ir žemėlapio sudarymo algoritmai. Analizuojami *Hector SLAM*, *Google Cartographer*, *GMapping* ir *Karto SLAM* pritaikymai bei literatūros šaltiniuose gauti metodų tikslumai.

Metodinėje dalyje yra atliekamas mobiliosios platformos projektavimas pasitelkiant lazerinį atstumo jutiklį *YDLIDAR X4* ir odometrijos duomenis gaunamus iš mobiliosios platformos enkoderių. Atliekamas *Hector SLAM* algoritmo derinimas siekiant gauti geriausias parametrų vertes roboto lokalizavimui ir žemėlapio sudarymui. Taip pat atliekami navigacijos eksperimentai pasitelkiant *move_base* modulį.

Eksperimentinėje dalyje pritaikomi ir tiriami *Google Cartographer*, *Hector SLAM*, *GMapping* ir *Karto SLAM* metodai. Atsižvelgiama į kiekvieno algoritmo lokalizacijos ir sudaryto žemėlapio tikslumą, įvertinami našumo rodikliai bei išrenkamas tiksliausias navigacijai algoritmas pagal gautus eksperimentų rezultatus.

Stasiukaitis, Edgaras. Development and Investigation of Navigation System for Mobile Robotic Platform. Master's Final Degree Project / supervisor doc. dr. Gintaras Dervinis; Faculty of Electrical and Electronics Engineering, Kaunas University of Technology.

Study field and area (study field group): Electronics Engineering, Engineering Science.

Keywords: *Hector SLAM*, *Google Cartographer*, *GMapping*, *Karto SLAM*, mobile navigation system.

Kaunas, 2024. 81 pages.

Summary

This thesis presents the Development and Investigation of Navigation System for Mobile Robotic Platform. Recently, the navigation problems of mobile robots have received a lot of attention due to their increasing popularity in automation applications. Often in confined environments, traditional means such as *GPS-based* robot positioning are not sufficient. *SLAM* techniques are increasingly being used to solve such problems.

The aim of this work is to find the most suitable algorithm from the analyzed localization and mapping methods by conducting experiments with different localization and mapping methods and parameters, so that the robot is able to carry out error-free navigation in dynamically changing environments and to transport loads.

The analytical part provides an overview of the state-of-the-art localization and mapping algorithms in use. Applications of *Hector SLAM*, *Google Cartographer*, *GMapping and Karto SLAM* are analyzed, as well as the accuracy of the results obtained in the literature.

In the methodological part, the design of the mobile platform is carried out using the *YDLIDAR X4* laser range sensor and odometry data obtained from the mobile platform encoders. The *Hector SLAM* algorithm is tuned to obtain the best parameter values for robot localization and mapping. Navigation experiments are also performed using the *move_base* module.

In the experimental part, the *Google Cartographer*, *Hector SLAM, GMapping* and *Karto SLAM* methods are applied and investigated. The accuracy of the localization and the map produced by each algorithm is considered, the performance is evaluated, and the most accurate algorithm for navigation is selected based on the experimental results.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpų ir terminų sąrašas

Santrumpos:

SLAM (angl. *Simultaneous Localization and Mapping)* – objekto lokalizacijos ir žemėlapio sudarymo sistema, sprendžianti navigacijos problemos uždavinius.

IMU (angl. *Inertial Measerument Unit*) – jutiklis, kuriuo galima išmatuoti kūno pasisukimą, tiesinį ir kampinį pagreitį (priklausomai nuo įrangos palaikymo).

ROS (angl. *Robot Operating System*) – robotų valdymo operacinė sistema, turinti gausybę bibliotekų, skirtų palengvinti robotų sąsają su skirtingais įrenginiais, kurie padeda įvykdyti tam tikrą užduotį.

UART (angl. *Universal asynchronous receiver-transmitter*) – sąsaja tarp kompiuterio ir periferinio įrenginio, paverčianti lygiagrečius duomenų bitus į nuoseklius.

I2C – nuoseklus komunikacijos protokolas palaikantis daugelį įrenginių vienu metu, kuriuo informacija yra perduodama bitais.

DAC (angl. *Digital-to-Analog Converter*) – skaitmeninis – analoginis keitiklis, paverčiantis skaitmeninį signalą į analoginį.

DRL (angl. *Deep Reinforcement Learning*) – mašininis mokymosi metodas, kuriuose siekiama apmokyti agentą kuo tiksliau pasiekti nustatytą tikslo funkciją, kuri apibrėžia siekiamą rezultatą.

DQN (angl. *Deep Q-Network*) – mašininis mokymosi metodas apjungiantis neuroninius tinklus ir Q giliojo mokymosi metodus (angl. *Q-learning*) siekiant aproksimuoti Q funkciją.

DDP (angl. *Distributed Data Parallelism*) – mašininis mokymosi metodas, skirtas apmokyti neuroninius tinklus daugelyje įrenginių vienu metu.

PPO (angl. *Proximal Policy Optimization*) – mokymosi su pastiprinimu gradientinis metodas, apmokantis agentą atlikti sudėtingas užduotis.

Terminai:

Lazerinis atstumo jutiklis (angl. *Light Detection And Ranging, LiDAR*) – jutiklis, kuriuo nustatomas atstumas iki šalia esančių objektų naudojant lazerį ir įvertinant laiką, per kurį atsispindėjusi šviesa sugrįžta į jutiklio imtuvą.

Uždaros kilpos aptikimas (angl. *Loop closure*) – lokalizacijos ir žemėlapio sudarymo metu naudojamo algoritmo galimybė atpažinti, kada robotas atsidūrė anksčiau buvusioje vietoje, siekiant sumažinti žemėlapio sudarymo netikslumus.

Kaštų žemėlapis (angl. *Costmap*) – roboto aplinkos interpretacinis pateikimas, kuriame kiekvienam laukeliui priskiriama kaštų vertė, priklausomai nuo užimtumo, atstumo iki kliūties ir kitų veiksnių.

Enkoderis (angl. *Encoder*) – mobiliuose robotuose įrenginys naudojamas įvertinti roboto poziciją ir greitį.

Odometrija (angl. *Odometry*) – robotikoje, tai roboto pozicijos pasikeitimo sekimas pasitelkiant jutiklių duomenis, kaip lazerinio atstumo jutiklio, ratų enkoderių ir/arba inercinio matavimo jutiklio.

Įvadas

Šiais laikais mobiliųjų robotų gebėjimas sudaryti kelią ir nukeliauti iki nustatyto taško yra viena iš pagrindinių optimalaus kelio radimo problemų. Tokio mobilaus roboto navigacijos tikslas yra surasti geriausią kelią iki nurodyto tikslo taško vengiant statinių ir dinaminių kliūčių. Navigacija dinamiškai kintančioje aplinkoje dažniausiai yra aktuali įvairiems sandėlių, namų bei aptarnavimo robotams, kurie privalo nuolatos stebėti aplinkos pokyčius bei prie jų prisitaikyti sprendžiant tinkamiausio kelio radimo problemą.

Pastaruoju metu mobiliųjų roboto navigacijos problemoms buvo skirta nemažai dėmesio dėl vis jų augančio patrauklumo automatizavimo srityse. Dažnai uždarose aplinkose nepakanka tradicinių priemonių, kaip roboto pozicijos nustatymo remiantis *GPS* dėl neaiškios vidinės aplinkos struktūros. Tokių problemų sprendimui vis dažniau pasitelkiami vienalaikės lokalizacijos ir kartografavimo metodai (angl. *Simultaneous Localization and Mapping, SLAM*), jog būtų įmanomą sudaryti uždaros aplinkos žemėlapį bei tiksliai išsiaiškinti aplinką ir joje esančias kliūtis. *SLAM* tipo algoritmais, dažniausiai neturint jokių pradinių duomenų apie aplinką, yra stengiamasi sudaryti aplinkos žemėlapį kartu nuolatos įvertinant roboto padėtį (lokalizaciją) aplinkoje. Toliau jau turint duomenis apie roboto poziciją ir aplinką yra sudaromas maršrutas iki nustatyto tikslo taško naudojant įvairius kelio planavimo metodus.

Darbo tikslas – sukurti originalios mobiliosios robotinės platformos navigacinę sistemą, kuri pasitelkiant roboto lokalizacijos metodais sudarytų žemėlapį uždaroje aplinkoje ir gebėtų pasiekti nurodytą tikslą vengiant kliūčių dinamiškai kintančioje aplinkoje.

Tikslo įgyvendinimui numatyti uždaviniai:

1. atlikti esamų robotų navigacijos po uždarą aplinką algoritmų analizę;
2. pritaikyti bei suderinti lokalizacijos ir žemėlapio sudarymo algoritmą mobiliai robotinei platformai;
3. įgyvendinti mobiliosios robotinės platformos autonominį judėjimą siekiant nustatyto tikslo ir vengiant aplinkoje esančių kliūčių;
4. atlikti lokalizacijos ir žemėlapio sudarymo tyrimus su suderintais *Google Cartographer*, *Hector SLAM*, *GMapping* ir *Karto SLAM* metodais;
5. atlikti lokalizacijos ir žemėlapio sudarymo algoritmų eksperimentų apibendrinimą.

Algoritmų analizei bus tiriami esami mobiliųjų robotų navigacijos po uždarą aplinką algoritmai bei palyginimas jų efektyvumas ir tikslumas pagal surinktus šaltinių rezultatus. Iš analizuotų algoritmų bus atrenkami problemos sprendimui tinkamiausi metodai bei pasirinktus algoritmus bus siekiama pritaikyti projekto realizacijai, patobulinant metodus siekiant išspręsti sudėtingesnius navigacijos po aplinką uždavinius, kaip dinaminių kliūčių įvertinimas bei atitinkamos elgsenos taikymas priklausomai nuo situacijos. Projekto tyrimas bus atliekamas palyginant *Google Cartographer*, *Hector SLAM*, *GMapping* ir *Karto SLAM* lokalizacijos bei žemėlapio sudarymo metodus atliekant bandymus skirtingais roboto judėjimo greičiais ir skirtingo dydžio aplinkose. Gauti eksperimentų rezultatai bus apibendrinami nustatant tiksliausią algoritmą mobiliai navigacinei sistemai.

1. Mobiliųjų robotų kelio planavimas

Roboto kelio planavimo užduotis apima tinkamiausio kelio nustatymo problema siekiant iš pradinės padėties pasiekti nustatytą tikslo tašką. Tam kad surasti geriausią maršrutą, prieš tai yra reikalinga pradinė informacija apie aplinką, kurioje atliekama navigacija. Žemiau yra pateikiami 3 pagrindiniai žingsniai, kurie itin svarbus kelio planavimo užduočiai išspręsti [1]:

- lokalizavimas,
- kelio planavimas,
- žemėlapio sudarymas bei interpretavimas.

Lokalizavimo etapas yra reikalingas tam, kad robotas galėtų žinoti savo tikslią poziciją aplinkoje kiekvienos pozicijos pakeitimo metu. Šioje dalyje robotas pasitelkdamas savo jutiklius nustato savo padėtį žemėlapyje, kurios tikslumas yra itin svarbus kelio planavimo žingsnyje. Kelio planavimo etape yra įvertinama informacija apie aplinką, jos kliūtys ir reljefą bei stengiamasi sudaryti tinkamiausią kelią iki nustatyto taško. Žemėlapio sudarymas ir interpretavimas yra reikalingas aplinkos, kurioje randasi robotas nustatymui bei užtikrinimui geriausio maršruto sudarymui, atsižvelgiant į objektus bei maršruto pravažumą. Egzistuoja skirtingi roboto lokalizavimo metodai, kurie gali būti skaidomi į šias kategorijas [1]:

- santykinė lokalizacija,
- absoliuti lokalizacija.

Santykinė lokalizacija yra pagrįsta roboto pozicijos nustatymu naudojant *IMU* bei nuvažiuotą ratų atstumą, kuris įvertinamas roboto judėjimo ir orientacijos keitimo pokyčio apskaičiavimu. Absoliučios lokalizacijos atveju roboto pozicijos nustatymui yra pasitelkiamos išorinės priemonės, tokios kaip GPS tikslios roboto padėties nustatymui aplinkoje [1].

Kelio planavimo būdas taip pat gali būti skirstomas į 2 tipus:

- vietinis,
- globalus.

Vietinis kelio planavimas apima savarankišką roboto judėjimo ir orientacijos sprendimų priėmimą ir valdymą, naudojant tokius jutiklius kaip ultragarsiniai, lazeriniai atstumo jutikliai, infraraudonųjų spindulių ir kameros. Vietinės navigacijos uždaviniams spręsti dažniausiai taikomi metodai yra neraiškioji logika, neuroniniai tinklai, genetiniai algoritmai, bei vienalaikės lokalizacijos ir kartografavimo algoritmai. Vietinis planavimo metodas yra ypatingai efektyvus, kai reikalingas spartus reagavimas į dinaminės aplinkos pokyčius, tokiu būdu sudarant naują maršrutą staiga atsiradus kliūčiai [1].

Globalus kelio planavimas yra tinkamas tada, kai aplinka, kurioje veikia robotas yra nekintanti, t. y. nėra judančių objektų, kurie galėtų trikdyti roboto navigaciją. Globali navigacija taip pat reikalauja išankstinių žinių apie aplinką. Globalaus planavimo metodo privalumas prieš vietinį yra tas, jog toks kelio planavimo tipas gali sudaryti pilną maršrutą nuo pradžios iki nustatyto taško, kadangi turima išankstinė informacija apie aplinką ir jos kliūtis, kur priešingai ją turi dinamiškai turi sudaryti vietinis planavimo metodas [1].

Į kelio planavimo metodus įeina ir roboto judesio planavimas, kuris nurodo kaip robotas turi judėti aplinkoje atsižvelgiant į jutiklių įvesties duomenis, kaip kliūtys, kurių turi vengti robotas. Judesio

planavime sukuriami tarpiniai taškai, kuriuos turi pasiekti robotas siekdamas nustatyto tikslo. Roboto judėjimą galima aprašyti naudojant tris skirtingas erdves [1]:

- dekarto erdvė – roboto padėtis ir orientaciją koordinačių sistemoje;
- sąnarių erdvė – roboto sąnarių arba variklių konfigūracija;
- pavaros erdvė – susijusi su valdymo signalais, siunčiamais pavaroms, pvz., varikliams ar servo pavarų varikliams, kad būtų sukurtas norimas judesys.

Kliūčių vengimo planavimas gali būti skirstomas į tris kategorijas, kurios veikia remiantis skirtingais kliūčių vengimo metodais [1]:

- reaktyvinis valdymas – apima tokius metodus kaip klajojimas aplinkoje, kliūčių apvažiavimas;
- reprezentacinis pasaulio modeliavimas – apima tokius metodus kaip „tikrumo“ tinkleliai aplinkos modeliui sukurti;
- reaktyvaus valdymo ir reprezentacinio pasaulio modeliavimo deriniai – vektoriaus lauko histograma.

Aukščiau paminėti metodai neužtikrina, jog visada bus rastas kelias, net jei ir atsižvelgiama į aplinkos kliūtys ir tokio kelio egzistavimą. Dažnai tam yra pasitelkiami aukštesnio lygmens navigacijos algoritmai, kaip uždaros kilpos įvertinamai tam, kad robotas nepasimestų aplinkoje. Uždaroje aplinkoje kliūčių vengimo metodų realizacija yra paprastesnė, nes tam nėra reikalingas papildomų veiksnių įvertinimas, kaip judančių objektų atpažinimas, kurių judėjimo greitis gali būti skirtingas.

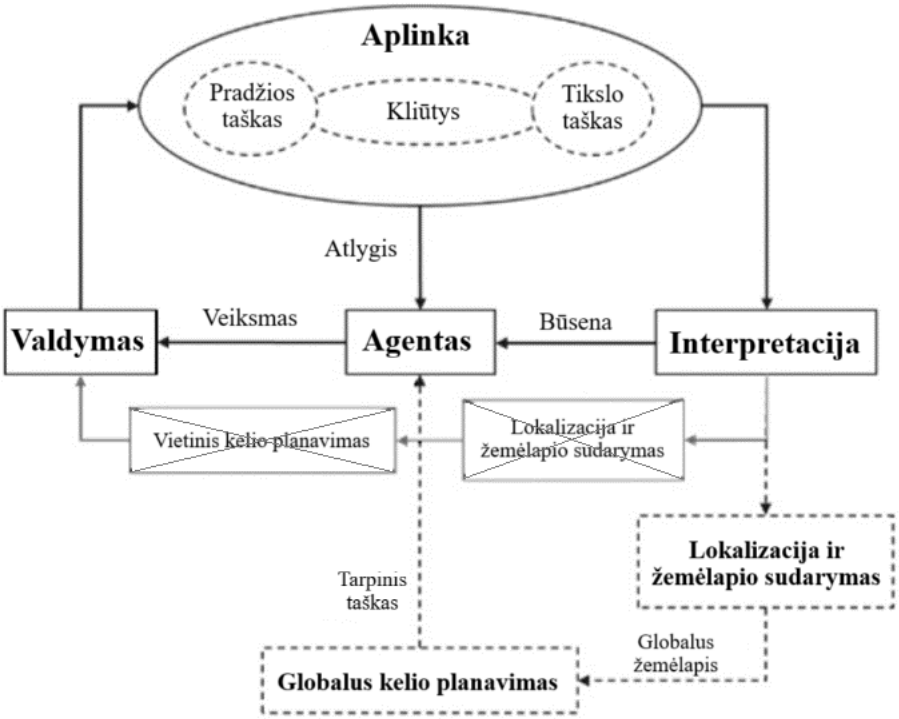
2. Mobiliųjų robotų navigacijos algoritmai

Pastaruoju metu buvo atrasta ir pasiūlyta nemažai robotų navigacijos algoritmų tiek statinėse, tiek dinaminėse aplinkose kliūčių vengimui ir kelio iki nurodyto tikslo suradimui. Kadangi tyrime numatytą spręsti navigacijos po uždarą aplinką problemą neturint jokių išankstinių žinių, tai analizės dalyje bus plačiau apžvelgiami vietinės navigacijos algoritmai.

2.1. Mokymosi su pastiprinimu algoritmai

Mokymosi su pastiprinimu (angl. *Deep Reinforcement Learning*) algoritmo taikymo autonominės navigacijos užduotyse tikslas – sudaryti geriausią maršrutą iki nustatyto tikslo, sąveikaujant su supančia aplinka. Įvairūs *DRL* algoritmai, kaip pavyzdžiui *DQN*, *DDP* ir *PPO* buvo išplėtoti siekiant sukurti *DRL* paremtas navigacijos sistemas. *DRL* metoduose navigacija atliekama pasitelkiant Markovo sprendimų procesą (angl. *Markov Decision Process*), kur iš jutiklių gauti duomenys yra panaudojami siekiant maksimizuoti tikėtiną veiksmų atlygį. *DRL* navigacijos privalumas yra tas, kad jai nereikia itin tikslių įvesties duomenų iš jutiklių dėl apsimokymo iš aplinkos galimybių bei išankstinio žemėlapio. *DRL* apsimokymas yra paremtas bandymų ir klaidų metodu. Tokio tipo apsimokymo metodams yra reikalingas gausus kiekis duomenų, jog navigacijos metu robotas galėtų tinkamai interpretuoti ir vengti kliūčių. Šiam tikslui *DRL* modeliai dažniausiai pirma yra apmokomi virtualioje aplinkoje prieš paleidžiant robotą į realų pasaulį, jog robotas gebėtų tiksliau priimti navigacinius sprendimus ir taip išvengti susidūrimų [2].

Daugelyje tyrimų buvo tiriama *DRL* pagrįsta navigacija. Tačiau *DRL* pagrįstos navigacijos skirstymas į kategorijas išlieka dviprasmiškas. Atvejuose kur yra naudojami roboto pozicijos nustatymo sprendimai pasitelkiant *GPS*, *DRL* pagrįsta navigacijos sistema gali nustatyti santykinę tikslo taško poziciją nesiremdama žemėlapio informacija. Tokio tipo metodai dar vadinami navigacija be žemėlapio. Kituose tyrimuose *DRL* metodas buvo taikomas apdorojant vietinių jutiklių duomenis, juos konvertuojant į vietinį žemėlapį. *DRL* metoduose, kuriuose atliekamas iš jutiklių gautų duomenų generavimas į žemėlapį dar vadinamas žemėlapiu pagrįstu metodu, kadangi tokio tipo algoritmuose nenaudojama absoliuti informacija, kaip *GPS* [2]. *DRL* pagrįstos navigacijos struktūros pavyzdys yra pateiktas 2.1 pav.



2.1 pav. *DRL* pagrįsta navigacijos sistema [2]

Paveiksle atsispindi, jog *DRL* agentas pakeičia lokalizaciją, žemėlapio sudarymo ir vietinio kelio planavimo žingsnius, kurie yra esminiai tradicinėje navigacijos sistemoje. Tačiau globalios informacijos pateikimas išlieka svarbi sistemos dalis ir *DRL* paremtose sistemose, nes vien vietiniai jutiklių duomenys neužtikrina, jog robotas neatsidurs uždarame cikle dėl aplinkos sudėtingumo.

DRL sistemą sudaro 3 pagrindinės dalys: būsenos erdvė (angl. *State Space*), veiksmo erdvė (angl. *Action Space*) ir atlygio funkcija (angl. *Reward Function*). Būsenos erdvė susideda iš pradžios ir tikslo taškų bei kliūčių, kur pradžios ir tikslo taškas, tai roboto pradinė pozicija ir tikslas atitinkamai.

Veiksmų erdvė apibūdina, kaip robotas turi judėti aplinkoje, jog pasiektų nustatytą tikslą. Judėjimą apima tiesinis roboto greitis, sukimosi greitis ir judėjimo veiksmai.

Atlygio funkcija yra svarbi robotui apsimokant, jog tinkamai pasiektų nustatytą tikslą. Priklausomai nuo roboto elgsenos atlygio funkcija yra padidinama, jei robotas pasiekia nustatytus taškus, išvengia kliūčių, o sumažinama atliekant priešingus prieš tai minėtus veiksmus – judant per lėtai arba visiškai sustojant ir taip nepasiekiant tikslo.

Vietinis kliūčių vengimas

Vietinių kliūčių vengimas yra pagrindinis *DRL* pagrįstų navigacijos sistemų taikymo atvejis, kuris taip pat gali būti patobulintas siekiant išspręsti sudėtingesnes navigacijos problemas. Vietiniam kliūčių vengimui įprastai taikomi reaktyvūs metodai. Pagrindinė reaktyvių metodų problema – reikalingi patikimi jutikliai, kurie gali tiksliai aproksimuoti vietinių kliūčių pozicijas. *DRL* metoduose netiesioginiam jutiklių duomenų apdorojimui yra naudojami neuroniniai tinklai, kuriuos tinkamai apmokant įmanoma išvengti tradicinių reaktyvių metodų kliūčių vengimo problemas.

Siekiant išspręsti didelio greičio susidūrimo problemas apmokymo metu buvo sukurtas neapibrėžtumo modeliu pagrįstas mokymosi algoritmas, kuris susidūrimo tikimybei įvertinti naudoja neuroninį tinklą. Algoritmas natūraliai pasirenka atsargiai elgtis nežinomoje aplinkoje ir padidina roboto greitį aplinkoje, kurioje jis yra labiau užtikrintas išvengti susidūrimo [3].

Daugelio robotų navigacija

Daugelio robotų navigacija papildo vietinių kliūčių vengimo problemą įvertinant judančius robotus aplinkoje. Tradicinius daugelio robotų navigacijos metodus galima suskirstyti į du tipus:

- centralizuotus,
- paskirstytus.

Centralizuotuose metoduose egzistuoja centrinis serveris, kuris naudojamas palengvinti duomenų dalinimąsi tarp robotų, kur kiekvienas robotas siunčia duomenis apie savo pozicijas ir atliekamus veiksmus, o serveris interpretuodamas kiekvieno roboto gautą informaciją atlieka optimizacijas ir suplanuoja tolimesnius robotų veiksmus. Paskirstytuose metoduose kiekvienas robotas individualiai priima sprendimus, tačiau tokio tipo metoduose kiekvienam robotui itin svarbu žinoti kitų robotų pozicijas ir judėjimo kryptis, jog išvengti susidūrimų. Dinamiškose aplinkose veikiant keliems robotams vienu metu tinkamai geba prisitaikyti *DRL* algoritmai paskirstytam kliūčių vengimui, kurie leidžia bendradarbiauti keliems robotams nekomunikaciniuose scenarijuose [4].

Socialinė navigacija

Socialinė navigacija susijusi su mobiliojo roboto judėjimu aplinkoje, kurioje yra pėsčiųjų. Šiame scenarijuje daugiausia dėmesio skiriama roboto gebėjimui judėti sudėtingoje aplinkoje esant gausiam žmonių kiekiui. Socialinė navigacija ir daugelio robotų navigacija turi panašumų, nes pėstieji gali būti traktuojami kaip nebendradarbiaujantys mobilieji robotai. Daugelį vietinių kliūčių vengimo ir paskirstytosios kelių robotų navigacijos metodų galima pritaikyti socialinės navigacijos scenarijams. Tačiau dėl didesnio dinaminių kliūčių tankio, didesni iššūkiai atsiranda tankiose aplinkose. Socialinius standartus atitinkančios navigacijos kiekybinis įvertinimas išlieka sudėtingas dėl nenuspėjamo žmonių elgesio pobūdžio, kas labiau didina atotrūkį tarp modeliuojamos ir realios aplinkos [5].

2.2. Genetiniai algoritmai

Genetiniai algoritmai įgijo savo populiarumą siekiant optimizuotų rezultatų. Genetinių algoritmų veikimo principas apima kandidatų į sprendimus populiacijos evoliuciją per natūralios atrankos procesą. Iš pradžių sukuriamas atsitiktinis galimų sprendinių rinkinys, vadinamas populiacija, skirtas sprendžiamai problemai spręsti. Tada kiekvieno sprendimo tinkamumo vertė įvertinama taikant problemos tikslo funkciją. Atrenkami tinkamiausi sprendimai ir jiems taikomos genetinės operacijos, tokios kaip mutacija ir kryžminimas, kurios rezultatas – nauja populiacija su patobulintais kandidatais. Toks ciklinis procesas tęsiamas tol, kol pasiekiamas didžiausias iteracijų skaičius arba gaunama patenkinama tinkamumo vertė. Genetiniai algoritmai plačiai taikomi, jog galėtų generuoti ir analizuoti gerai optimizuotus rezultatus įvairioms taikomosioms programoms [6].

Genetinis algoritmas pradedamas neturint jokių išankstinių žinių apie siekiamą rezultatą, o norimo tikslo generavimui naudojamas tik grįžtamasis ryšys iš aplinkos ir operatorių, kaip mutacija ir kryžminimas. Vienu metu inicijuodamas kelis kandidatų sprendimus ir vengdamas vietinių

minimumų, algoritmas konverguoja prie neoptimalių sprendimų. Šis metodas paprastai taikomas generuojant taškus po objekto aptikimo pasitelkiant regos sistema. Sugeneruotų kelių tinkamumas vertinamas pagal tokius veiksnius kaip apsauga nuo dinaminių kliūčių ir artumas iki tikslo. Genetiniai algoritmai plačiai taikomi planuojant kelius [6].

Dinamiškoje aplinkoje genetinis algoritmas naudoja vietinę chromosomų atmintį. Tačiau jam keliami dideli skaičiavimo reikalavimai, o jo veikimui įtakos turi dinaminės aplinkos konfigūracija. Pagrindinis genetinio algoritmo apribojimas kelio planavime yra ribotas jo pritaikomumas dinaminėse aplinkose. Kadangi genetiniai algoritmai priklauso nuo tinklinių žemėlapių arba fiksuotos paieškos srities skiriamosios gebos, taip pat dėl populiacijos įvairovės kontrolės trūkumo, dėl kurio įvyksta ankstyva konvergencija [6].

Pagrindinis genetinio algoritmo trūkumas yra jo priklausomybė nuo silpnai žinomų tinkamumo funkcijų, dėl to gali būti generuojami netinkami chromosomų blokai. Tiesiog kryžminant gerus chromosomų blokus tam tikros optimizavimo problemos nėra veiksmingai sprendžiamos.

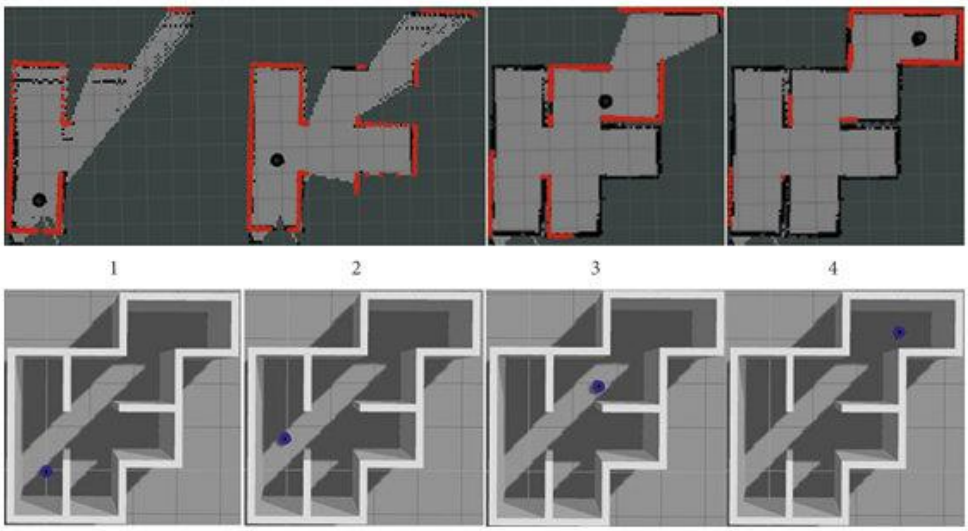
2.3. Vienalaikės lokalizacijos ir kartografavimo algoritmai

Tikslus padėties nustatymas ir žemėlapių sudarymas yra mobiliųjų robotų navigacijos pagrindas. Atvirose aplinkose robotai gali pasikliauti *GPS*, kad nustatytų savo absoliučią padėtį, o navigacijai naudoti žinomus kelių žemėlapius. Tačiau nežinomoje aplinkoje, kaip uždarose patalpose, gauti esamos aplinkos žemėlapį yra sudėtingas procesas, kadangi vien *GPS* negali užtikrinti patikimos vietos nustatymo. Šiai problemai spręsti naudojami lokalizacijos ir kartografavimo *SLAM* metodai. Pasitelkiant inercinių matavimo vienetų jutiklius, lazerinius atstumo jutiklius, kameras, robotas gali orientuotis nežinomoje aplinkoje. Taip pat jis gali nustatyti savo padėtį ir sudaryti aplinkos žemėlapį [7].

SLAM – tai mobiliųjų robotų naudojamas metodas, leidžiantis sudaryti aplinkos žemėlapį ir kartu nustatyti jo padėtį aplinkoje. Šis procesas apima du pagrindinius aspektus: kartografavimą ir lokalizavimą. Žemėlapio sudarymo metu robotas tyrinėja aplinką ir kuria kuo labiau ją atspindintį žemėlapį. Lokalizavimas apima sukurto žemėlapio naudojimą, siekiant nustatyti roboto padėtį joje ir suplanuoti kelią iki nustatyto taško [8].

Viena iš svarbiausių *SLAM* savybių – galimybė vienu metu atlikti kartografavimą ir lokalizavimą. Judėdamas mobilusis robotas nuolat atnaujina kuriamą žemėlapį, todėl galima realiu laiku įvertinti objektų padėtį ir roboto trajektoriją. Ši galimybė leidžia *SLAM* generuoti nuoseklius ir tikslius žemėlapius kartu nustatant roboto buvimo vietą. *SLAM* yra vienas veiksmingiausių mobiliųjų robotų metodų, leidžiančių robotams autonomiškai veikti nežinomoje aplinkoje neturint jokių išankstinių žinių apie aplinką [9].

2.2 pav. yra pateiktas *Hector SLAM* algoritmu palaipsniui sudaromas aplinkos žemėlapis [10]. Viršuje yra matomas sudarytas aplinkos žemėlapis (kaip robotas mato aplinką), o žemiau yra atvaizduojamas realios vietos vaizdas. Pilkas plotas žymi pravažiuojama dalį, juodais kontūrais žymimos sienos ir objektai, juodu apskritimu atvaizduojama dabartinė roboto padėtis, o raudoni taškai atspindi realiu laiku matomus objektus lazeriniu atstumo jutikliu.



2.2 pav. *Hector SLAM* metodu sudarytas aplinkos žemėlapis [10]

SLAM metodus sudaro trys pagrindinės dalys [9]:

- žemėlapių sudarymas,
- lokalizacija,
- navigacija.

Prieš pradėdamas tyrinėti nežinomą aplinką, robotas turi turėti išankstines žinias apie teritoriją. Žemėlapio sudarymas leidžia robotui sukurti aplinkos atvaizdą gautą iš jutiklių. Šie duomenys naudojami kuriant įvairių tipų žemėlapius, pavyzdžiui, topologinį, geometrinį, tinklinį arba šių atvaizdavimų derinį. Žemėlapis veikia kaip nuoroda, pagal kurią robotas nustato savo buvimo vietą ir atpažįsta aplink esančius objektus [9].

Lokalizacija yra esminė *SLAM* funkcija, leidžianti mobiliajam robotui pagal sukurtą žemėlapį apskaičiuoti ir įvertinti savo trajektoriją ir orientyrų padėtį. Lokalizacija apima roboto gebėjimą nustatyti savo buvimo vietą, suprasti jį supančią aplinką ir išvengti kliūčių naudojant informaciją, gautą iš žemėlapio sudarymo proceso [9].

Navigacija apima ir kartografavimo, ir lokalizavimo funkcijas, nes mobilusis robotas naudoja žemėlapio sudarymo ir lokalizavimo procesų metu gautą informaciją, kad suplanuotų kelią. Veikdamas aplinkoje, robotas nuolat atnaujina savo žinias apie aplinką. Roboto vykdomas navigacijos planavimas apima kelio planavimą pagal gautą informaciją, reagavimą į aplinkos pokyčius ir gebėjimą po tyrinėjimo grįžti į pradinį tašką [9].

Taikant *SLAM* reikia spręsti keletą svarbių uždavinių, įskaitant neapibrėžtumą, atitikimą, duomenų susiejimą ir laiko sudėtingumą. Neapibrėžtumas apima du svarbius *SLAM* iššūkius: vietos ir techninės įrangos neapibrėžtumą. Vietos neapibrėžtumas kelia sunkumų *SLAM,* nes nuo jo priklauso mobilaus roboto gebėjimas judėti daugeliu kelių aplinkoje. Judėti iš vieno taško į kitą tiesiu keliu ir grįžti į pradinį tašką yra gana paprasta, kai kelias yra aiškus ir lengvai atpažįstamas. Tačiau sudėtingoje aplinkoje egzistuoja daugybė kelių, todėl robotui sunku pasirinkti tinkamą kelią ir tiksliai nustatyti savo padėtį [11].

Įrangos neapibrėžtumas reiškia roboto aparatūros komponentų sukeliamus netikslumus. Pavyzdžiui, lazerinių atstumo jutiklių, išgaunamoje informacijoje esantys triukšmai gali lemti netikslius matavimus. Vėliau šie matavimai naudojami mobiliojo roboto padėčiai ir orientyrams įvertinti. Dėl atsirandančio įrangos triukšmo darosi sudėtinga atskirti naujus orientyrus nuo anksčiau pastebėtų, todėl kyla problemų įrangos bendradarbiavimo metu [11].

Atitiktis yra pagrindinė *SLAM* problema, ypač nustatant objektus. *SLAM* turi turėti galimybę atskirti atskirus orientyrus ir atpažinti jų unikalumą, palyginti su kitais identifikuotais orientyrais. Robotai yra itin priklausomi nuo techninės įrangos, kad galėtų suvokti ir įvertinti aplinką. Išgaunant informaciją iš roboto techninės įrangos, pavyzdžiui, lazerinio jutiklio, kyla sunkumų atpažįstant, ar naujai pastebėtas objektas skiriasi nuo anksčiau pastebėtų [12].

Duomenų susiejimo problemų kyla, kai mobilusis robotas, ištyręs aplinką, turi grįžti į savo pradinį tašką. Iššūkis yra susieti dabartinį orientyrą su anksčiau stebėtais orientyrais, kad būtų galima orientuotis atgal. Duomenų susiejimo procesai naudojami orientyrų atitikmenims įvertinti, kad robotas galėtų grįžti atgal pagal ankstesnį žemėlapį ir nustatytus orientyrus [12].

Laiko sudėtingumo klausimai susiję su greičiu, kuriuo įgyvendintas *SLAM* algoritmas ar metodai apdoroja ir apskaičiuoja gautą informaciją, kad mobilusis robotas gautų reikiamus rezultatus. Navigacijos metu *SLAM* kartografavimą ir lokalizavimą atlieka vienu metu. Norint kontroliuoti kelis vykdomus procesus per trumpą laiką, reikia veiksmingai juos valdyti. Todėl *SLAM* algoritmo našumas ir laiko sudėtingumas yra labai svarbūs siekiant gauti patikimus rezultatus, kad mobilusis robotas galėtų sėkmingai tyrinėti aplinką ir kartu sumažinti paklaidą [13].

2.3.1. Filtru paremtas *SLAM*

Filtru pagrįstas *SLAM* yra vienas seniausių ir labiausiai įsitvirtinusių algoritmų vienalaikės lokalizacijos ir kartografavimo srityje. Algoritmas veikia Bajeso įvertinimo principais, kai roboto sekanti padėtis iš pradžių įvertinama pagal gaunamą valdymo informaciją. Vėliau ši apskaičiuota padėtis tikslinama ir koreguojama naudojant stebėjimo informaciją. Šiuo metu pagrindiniai filtrais grindžiami *SLAM* algoritmai yra išplėstinis Kalmano filtro *SLAM* (*EKF-SLAM*) ir dalelių filtro SLAM (*PF-SLAM*). *SLAM* problema paverčiama būsenos vektoriaus įvertinimu naudojant išplėstinį Kalmano filtrą. Naudojant išplėstinį Kalmano filtrą galima vienu metu numatyti ir atnaujinti roboto padėtį ir aplinkos požymių žemėlapį, kad būtų galima juos įvertinti [14].

EKF-SLAM naudoja Kalmano filtro principus roboto prognozavimo ir stebėjimo lygtims nustatyti. Prognozavimo lygtis įvertina roboto padėtį ir orientaciją sekančiame veiksme, o stebėjimo lygtis koreguoja įvertintą padėtį ir požymių žemėlapį. Iš esmės *EKF-SLAM* sujungia šias lygtis, kad iteratyviai patikslintų roboto padėties įvertinimą ir atnaujintų stebimų požymių žemėlapį [14].

EKF-SLAM yra gerai išvystytas *SLAM* algoritmas, kuriam naudingas tvirtas tikimybių teorijos ir išplėstinio Kalmano filtravimo pagrindas. Jis plačiai suprantamas ir naudojamas. Tačiau išplėstinių Kalmano filtrų taikymas sprendžiant *SLAM* problemas turi tam tikrų apribojimų, kurie visų pirma kyla dėl filtre naudojamo matematinio modelio trūkumų. Pirma, kadangi lygtys yra netiesinės, jas reikia ištiesinti, o ištiesinant atsiranda būdingų paklaidų. Be to, išplėstiniame Kalmano filtre reikia saugoti būsenos vektoriaus klaidų kovariacijos sumos matricą. Robotui tyrinėjant aplinką, požymių taškų skaičius žemėlapyje palaipsniui didėja, todėl gerokai padidėja užimama atmintis. Tai gali kelti didelių sunkumų, nes sunaudojama nemažai kompiuterio atminties išteklių [14].

Siekiant išspręsti didėjančio skaičiavimo sudėtingumo problemą, su kuria susiduria robotai atlikdami didelės apimties tyrimus, *EKF-SLAM* algoritmą galima patobulinti integruojant jį su vietiniu žemėlapiu. Taikant šį metodą pagal stebimus požymių taškus sukuriami keli vietiniai žemėlapiai. Robotui toliau judant, požymių taškų skaičius kiekviename vietiniame žemėlapyje palaipsniui didėja. Kai požymių taškų skaičius viršija nustatytą ribą, vietinis žemėlapis sujungiamas su visuotiniu žemėlapiu. Vėliau inicijuojamas naujas vietinis žemėlapis. Šis metodas, palyginti su tradiciniu *EKF-SLAM*, yra labai naudingas, nes veiksmingai sumažina skaičiavimo sudėtingumą ir padidina algoritmo greitį [14].

2.3.2. Dalelių filtru paremtas *SLAM*

PF-SLAM algoritmas, kuriame naudojamas dalelių filtras, naudoja svertinių dalelių rinkinį, kad atvaizduotų įvertintos būsenos aposteriorinį tikimybės pasiskirstymą. Kiekviena filtro dalelė atspindi galimą roboto padėties trajektoriją ir turi susijusį aplinkos žemėlapį. Esant pakankamai dideliam dalelių skaičiui, dalelių filtras gali tiksliai atvaizduoti bet kokią tikimybės tankio funkciją. Naudojant valdymo ir stebėjimo informaciją, įvertinamos roboto padėtys, o vėliau, remiantis žinomomis roboto padėtimis, įvertinamas aplinkos žemėlapis. Šį algoritmą galima suskirstyti į keturis pagrindinius etapus: mėginių ėmimą, svorio apskaičiavimą, pakartotinį mėginių ėmimą ir žemėlapio įvertinimą. Pirmuose trijuose etapuose daugiausia dėmesio skiriama roboto padėčiai įvertinti taikant dalelių filtro metodą [15].

PF-SLAM privalumas yra tas, kad juo galima aproksimuoti bet kokį tikimybinį pasiskirstymą, nesiremiant linijiškumo prielaidomis ar Gauso triukšmu, todėl tai yra universalus sprendimas. Tačiau šis algoritmas susiduria su pastebimu trūkumu: didėjant dalelių skaičiui, didėja ir PF-SLAM skaičiavimo sudėtingumas, ir atvirkščiai, mažinant dalelių skaičių mažėja algoritmo tikslumas. Kitas iššūkis, kylantis perrinkimo etape, yra dalelių išsekimo problema [15].

Naudojant *RBPF-SLAM*, uždavinys sumažinti dalelių skaičių kyla dėl to, kad kiekviena dalelė turi atskirą aplinkos žemėlapį. Šiam uždaviniui spręsti įvedamas adaptyvus metodas. Iš pradžių siūlomas tikslus siūlomo pasiskirstymo apskaičiavimo metodas, kuriame atsižvelgiama ir į roboto judėjimą, ir į naujausius stebėjimus. Šis metodas gerokai sumažina neapibrėžtumą, susijusį su roboto pozicija per filtro prognozavimo etapą. Be to, dalelių išsekimo problemai sušvelninti siūlomas selektyvus pakartotinės atrankos metodas, kuriuo pateikiamas sprendimas, kaip išlaikyti pakankamą dalelių skaičių [15].

2.3.3. Grafais paremtas *SLAM* – *Karto SLAM* metodas

Šiuo metu grafais pagrįstas *SLAM* algoritmas yra plačiai naudojamas ir laikomas labiausiai paplitusiu metodu. Jį paprastai sudaro trys pagrindiniai etapai: priekinis tarpkadrinis įvertinimas, uždarojo ciklo aptikimas ir grafo optimizavimas.

Iš pradžių požymių taškų išskyrimo algoritmas naudojamas požymių taškams iš vaizdo išgauti ir šiems atitinkamiems taškams sukuriami deskriptoriai. Šie požymių taškai yra sukurti taip, kad būtų invariantiški vaizdo kampo ir roboto padėties pokyčiams. Tada, sutapatinus požymių taškų poras tarp vienas po kito einančių kadrų, įvertinama roboto laikysenos transformacija tarp šių kadrų, todėl galima apskaičiuoti roboto padėtį skirtingais laiko momentais. Šis procesas paprastai vadinamas vizualine odometrija. Laikui bėgant vizualinės odometrijos kaupiamoji paklaida palaipsniui didėja. Tačiau, kai robotas grįžta į anksčiau buvusią vietą, įsijungia uždaro ciklo aptikimas, kas leidžia

nustatyti ir ištaisyti susikaupusias klaidas. Po to galiniame grafo optimizavimo etape optimizuojamos roboto padėtys, kad būtų pasiektas visuotinis nuoseklumas. Gavus visuotinai nuoseklias roboto padėtis, kiekvieno kadro stebimas vertes galima transformuoti į visuotinę koordinačių sistemą ir taip gauti visuotinai nuoseklų žemėlapį. *SLAM* našumas didinamas tobulinant įvairius komponentus, pavyzdžiui, požymių taškų išskyrimo algoritmą, tarpkadrinio atitikimo algoritmą, uždarojo kontūro aptikimo algoritmą, grafo optimizavimo algoritmą [16].

Plačiai naudojamas grafu pagrįstas *SLAM* optimizavimo algoritmas yra *Karto SLAM*. Grafinis optimizavimas apima įprastinės optimizavimo problemos atvaizdavimą grafo pavidalu. Grafu pagrįstame SLAM roboto padėtis vaizduojama kaip mazgas arba viršūnė, o ryšiai tarp pozų sudaro grafo briaunas. Šie ryšiai gali apimti odometrijos matavimus tarp vienas po kito einančių laiko žingsnių arba padėties transformacijos matricas, apskaičiuotas naudojant regos metodus. Sukūrus grafą, roboto padėtis koreguojama taip, kad atitiktų šių briaunų nustatytus apribojimus. Grafo optimizavimo SLAM problemą galima suskirstyti į du pagrindinius uždavinius [17]:

- grafo konstravimas,
- grafo optimizavimas.

Grafo konstravimo etape roboto padėtis yra mazgai, o ryšiai tarp padėčių vaizduojami kaip briaunos. Šiame etape, dažnai vadinamame priekine dalimi, kaupiama jutiklių informacija grafui konstruoti.

Grafo optimizavimo etape, viršūnės, vaizduojančios roboto poziciją, koreguojamos taip, kad geriausiai atitiktų briaunų nustatytus apribojimus. Šiame žingsnyje daugiausia dėmesio skiriama grafo optimizavimui, koreguojant roboto padėtį, kad būtų sumažintos santykių tarp padėčių paklaidos.

Šiuo metu grafo optimizavimu grindžiamas *SLAM* nors ir yra plačiai taikomas, tačiau vis dar turi nemažai trūkumų. Kai aplinkoje trūksta tekstūros arba ją sudaro pavieniai elementai, dėl riboto požymių taškų skaičiaus sumažėja judėjimo tarp kadrų įvertinimo tikslumas. Be to, kadangi požymių taškų išskyrimo algoritmas parenkamas rankiniu būdu, dažnai kyla problemų dėl optimalaus algoritmo nustatymo. Kadangi sukurtame žemėlapyje trūksta semantinės informacijos, jis riboja robotų navigacijos galimybes. Tam tikru mastu giliojo mokymosi ir *SLAM* integracija padeda spręsti šiuos minėtus iššūkius [18, 19].

2.3.4. Skenavimo atitikimo metodai

Algoritmo veikimas yra pagrįstas atliekant ciklą per pradinį padėties įvertinimą, kurį pateikia odometrija, o po to suderinant *2D* lazerinio atstumo jutiklio skenavimo informaciją, kad būtų sukurtas tikslus ir skaičiavimo požiūriu efektyvus žemėlapis. Algoritmas remiasi nedideliu roboto pradinės ir tikrosios padėties neatitikimu, kad būtų pasiektas visuotinai optimalus atitikimas. Skenavimo atitikimo algoritmas naudoja tris pagrindinius metodus [15, 20]:

- požymio į požymį atitikimas (angl. *Feature-to-Feature matching*),
- taško į požymį atitikimas (angl. *Point-to-Feature matching*),
- taško į tašką atitikimas (angl. *Point-to-Point matching*).

Požymio į požymį atitikimo metodas per trumpiausią įmanomą laiką sumažina šimtus nuotolio taškų iki mažesnio funkcijų rinkinio. Jis gali naudoti linijų segmentus ir kampinius požymius patalpų atvaizdavimui. Be to, požymio atitikimas požymiui gali būti naudojamas miestų kartografavimui, kai

požymiams nustatyti naudojamas lazerinio atstumo jutiklio signalo atspindžio intensyvumas ir geometriniai primityvai [20].

Tais atvejais, kai požymiai pagal funkciją požymis į požymį nėra patikimai nustatomi ir juose yra neapibrėžtumo, naudojant funkciją taškas į požymį sumažinamas informacijos kiekis. Atliekant taško į požymį atitikimą pasiekiama pusiausvyra tarp greičio ir tikslumo. Paprastai jis apima nuskaitytų taškų atitikimą linijiniams požymiams arba linijinių ir abstrakčių požymių deriniui. Požymis vaizduojamas kaip Gauso skirstinys su vidurkiu ir dispersija, o kvadratinis narys apskaičiuojamas nagrinėjant nuskaitymo tašką, patenkantį į tinklelio vienetą [20].

Dažniausiai naudojami šie taško į tašką atitikimo metodai: *ICP* (angl. *Iterative Closest Point*), *IMRP* (angl. *Iterative Matching Range Point*) ir *IDC* (angl. *Iterative Dual Correspondence*). Šiais metodais daugiausia dėmesio skiriama taškų debesų suderinimui ciklišk ai tikslinant skirtingų skenavimų taškų atitikimą [20].

2.3.5. *Rao-Blackwellized* dalelių filtras

Rao-Blackwellized algoritmo dalelių filtro dalelių skaičiaus mažinimas yra pagrindinė problema siekiant gauti tikslų modelį. Norint pasiekti tenkinamą atvaizdavimo rezultatą paprastai reikia didelio dalelių skaičiaus, kas atitinkamai padidina skaičiavimo sudėtingumą ir resursų sąnaudas. Dažniausiai naudojamas dalelių filtro algoritmas yra atrankos ir svarbos perrinkimo (angl. *Sampling Importance Resampling, SIR*) filtras, kurį sudaro šie keturi žingsniai [21]:

- prognozavimas,
- koregavimas,
- pakartotinė atranka,
- žemėlapio įvertinimas.

Prognozavimo etape iš pradžių dalelių filtras, prognozuodamas būsenos perėjimo funkciją, sukuria didelį kiekį imčių. Šios imtys, vadinamos dalelėmis, sujungiamos su svoriais, siekiant aproksimuoti aposteriorinį tikimybės tankį [21].

Koregavimo etape gavus naujų stebėjimų, apskaičiuojami kiekvienos dalelės svarbos svoriai. Šie svoriai parodo tikimybę stebėti prognozuojamą pozą, atsižvelgiant į pirmąją dalelę. Kuo didesnė dalelės stebėjimo tikimybė, tuo didesnis jos svoris [21].

Pakartotinės atrankos žingsnyje atrinktos dalelės perskirstomos pagal jų svorius. Šis etapas labai svarbus dėl riboto dalelių skaičiaus ištisiniuose pasiskirstymuose. Kitame filtravimo etape pakartotinai atrinktų dalelių rinkinys naudojamas kaip įvestis į būsenos perėjimo lygtį, kad būtų gautos naujos prognozuojamos dalelės [21].

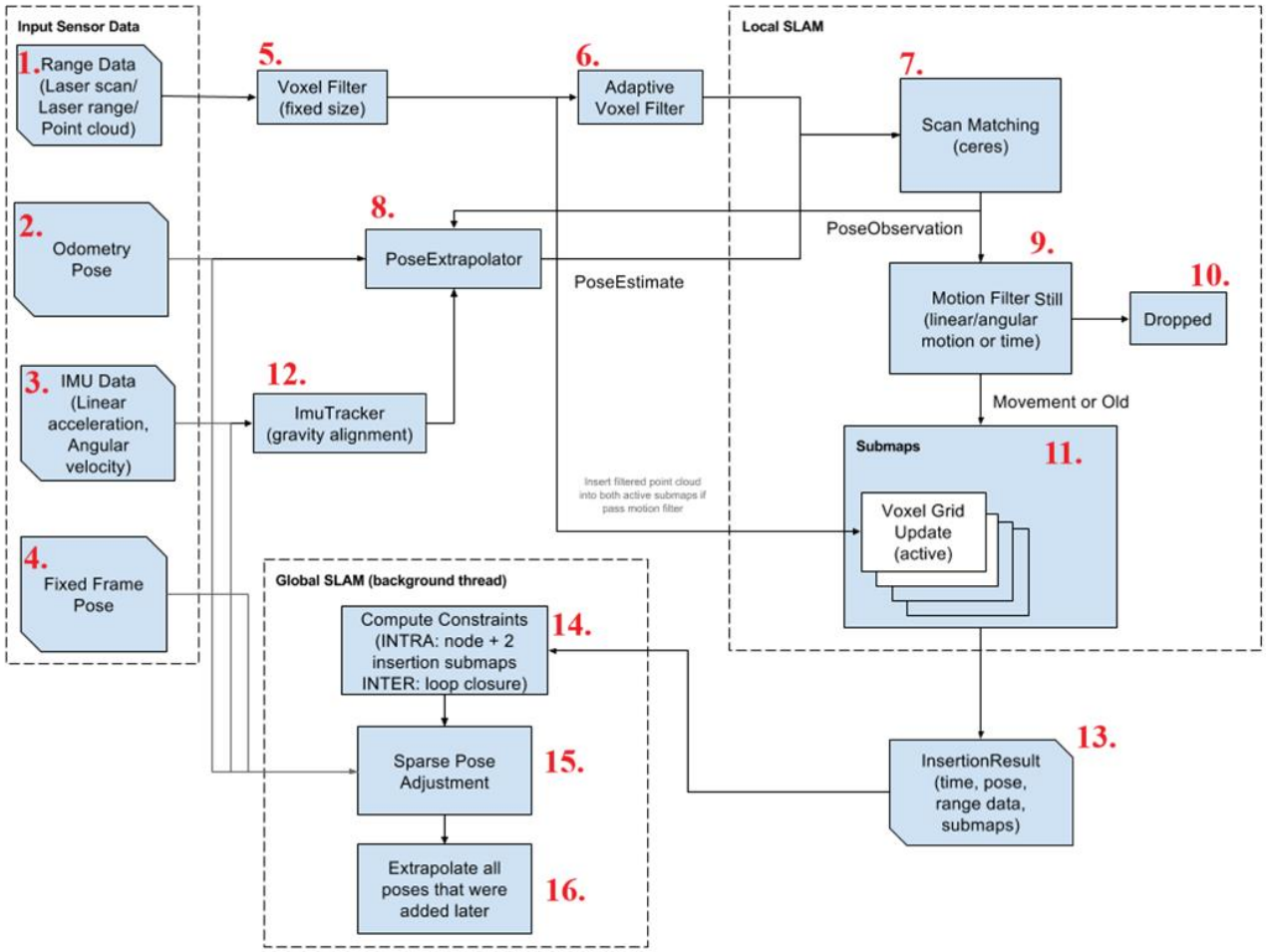
Žemėlapio įvertinimo metu kiekvienai atrinktai dalelei atitinkamas žemėlapio įvertis apskaičiuojamas remiantis trajektorija ir stebėjimais per atranką. Gavus naujų stebėjimų, SIR algoritmas turi iš naujo įvertinti dalelių svorius nuo pat pradžių. Laikui bėgant trajektorijos ilgis didėja, todėl šio proceso skaičiavimo sudėtingumas didėja. Svarbos tikimybės tankio funkcija apribojama, kad būtų galima išvesti rekursyvią svarbos svorių skaičiavimo formulę [21].

Stengiantis optimizuoti dalelių filtro algoritmą, siekiama rasti būdų, kaip sumažinti dalelių skaičių, kartu išlaikant tikslius atvaizdavimo rezultatus. Iššūkis – rasti pusiausvyrą tarp skaičiavimo efektyvumo ir atvaizdavimo veiksmingumo.

2.3.6. *Google Cartographer* metodas

Google Cartographer yra vienas iš *SLAM* metodų skirtų dvimatės arba trimatės aplinkos lokalizacijai ir žemėlapio sudarymui. Sistema palaiko lazerinių atstumo, inercinių matavimo vienetų, kamerų ir odometrijos jutiklių šaltinių kombinavimą siekiant išgauti kuo tikslesnę roboto poziciją bei realią aplinką atspindintį aplinkos žemėlapį.

Google Cartographer yra sudarytas iš 2 posistemių: globalaus ir vietinio *SLAM*. Vietinė *SLAM* posistemė yra atsakinga už žemėlapių fragmentų sudarymą, o globali posistemė apjungia sudarytas žemėlapių dalis, siekiant išvengti lokalizacijos paklaidos didėjimo, kurią sukuria vietinė posistemė, kai robotas keletą kartų aplanko tą pačią poziciją. Kad lokalizacijos paklaida būtų kuo mažesnė, taip pat yra naudojami papildomi minėti jutikliai ir informacijos šaltiniai, kaip *IMU* ir odometrija, kurie padeda sumažinti paklaidos didėjimą [22]. 2.3 pav. yra pateikiama *Google Cartographer* posistemių komunikacijos schema apibūdinanti sistemos procesų veikimą.



2.3 pav. *Google Cartographer* sistemos struktūra [22]

1. Lazerinio atstumo jutiklio duomenys (lazeriniai skenavimai arba taškų debesys).
2. Odometrijos duomenys.
3. *IMU* duomenys (tiesinis, kampinis pagreitis).
4. Fiksuotos konstrukcijos padėtis.

5. *Voxel* filtras (nustatyto dydžio), skirtas skenavimo duomenų sumažinimui išrenkant aktualiausius įrašus.
6. Adaptyvus *Voxel* filtras, bandantis nustatyti ir parinkti geriausią kiekį duomenų.
7. Skenavimo atitikimas (*Ceres* algoritmas).
8. Pozicijos ekstrapoliatorius, nuspėjantis, kokiam daliniam žemėlapiui priklausys skenavimo duomenys.
9. Judesio filtras (tiesinis, kampinis judesys arba laikas), užtikrinantis, jog judesio metu bus įtraukta tik svarbi informacija į dalinį žemėlapį.
10. Judesio metu atmesta mažiau aktuali informacija.
11. *Voxel* užimtumo žemėlapio atnaujinimas pagal gautų dalinių žemėlapių informaciją.
12. *IMU* sekimas (gravitacijos suderinimas).
13. Laiko, roboto padėties, lazerinio atstumo jutiklio skenavimo ir dalinių žemėlapių informacijos patalpinimas.
14. Apribojimų skaičiavimo procesas siekiant optimizuoti žemėlapio vidinį atvaizdavimą.
15. Įvertintų roboto pozicijų patikslinimas remiantis gautais stebėjimais iš jutiklių.
16. Visų vėliau pridėtų pozicijų ekstrapoliacija.

Pagrindinis informacijos šaltinis nuo kurio priklauso sistemos veikimas yra lazerinio atstumo jutiklių duomenys. Kadangi gauti duomenys iš jutiklių turi nuokrypius, tam yra reikalingas filtras, kuriuo pašalinami minimalių ir maksimalių verčių slenksčius peržengiantys duomenys. Taip pat priklausomai nuo lazerinio atstumo jutiklio pajėgumų, kai kurie jutikliai gali pateikti žymiai didesnį duomenų kiekį, todėl yra reikalingas duomenų normalizavimas, jog nesulėtinti sistemos veikimo ir kartu išlaikyti gaunamos informacijos vientisumą neprarandant svarbių duomenų. Po jutiklio duomenų filtravimo, vietinės posistemės duomenys yra paverčiami į pradinį žemėlapio fragmentą pasitelkiant skenavimo atitikmenų algoritmą. Gauta informacija toliau yra perduodama į judesio filtrą, kuris surenka tik svarbiausius duomenys, t. y. tuos skenavimo atitikmenis, kurie labiausiai skiriasi nuo praeito atlikto pasisukimo bei atstumo laike. Vietinės posistemės sudaryti žemėlapio fragmentai yra perduodami į globalią posistemę, kuri atlieka optimizacijas sudarydama vieną bendrą aplinkos žemėlapį [22].

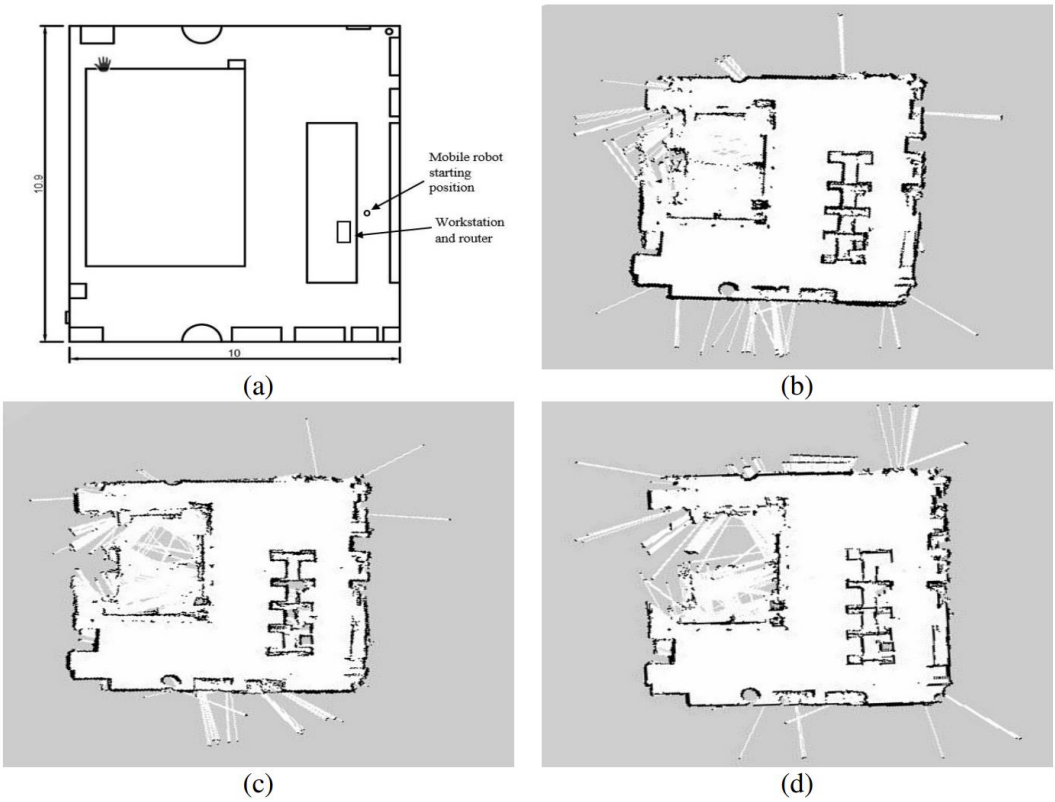
2.3.7. *GMapping* metodas

GMapping metodas yra paremtas dalelių filtro algoritmais. Jame taikomas *Rao-Blackwellized* dalelių filtro (*RBPF*) metodas, pagal kurį pirmiausia dėmesys sutelkiamas į padėties nustatymą, o tik po to į žemėlapio sudarymą. Tam yra naudojamas teorinis sprendimo metodas, kad nustatyti, ar reikalinga pakartotina duomenų imtis, jog sumažinti pakartotinio ėmimo skaičių ir informacijos praradimą. Pakartotinis ėmimas atliekamas tik tada, kai dalelių skaičius nukrenta žemiau tam tikros ribos, taip sumažinant imčių skaičių ir dalelių nykimą. Šis metodas gerokai sumažina svarbios dalelių informacijos atmetimo problemą [15].

GMapping algoritmas yra tinkamas tiek uždaros, tiek atviros aplinkos lokalizacijai ir žemėlapio sudarymui. Pritaikymas kartu su minėtu *RBPF* metodu leidžia pasiekti tikslių rezultatų nustatant roboto poziciją nežinomoje aplinkoje ir tuo pačiu atliekant aplinkos kartografavimą. *GMapping* yra itin priklausomas nuo odometrijos šaltinių tikslumo, kad sumažintų dalelių skaičių, todėl vien tik

lazerinis atstumo jutiklis nėra pakankamas. Metode taip pat nėra kilpos aptikimo (angl. *Loop Closure*), todėl gali nukentėti sudaromo žemėlapio kokybė. Ypatingai neidealiais reljefo atvejais atsiranda didesnė tikimybė gauti neteisingus sudaryto žemėlapio rezultatus, todėl sudėtingesnei aplinkai reikalingas didesnis dalelių skaičius, kas reikalauja didesnių skaičiavimo pajėgumų. *GMapping* metodas labiau yra tinkamas mažesnės aplinkos kartografavimui dėl anksčiau minėtų problemų [15].

GMapping algoritmo lokalizacijos ir žemėlapio sudarymo tikslumas buvo įvertintas tyrime [23]. Tyrime 3 skirtingais atvejais buvo sudaromos tos pačios uždaros aplinkos žemėlapis su skirtingu roboto judėjimo greičiu, žemėlapio atnaujinimo intervalu bei daleliu filtru. Sudaryto žemėlapio rezultatai pateikiami 2.4 pav. Raide (a) yra pažymėtas uždaros aplinkos planas, (b) – pirmo atvejo testavimas robotui judant 0,1 m/s greičiu, atnaujinant žemėlapį kas 5 sekundes. Atveju (c) pasirinktas roboto greitis – 0,133 m/s, o žemėlapio atnaujinimo intervalas – 1 sekundė. Atveju (d) robotas judėjo 0,161 m/s greičiu ir atnaujino aplinkos žemėlapį kas 0,1 sekundę. Tikslumas buvo apskaičiuotas padalinant sudarytos aplinkos žemėlapio ilgį iš realios aplinkos ilgio ir padauginant iš 100 %.



2.4 pav. *GMapping* algoritmo uždaros aplinko žemėlapio sudarymo tikslumo tyrimas [23]

Iš atlikto tyrimo geriausi rezultatai tiksliausiai atspindintys realios aplinkos planą gauti (b) atveju robotui judant lėčiausiai su 92,96 % žemėlapio atitikties tikslumu, tačiau robotas užtruko ilgiausiai sudaryti aplinkos žemėlapį – ~24 min. Testavimo atveju (b) gautas žemėlapio tikslumas truputį mažesnis nei (a) – 91,21 %, bet žemėlapis buvo sudarytas dvigubai greičiau – ~12 min. Atveju (c) tikslumas buvo 86,59 %, o pilnai užbaigti žemėlapio sudarymą užtruko ~ 6 min [23].

2.3.8. *Hector SLAM* metodas

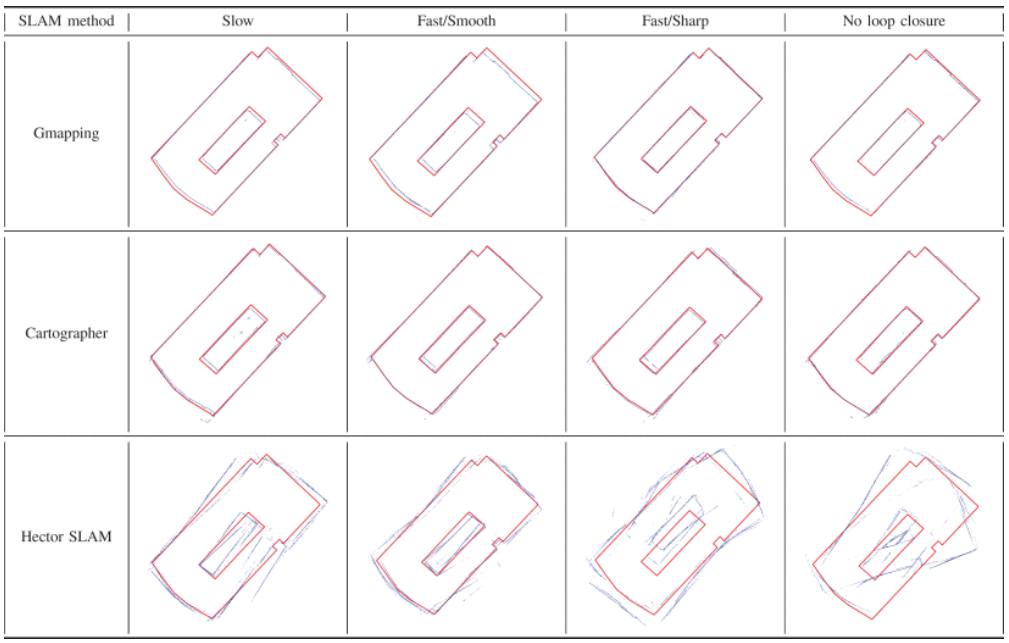
Hector SLAM yra vienas iš kelių skenavimo atitikties algoritmų. *Hector SLAM* algoritme naudojamas didelis *2D* lazerinio atstumo jutiklio mėginių ėmimo dažnis, jog realiu laiku būtų galima tiksliai

nustatyti roboto padėtį. Metodas neturi uždaros kilpos aptikimo, tačiau yra pakankamai tikslus daugeliui realaus pasaulio scenarijų.

Skenavimo atitikimo *SLAM*, naudojamas *Hector SLAM*, įvertina roboto poziciją ir pasukimą tarp dviejų skenavimų, remdamasis artimiausio kaimyno skenavimo atitikimu. *Hector SLAM* sukuriamas žemėlapis yra tapatus *GMapping* sukurtam žemėlapiui. Optimizuodamas lazerio spindulių tinklelį ir atvaizduodamas lazerio taškus žemėlapyje, algoritmas įvertina aplinkos užėmimo tikimybę. Skenavimo atitikimo uždavinys sprendžiamas taikant Gauso-Niutono metodą, kai lazerio taškų aibė atvaizduojama pagal esamo žemėlapio standžiojo kūno transformaciją (x, y, θ). Kad būtų išvengta vietinių minimumų ir pasiektas globalus optimalumas, naudojami kelių skiriamųjų gebų žemėlapiai [24].

Siekiant pagerinti būsenos įvertinimą navigacijos metu, *Hector SLAM* algoritme galima pridėti inercinę matavimo sistemą (*IMU*) naudojant Kalmano filtrą. Kai naudojamas *2D* lazerinis atstumo jutiklis su dideliu mėginių ėmimo dažniu, *Hector SLAM* ypač veiksmingas kartografuojant nedideles vidaus patalpas ir nereikalauja sudėtingos geografinės aplinkos. Kadangi algoritmas nepriklauso nuo odometrijos duomenų rinkimo, jis puikiai tinka robotams, kurių skaičiavimo galimybės ribotos [24].

2.5 pav. atvaizduojamas *GMapping, Google Cartographer* ir *Hector SLAM* žemėlapio sudarymo palyginimas robotui judant skirtingais greičiais, atliekant staigesnius posūkius bei kai metoduose yra išjungtas uždaros kilpos aptikimo funkcionalumas. Raudoni kontūrai žymi realios aplinkos sienas, o mėlyni – *SLAM* algoritmo nustatytas sienas. Paklaidos buvo apskaičiuotos palyginant sudaryto aplinkos žemėlapio kontūrus su tikrosios aplinkos. Atliktuose tyrimuose buvo nustatyta, kad *Google Cartogpraher* algoritmas tiksliausiai sudarė aplinkos žemėlapį, kai robotas judėjo lėtai, greitai su švelniais posūkiais ir be įjungtos uždaros kilpos aptikimo su 7,41 cm, 5,35 cm ir 4,97 cm paklaida atitinkamai. *GMapping* metodas buvo tikslesnis už *Google Cartographer*, kai robotas judėjo greitai ir atliko staigius posūkius su 3,21 cm paklaida. *Hector SLAM* paklaida visais atvejais buvo didžiausia iš kitų tirtų metodų ir visais atvejais buvo didesnė už kitus metodus beveik 4 kartus. Tiksliausius rezultatus *Hector SLAM* pasiekė robotui judant greičiau ir atliekant nestaigius posūkius su 19,36 cm paklaida [25].



2.5 pav. *GMapping, Google Cartographer* ir *Hector SLAM* algoritmų palyginimas [25]

2.3.9. Algoritmų palyginimas

2.1 lentelėje pateikiami populiariausių *ROS* sistemoje naudojamų *SLAM* algoritmų palyginimas. Algoritmų palyginimui buvo išskirti 7 kriterijai:

- reikalingas lazerinis atstumo jutiklis,
- reikalingas inercinis matavo jutiklis,
- reikalinga odometrija,
- uždaros kilpos aptikimas,
- skaičiavimo resursų sąnaudos,
- žemėlapio tikslumas,
- palaikomos dimensijos.

2.1 lentelė. Lokalizacijos ir žemėlapio sudarymo algoritmų palyginimas

Metodas / **Kriterijus**	***Hector SLAM***	***Google Cartographer***	***GMapping***	***Karto SLAM***
Reikalingas lazerinis atstumo jutiklis	+	+	+	+
Reikalingas inercinis matavo jutiklis	–	–	–	–
Reikalinga odometrija	–	–	+	+
Uždaros kilpos aptikimas	–	+	–	+
Skaičiavimo resursų sąnaudos	Mažos	Didelės	Vidutinės	Vidutinės
Žemėlapio tikslumas	Vidutinis	Labai didelis	Didelis	Didelis
Palaikomos dimensijos	2D	2D ir 3D	2D	2D

SLAM metodams lazerinis atstumo jutiklis yra neatsiejama dalis norint atlikti žemėlapio sudarymą. Nagrinėtuose metoduose lazerinis atstumo jutiklis naudojamas ne tik žemėlapio sudarymui, bet ir roboto lokalizacijai, nes vien odometrijos duomenų nepakanka tiksliai nustatyti roboto poziciją tuo atveju, kai robotas nuvažiuoja didelį atstumą. Odometrija neužtikrina, jog roboto nuvažiuotas kelias bus apskaičiuotas tiksliai ir be paklaidos, kuri laikui bėgant mažina lokalizacijos tikslumą. Lazerinis atstumo jutiklis padeda išspręsti šią problemą įvertinant anksčiau aptiktų objektų panašumus.

Inercinis matavimo jutiklis naudingas, kad robotas judėdamas dideliu greičiu ir atlikdamas posūkius kuo tiksliau galėtų įvertinti atliktą roboto pasisukimą. Vien tik lazerinio atstumo jutiklio ir odometrijos informacija nėra tiksli, kai atliekami didelio greičio posūkiai, kurie be inercinio jutiklio gali padidinti lokalizacijos paklaidą ir pakenkti sudaromo žemėlapio kokybei. Nors nei viename iš analizuotų metodų *IMU* nėra privalomas, tačiau visuose *SLAM* algoritmuose jutiklis prisideda prie roboto orientacijos nustatymo tikslumo.

Odometrija yra svarbi nustatyti roboto nuvažiuotą atstumą, kurio tiksliai negali įvertinti lazerinis atstumo jutiklis dėl nuolat generuojamo triukšmo. *Hector SLAM* ir *Google Cartographer* atveju odometrija nėra privaloma, kadangi jiems pakanka duomenų iš lazerinio atstumo jutiklio. Tačiau informacija naudojama tik iš lazerinio jutiklio dažnai nėra efektyvi didelėse teritorijose, nes nėra galimybės užtikrinti, kad robotas atsidūręs anksčiau buvusioje pozicijoje nepraras orientacijos ir nepradės traktuoti gaunamos informacijos kaip iš naujos vietovės dėl neatpažįstamų orientyrų arba jų trūkumo aplinkos unikalumui nustatyti.

Uždaros kilpos aptikimas, tai algoritmo gebėjimas nustatyti ar robotas anksčiau nesilankė toje pačioje vietovėje. Šis metodas padeda sumažinti lokalizacijos paklaidą ypač aplinkose, kuriose yra unikalių objektų pagal kuriuos robotas gali atpažinti savo poziciją. Uždaros kilpos aptikimu pasinaudoja *Google Cartographer* ir *Karto SLAM* metodai gebėdami tiksliau lokalizuoti robotą aplinkoje.

Skaičiavimo resursų sąnaudos buvo įvertintos iš tirtų šaltinių pagal kuriuos *Hector SLAM* algoritmas turi mažiausias sąnaudas dėl savo įgyvendinimo paprastumo. *GMapping* ir *Karto SLAM* reikalaujamas skaičiavimo išteklių kiekis yra didesnis nei *Hector SLAM*. O *Google Cartographer* yra vienas iš intensyvesnių metodų ypač *3D* žemėlapio sudarymo procese.

Žemėlapio tikslumas nustatytas pagal šaltiniuose pastebėtus rezultatus. Tyrimo rezultatuose [25] buvo nustatyta, kad tiksliausiai robotą lokalizuoja ir sudaro žemėlapį *GMapping* ir *Google Cartographer* algoritmai. *Hector SLAM* algoritmas net ir nedideliais greičiais bei lėtai atliekamais roboto posūkiais gaudavo rezultatus su beveik 4 kartus didesnėmis paklaidomis nei *GMapping* ir *Google Cartographer* metodai. *Google Cartographer* algoritmas tiksliausiai atspindėjo uždaros aplinkos žemėlapį mažesniais greičiais, o *GMapping* rezultatai buvo geresni už *Google Cartographer* didesniais greičiais ir staigesniais posūkiais. Šaltinyje [26] iš gautų eksperimentų su *GMapping*, *Hector SLAM* ir *Karto SLAM* algoritmais buvo nustatyta, kad geriausia žemėlapio kokybė gaunama su *Karto SLAM* metodų naudojant lazerinį atstumo jutiklį, *IMU* ir odometriją.

Iš dažniausiai taikomų *ROS* lokalizacijos ir žemėlapio sudarymo metodų, *Google Cartographer* turi galimybę atlikti lokalizaciją ir žemėlapio sudarymą trimatėje (*3D*) erdvėje, kur likę metodai yra pritaikyti dvimatės aplinkos kartografavimui ir roboto lokalizavimui.

3. Metodinė dalis

3.1. Naudojama įranga

Tyrimo metu naudota aparatūrinė įranga, skirta atlikti roboto lokalizacijos ir navigacijos užduotis:

- mobilioji robotinė platforma (*STM32F103RCT4*, važiuoklė),
- lazerinis atstumo jutiklis (*YDLIDAR X4*),
- *Arduino Mega 2560,*
- *Raspberry Pi 4,*
- 2x – *MCP4725* (*DAC*).

Naudojama mobilioji robotinė platforma (3.1 pav.), tai savadarbė mašina, kuri sukonstruota iš riedžio *Polaris* paliekant *STM32* valdiklį ir du ratus ant kurių statoma stačiakampė roboto konstrukcija/platforma su papildomais ratukais priekyje ir gale, skirtais išlaikyti platformos balansą. Mobiliosios robotinės platformos matmenys: 450x235x210 mm.

Mobiliosios platformos valdymas atliekamas naudojant *Arduino Mega 2560* valdiklį, į kurį prijungiami du *MCP4725* skaitmeninį į analoginį signalą konvertuojantys keitikliai. Tam naudojamas *I2C* protokolas, kur abu keitikliai yra adresuojami skirtingomis adresų vertėmis.

Kad *Arduino* valdiklis galėtų tinkamai valdyti ratų variklius, yra naudojamas *Raspberry Pi 4* valdiklis, kuriuo atliekami skaičiavimai roboto lokalizacijos ir navigacijos problemoms spręsti. Šiam tikslui įgyvendinti *Raspberry Pi* valdiklyje yra įdiegta *Ubuntu 20.04* operacinė sistema. Siekiant palengvinti roboto lokalizacijos ir navigacijos uždavinių atlikimą yra naudojama *ROS Noetic* programinė įranga, kuri suteikia galimybę pasinaudoti egzistuojančiomis SLAM bibliotekomis bei palengvina skirtingų įrenginių komunikacijos apjungimą: lazerinio atstumo jutiklio, *Arduino* valdiklio, tinklo konfigūraciją (norint valdyti mobiliąją platformą iš kito įrenginio egzistuojančio tame pačiame tinkle).

Prie *Raspberry Pi* valdiklio yra prijungiamas lazerinis atstumo jutiklis *YDLIDAR X4*. Lazerinio atstumo jutikliu yra atliekamas žemėlapio sudarymas. Lazerinis atstumo jutiklis yra *2D* tipo, todėl galimas tik dvimatės aplinkos kartografavimas, kurios pakanka uždaros aplinkos navigacijos uždaviniui išspręsti.

Lokalizacijai ir navigacijai įgyvendinti yra naudojamos ROS *hector_slam*[1] ir *move_base*[2] bibliotekos, kuriomis naudojantis yra išgaunama informacija siekiant valdyti roboto judėjimą, prieš tai pateikiant bibliotekoms jutiklių duomenys, iš kurių atliekami roboto lokalizacijos ir navigacijos uždaviniai.

Šiame eksperimente pasirinkto *Hector SLAM* algoritmo privalumas yra tas, kad jam nereikia jokios kitos odometrijos informacijos (nereikia ratų enkoderių informacijos, inercinio matavimo jutiklio, skirto pasisukimo krypčiai ir pagreičiui nustatyti) ir pakanka gautų duomenų iš lazerinio jutiklio. *Hector SLAM* algoritmas iš turimų lazerinio jutiklio taškų sudaro aplinkos žemėlapį ir atlieka roboto lokalizaciją įvertindamas taškų atitikmenis. Tačiau tokia metodika turi savo trūkumų kaip apribotas roboto judėjimo greitis, dėl kurio sudėtinga gauti tikslų aplinkos žemėlapį ir roboto poziciją.

Minėtos naudojamos įrangos specifikacija yra pateikiama 3.1 lentelėje.

[1] Daugiau informacijos: https://wiki.ros.org/hector_slam

[2] Daugiau informacijos: https://wiki.ros.org/move_base

3.1 lentelė. Naudojamos įrangos parametrai

Komponentas	**Parametrai**
Raspberry Pi 4	Procesorius: 4 branduolių *Cortex-A72 (ARM v8)* 64x 1.8GHz; Operatyvioji atmintis: 4GB; Tinklas: 2,4 GHz ir 5 GHz 802.11ac bevielis ryšys, *Bluetooth 5.0*; Prievadai: 2 – USB 3.0, 2 - USB 2.0, 2 – *micro-HDMI* Atmintis: *Micro-SD* kortelė
Arduino Mega 2560	Procesorius: *ATmega2560*, 16 MHz; Atmintis: 256 KB; Skaitmeninės įvestys/išvestys: 54 (15 iš jų *PWM*); Analoginės įvestys: 16
YDLIDAR X4	Veikimo atstumas: 0,12 – 10 m; Sisteminė paklaida: 2 cm; Variklio dažnis: 6 – 12 Hz; Stebėjimo kampas: 0 – 360 laipsnių; Įtampa: 4,8 – 5,2 V
Ekranas	Įstrižainė: 10,1 colių; Rezoliucija: 1024×600 Jungtis: *HDMI* Ekrano tipas: liečiamas ekranas *LCD (H)*
MCP4725	Įtampa: 2,7 – 5,5 V; Jungtys: • *OUT* – analoginė išvestis; • *SDA* – *I2C* magistralės duomenų linija; • *SCL* – *I2C* magistralės laikmačio linija; • *VCC* – maitinimo įtampa; • *GND* – įžeminimas

3.1 pav. Mobili robotinė platforma su lazeriniu atstumo jutikliu ir ekranu valdymui

3.2. Projektavimas ir realizacija

3.2 pav. yra matoma bendra mobiliosios robotinės platformos, valdiklių ir jutiklių sujungimo schema. Schemoje atvaizduojamos naudojamos sąsajos tarp valdiklių bei komunikacija su prijungtais įrenginiais. Kaip minėta anksčiau, robotinei platformai valdyti naudojamas ROS programinis paketas, kuris suteikia plačias galimybes atlikti skirtingas su robotų projektavimu susijusias užduotis. Ši programinė įranga yra diegiama į *Raspberry Pi 4* valdiklį su įrašyta *Ubuntu 20.04* operacine sistema.

Siekiant gauti duomenų apie esamą aplinką, prie *Raspberry Pi 4* valdiklio yra prijungiamas lazerinis atstumo jutiklis *YDLIDAR X4* per *UART* sąsają. Lazerinis atstumo jutiklis bus naudojamas *SLAM* metodam realizuoti, nes pateikiama informacija iš jutiklio gali būti panaudota ne tik žemėlapiui sudaryti, bet ir lokalizuoti robotą iš lazerinio atstumo jutiklio duomenų apie aplinką.

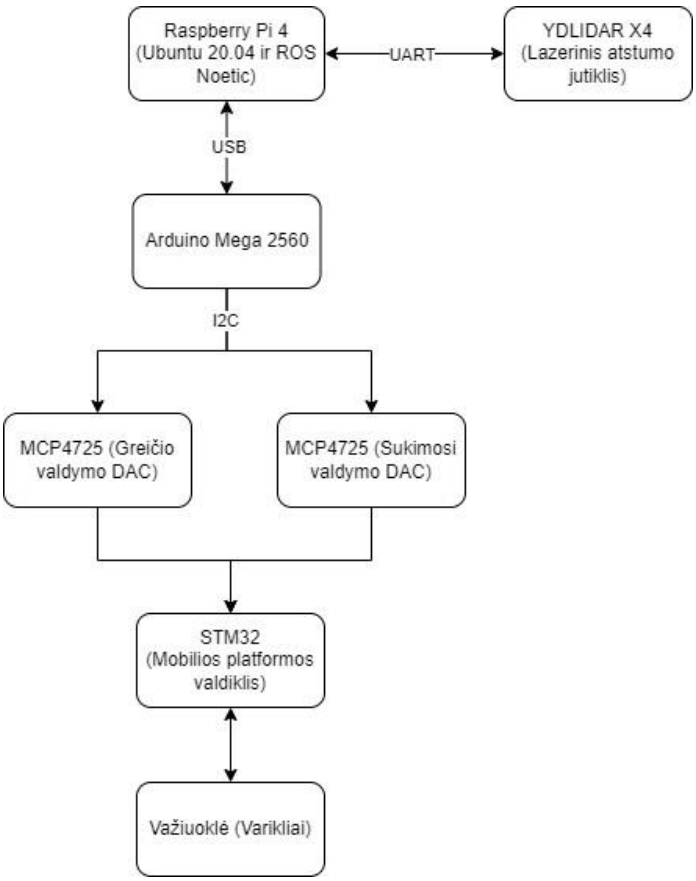
Papildomai prie *Raspberry Pi* yra prijungiamas *Arduino* valdiklis, kuris atsakingas už nustatytų signalo verčių išsiuntimą atitinkamai į abu *MCP4725* skaitmeninio į analoginio signalo keitiklius. Keitikliais perduodamas signalas į *STM32* valdiklį, prie kurio tiesiogiai yra prijungti varikliai, skirti ratų sukimosi greičiui valdyti. Vienas keitiklis yra naudojamas roboto sukimosi krypčiai keisti, o kitas – greičiui reguliuoti. *MCP4725* keitikliai priima įtampas nuo 0–5V. Per *Arduino* valdiklį yra paduodama tam tikra įtampa, kur, priklausomai nuo vertės, yra atliekamas roboto valdymas. Derinimo metu nustatytos reikšmės skirtos pasukti robotą į kairę arba dešinę, važiuoti tiesiai arba atbuline eiga. Vertėms nustatyti buvo naudojamas potenciometras, tokiu būdu atrandant ratų variklių judėjimo kryptis ir reikalingas įtampas. Toliau pateikiama 3.2 lentelė su nustatytomis vertėmis (žr. 3.2 lentelę).

3.2 lentelė. Derinimo metu nustatytos keitiklių įtampų vertės mobilios platformos judėjimo veiksmams atlikti

Veiksmas	**Sukimosi signalo keitiklis (0x60)**	**Greičio signalo keitiklis (0x61)**
Važiuoti tiesiai	2,3–2,5V	2,1–2,35V
Važiuoti atgal	2,5V	1V
Sukti kairėn	1–3,5V	1,8–2V
Sukti dešinėn	1,4–4V	1,8–2V
Sustoti	1,1V	1,1V

MCP4725 išėjimo signalų vertės yra perduodamos į važiuoklės *STM32* valdiklį, kuris šias gautas analogines vertes perduoda į ratų variklius. Priklausomai nuo įtampos signalų vertės yra atliekamas mobiliosios platformos pasisukimas arba važiavimas tiesia ar atbuline eiga.

Kad *Raspberry* ir *Arduino* valdikliai galėtų tarpusavyje komunikuoti, t. y. keistis vienas su kitu informacija yra naudojama *ROS* integruota skelbimo (angl. *Publisher*) ir prenumeravimo (angl. *Subscriber*) sistema. Ši komunikacijos sistema leidžia nesudėtingai susikonfigūruoti atskirus mazgus bei taip dalintis reikiama informacija, kuri gali būti pasiekiama iš vieno ir kito valdiklio užtikrinant duomenų apsikeitimą.



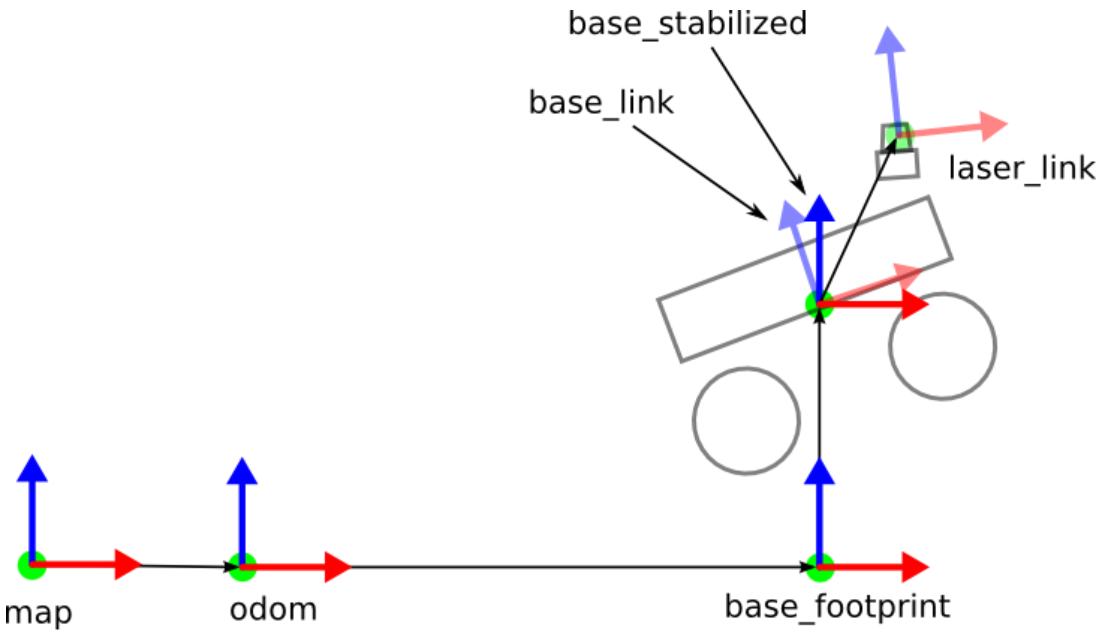
3.2 pav. Robotinės platformos valdiklių ir jutiklių sujungimo schema

Siekiant atlikti uždaros aplinkos kartografavimą ir nustatyti roboto poziciją joje reikalingi algoritmai, kurie sugeba lazerinio atstumo jutiklio gaunamus duomenis panaudoti žemėlapio kurti ir robotui lokalizuoti. *ROS Noetic* programinė įranga suteikia šias galimybes su gausybe bibliotekų, skirtų lokalizavimo ir navigacijos problemoms spręsti.

Tam, kad tinkamai susikonfigūruoti naudojamos bibliotekos *hector_slam* paketą, svarbu nustatyti teisingas koordinačių plokštumos transformacijas, kurios atvaizduojamos 3.3 pav. Paveiksle pateikiamos roboto transformacijos dvimatėje erdvėje, kurios bus naudojamos roboto atskaitos taškui ir dabartinei pozicijai nustatyti nežinomoje aplinkoje [27, 28]:

- ***map*** **–** tai fiksuota koordinačių plokštuma, kur *z* ašis yra nukreipta į viršų. Plokštuma nėra tęstinė, todėl roboto padėtis gali keistis tam tikrais diskrečiais šuoliais bet kuriuo laiko momentu. Šioje plokštumoje roboto pozicija yra nuolatos perskaičiuojama priklausomai nuo naudojamų jutiklių informacijos – šiuo atveju remiantis lazerinio atstumo jutiklio duomenimis.
- ***odom*** **–** koordinačių ašyje ši plokštuma priklauso nuo *map* plokštumos. Ši transformacija nusako roboto padėtį *map* plokštumoje. Robotui judant, *odom* ašis taip pat kinta, tačiau kitimas vyksta be diskretinių šuolių, o pastoviai. Ašis *odom* yra apskaičiuojama iš odometrijos šaltinių, kaip ratų enkoderių ir inercinių matavimo jutiklių.
- ***base_footprint*** **–** apibūdina roboto pradinę padėtį dvimatėje erdvėje, pateikiant jo poziciją ir orientaciją.

- ***base_link*** – koordinačių ašis, kuri yra susieta su roboto važiuoklės konstrukcija. Ji naudojama kaip atskaitos taškas kitoms koordinačių ašims. Ši ašis nusako išilginį ir šoninį roboto pasisukimą erdvėje.
- ***base_stabilized*** – įtraukiamas roboto aukščio parametras priklausomai nuo *map* ir *odom* koordinačių plokštumų. Tyrime ši ašis nebus naudojama bei bus sulyginta su *base_link* ašimi, nes aukštis yra nekintantis parametras ir informacija apie dvimatę erdvę yra pakankama navigacijos problemai išspręsti.
- ***laser_link*** – tai lazerinio atstumo jutiklio pozicija koordinačių sistemoje, kuri pateikiama susikonfigūravus ir paleidus lazerinio atstumo jutiklį.



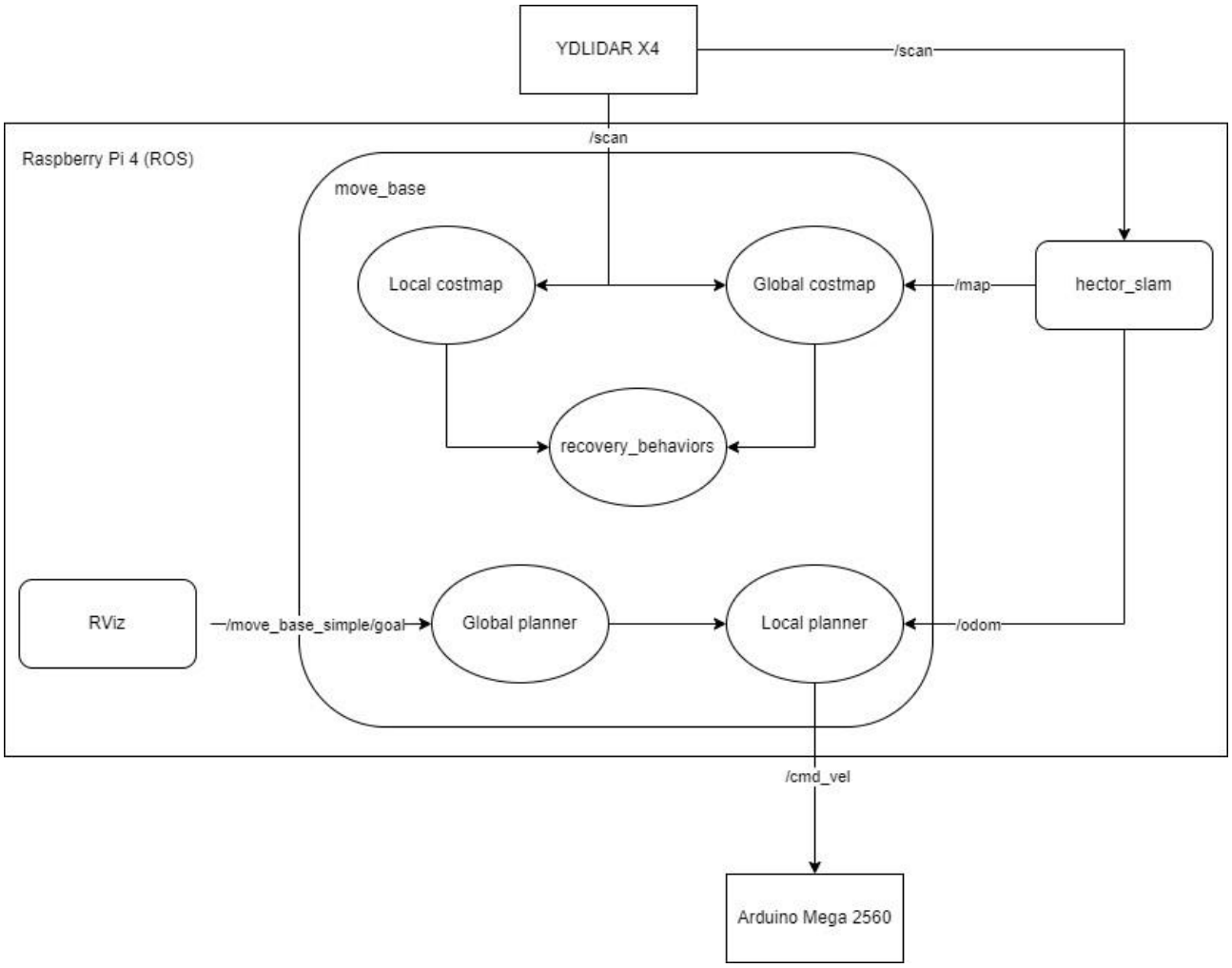
3.3 pav. Roboto koordinačių ašių plokštumos [28]

Visų pirma, jog būtų galima tikslingai panaudoti lazerinio atstumo jutiklio duomenis reikia tvarkyklės galinčios šiuos duomenis paversti į standartinę struktūrą, kurią pagal bendrai priimtą standartą geba interpretuoti *ROS* bibliotekos. Tam yra įsirašoma *ydlidar_ros_driver*[3] tvarkyklė, kurią susikonfigūravus pagal turimo lazerinio atstumo jutiklio parametrus ir susikūrus ROS konfigūracinį failą bei atlikus jo paleidimą, startuojamas lazerinis atstumo jutiklis ir gaunami apdoroti jo duomenys, kurie gali būti pasiekiami iš kitų *ROS* servisų [29].

Turint duomenis reikalingas algoritmas, galintis juos panaudoti žemėlapiui sudaryti ir robotui lokalizuoti. Šiai problemai spręsti pasirinkta *hector_slam* biblioteka. Naudojamam algoritmui nereikalingi jokie papildomi duomenys ir pakanka gaunamos informacijos iš lazerinio atstumo jutiklio. *Hector SLAM* veikimo principas yra pagrįstas pagal surinktų skenavimo taškų atitikimo metodiką, kur anksčiau gauti taškai yra lyginami su ankstesniais ir pagal tai įvertinimas roboto pozicijos pasikeitimas. Siekiant užtikrinti didesnį algoritmo tikslumą ir sumažinti paklaida yra naudojamas uždaros kilpos nustatymo metodas, kuriuo stengiamasi pašalinti įvertintos pozicijos nuokrypį per tam tikrą laiką [30, 31].

[3] Daugiau informacijos: https://github.com/YDLIDAR/ydlidar_ros_driver

Roboto navigacijai aplinkoje bus integruojama *move_base* biblioteka. Paketas *move_base* suteikia galimybę atlikti navigaciją jau iš turimų *hector_slam* bibliotekos algoritmo duomenų, nes pats vienas *move_base* servisas negali funkcionuoti dėl trūkstamos informacijos apie aplinką. Toliau pateikiama schema, kurioje atvaizduojama sąveika tarp naudojamų bibliotekų, įrenginių ir jų pateikiamos informacijos, reikalingos suplanuoti kelią iki nustatyto tikslo taško žemėlapyje (3.4 pav.).



3.4 pav. Komunikacijos tarp įrenginių ir *ROS* bibliotekų schema

Biblioteka *move_base* naudoja tiek vietinius duomenis apie aplinką, kurie nuolatos yra pateikiami ir atnaujinami iš jutiklių, siekiant, kad robotas tiksliau gebėtų judėti aplinkoje įvertinant dinaminius pokyčius, tiek globalius duomenis, kaip aplinkos žemėlapis, kuris sudaromas tik kartą.

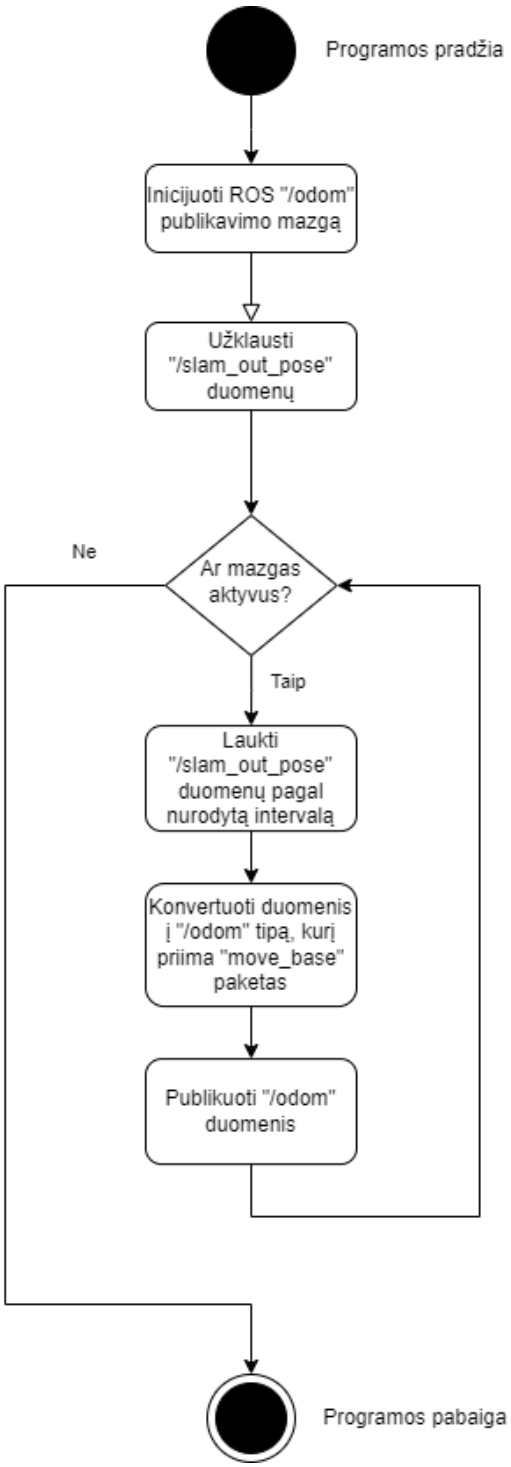
Navigacijos sistemai yra reikalingi sudaryto žemėlapio */map* ir dabartinės roboto pozicijos */odom* duomenys. Visa ši informacija jau yra gaunama iš *hector_slam* bibliotekos, kuri iš lazerinio atstumo jutiklio *YDLIDAR X4* sudaro aplinkos žemėlapį ir nustato roboto padėtį. Tikslo taškui nustatyti yra naudojama *RViz* programa, suteikianti grafinę sąsaja naudotojui ir leidžianti peržiūrėti sudarytą aplinkos žemėlapį */map*, dabartinę roboto poziciją */odom* ir nustatyti tikslo tašką */move_base_simple/goal*, į kurį turės nuvažiuoti robotas. Visa minėta gaunama informacija yra apdorojama *move_base* bibliotekos.

Apie aplinkos kliūtis ir jų pravažumą informacija yra talpinama vietinio (angl. *Local costmap*) ir globalaus (angl. *Global costmap*) kaštų žemėlapiuose. Globalus aplinkos žemėlapio įvertinimas yra

naudojamas ilgalaikiam aplinkos planavimui, t. y. turima informacija apie aplinkos išdėstymą yra naudojama globaliam keliui iki tikslo suplanuoti. O vietinis aplinkos kaštų žemėlapis yra naudojamas kliūtims išvengti, kurios gali atsirasti vėliau po aplinkos žemėlapio sudarymo etapo. Abu kaštų žemėlapiai yra konfigūruojami failuose nurodant parametrus, kurie apima roboto dimensijas, maksimalų atstumą iki kliūties, kurios turi vengti robotas, žemėlapio informaciją, dabartinę roboto poziciją ir kitus parametrus [32, 33].

Vietinis planavimo modulis (angl. *Local planner*) yra atsakingas už roboto greičio ir pasisukimo nustatymą planuojant kelią iki nustatyto tikslo. Tam atitinkamai egzistuoja konfigūraciniai nustatymai, kurie nurodomi pagal roboto techninius parametrus. Parametruose nurodomas maksimalus ir minimalus roboto greitis bei pagreitis, pasisukimo kampas. Vietinio planavimo modulio duomenys yra pasiekiami per */cmd_vel* kreipinį, kurį naudoja *Arduino* valdiklis ir pagal gautas komandas atlieka judėjimo veiksmus.

Kad duomenų apsikeitimas tarp *hector_slam* ir *move_base* paketų veiktų tinkamai, reikalingas *hector_slam* bibliotekos duomenų */slam_out_pose* konvertavimas į *move_base* priimamus */odom* duomenis. Kadangi formatai tarp šių dviejų duomenų struktūrų yra skirtingi, tačiau pateikiami duomenys yra analogiški, pakanka duomenų struktūros formato pakeitimo, kuris būti tinkamai interpretuojamas *move_base* modulio. Minėto duomenų konvertavimo veiklos diagrama atvaizduojama 3.5 pav.

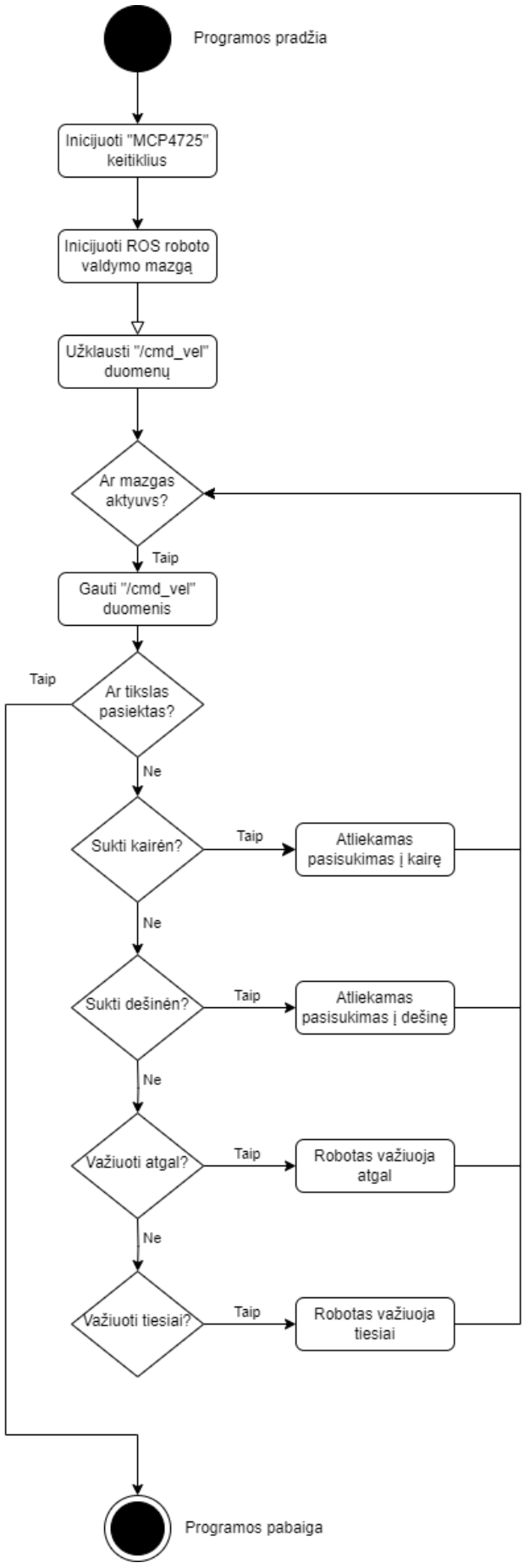


3.5 pav. Bibliotekos *hector_slam* duomenų struktūros konvertavimas į *move_base* interpretuojama struktūrą

Iš pateiktos diagramos yra inicijuojamas naujas *ROS* mazgas, kuriame nuolatos atliekama duomenų konversija *hector_slam* modulio paleidimo metu. Mazgas prenumeruoja *hector_slam* skelbiamus */slam_out_pose* taip nuskaitant gaunamos struktūros duomenis. Kiekvieną kartą yra patikrinama ar pradėtas mazgas yra vis dar aktyvus ir laukiami */slam_out_pose* duomenys kas nustatytą intervalą. Toliau šie duomenys yra paverčiami į */odom* tipą ir atitinkamai skelbiami šio mazgo, kad būtų pasiekiami kitų *ROS* mazgų, šiuo atveju reikalingi *move_base* moduliui. Šis ciklas nuolatos kartojamas iki kol mazgas yra aktyvus. Programinis kodas pateikiamas 1 priede.

3.6 pav. žemiau pateikiama *Arduino Mega 2560* valdiklio mobiliosios platformos valdymo diagrama, kurioje atliekamas roboto variklių sukimosi krypties ir greičio keitimas naudojant du *MCP4725* skaitmeninį į analoginį signalą konvertuojančius keitiklius. *Arduino* valdiklis naudoja */cmd_vel* duomenis, kuriuos grąžina *Raspberry Pi 4 move_base* modulis. Kaip minėtą anksčiau, */cmd_vel* gaunamame pranešime yra apibrėžta, kaip robotas turi elgtis, kad pasiektų nurodytą tikslo tašką. Gaunama duomenų struktūra yra sudaryta iš tiesinio greičio ir pasisukimo kampo verčių.

Programos startavimo metu pirmiausia yra inicializuojami *MCP4725* moduliai. *MCP4725* modulių paruošimo metu yra nustatomos adresų vertės, jog būtų galima siųsti signalus į greičio ir sukimosi valdymo modulius. Toliau yra inicializuojamas roboto valdymo *ROS* programos mazgas, jog būtų galima gauti */cmd_vel* duomenis, apdorotus iš *move_base* serviso. Paskui kiekvieną kartą yra patikrinama ar dabartinis programos mazgas yra aktyvus. Jei mazgas aktyvus, tai toliau yra laukiami duomenys iš */cmd_vel*. Priklausomai nuo gautos informacijos iš */cmd_vel*, visų pirma yra nustatoma, ar nėra pasiektas nurodytas tikslo taškas žemėlapyje. Jei tikslas pasiektas – mazgo darbas yra baigiamas. Priešingu atveju pagal gautas tiesinio ir kampinio greičio reikšmės iš */cmd_vel* yra atliekamas roboto judėjimas tikslo taško link. Tam atitinkamai *Arduino* valdiklyje apsirašomi metodai, kuriuose nurodomas robotas judėjimas skirtingomis kryptimis: kairę, dešinę, atgal ir į priekį. Roboto judėjimo *Arduino* programinė realizacija pateikiama 2 priede.



3.6 pav. *Arduino* roboto judėjimo valdymo veiklos diagrama pagal gautą */cmd_vel* pranešimą

3.3. Navigacijos *move_base* derinimas

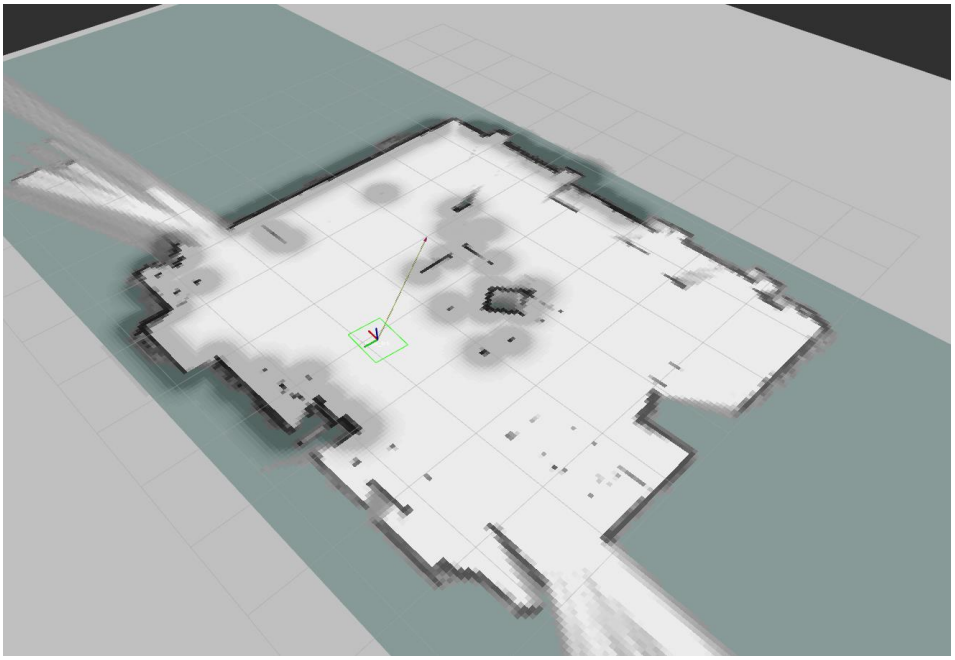
Po lokalizacijos ir žemėlapio sudarymo, sėkmingai navigacijai yra reikalinga roboto galimybė iš gautos apdorotos aplinkos informacijos sudaryti geriausią maršrutą iki nustatyto tikslo. Tam pasitelkiamas *move_base* paketas. Kaip ir lokalizacijos algoritmai, navigacijos metodai turi gausybę parametrų, skirtų suderinti skirtingų techninių parametrų robotų veikimą. Pagrindiniai parametrai į kuriuos bus atsižvelgiama *move_base* navigacijos derinimui, tai analizuojamos aplinkos dydis, kaštų žemėlapio tikslumas ir roboto gebėjimas įveikti aplinkoje esančias kliūtis.

Naudojamo modulio *move_base* parametruose yra itin svarbu nusirodyti tinkamą mobiliosios platformos dydį, nes pagal nurodytas roboto dimensijas parenkamas tinkamiausias maršrutas. Tačiau reikia atsižvelgti ir į tai, jog naudojamas lazerinis atstumo jutiklis nėra idealus ir generuoja matavimų paklaidą, kuri pagal *YDLIDAR X4* techninės įrangos dokumentaciją yra iki 2 cm. Papildomai svarbu atkreipti dėmesį į tai, kad robotui atliekant pasisukimą ir bandant išvengti kliūties, pasisukimo metu mobilioji platforma gali priartėti prie kliūties dėl pajudėjimo į priekį.

Be roboto dydžio nustatymo, esminė problema yra tinkamai nurodyti sudaromo kaštų žemėlapio tikslumą ir plotą. Modulio *move_base* sudaromas kaštų žemėlapis yra 2 tipų: globalus ir vietinis.

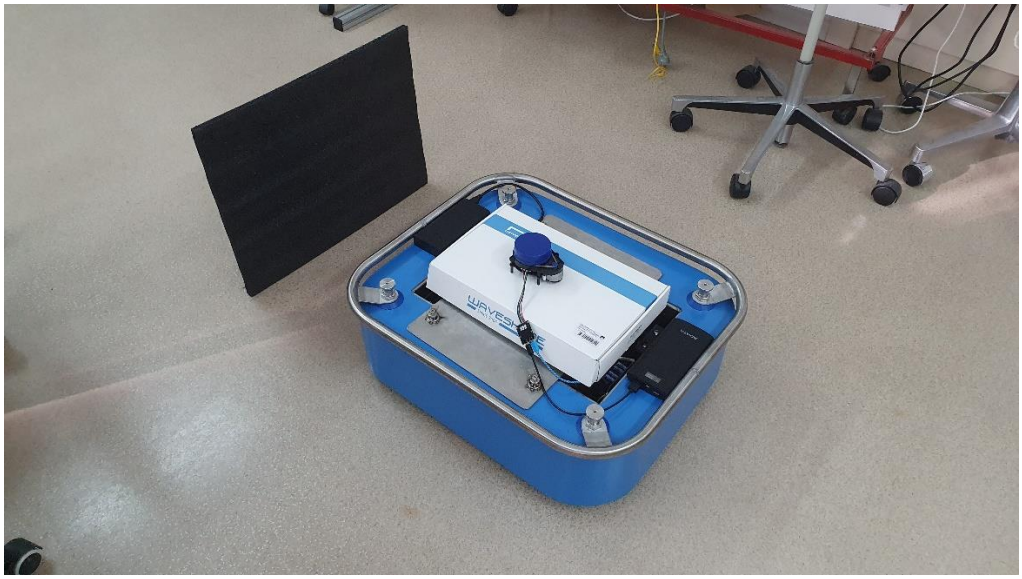
Globalus kaštų žemėlapis yra atnaujinimas rečiau (3.7 pav.), tačiau pateikia duomenis apie globalios aplinkos struktūrą. Globalus kaštų žemėlapis yra esminis pirminio maršruto sudarymo etape, kur pagal turimą informaciją apie bendrą aplinką yra sudaromas preliminarus maršrutas. 3.7 pav. pateiktame paveikslėlyje yra atvaizduojami globalaus kaštų žemėlapio pažymėtos kliūtys tamsesne pilka spalva ir apskritimo forma. Roboto tikslas yra kuo įmanoma labiau vengti priartėjimo prie šių kliūčių ir taip išvengti susidūrimo.

Vietinis kaštų žemėlapis yra atnaujinamas dažnesniu intervalu ir dinamiškai kintančioje aplinkoje leidžia robotui įvertinti naujai atsiradusias kliūtis. Vietinis kaštų žemėlapis dėl dažnesnio atnaujinimo taip pat reikalauja didesnių skaičiavimo pajėgumų, todėl tokio tipo kaštų žemėlapis dažnai interpretuoja tik tam tikrą ploto dalį.

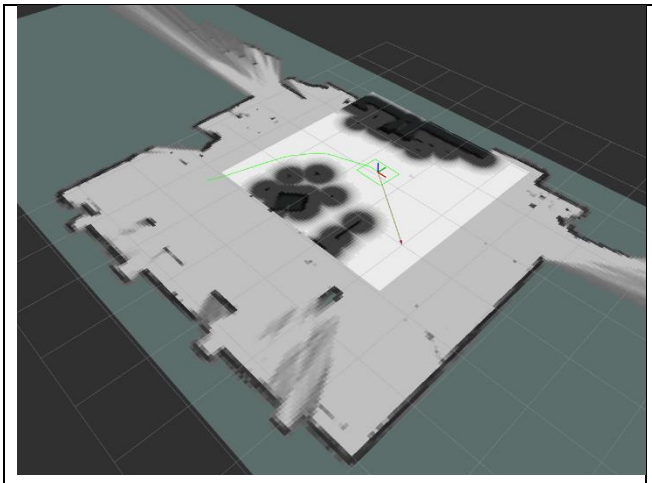


3.7 pav. Globalus kaštų žemėlapis iki 20 m su 0,05 m tikslumu

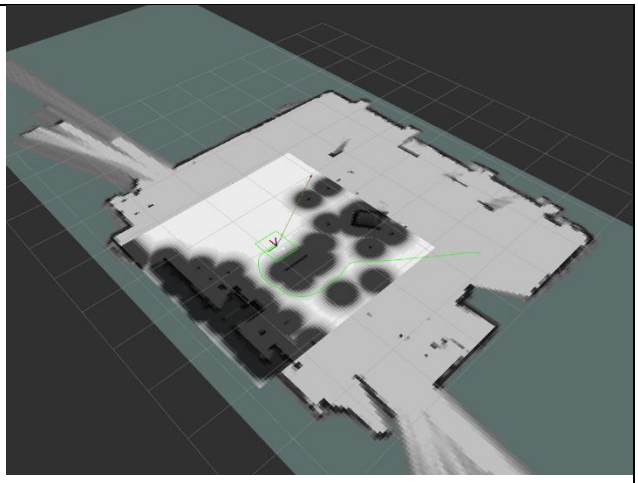
3.8 pav. priešais robotą yra pastatoma kliūtis, kuri atsispindi 3.10 pav. dinamiškai atsinaujinus kaštų žemėlapiui dėl atsiradusio objekto. 3.9 pav. yra pateikta situacija, kai vietinis kaštų žemėlapis, kuris yra 4x4 m dydžio ir 0,05 m tikslumo įvertina aplink esančias kliūtis ir pagal tai sudaro maršrutą (žalia trajektorija) iki nustatytos padėties. Atsiradus kliūčiai (3.10 pav.) maršrutas atitinkamai pasikeičia ir robotas bando įveikti kliūtį ją apvažiuodamas kitu keliu.



3.8 pav. Priešais robotą pastatytos kliūties įvertinimas

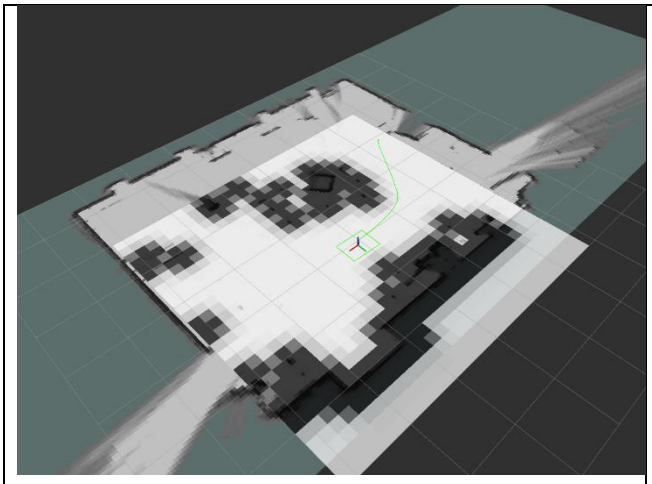


3.9 pav. Vietinis kaštų žemėlapis iki 4 m su 0,05 m tikslumu

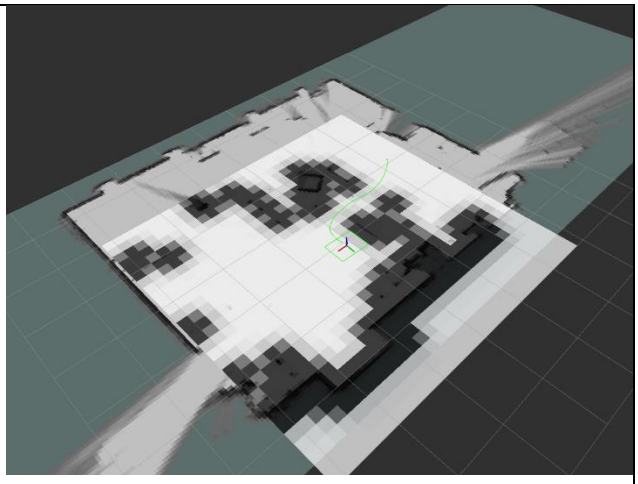


3.10 pav. Vietinis kaštų žemėlapis iki 4 m su 0,05 m tikslumu ir kliūties įvertinimas

3.11 pav. ir 3.12 pav. padidinama vietinio kaštų žemėlapio interpretuojama teritorija iki 6x6 m dydžio, o tikslumas sumažinimas iki 0,2 m. Analizuojamos teritorijos padidinimas leidžia kelio planavimo moduliui atsižvelgti į alternatyvius kelio sudarymo būdus siekiant tikslo. Tikslumo sumažinimas padidina neužtikrintumą įveikiant kliūtis, kuris atsiranda dėl abstraktesnio kliūties įvertinimo ir tokiu būdu padidinama paklaida robotui bandant įveikti maršrutą. 3.12 pav. priešais robotą atsiradus kliūčiai, planavimo modulis bando įveikti kliūtį apvažiuodamas ją kitu maršrutu nei prieš tai bandytu 3.10 pav.

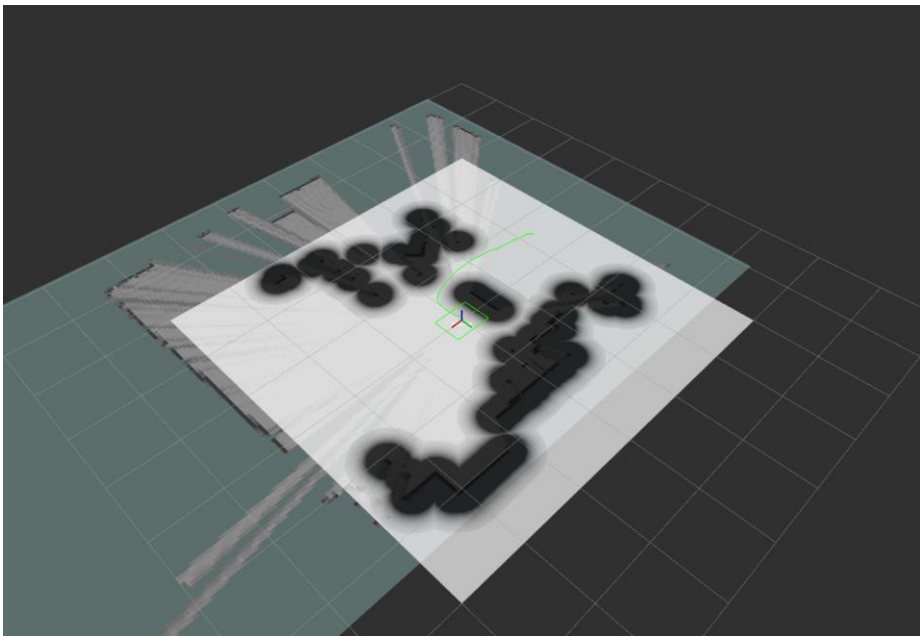


3.11 pav. Vietinis kaštų žemėlapis iki 6 m su 0,2 m tikslumu



3.12 pav. Vietinis kaštų žemėlapis iki 6 m su 0,2 m tikslumu ir kliūties įvertinimas

3.13 pav. atsispindi vietinis kaštų žemėlapis, kai nustatytas dydis yra 6x6 m, o tikslumas 0,01 m. Šiuo atveju galima tikslesnė aplinkos interpretacija, nes kiekvienas žemėlapio langelis yra apdorojamas 0,01 tikslumu, tačiau toks apdorojimas turi trūkumų, kaip didesnių skaičiavimo resursų sunaudojimas.

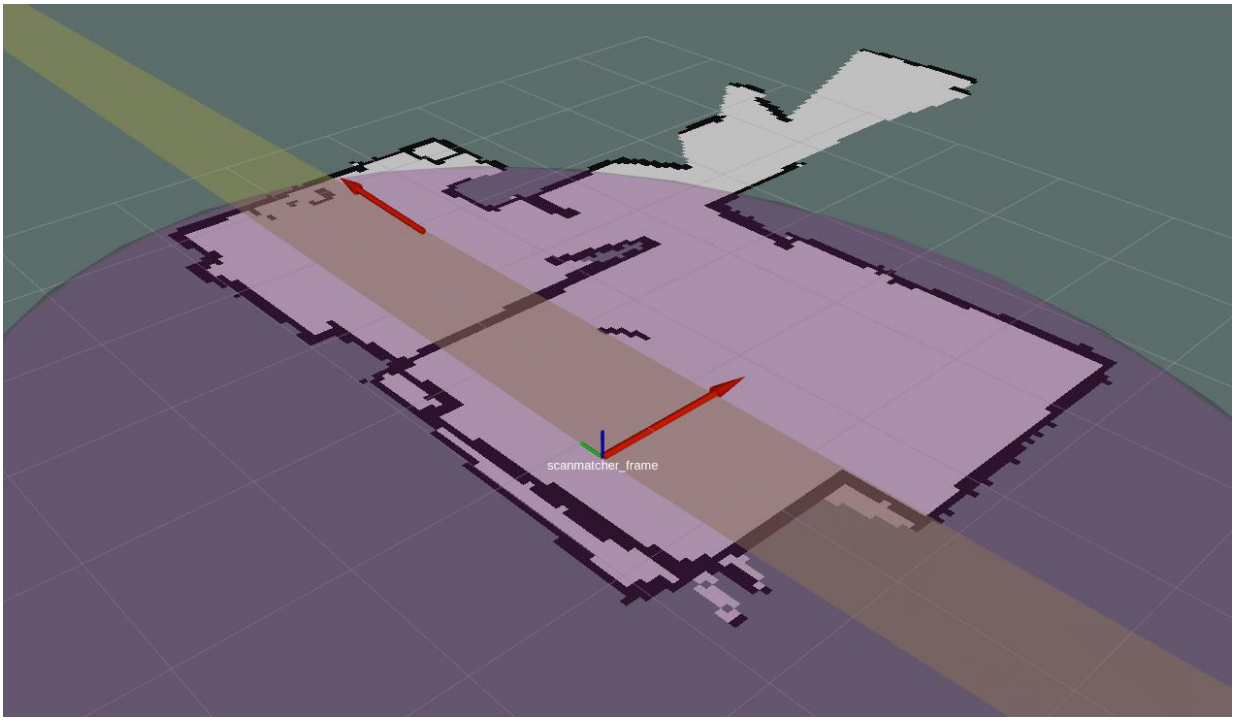


3.13 pav. Vietinis kaštų žemėlapis iki 6 m su 0,01 m tikslumu ir kliūties įvertinimas

Lyginant su kaštų žemėlapiu, kurio parinktas langelio tikslumas yra 0,05 m ir 4x4 m dydžio vidutiniškai sunaudojama 19 % *CPU*, o 6x6 m dydžio ir 0,01 m langelio tikslumo atveju – 25 % vien tik navigacijai. Nors 6 % skirtumas neatrodo didelis, bet riboto skaičiavimo resursų valdikliuose didesnė apkrova gali lemti didesnį skaičių klaidų lokalizacijoje ir žemėlapio sudaryme dėl netikėtai išaugusio procesoriaus naudojimo. Siekiant išvengti minėtų problemų buvo pasirinktas 4x4 m dydžio vietinio kaštų žemėlapio sudarymas su 0,05 m langelio tikslumo dėl mažesnių resursų sąnaudų ir pakankamo tikslumo roboto navigacijai dinamiškai kintančioje aplinkoje.

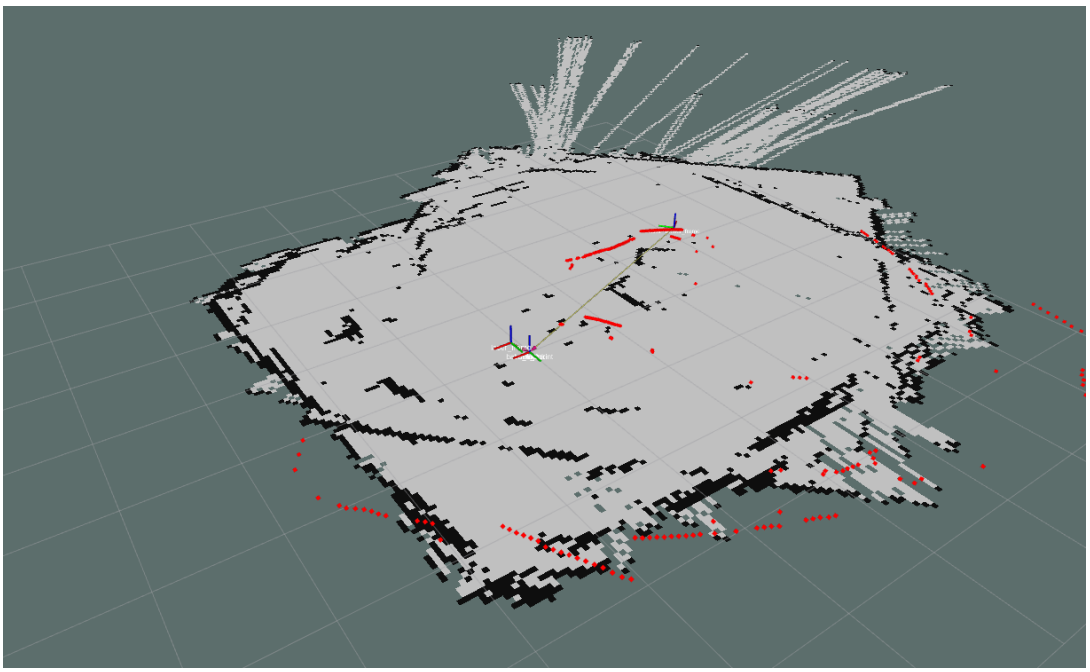
3.4. *Hector SLAM* metodo testavimas

3.14 pav. yra pateikiamas lazerinio atstumo jutiklio kartu su *ROS Hector SLAM* biblioteka bandymas naudojant *RViz* programinę įrangą skirtą atvaizdavimui. Tam rankiniu būdu yra sudaromas aplinkos žemėlapis, keičiant lazerinio atstumo jutiklio pozicijas. Juodi kontūrai žemėlapyje žymi nepravažiuojamas vietas arba kliūtis, o pilkas plotas – pravažiuojamą dalį. Objektas *scanmatcher_frame* nurodo dabartinę lazerinio atstumo jutiklio pozicija aplinkoje. Testavimo metu buvo pastebėta, jog tikslesni rezultatai gaunami tuo atveju, jei lazerinis atstumo jutiklis yra perkeliamas iš vienos vietos į kitą bei pasukamas lėtesniu greičiu. Didesnio greičio atveju dažnai buvo prarandama tiksli objekto pozicija ir gaunamas netaisyklingai sudarytas žemėlapis, kuris pastebimas tolimesniuose testavimo rezultatuose – 3.15 pav. ir 3.16 pav.

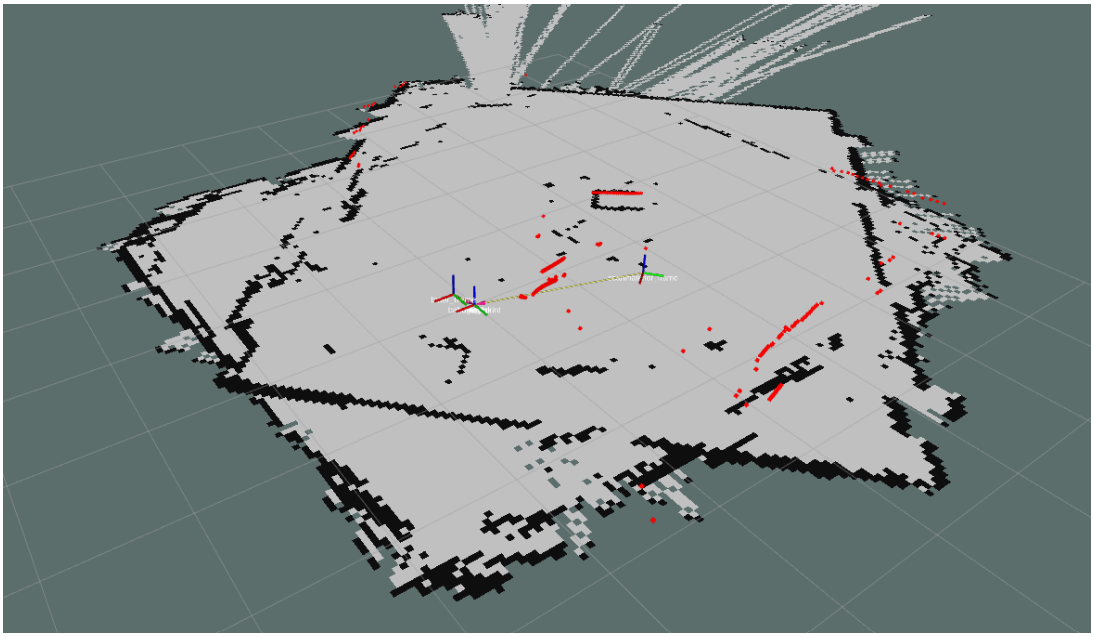


3.14 pav. Lazerinio atstumo jutiklio ir *Hector SLAM* paleidimo bandymas

3.15 pav. ir 3.16 pav. yra atvaizduojamas sudarytas aplinkos žemėlapis pirmu bandymu, kai robotas juda ~12 cm/s greičiui. Šiuo atveju robotas yra baigęs sudaryti aplinkos žemėlapį ir randasi prie nustatyto tikslo taško 3.15 pav., kuris žymimas *scanmatcher_frame* ašimi. Roboto pradinė pozicija žemėlapyje (pozicija, nuo kuriuos buvo pradėtas žemėlapio sudarymas) yra pažymėta *base_footprint* ašimi. Raudoni taškai žymi iš lazerinio atstumo jutiklio nustatytas kliūtys, kurios yra pastoviai atnaujinamos robotui judant. Dėl duomenų iškraipymo, pradinė pozicija yra pakitusi, kas veda prie gana didelių skaičiavimo netikslumų. 3.16 pav. atsispindi roboto grįžimas į pradinę poziciją žemėlapyje. Dėl staigesnių mobiliosios platformos posūkių matomas sudaryto žemėlapio susidubliavimas, kuris lemia neteisingą roboto pozicijos interpretavimą. Tokiu atveju reikalingas pakartotinis žemėlapio sudarymas arba rankinis pradinės roboto pozicijos atstatymas, nes robotas nebesugeba teisingai orientuotis aplinkoje dėl netikslios globalios aplinkos žemėlapio.

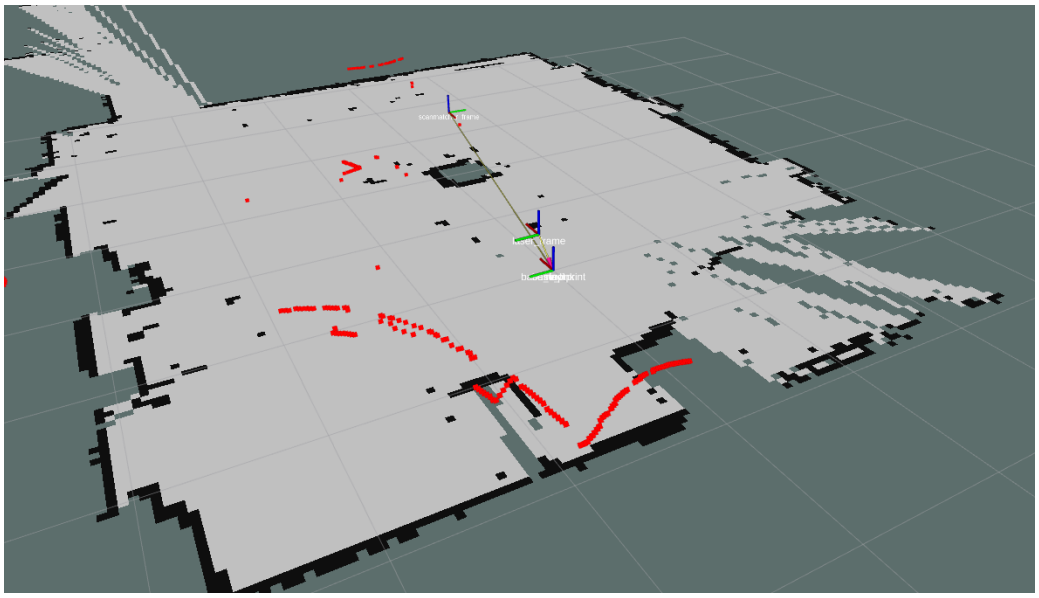


3.15 pav. *Hector SLAM* lokalizacijos ir navigacijos aplinkoje pirmas bandymas (tikslo taškas)

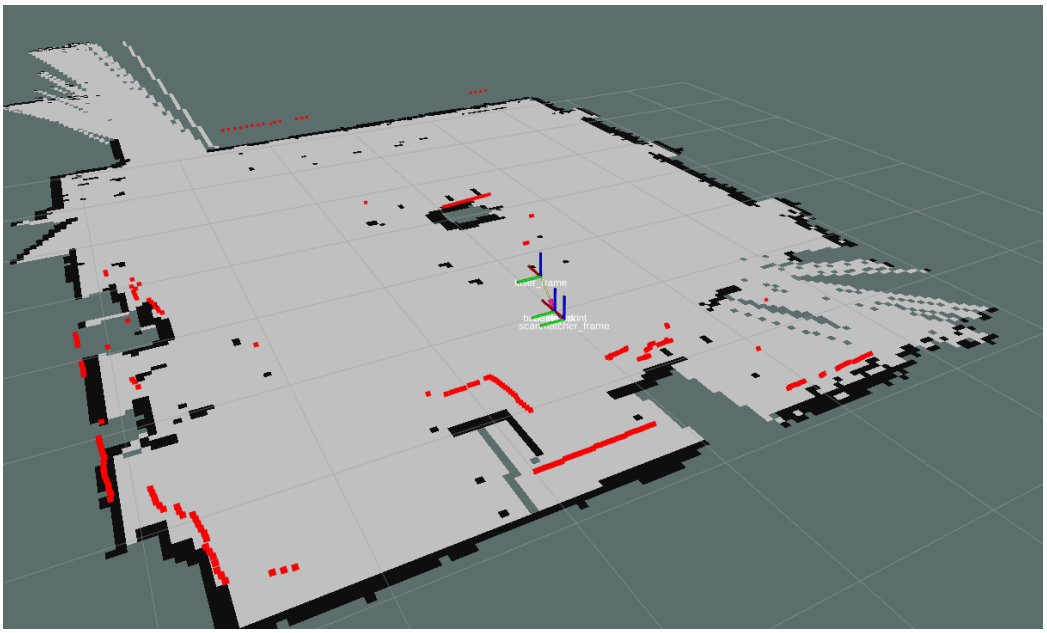


3.16 pav. *Hector SLAM* lokalizacijos ir navigacijos aplinkoje pirmas bandymas (pradinė pozicija)

3.17 pav. ir 3.18 pav. atliekant antrąjį bandymą robotas buvo nustatytas judėti ~5 cm/s greičiu, siekiant išvengti neteisingos roboto pozicijos nustatymo ir gauti tikslesnius duomenis apie aplinką. Gautas žemėlapis buvo žymiai tikslesnis ir roboto pozicija aplinkoje atitinkamai buvo nustatyta gana tiksliai lyginant su praeitu bandymu. 3.17 pav. atvaizduojamas robotas, kuris randasi nustatytame tikslo taške ir kaip anksčiau yra žymimas *scanmatcher_frame* ašimi. Pradinė roboto pozicija, iš kurios buvo pradėtas sudaryti žemėlapis yra žymima – *base_footprint* ašimi.



3.17 pav. *Hector SLAM* lokalizacijos ir navigacijos aplinkoje antras bandymas (tikslo taškas)



3.18 pav. *Hector SLAM* lokalizacijos ir navigacijos aplinkoje antras bandymas (pradinė pozicija)

4. Eksperimentinė dalis

Be *Hector SLAM* algoritmo pritaikymo buvo pasirinkti realizuoti 3 lokalizacijos bei žemėlapio sudarymo algoritmai. Tyrimai ir palyginimai atliekami su šiais metodais: *Google Cartographer, Hector SLAM, GMapping* ir *Karto SLAM*. Kiekvienas iš šių algoritmų buvo pritaikytas veikimui su naudojama mobiliaja robotine platforma (enkoderių duomenys odometrijai) ir lazeriniu atstumo jutikliu *YDLIDAR X4*. Realizacija yra panaši kaip ir *Hector SLAM* algoritmo, kuri apibendrintai pateikiama 3.4 pav., parenkant atitinkamus parametrus kiekvienam iš algoritmų pagal lazerinio atstumo jutiklio ir mobiliosios platformos technines galimybes.

Tyrimas buvo atliktas laboratorijoje pasirenkant 3 skirtingus roboto judėjimo greičius: 5 cm/s, 10 cm/s ir 15 cm/s. Kiekvienu atveju buvo fiksuojamas *Raspberry Pi 4* procesoriaus ir atminties naudojimas. Be našumo rodiklių buvo atsižvelgiama į lokalizacijos tikslumą, išmatuotos aplinkos ilgį ir plotį, siekiant įvertinti žemėlapio teisingumą atsižvelgiant į gautus aplinkos išmatavimus. Lokalizacijos tikslumas įvertinimas atsižvelgiant į paklaidą, kuri gaunama robotui grįžus į savo pradinę poziciją (vieta, nuo kurios buvo startuotas žemėlapio sudarymas), o paklaida įvertinama išmatavus atstumą nuo pradinio taško iki roboto padėties *RViz* programoje. Sudaryto žemėlapio ilgis ir plotis atitinkamai nustatomi *RViz* programoje. Kiekvieno eksperimento metu buvo apskaičiuojamas procesoriaus naudojimo vidurkis ir reikalingas operatyviosios atminties kiekis. Sunaudotas operatyviosios atminties kiekis buvo apskaičiuojamas atimant paskutinę procentinę vertę (po užbaigto žemėlapio sudarymo) iš pirmos procentinės vertės (žemėlapio sudarymo pradžios), nes atminties sunaudojimas auga priklausomai nuo aplinkos dydžio.

Remiantis anksčiau minėta metodika taip pat su kiekvienu algoritmu buvo sudaromas didesnės aplinkos 2 laboratorijų žemėlapis bei išmatuojami aprašyti rodikliai. Gauti duomenys buvo analizuojami ir palyginami, nustatant tiksliausią ir mažiausiai triukšmo generuojantį lokalizacijos ir žemėlapio sudarymo algoritmą. Taip pat buvo atsižvelgiama į tai, kiek kiekvienas iš algoritmų sunaudoja resursų tos pačios aplinkos žemėlapio sudarymui.

Žemėlapiai sudaromi naudojant *move_base* servisą, kuris robotui judant aplinkoje ir dinamiškai atnaujinant žemėlapį bei savo poziciją atitinkamai atsižvelgia į šiuos parametrus ir sudaro maršrutą iki nustatyto tikslo taško, kurį turi pasiekti robotas pagal gaunamas judėjimo komandas.

Pateikti žemėlapiai iš skirtingų lokalizacijos ir kartografavimo algoritmo turi analogišką legendos struktūrą: juodi kontūrai žymi atpažintas sienas ir objektus, kurie nėra pravažiuojami, pilkas plotas nurodo laisvas ir pravažiuojamas aplinkos dalis, žalia stačiakampė platforma ir/arba *odom* vadinama ašis, tai roboto dabartinė nustatyta padėtis aplinkoje. Žalia linija žymi *move_base* suplanuotą judėjimo trajektoriją iki nustatyto tikslo.

4.1 pav. pateikiami laboratorijos, kurioje atliekami minėti eksperimentai atvaizdai.

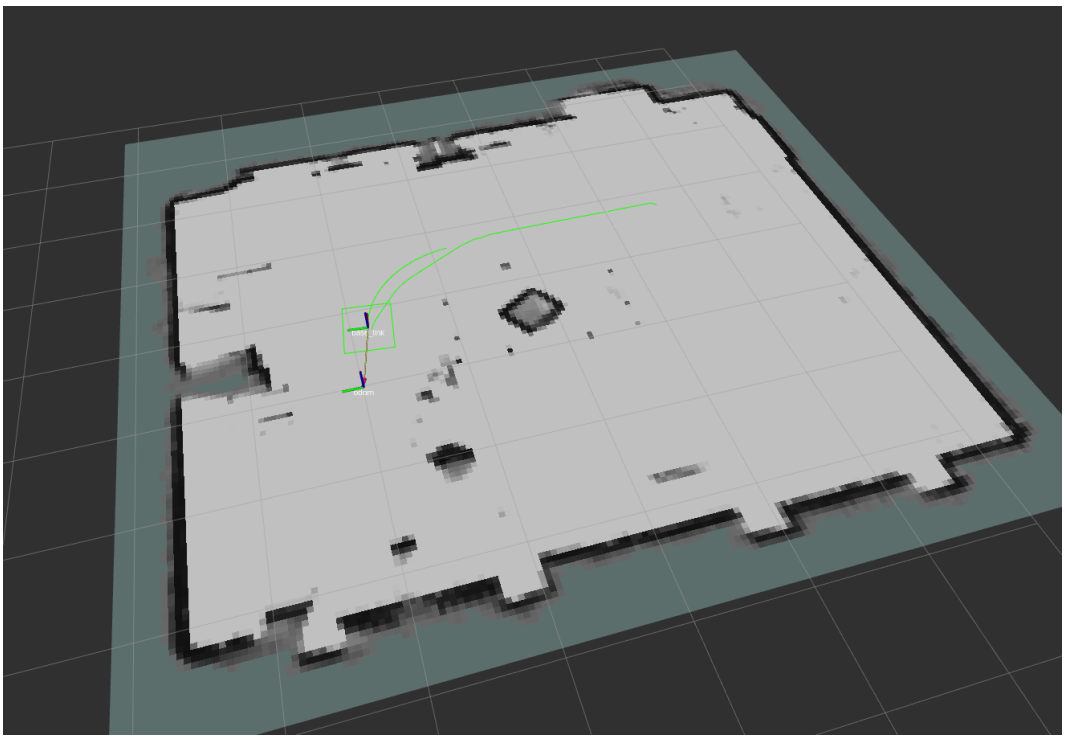
4.1 pav. Laboratorija, kurioje atliekami lokalizacijos, žemėlapio sudarymo ir navigacijos eksperimentai

4.1. *Google Cartographer* metodas

Testavimo metu buvo naudojamas lazerinis atstumo jutiklis *YDLIDAR X4* roboto pozicijos nustatymui ir žemėlapio sudarymui.

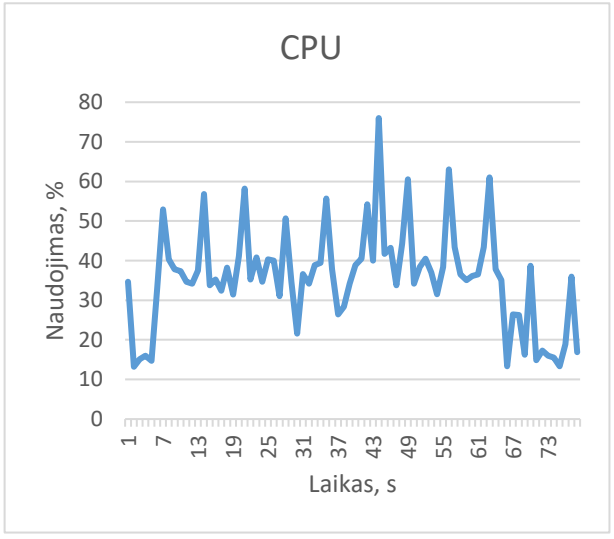
Žemėlapio sudarymas ir lokalizacija mažoje aplinkoje

4.2 pav. pateikiamas *Google Cartographer* sudarytas aplinkos žemėlapis robotui judant aplinkoje ir grįžtant į savo pradinę padėtį. Bendras žemėlapis turi mažai triukšmo, nes nepravažiuojamos dalys (juodi kontūrai) yra apibrėžtos gana aiškiai ir atspindinčios panašias kliūčių padėtis. Robotui sugrįžus į savo pradinę padėtį po žemėlapio sudarymo etapo buvo išmatuota 0,07 m lokalizacijos paklaida, kas rodo tikslią įvertinimo poziciją ir labai mažą nuokrypį.

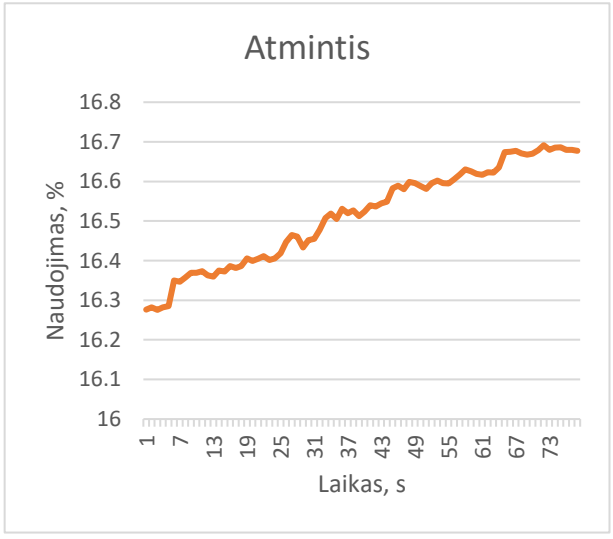


4.2 pav. *Google Cartographer* sudarytas aplinkos žemėlapis robotui judant 5 cm/s greičiu

4.3 pav. ir 4.4 pav. atsispindi sunaudoti skaičiavimo ištekliai žemėlapio sudarymo ir navigacijos metu. Maksimali *CPU* sunaudojimo vertė 76 % pasiekta 44 sekundę. Viso eksperimento metu *CPU* naudojimo vidurkis buvo 35,64 %, o atminties sunaudojimas 16,05 MB.

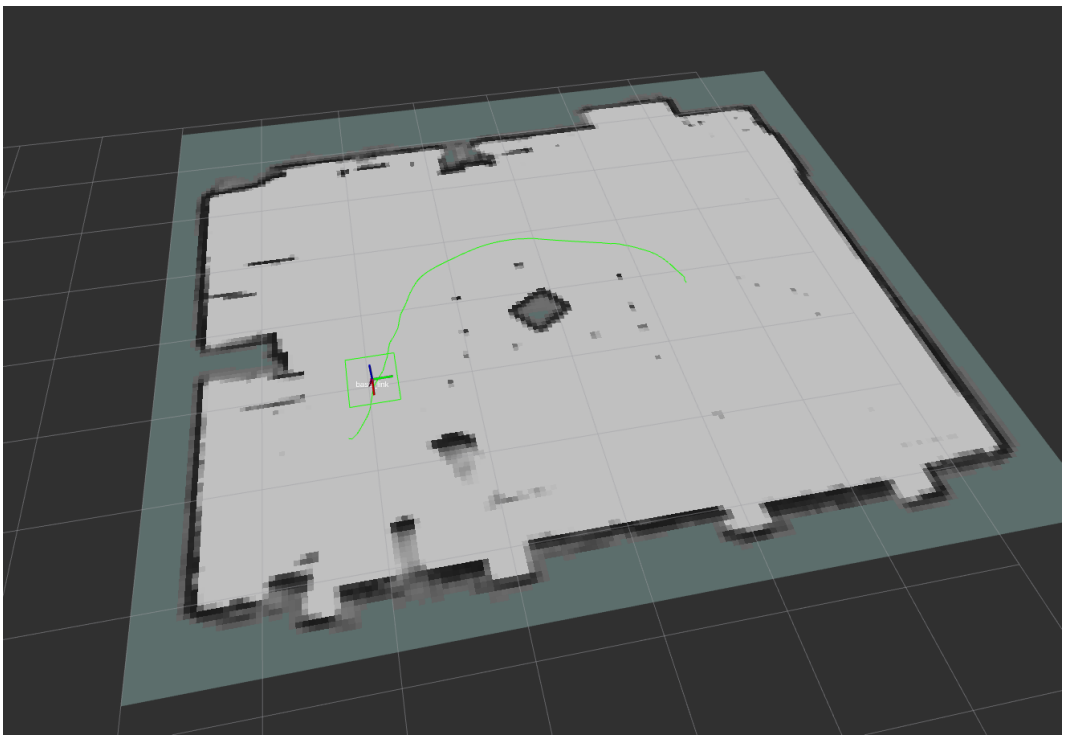


4.3 pav. *Google Cartographer CPU* naudojimas robotui judant 5 cm/s



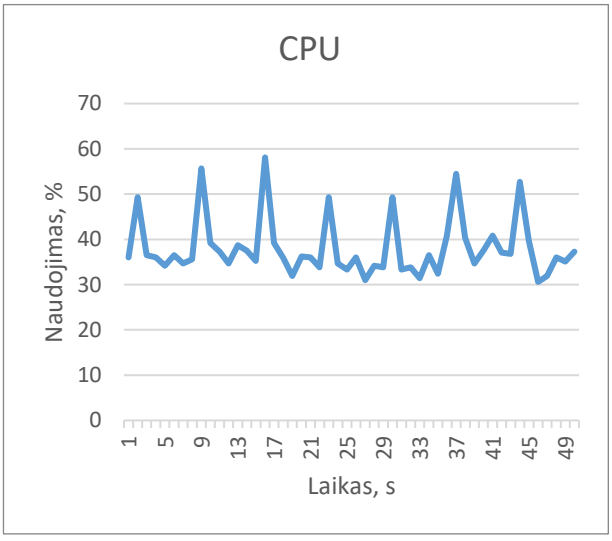
4.4 pav. *Google Cartographer* operatyviosios atminties naudojimas robotui judant 5 cm/s

4.5 pav. atvaizduojamas sudarytas aplinkos žemėlapis, kai robotas nustatytas judėti 10 cm/s greičiu. Sudarytame žemėlapyje triukšmo yra mažai lyginant su praeitu, net ir robotui judant dvigubai didesniu greičiu. Roboto lokalizacijos tikslumas taip pat išliko beveik nepakitęs su 0,09 m paklaida.

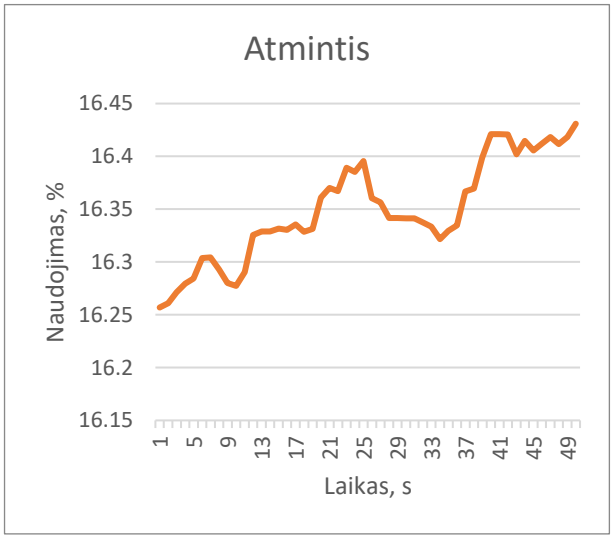


4.5 pav. *Google Cartographer* sudarytas aplinkos žemėlapis robotui judant 10 cm/s greičiu

4.6 pav. ir 4.7 pav. atsispindi *CPU* ir atminties sunaudojimas atitinkamai robotui judant 10 cm/s greičiu. Procesoriaus naudojimas buvo tolygesnis, kur maksimali užfiksuota reikšmė 58,1 % 14 sekundę. Vidutiniškai procesoriaus panaudojimas buvo truputį didesnis 38,06 % nei robotui judant lėtesniu greičiu. Atminties sunaudojimas šiuo atveju buvo užfiksuota itin žemas – 6,96 MB.



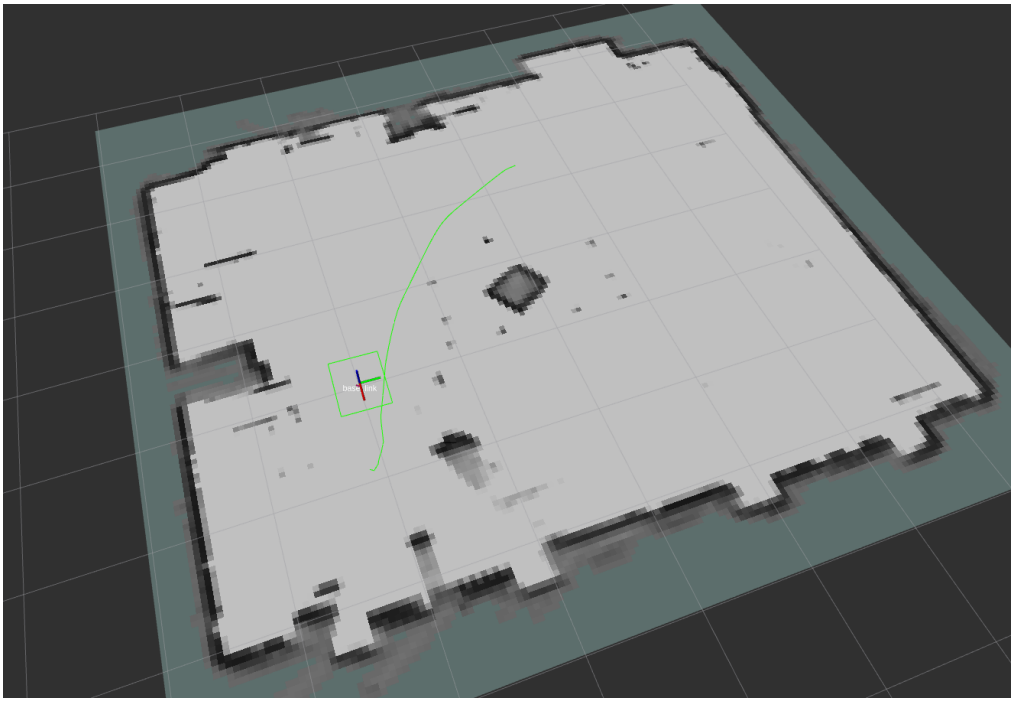
4.6 pav. *Google Cartographer CPU* naudojimas robotui judant 10 cm/s



4.7 pav. *Google Cartographer* operatyviosios atminties naudojimas robotui judant 10 cm/s

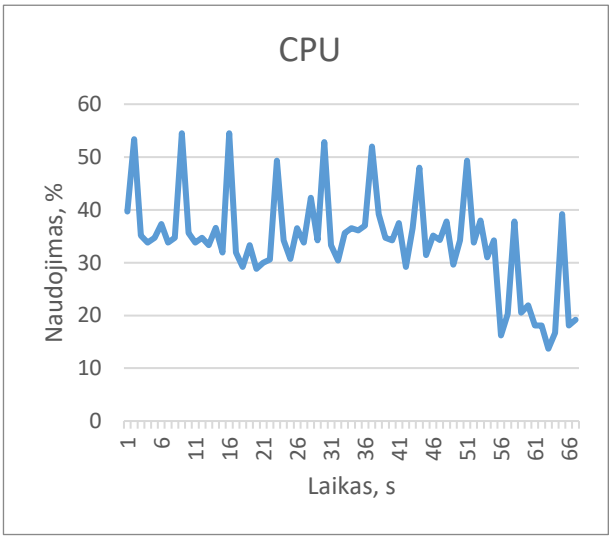
4.8 pav. pateikiamas sudarytas aplinkos žemėlapis robotui judant 15 cm/s greičiui. Iš sudaryto žemėlapio atsispindi, jog triukšmas yra truputį padidėjęs dėl už kontūro ribų egzistuojančių pilkų

zonų, tačiau pravažiuojama aplinkos dalis yra įvertinta panašiu tikslumu kaip ir praeitais eksperimento atvejais. Nustatyta lokalizacijos tikslumo paklaida išaugo iki 0,14 m., kas šiuo atveju gali sukelti didesnių navigacijos problemų ypač aplinkose, kuriose reikalaujamas aukštas roboto lokalizacijos tikslumas.

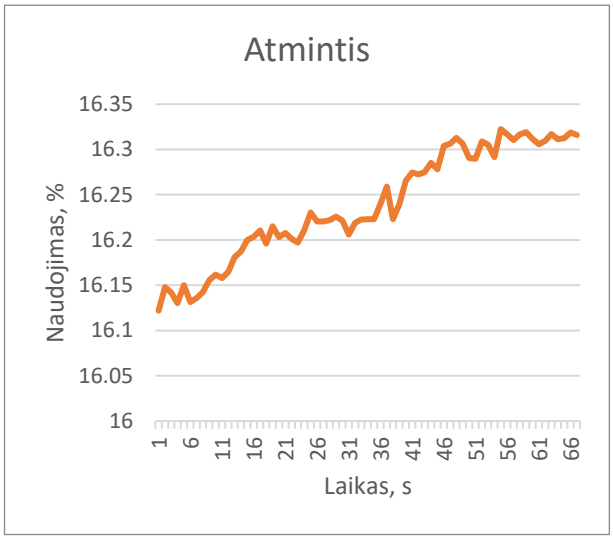


4.8 pav. *Google Cartographer* sudarytas aplinkos žemėlapis robotui judant 15 cm/s greičiu

4.9 pav. ir 4.10 pav. pateikiami *CPU* ir atminties sunaudojimo grafikai. Grafikuose užfiksuota maksimali procesoriaus sunaudojimo vertė 54,5 % 16 sekundę. Vidutiniškai procesoriaus naudojimas buvo 34,09 %. Sunaudotas operatyviosios atminties kiekis – 7,76 MB.



4.9 pav. *Google Cartographer CPU* naudojimas robotui judant 15 cm/s



4.10 pav. *Google Cartographer* operatyviosios atminties naudojimas robotui judant 15 cm/s

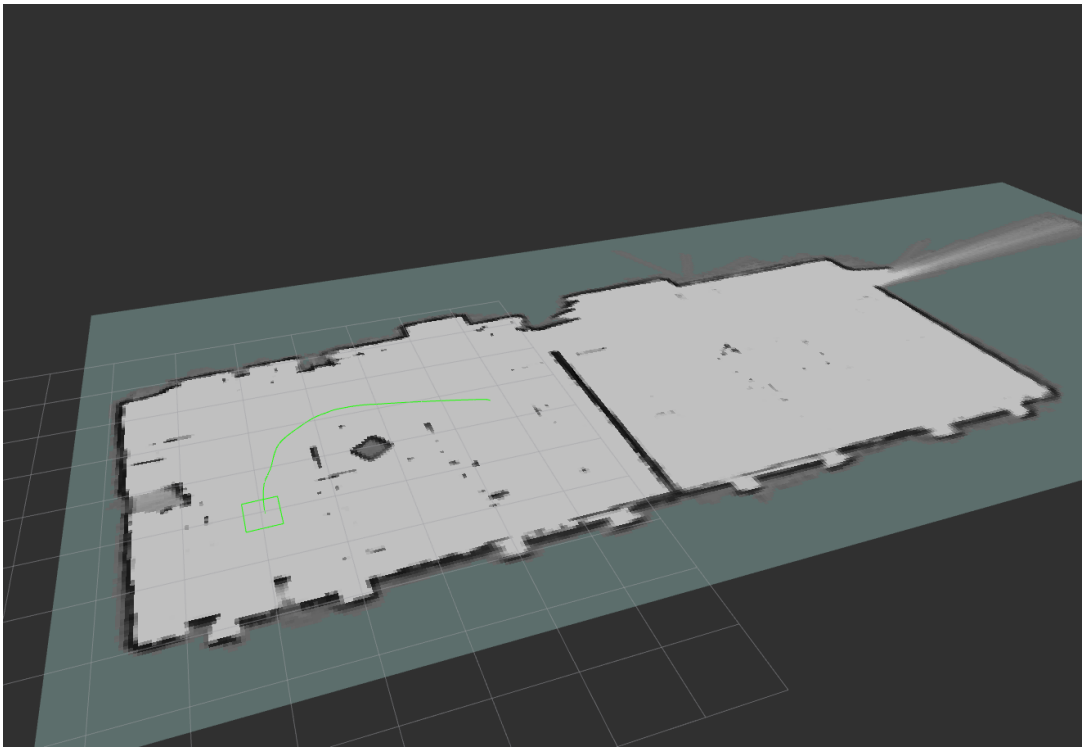
4.1 lentelėje pateikiama nustatytų rodiklių informacija *Google Cartographer* algoritmo skirtingais ištirtais roboto judėjimo greičiais. Visais roboto judėjimo greičiais aplinkos ilgis ir plotis buvo išmatuotas panašus ir su nedidele paklaida. Mažiausia lokalizacijos paklaida 0,07 m užfiksuota robotui judant 5 cm/s greičiui. Kadangi robotas judėjo lėčiau ir posūkius atliko švelniau buvo galima tiksliau įvertinti roboto poziciją kiekvieno lazerinio atstumo jutiklio skenavimo metu ir tokiu būdu *Google Cartographer* algoritmas galėjo geriau įvertinti roboto poziciją pagal skenavimo taškų atitiktį žemėlapyje. Vidutiniškai *Google Cartographer* kartu su *move_base* navigacijos moduliu sunaudojo 35,93 % procesoriaus ir 10,26 MB operatyviosios atminties (*RAM*).

4.1 lentelė. *Google Cartographer* rodikliai skirtingais judėjimo greičiais mažoje aplinkoje

Greitis, cm/s	**Lokalizacijos paklaida, m**	**Ilgis, m**	**Plotis, m**	**CPU, %**	**RAM, MB**
5	0,07	7,04	5,89	35,64	16,05
10	0,09	7,00	5,90	38,06	6,96
15	0,14	7,02	5,83	34,09	7,76
Vidutinė reikšmė:	**0,10**	**7,02**	**5,87**	**35,93**	**10,26**

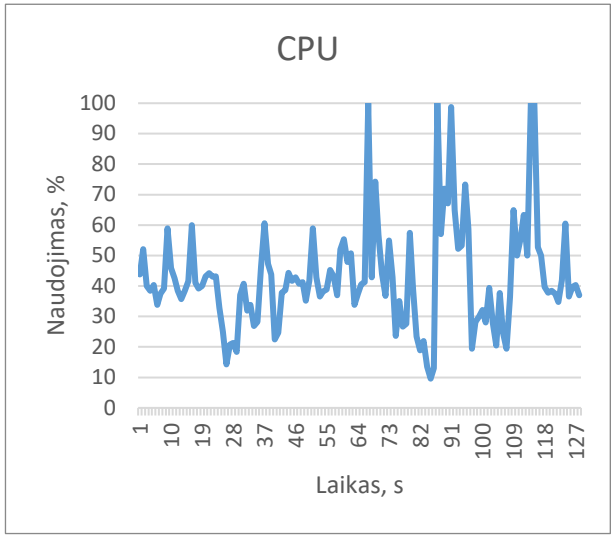
Žemėlapio sudarymas ir lokalizacija didelėje aplinkoje

4.11 pav. atsispindi dviejų laboratorijų sudarytas aplinkos žemėlapis robotui judant 5 cm/s greičiu. Pačiame sudarytame žemėlapyje yra mažai triukšmo, tačiau padidinus žemėlapio sudarymo ir lokalizacijos plotą pastebėtas lokalizacijos tikslumo nukentėjimas. Robotui grįžus į savo pradinę poziciją nustatyta 0,22 m lokalizacijos paklaida nuo tikrosios pradžios pozicijos, kas rodo, jog informacija gaunama iš lazerinio atstumo jutiklio nėra pakankama tiksliam pozicijos įvertinimui, kai aplinkos plotas yra didelis.

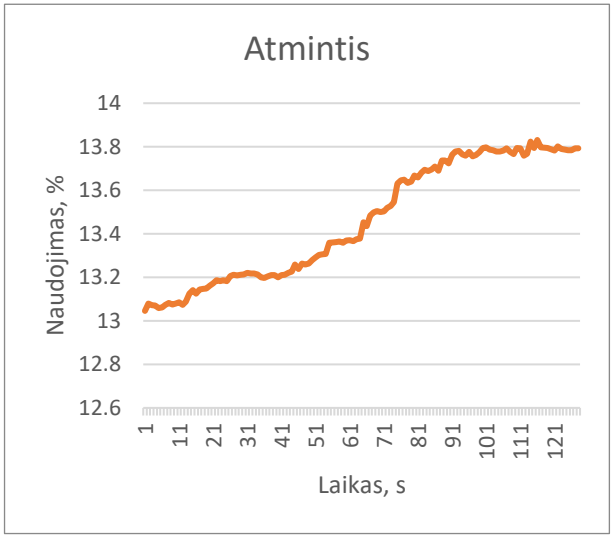


4.11 pav. *Google Cartographer* sudarytas 2 laboratorijų aplinkos žemėlapis robotui judant 5 cm/s greičiu

4.12 pav. ir 4.13 pav. pateikiami *CPU* ir atminties sunaudojimo grafikai. Padidėjus aplinkai nustatytas didesnis procesoriaus naudojimas, kur 67, 87, 91 114 ir 115 sekundėmis užfiksuotas maksimalus galimas jo naudojimas. Dažnas maksimalus procesoriaus resursų sunaudojimas galėjo turėti įtakos dėl roboto lokalizacijos tikslumo įvertinimo, nes procesoriui nepakanka laiko apdoroti lazerinio atstumo jutiklio informacijos ir dėl to atsiranda didesnė paklaida. Vidutiniškai užfiksuotas 42,62 % procesoriaus sunaudojimas. Atminties sunaudojimas atitinkamai išaugo iki 29,87 MB padidėjus tiriamos aplinkos plotui, nes reikalinga saugoti didesnį aplinkos žemėlapį.

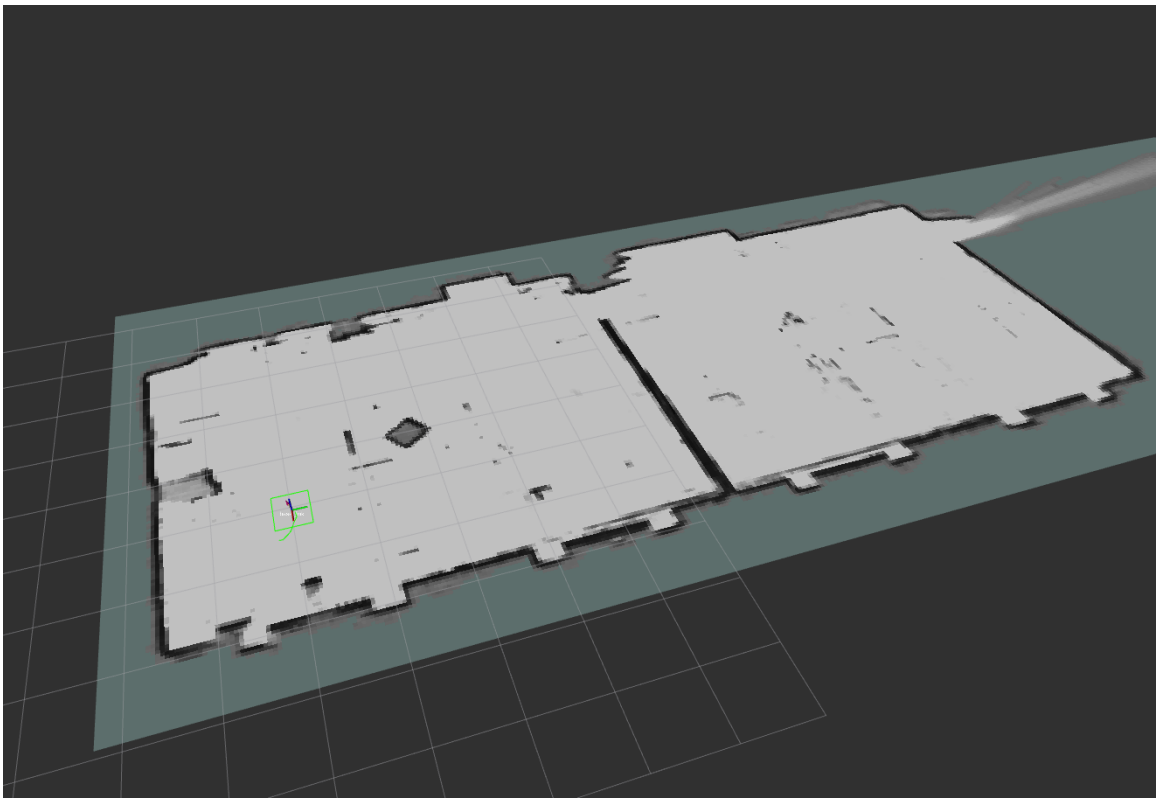


4.12 pav. *Google Cartographer CPU* naudojimas robotui judant 5 cm/s



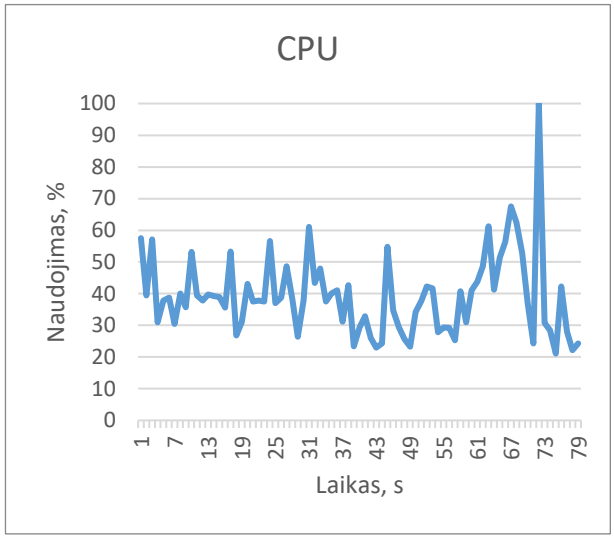
4.13 pav. *Google Cartographer* operatyviosios atminties naudojimas robotui judant 5 cm/s

4.14 pav. atvaizduojamas atliktas eksperimentas, kai roboto judėjimo greitis yra 10 cm/s. Sudaryto žemėlapio kokybei didelių netikslumų lyginant su pastaruoju žemėlapiu nematyti. Lokalizacijos paklaida atitinkamai išaugo iki 0,25 m., kas toliau padidina roboto galimybę tiksliai naviguoti aplinkoje dėl sumažėjusio lokalizacijos tikslumo.

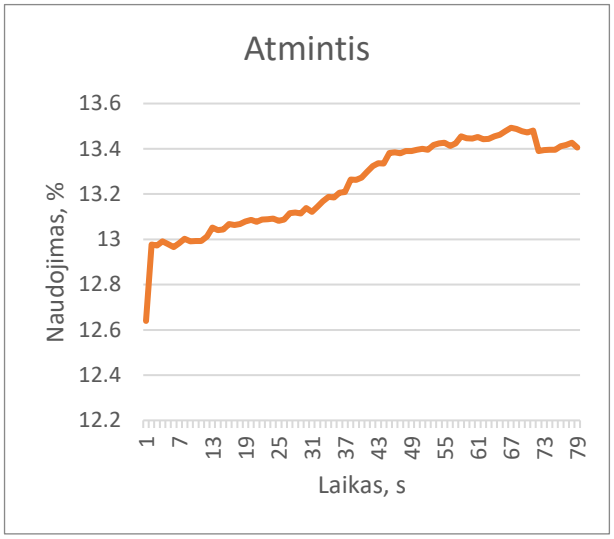


4.14 pav. *Google Cartographer* sudarytas 2 laboratorijų aplinkos žemėlapis robotui judant 10 cm/s greičiu

4.15 pav. ir 4.16 pav. atitinkamai atvaizduojami *CPU* ir atminties naudojimo grafikai. Šiuo atveju buvo užfiksuotas vienas procesoriaus naudojimo šuolis iki maksimumo 72 sekundę. Procesoriaus naudojimo vidurkis – 39,21 %. Sunaudota atmintis išliko panaši 30,64 MB, kadangi nebuvo pakeistas tiriamos aplinkos plotas.

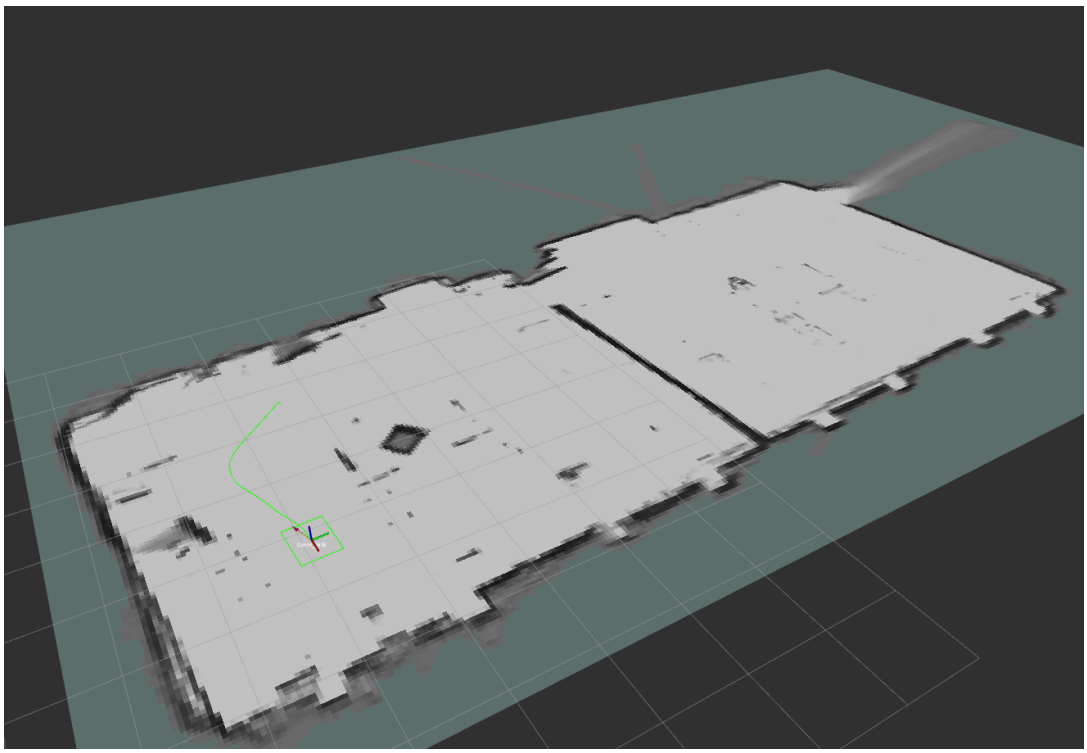


4.15 pav. *Google Cartographer CPU* naudojimas robotui judant 10 cm/s



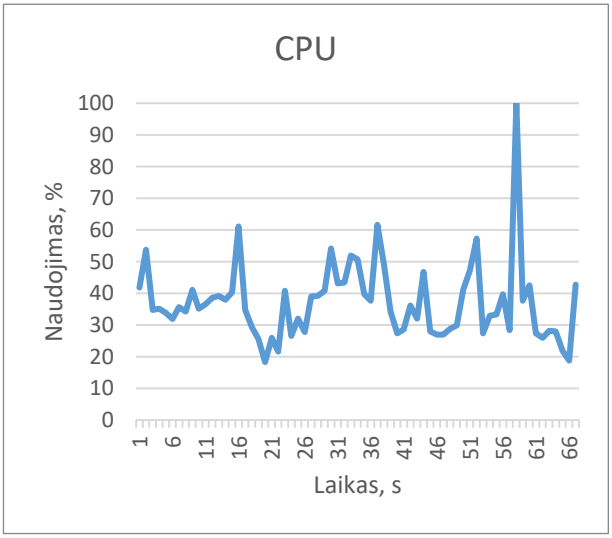
4.16 pav. *Google Cartographer* operatyviosios atminties naudojimas robotui judant 10 cm/s

4.17 pav. atvaizduojamas sudarytas aplinkos žemėlapis, kai robotas nustatytas judėti 15 cm /s greičiu. Nuo pradinės matavimo pozicijos yra pastebimas sudaryto žemėlapio netikslumas dėl sienų susidubliavimo, kas gali lemti neteisingą orientyrų interpretavimą. Atitinkamai padidinus judėjimo greitį, lokalizacijos paklaida išaugo iki 0,33 m, kas toliau apsunkina orientavimąsi aplinkoje.

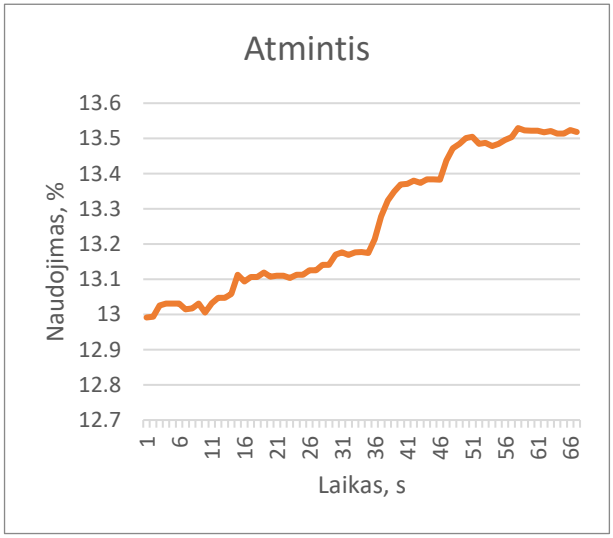


4.17 pav. *Google Cartographer* sudarytas 2 laboratorijų aplinkos žemėlapis robotui judant 15 cm/s greičiu

4.18 pav. atsispindi, jog procesoriaus sunaudojimas pasiekė maksimalią vertę 58 sekundę. Vidutiniškai buvo nustatytas 37,16 % naudojimas, kuris yra mažesnis nei pastarųjų 5 cm/s ir 10 cm/s eksperimentų. Sunaudotas atminties kiekis taip pat buvo truputį mažesnis – 21,06 MB.



4.18 pav. *Google Cartographer CPU* naudojimas robotui judant 15 cm/s



4.19 pav. *Google Cartographer* operatyviosios atminties naudojimas robotui judant 15 cm/s

4.2 lentelėje pateikiami gauti *Google Cartographer* rodikliai robotui veikiant skirtingais greičiais padidintoje tiriamoje aplinkoje. Lyginant gautus rezultatus su vienos laboratorijos gautais rezultatais pateiktais 4.1 lentelėje, galima įžvelgti, jog lokalizacijos paklaida padidėjo apie 3 kartus visais skirtingais roboto judėjimo greičiais. Atitinkamai ~4 % padidėjo vidutinis *CPU* ir ~17 MB atminties naudojimas. Išmatuotos aplinkos ilgis ir plotis visais atvejais išliko panašus. *CPU* ir atminties sunaudojimo išaugimas yra tiesiogiai susijęs su matuojamos aplinkos plotu, nes atsiranda didesnis informacijos kiekis, kuris turi būti saugomas ir apdorojamas valdiklio. Iš gautų *Google Cartographer* rezultatų galima teigti, kad naudojamas algoritmas su lazeriniu atstumo jutikliu yra labiau tinkamas nedidelėse aplinkose, nes lokalizacijos paklaida yra trigubai mažesnė nei dvigubo ploto aplinkoje.

4.2 lentelė. *Google Cartographer* rodikliai skirtingais judėjimo greičiais didelėje aplinkoje

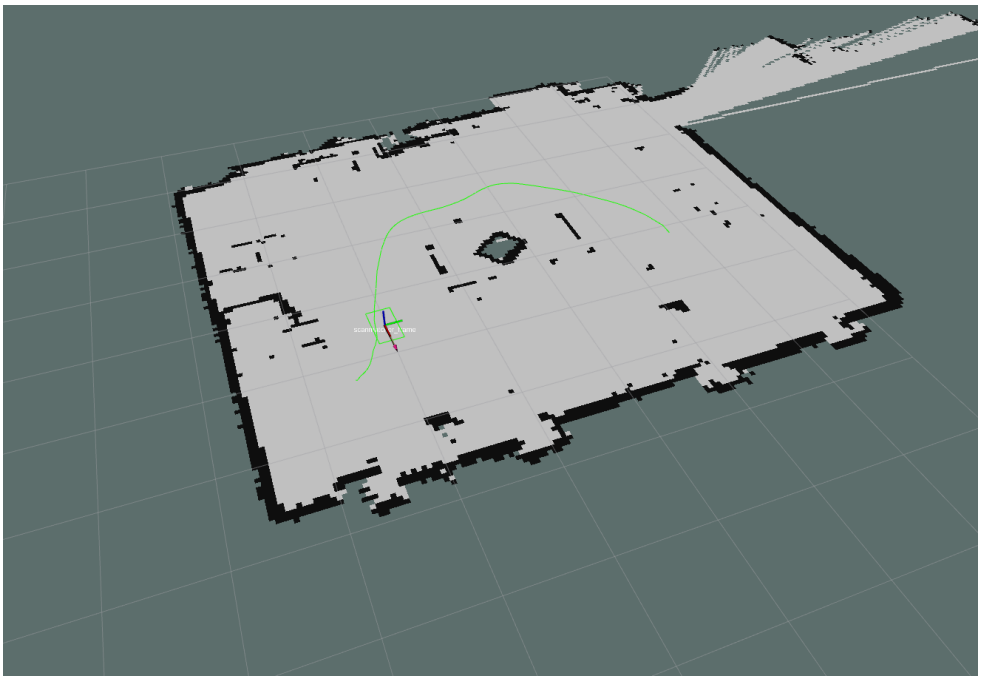
Greitis, cm/s	**Lokalizacijos paklaida, m**	**Ilgis, m**	**Plotis, m**	**CPU, %**	**RAM, MB**
5	0,22	14,14	5,88	42,62	29,87
10	0,25	14,12	5,81	39,21	30,64
15	0,33	14,15	5,84	37,16	21,06
Vidutinė reikšmė:	**0,27**	**14,14**	**5,84**	**39,66**	**27,19**

4.2. *Hector SLAM* metodas

Algoritmo tyrime buvo naudojamas tik lazerinis atstumo jutiklis.

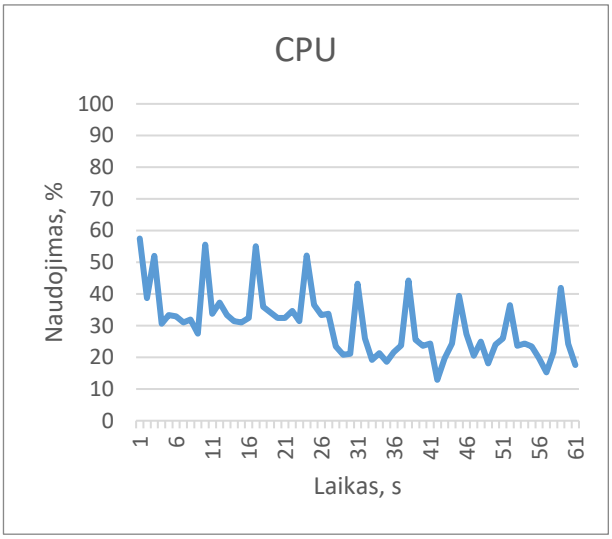
Žemėlapio sudarymas ir lokalizacija mažoje aplinkoje

4.20 pav. atvaizduojamas *Hector SLAM* algoritmu sudarytas aplinkos žemėlapis, kai robotas juda 5 cm/s greičiu. Žemėlapis yra sudarytas gana tiksliai ir atspindi tikrosios aplinkos kontūrus. Nustatyta lokalizacijos paklaida – 0,45 m., kas veda prie didelių netikslumų ir sunkumų tinkamai naviguojant aplinkoje, nes su nustatyta lokalizacijos paklaida tampa sudėtinga tikslią roboto poziciją ir objektus pažymėtus žemėlapyje.

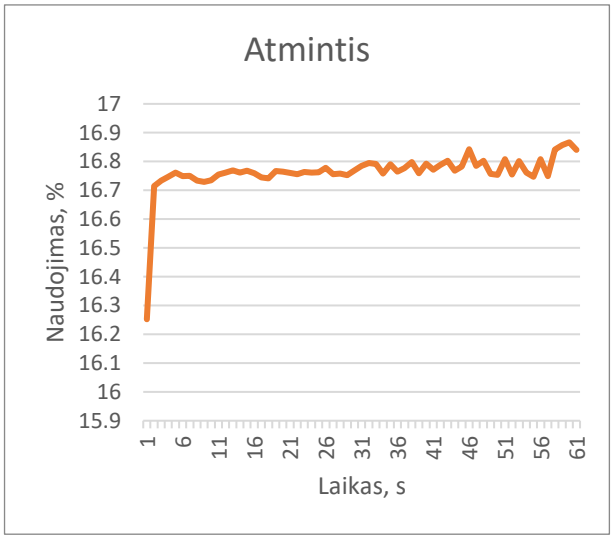


4.20 pav. *Hector SLAM* sudarytas aplinkos žemėlapis robotui judant 5 cm/s greičiu

4.21 pav. pateiktame procesoriaus naudojimo grafike atsispindi, jog maksimali 57,5 % vertė pasiekta pirmą sekundę. Vidutinis *CPU* naudojimas buvo 30,24 %, o sunaudota atmintis pasiekė 23,51 MB (4.22 pav.). Pagal našumo rodiklius valdiklio parametrai negalėjo turėti didelės įtakos roboto lokalizavimui, nes eksperimento metu nebuvo pasiektas maksimalus procesoriaus ar atminties naudojimas.

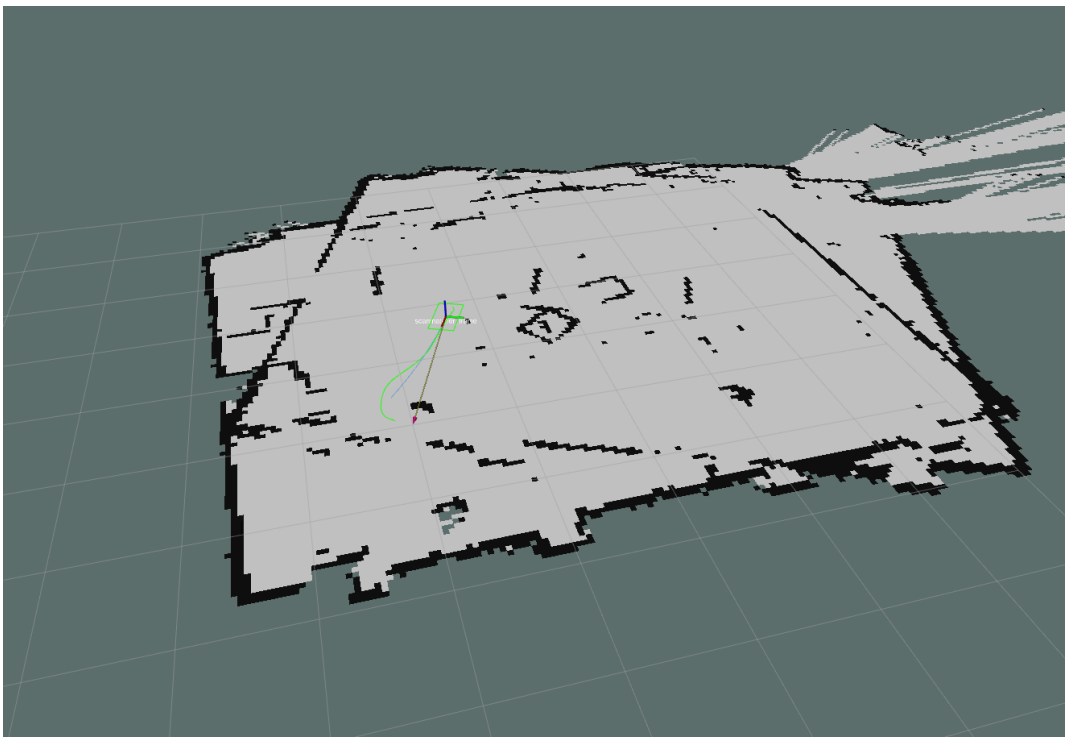


4.21 pav. *Hector SLAM CPU* naudojimas robotui judant 5 cm/s



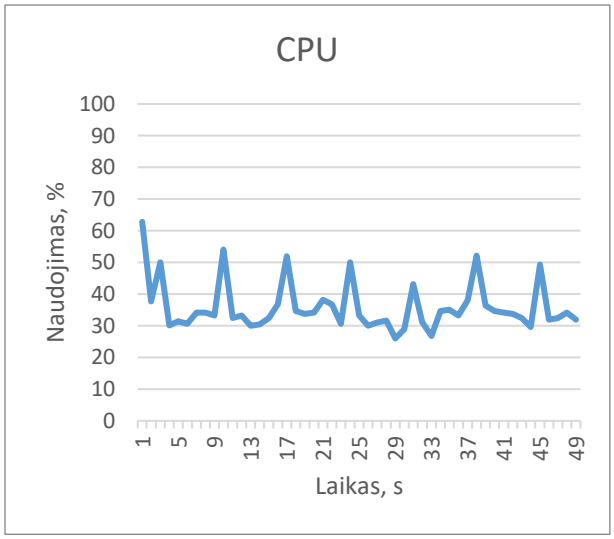
4.22 pav. *Hector SLAM* operatyviosios atminties naudojimas robotui judant 5 cm/s

4.23 pav. pateikiamas *Hector SLAM* sudarytas žemėlapis mobiliai platformai judant 10 cm/s greičiu. Dėl didesnio greičio sudaryto žemėlapio kokybė tapo prastesnė, ypač dėl staigesnių įvykdytų posūkių. Lokalizacijos paklaida atitinkamai išaugo iki 1,82 m robotui grįžus į savo pradinę poziciją.

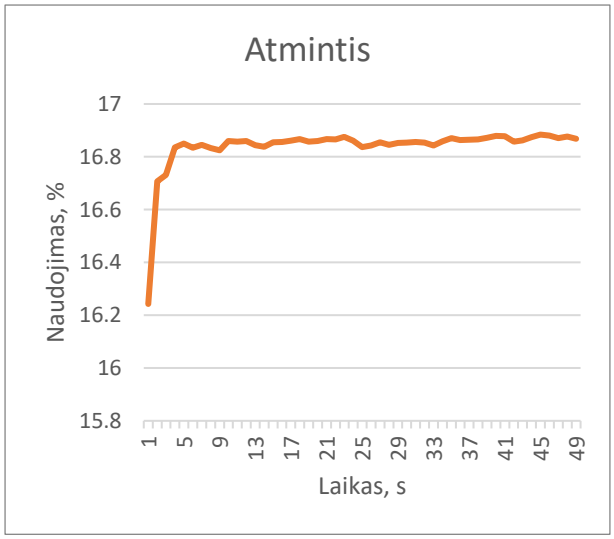


4.23 pav. *Hector SLAM* sudarytas aplinkos žemėlapis robotui judant 10 cm/s greičiu

4.24 pav. procesoriaus sunaudojimo grafike didelių šuolių nepastebima ir *Raspberry* valdiklis veikia techninių parametrų galimybių ribose. Maksimali pasiekta *CPU* vertė – 62,8 % pirmąją matavimo sekundę, o vidutiniškai nustatytas 35,91 % naudojimas. 4.25 pav. pateikti reikalingos atminties sunaudojimo rezultatai padidėjo nežymiai iki 25,02 MB.



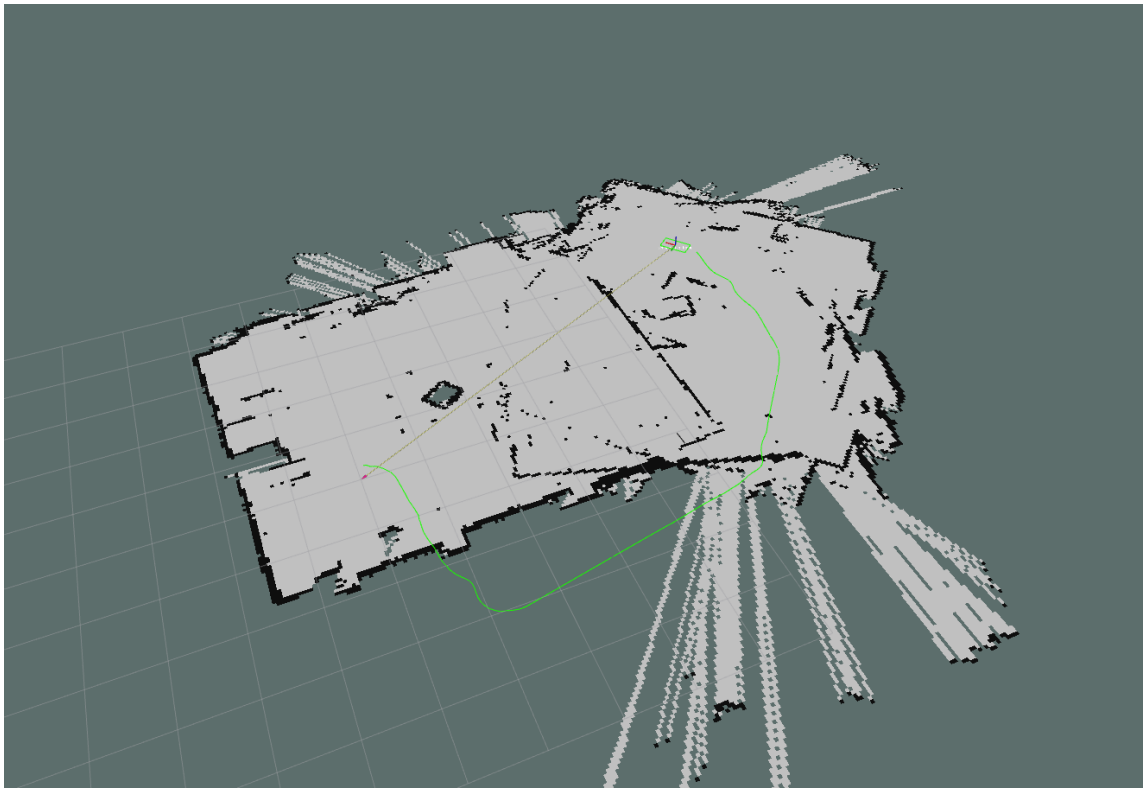
4.24 pav. *Hector SLAM CPU* naudojimas robotui judant 10 cm/s



4.25 pav. *Hector SLAM* operatyviosios atminties naudojimas robotui judant 10 cm/s

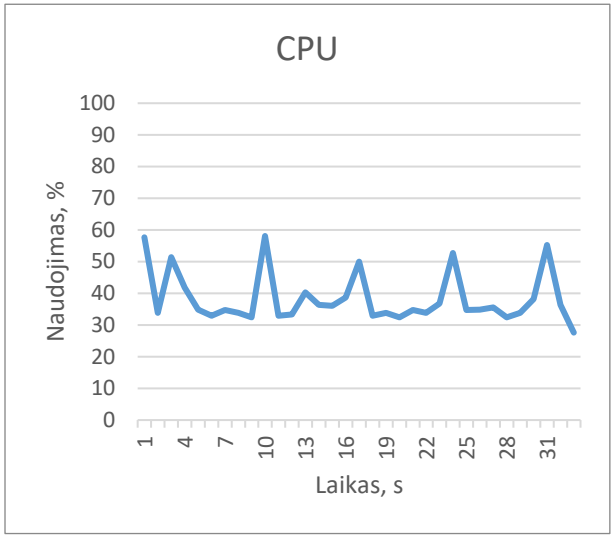
4.26 pav. praktiškai judėjimas aplinkoje 15 cm/s su *Hector SLAM* algoritmu sukelia didelių problemų žemėlapio ir lokalizacijos tikslumui. Mobilios platformos judėjimas dideliu greičiu sukelia problemų

tiksliau roboto pozicijos nustatymui, o pats žemėlapis taip pat nebeatspindi realybės dėl netinkamo roboto lokalizavimo.

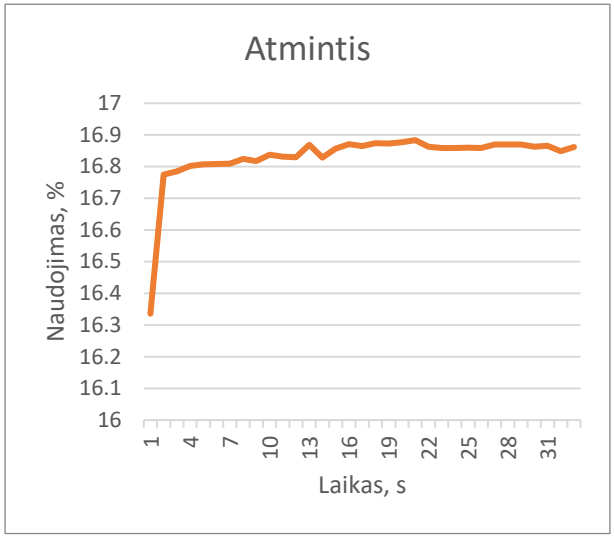


4.26 pav. *Hector SLAM* sudarytas aplinkos žemėlapis robotui judant 15 cm/s greičiu

4.27 pav. pateiktame procesoriaus naudojimo grafike, *CPU* naudojimas išlieka tarp ribų ir didžiausia naudojimo vertė 58,1 % pasiekiama 10 sekundę. Vidutiniškai procesoriaus naudojimas pakilo iki 38,33 % padidinus roboto judėjimo greitį iki 15 cm/s. 4.28 pav. atminties sunaudojimas buvo truputį mažesnis nei pastarųjų *Hector SLAM* eksperimentų ir siekė 21,04 MB.



4.27 pav. *Hector SLAM CPU* naudojimas robotui judant 15 cm/s



4.28 pav. *Hector SLAM* operatyviosios atminties naudojimas robotui judant 15 cm/s

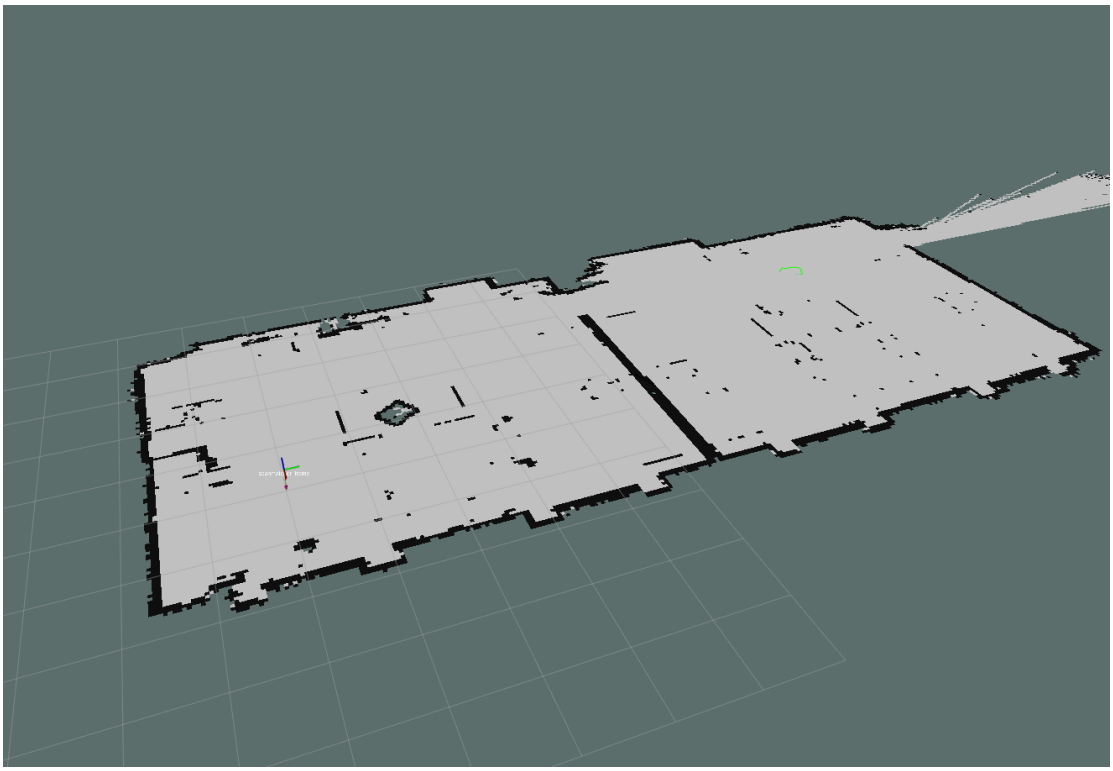
4.3 lentelėje pateikiami *Hector SLAM* metodo lokalizacijos ir žemėlapio sudarymo nustatytų rodiklių rezultatai skirtingais mobiliosios platformos veikimo aplinkoje greičiais. Iš lokalizacijos paklaidos atsispindi, jog *Hector SLAM* algoritmas yra tinkamas tik tuo atveju, jei nustatytas roboto greitis yra itin mažas – 5 cm/s, tačiau ir tuo atveju lokalizacijos paklaida yra gana aukšta (0,45 m) nei tarkim *Google Cartographer* algoritmo (žr. 4.1 lentelę), kuris žymiai tiksliau atlieka lokalizaciją (0.07 m) tokio pačio dydžio aplinkoje ir su vienoda įranga. Vidutiniškai *Hector SLAM* procesoriaus sunaudojimas 34,83 % taip pat buvo nedaug mažesnis nei *Google Cartographer* – 35,93 %, o operatyviosios atminties *Hector SLAM* vidutiniškai sunaudojo dvigubai daugiau nei (23,19 MB) nei *Google Cartographer* metodas(10,26 MB).

4.3 lentelė. *Hector SLAM* rodikliai skirtingais judėjimo greičiais mažoje aplinkoje

Greitis, cm/s	**Lokalizacijos paklaida, m**	**Ilgis, m**	**Plotis, m**	**CPU, %**	**RAM, MB**
5	0,45	6,86	5,92	30,24	23,51
10	1,82	6,88	5,95	35,91	25,02
15	7,85	–	–	38,33	21,04
Vidutinė reikšmė:	**3,37**	**6,87**	**5,93**	**34,83**	**23,19**

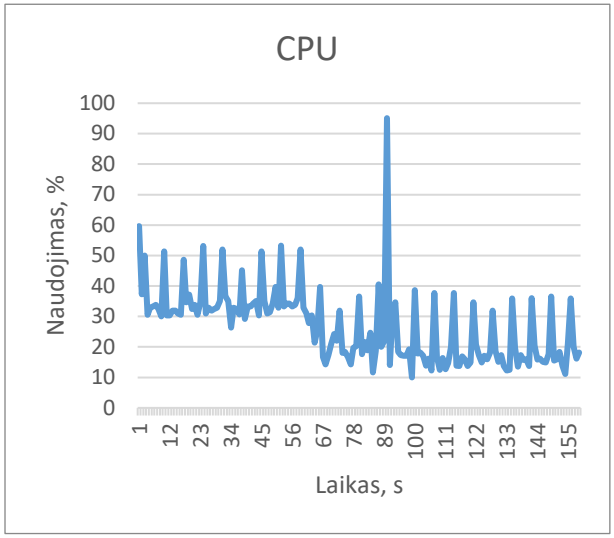
Žemėlapio sudarymas ir lokalizacija didelėje aplinkoje

4.29 pav. *Hector SLAM* metodas gana tiksliai sudaro dviejų laboratorijų aplinkos žemėlapį judant 5 cm/s greičiu. Lokalizacijos paklaida taip pat išlieka panaši kaip ir vienos laboratorijos žemėlapio sudarymo atžvilgiu – 0,48 m, kas rodo, jog net dvigubai padidinus tiriamos aplinkos plotą, *Hector SLAM* pozicijos nustatymo galimybės išlieka beveik vienodos kaip ir vienos tiriamos laboratorijos.

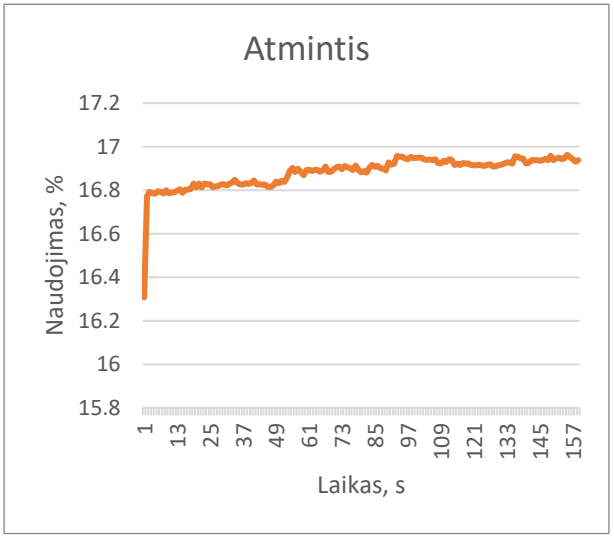


4.29 pav. *Hector SLAM* sudarytas 2 laboratorijų aplinkos žemėlapis robotui judant 5 cm/s greičiu

4.30 pav. pateiktame procesoriaus naudojimo grafike užfiksuotas didžiausias naudojimas 90 sekundę su 95,1 %. Vidutinis *CPU* naudojimas buvo 26,69 %, kas yra ~4 % mažiau nei to pačio *Hector SLAM* atlikto tyrimo vienoje laboratorijoje. 4.31 pav. operatyviosios atminties iš viso buvo sunaudota 25,22 MB žemėlapiui sudaryti.

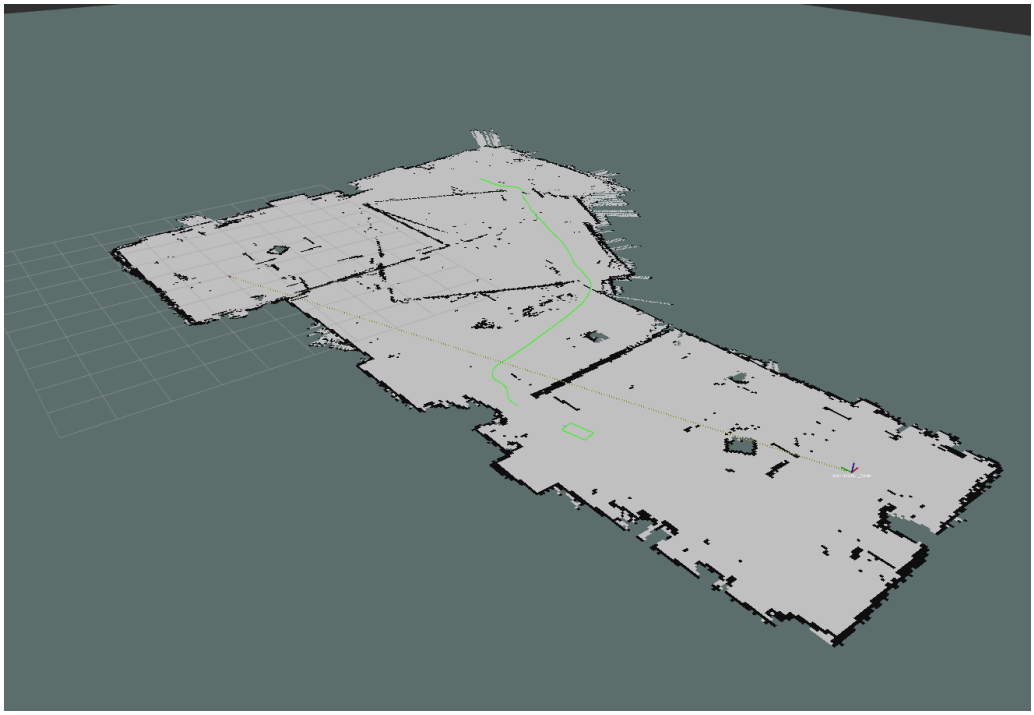


4.30 pav. *Hector SLAM CPU* naudojimas robotui judant 5 cm/s



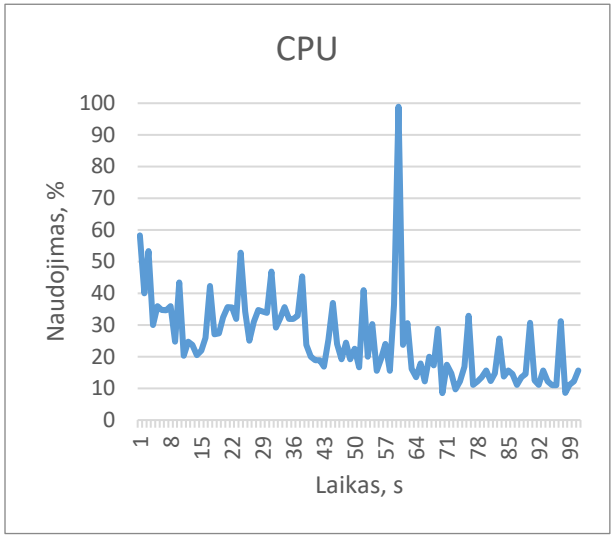
4.31 pav. *Hector SLAM* operatyviosios atminties naudojimas robotui judant 5 cm/s

4.32 pav. padidinus roboto greitį iki 10 cm/s matoma analogiška problema kaip ir vienos laboratorijos žemėlapio sudarymo atveju (4.23 pav.). Tik šiuo atveju sudarytas žemėlapis yra visiškai netikslus ir pasikartoja bei persidengia keletą kartų. Atitinkamai roboto pozicija tampa netinkamai interpretuojama, kas veda prie naujų žemėlapių kūrimo ir praktiškai algoritmo nebūtų galima panaudoti dėl didelių lokalizacijos netikslumų.

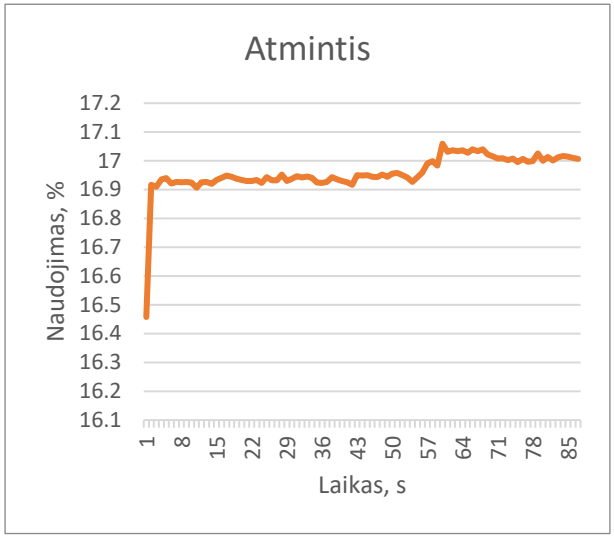


4.32 pav. *Hector SLAM* sudarytas 2 laboratorijų aplinkos žemėlapis robotui judant 10 cm/s greičiu

4.33 pav. CPU naudojimas viso eksperimento metu išliko pastovus ir tik 60 sekundę pasiekė 98,9 % naudojimą. Vidutiniškai *CPU* naudojimas buvo 25,18 %. 4.34 pav. operatyviosios atminties sunaudojimas pasiekė 21,76 MB.

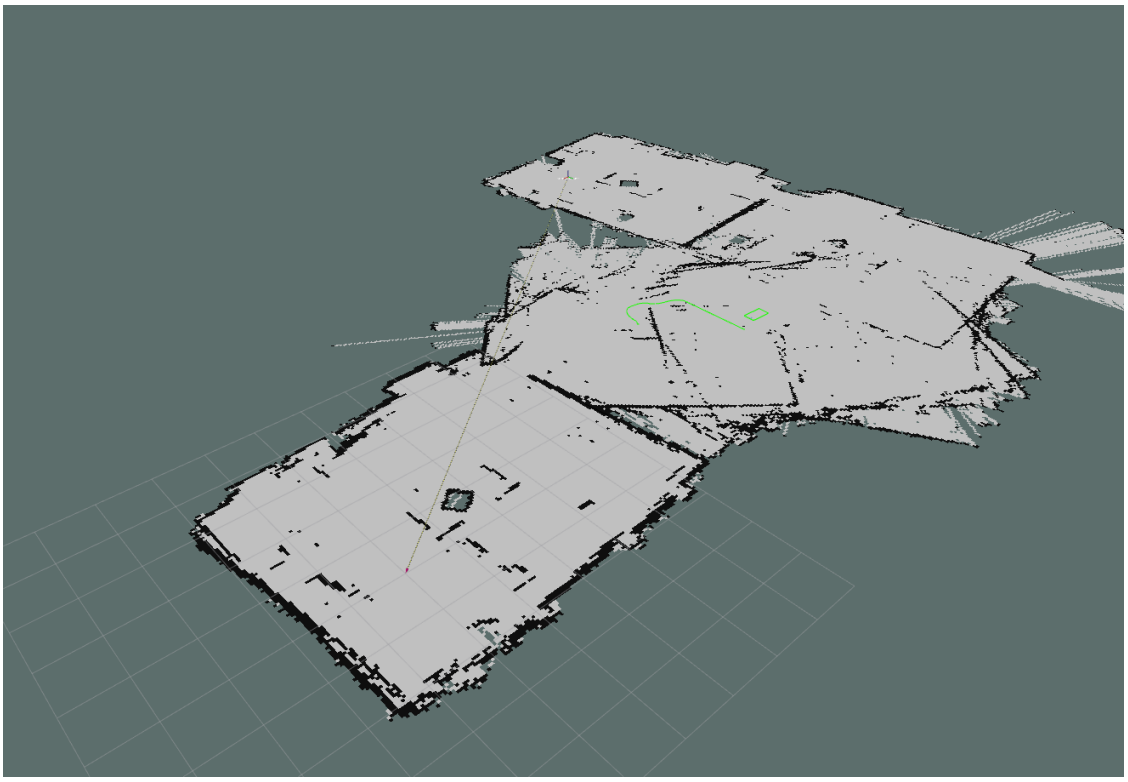


4.33 pav. *Hector SLAM CPU* naudojimas robotui judant 10 cm/s



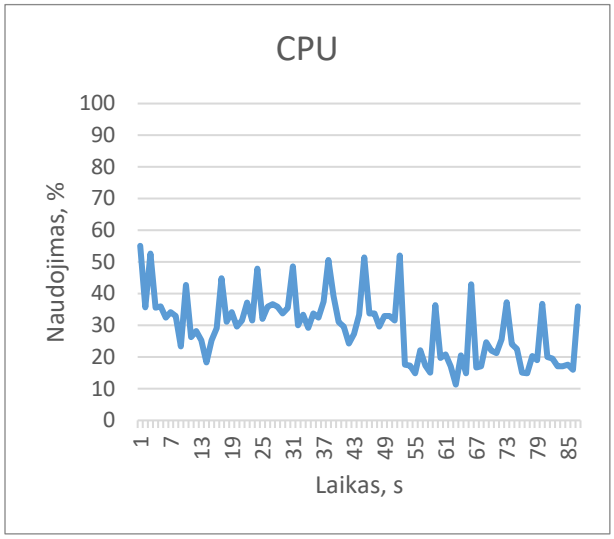
4.34 pav. *Hector SLAM* operatyviosios atminties naudojimas robotui judant 10 cm/s

4.35 pav. atliktame eksperimente, kai robotas veikia 15 cm/s greičiu susiduriama su dar didesnėmis lokalizacijos problemomis. Veikdamas dideliu greičiu robotas praranda savo padėtį ir pradeda netinkamai interpretuoti aplink ją supančią aplinką. Kaip ir 10 cm/s judėjimo greičiu, *Hector SLAM* tampa nebetinkamas lokalizacijai ir žemėlapio sudarymui itin dideliais greičiais.

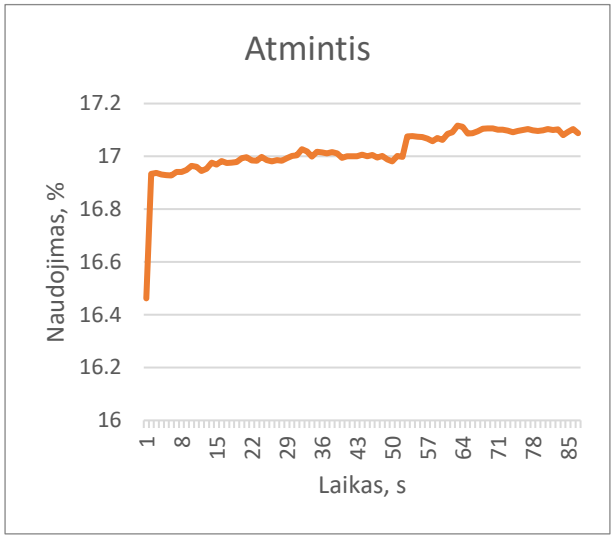


4.35 pav. *Hector SLAM* sudarytas 2 laboratorijų aplinkos žemėlapis robotui judant 15 cm/s greičiu

4.36 pav. procesoriaus naudojimo didžiausia reikšmė pasiekta pirmą sekundę – 55,1 %. Vidutiniškai *CPU* sunaudojimas buvo 29,39 % 4.37 pav. pateiktame *RAM* naudojimo grafike sunaudota atminties žemėlapio sudarymui pasiekė 25,02 MB.



4.36 pav. *Hector SLAM CPU* naudojimas robotui judant 15 cm/s



4.37 pav. *Hector SLAM* operatyviosios atminties naudojimas robotui judant 15 cm/s

4.4 lentelėje atsispindi, kad *Hector SLAM* didelėse aplinkose yra tinkamas tik mobiliai platformai veikiant nedideliais greičiais (iki 5 cm/s). Nors lokalizacijos paklaida beveik nepasikeitė nuo vienos laboratorijos 5 cm/s greičio matavimų, kitais atvejais *Hector SLAM* algoritmas nesugeba teisingai įvertinti roboto pozicijos dėl staigiai įvykstančių aplinkos pasikeitimų. Šios problemos sprendimui padėtų didesnio dažnio ir tikslesnis lazerinis atstumo jutiklis, kuris dažniau pateikia informaciją apie robotą supančius orientyrus. Taip pat pagelbėtų *IMU* jutiklis, kuris kompensuotų staigių posūkių įvertinimą.

Procesoriaus naudojimas sumažėjo lyginant su atliktu tyrimu vienoje laboratorijoje, tačiau toks sumažėjimas galėjo atsirasti dėl to, kad žemėlapio sudarymas vyko ilgesnį laiką, kur atsiradę didesni naudojimo šuoliai turėjo mažesnę įtaką bendram *CPU* sunaudojimui. Taip pat šis sumažėjimas galėjo atsirasti ir dėl to, jog *move_base* navigacija automatiškai išsijungdavo roboto įvertintai pozicijai atsidūrus už žemėlapio ribų.

Apibendrinant gautus rezultatus galima teigti, kad *Hector SLAM* algoritmas gali būti tinkamai pritaikytas su lazerinio atstumo jutikliu tik nedideliais greičiais ir atliekant švelnius posūkius. Kita vertus, tas pats *Google Cartographer* algoritmas pasiekė žymiai tikslesnius rezultatus naudojant vienodą lazerinį atstumo jutiklį (žr. 4.2 lentelę)

4.4 lentelė. *Hector SLAM* rodikliai skirtingais judėjimo greičiais didelėje aplinkoje

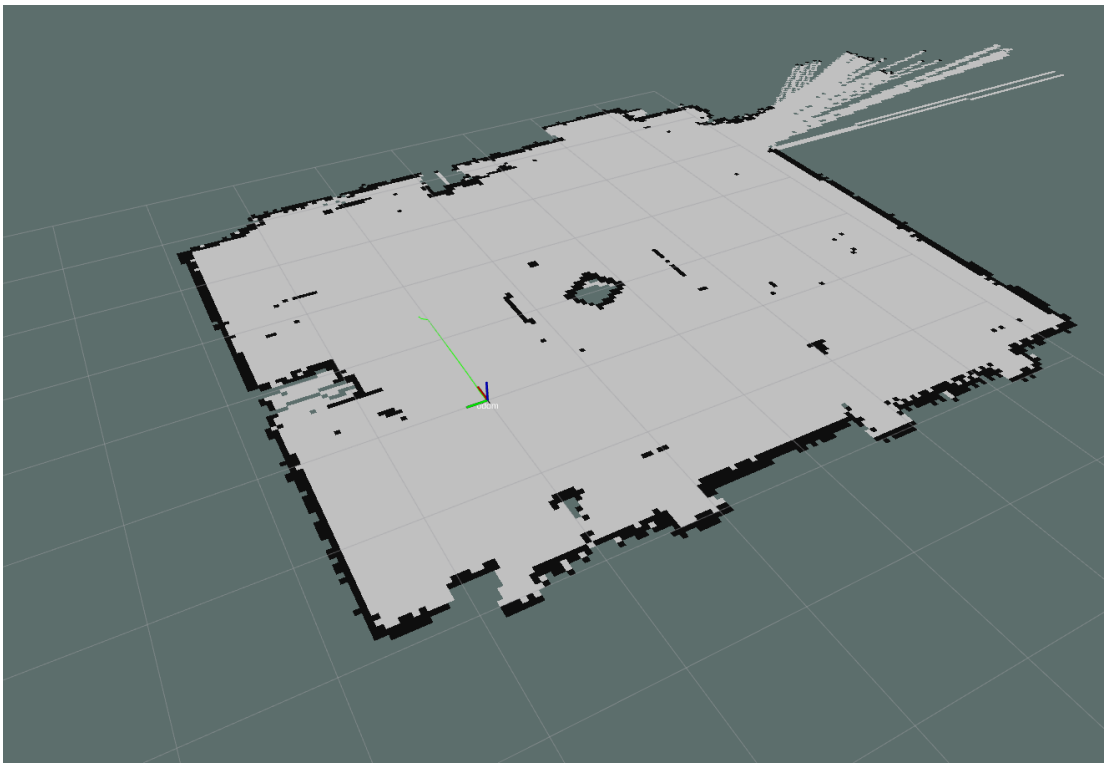
Greitis, cm/s	**Lokalizacijos paklaida, m**	**Ilgis, m**	**Plotis, m**	**CPU, %**	**RAM, MB**
5	0,48	14,21	5,89	26,69	25,22
10	15,20	–	–	25,18	21,76
15	16,50	–	–	29,39	25,02
Vidutinė reikšmė:	**10,73**	**14,21**	**5,89**	**27,09**	**24,00**

4.3. *GMapping* metodas

Eksperimentuose naudojamas lazerinis atstumo jutiklis ir mobiliosios platformos enkoderiai roboto odometrijai įvertinti.

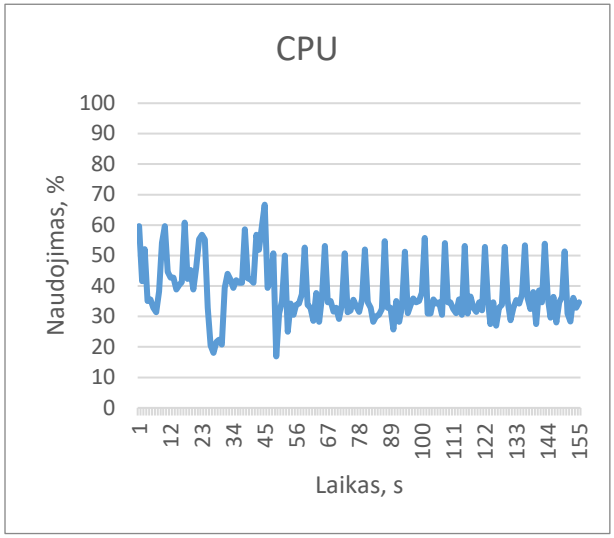
Žemėlapio sudarymas ir lokalizacija mažoje aplinkoje

4.38 pav. atvaizduojamas sudarytas žemėlapis su *GMapping* algoritmu ir mobiliai platformai veikiant 5 cm/s greičiu. Sudarytas žemėlapis tiksliai atspindi realios aplinkos kontūrus ir jame pastebima mažai triukšmo. Nustatyta lokalizacijos paklaida robotui grįžus į pradinę poziciją – 0,05 m. Gauta paklaida yra itin maža, todėl galima tiksli roboto navigacija aplinkoje teisingai įvertinant aplinkos kliūtis.

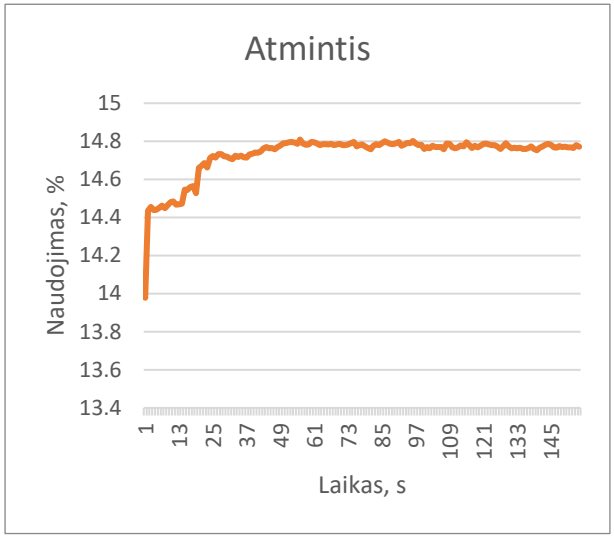


4.38 pav. *GMapping* sudarytas aplinkos žemėlapis robotui judant 5 cm/s greičiu

4.39 pav. pateiktame procesoriaus naudojimo grafike atsispindi didžiausia pasiekta reikšmė 66,7 % 45 sekundę. Vidutinis CPU naudojimas buvo – 37,93 %. 4.40 pav. atsispindi atminties sunaudojimas, kuris viso lokalizacijos ir aplinkos kartografavimo proceso metu pasiekė 31,75 MB. Procesoriaus sunaudojimas išliko panašus kaip ir *Google Cartographer* bei *Hector SLAM* metodų, o sunaudotas atminties kiekis yra didesnis už abu metodus.

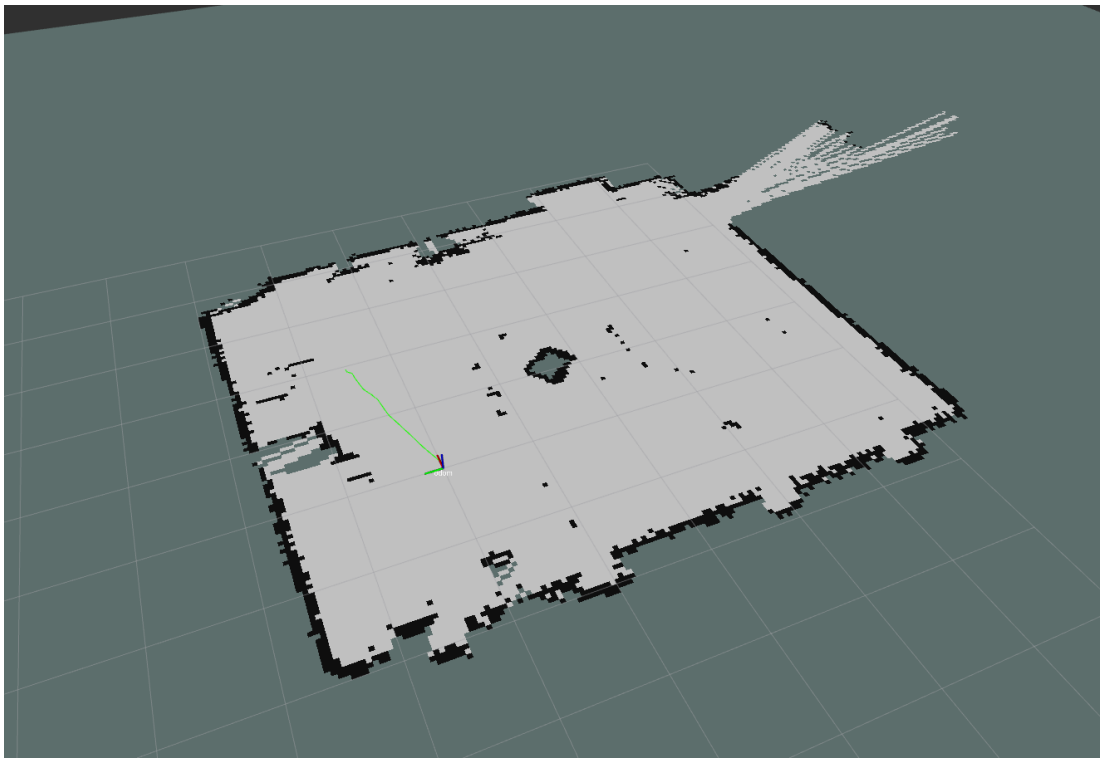


4.39 pav. *GMapping CPU* naudojimas robotui judant 5 cm/s



4.40 pav. *GMapping* operatyviosios atminties naudojimas robotui judant 5 cm/s

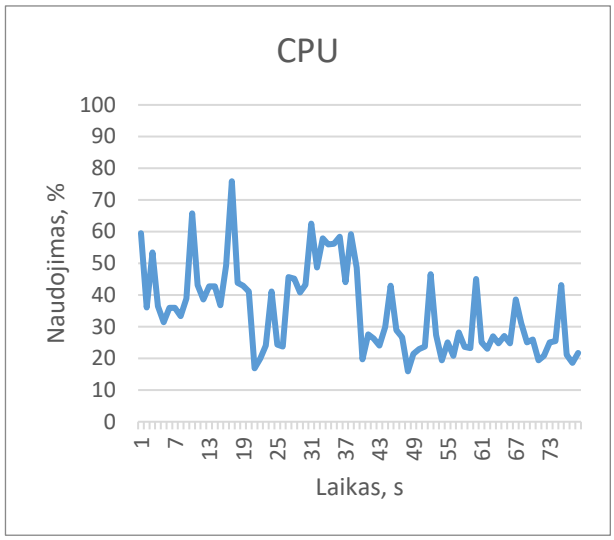
4.41 pav. padidinus roboto judėjimo greitį iki 10 cm/s, didesnis triukšmas sudarytame žemėlapyje nėra pastebimas ir jo tikslumas išlieka panašus kaip ir su lėtesniu greičiu. Gauta lokalizacijos paklaida iš esmės nepakito – 0,06 m. Net ir didesniu greičiu algoritmas sugebėjo sudaryti tikslų žemėlapį bei lokalizuoti robotą su maža paklaida.



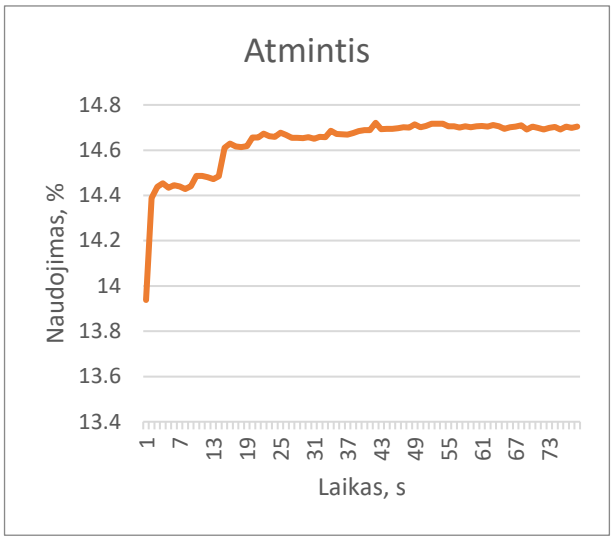
4.41 pav. *GMapping* sudarytas aplinkos žemėlapis robotui judant 10 cm/s greičiu

4.42 pav. procesoriaus naudojimo grafike pastebimas didžiausias naudojimas 17 sekundę su 75,9 %. Vidutiniškai CPU naudojimas siekė 35,09 %, kas iš esmės nepakito nuo praeito matavimo. 4.43 pav.

pateiktas atminties naudojimo grafikas atspindi, kad reikalingas operatyviosios atminties kiekis išliko beveik vienodas – 30,67 MB.

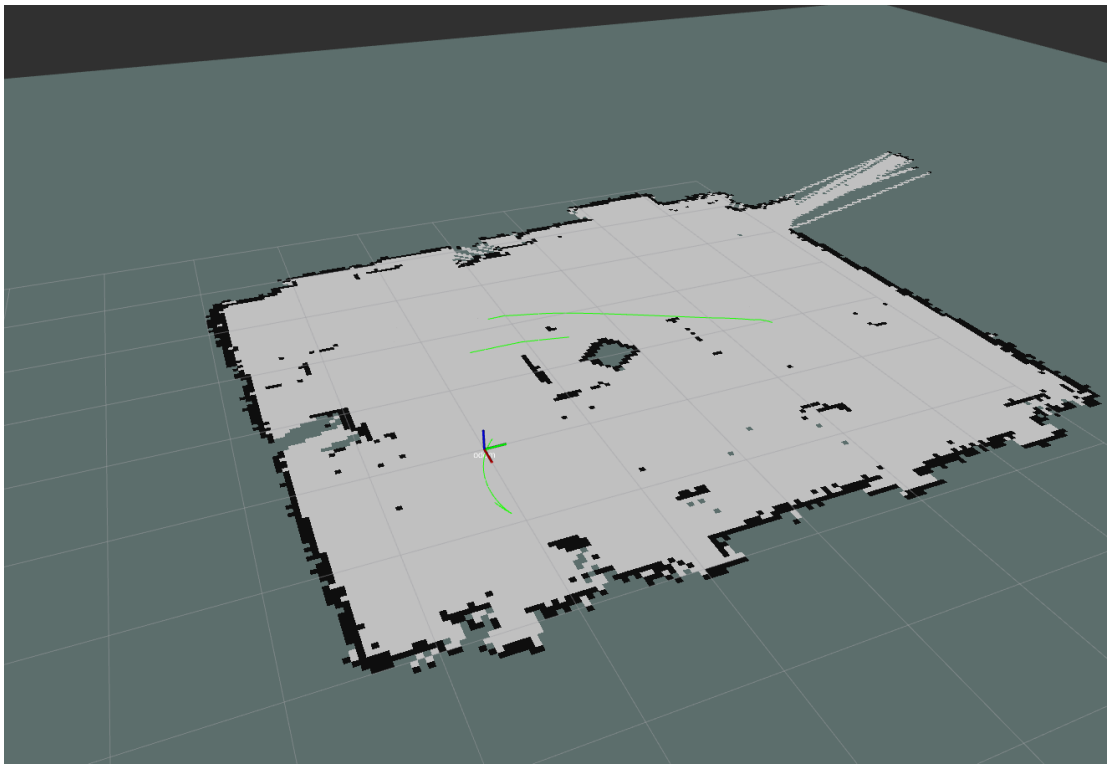


4.42 pav. *GMapping CPU* naudojimas robotui judant 10 cm/s



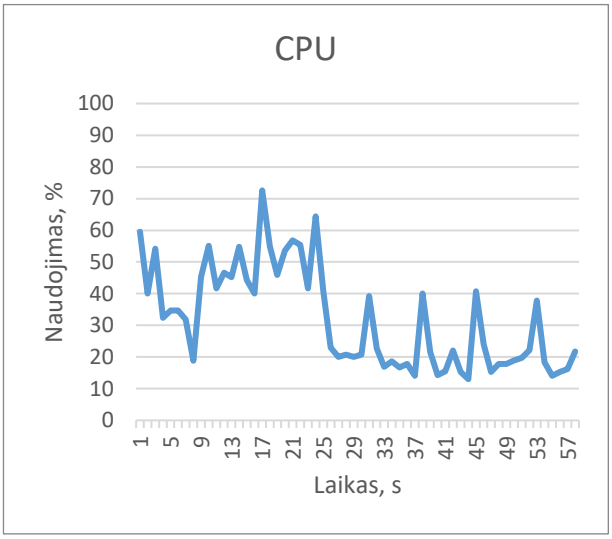
4.43 pav. *GMapping* operatyviosios atminties naudojimas robotui judant 10 cm/s

4.44 pav. atvaizduojama žemėlapio kartografija, kai robotas aplinkoje juda 15 cm/s greičiu. Žemėlapis išliko analogiškas kaip ir 10 cm/s atveju. Triukšmas žemėlapyje yra šiek tiek didesnis, nes kai kuriose vietose matomi neužpildyti taškai, kurių nėra robotui veikiant mažesniais greičiais. Roboto lokalizacijos paklaida išliko maža ir su didesniu greičiu nepakito nuo 10 cm/s eksperimento – 0,06 m.

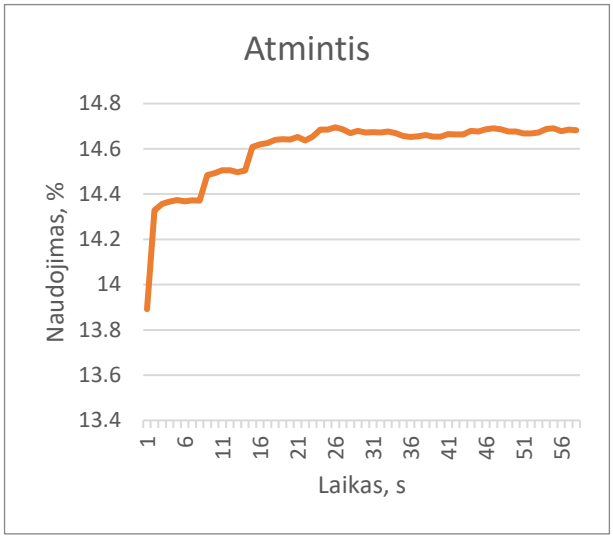


4.44 pav. *GMapping* sudarytas aplinkos žemėlapis robotui judant 15 cm/s greičiu

4.45 pav. didžiausia pasiekta *CPU* naudojimo reikšmė 72,6 % 17 sekundę. Vidutiniškai *CPU* naudojimas sumažėjo nuo lėtesniu greičiu atliktų *GMapping* eksperimentų – 32,03%. 4.46 pav. operatyviosios atminties sunaudojimas išliko panašus kaip ir ankstesniais eksperimentų atvejais – 31,63 MB.



4.45 pav. *GMapping CPU* naudojimas robotui judant 15 cm/s



4.46 pav. *GMapping* operatyviosios atminties naudojimas robotui judant 15 cm/s

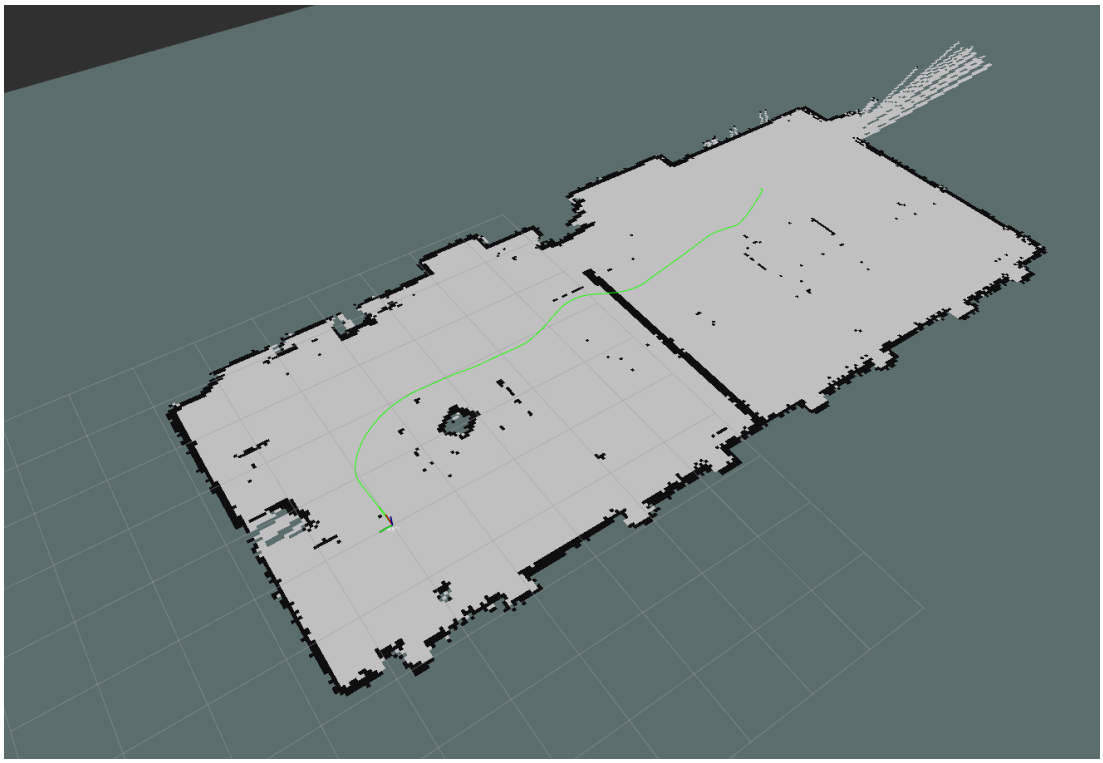
4.5 lentelėje pateikiami naudojamo *GMapping* algoritmo gauti rodikliai skirtingais mobiliosios platformos veikimo greičiais. Iš lokalizacijos paklaidos atsispindi, jog *GMapping* algoritmui greičio padidėjimas neturi įtakos roboto lokalizavimui, nes visais atvejais paklaida išliko itin maža. *GMapping* algoritmas geba tiksliau įvertinti roboto poziciją nei pastarieji tirti *Google Cartographer* (žr. 4.1 lentelę) ir *Hector SLAM* (žr. 4.3 lentelę) metodai.

4.5 lentelė. *GMapping* rodikliai skirtingais judėjimo greičiais mažoje aplinkoje

Greitis, cm/s	**Lokalizacijos paklaida, m**	**Ilgis, m**	**Plotis, m**	**CPU, %**	**RAM, MB**
5	0,05	6,95	5,90	37,93	31,75
10	0,06	6,94	5,81	35,09	30,67
15	0,06	6,98	5,83	32,03	31,63
Vidutinė reikšmė:	**0,06**	**6,96**	**5,85**	**35,02**	**31,35**

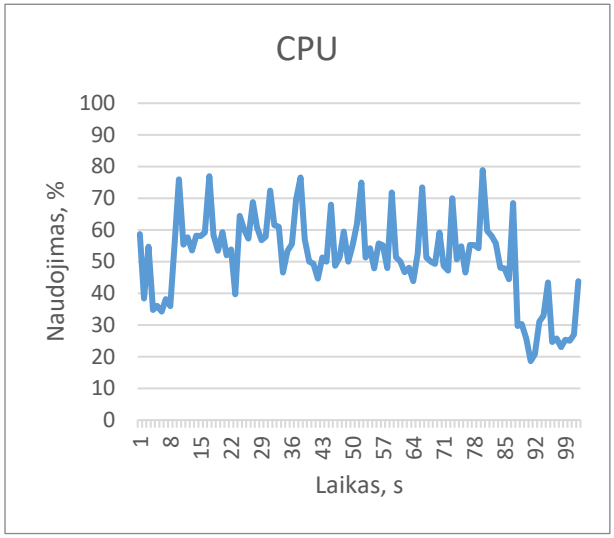
Žemėlapio sudarymas ir lokalizacija didelėje aplinkoje

4.47 pav. *GMapping* sudarytas aplinkos žemėlapis robotui judant 5 cm/s greičiu atsispindi *GMapping* sudarytas aplinkos žemėlapis robotui važiuojant 5 cm/s greičiu. Eksperimento metu nustatyta roboto lokalizacijos paklaida robotui grįžus į pradinę poziciją – 0,1 m. Gauta paklaida yra 2 kartus didesnė nei 4.38 pav. mažesnėje aplinkoje. Žemėlapio kokybė tiksliai pažymi aplinkos objektus ir sudarytame žemėlapyje nėra didelio triukšmo.

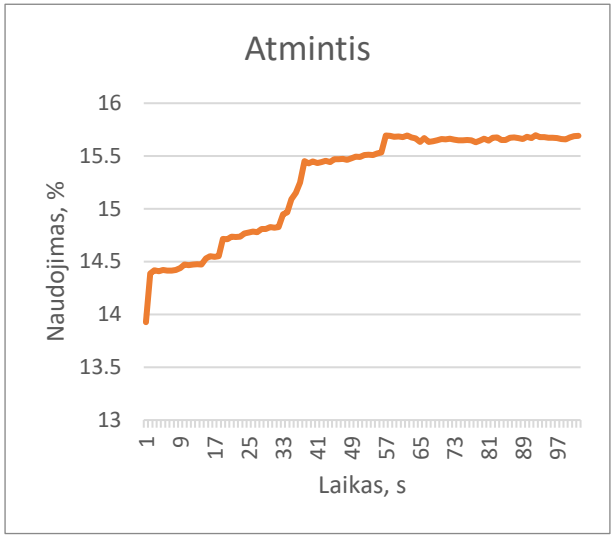


4.47 pav. *GMapping* sudarytas aplinkos žemėlapis robotui judant 5 cm/s greičiu

4.48 pav. maksimalus pasiektas *CPU* naudojimas buvo 78,9 % 80 eksperimento sekundę. Nustatytas vidutinis procesoriaus naudojimas – 51,72 %. Matomai dėl didesnės aplinkos padidėjo resursų sąnaudos lyginant su mažesne aplinka apie ~14 % (žr. 4.5 lentelę). 4.49 pav. operatyviosios atminties naudojimas pasiekė 70,51 MB, kas yra daugiau nei 2 kartus su gautais naudojimo rodikliais mažesnėje aplinkoje.

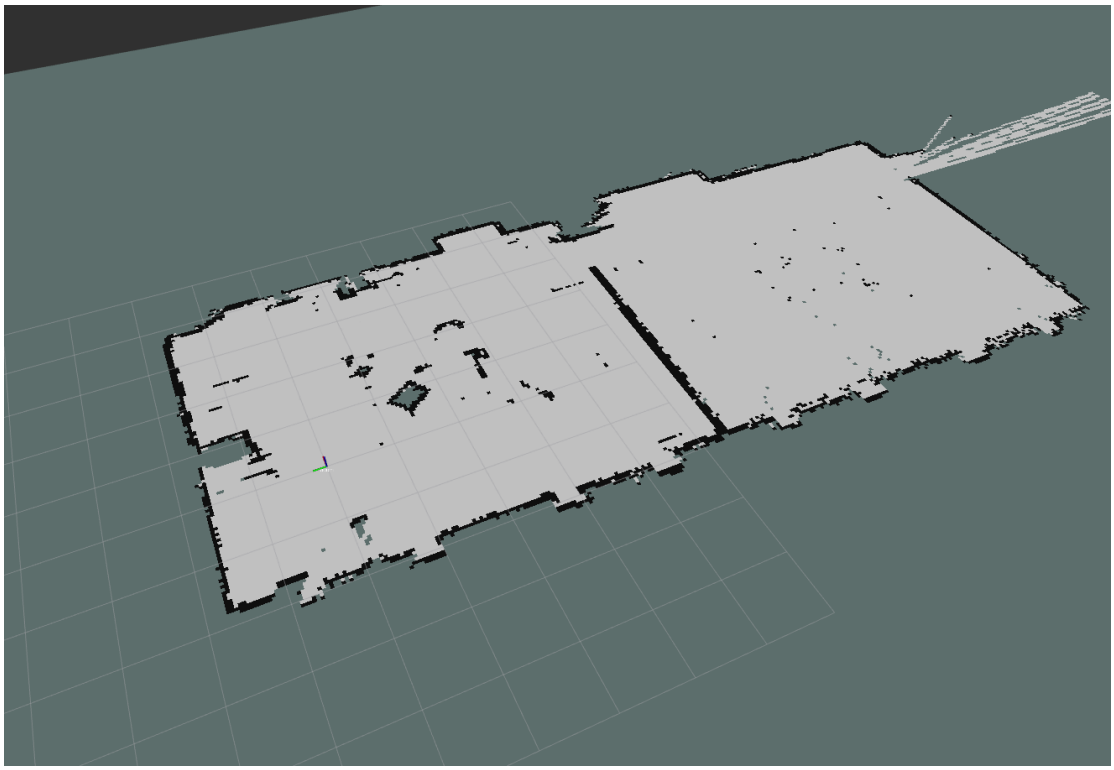


4.48 pav. *GMapping CPU* naudojimas robotui judant 5 cm/s



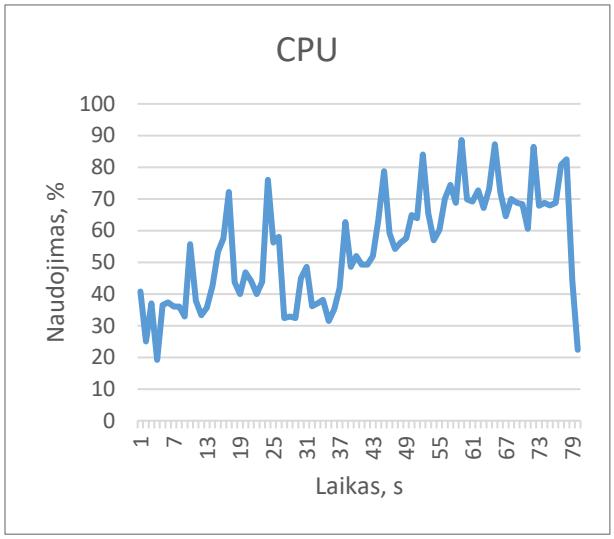
4.49 pav. *GMapping* operatyviosios atminties naudojimas robotui judant 5 cm/s

4.50 pav. didesnis triukšmas gautame žemėlapyje nėra įžvelgiamas su padidintu mobilios platformos judėjimu greičiui. Lokalizacijos paklaida lyginant su ankstesniu eksperimentu atliktu mažesniu greičiu padidėjo iki 0,16 m.

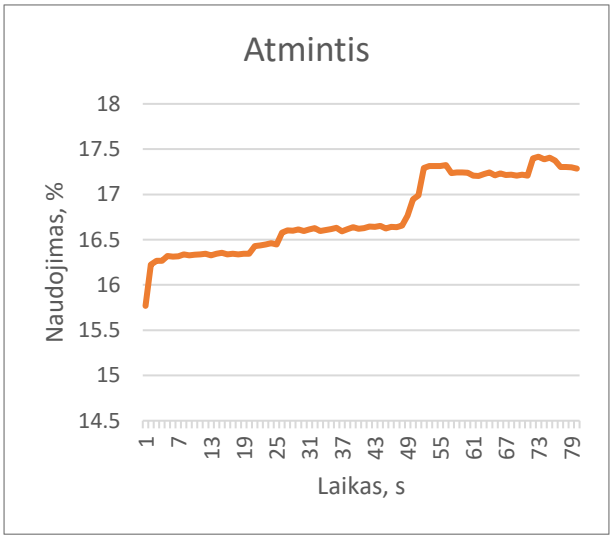


4.50 pav. *GMapping* sudarytas aplinkos žemėlapis robotui judant 10 cm/s greičiu

4.51 pav. užfiksuotas maksimalus *CPU* naudojimas – 88,6 % 59 matavimo sekundę. Vidutinis *CPU* sunaudojimas – 54,43 %. Procesoriaus naudojime pastebimi didesni šuoliai antroje matavimo dalyje nuo 40 sekundės, kur robotas tuo metu pradeda antrosios aplinkos žemėlapio sudarymą. 4.52 pav. nustatytas operatyviosios atminties sąnaudos – 60,62 MB, kas yra ~10 MB mažiau nei lėtesniu greičiu atlikto žemėlapio sudarymo.

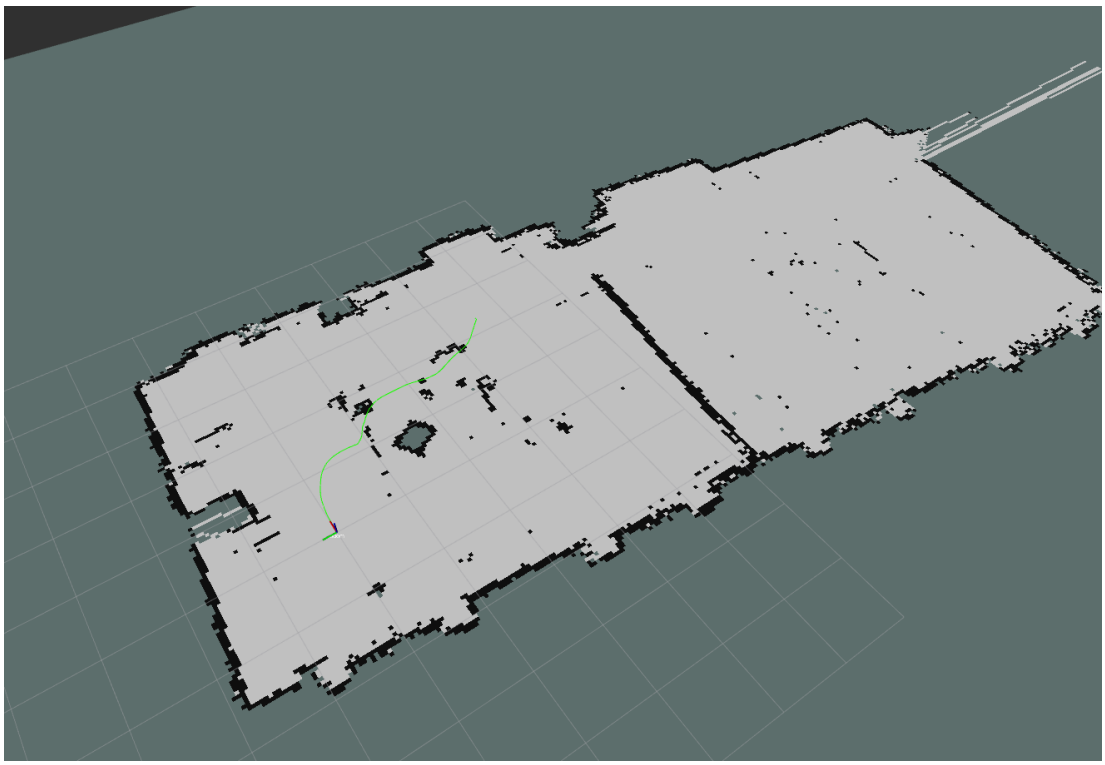


4.51 pav. *GMapping CPU* naudojimas robotui judant 10 cm/s



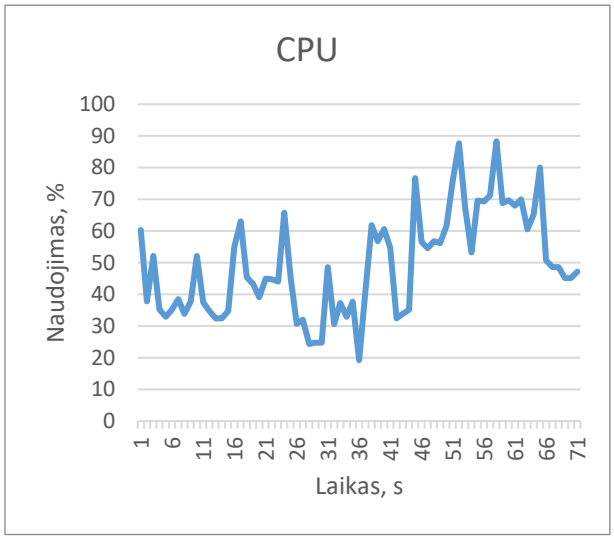
4.52 pav. *GMapping* operatyviosios atminties naudojimas robotui judant 10 cm/s

4.53 pav. padidinus roboto judėjimo greitį iki 15 cm/s pastebimas truputį didesnis triukšmas, kuris atsispindi ties sienų kontūrais, ypač gretimoje matuotoje aplinkoje. Lokalizacijos paklaida buvo nežymiai mažesnė už 10 cm/s atlikto eksperimento – 0,14 m, tačiau net ir tokia paklaida dar leidžia tinkamai lokalizuoti robotą aplinkoje.

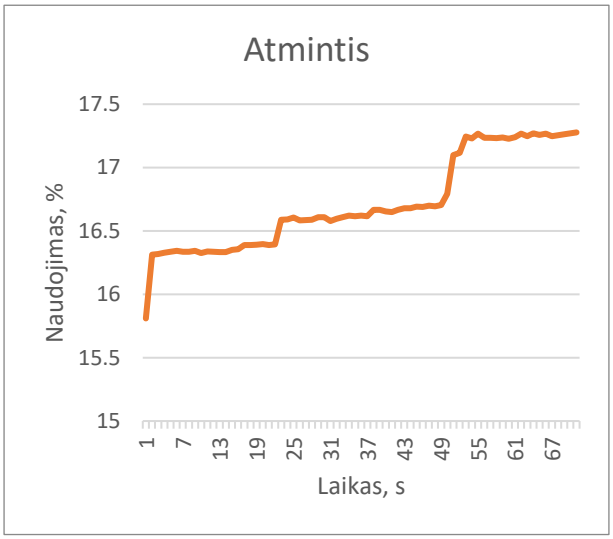


4.53 pav. *GMapping* sudarytas aplinkos žemėlapis robotui judant 15 cm/s greičiu

4.54 pav. nustatytas maksimalus procesoriaus naudojimas 58 sekundę su 88,3 %. Vidutiniškai *CPU* naudojimas buvo mažesnis už mažesnių greičių eksperimentų – 49,41 %. 4.55 pav. reikalingas operatyviosios atminties kiekis – 58,71 MB.



4.54 pav. *GMapping CPU* naudojimas robotui judant 15 cm/s



4.55 pav. *GMapping* operatyviosios atminties naudojimas robotui judant 15 cm/s

4.6 lentelėje pateikiami gauti eksperimentų rezultatai didesnėje aplinkoje ir skirtingais roboto judėjimo greičiais. Lyginant su mažesnės aplinkos rezultatais pateiktais 4.5 lentelėje, roboto lokalizacijos paklaida išaugo daugiau nei 2 kartus, kas rodo, jog roboto lokalizavimas ir žemėlapio kartografavimas didesnėje aplinkoje yra tiesiogiai susijęs. Atitinkamai nustatyti didesni resursų naudojimai: *CPU* naudojimas padidėjo apie ~15 % skirtingais greičiais lyginant su *GMapping* rodikliais mažesnėje aplinkoje, RAM naudojimas taip pat išaugo 2 kartus. Nors resursų sunaudojimas padidėjo, tačiau *GMapping* algoritmas yra tikslesnis už atliktus eksperimentus su *Hector SLAM* ir *Google Cartographer* metodais.

4.6 lentelė. *GMapping* rodikliai skirtingais judėjimo greičiais didelėje aplinkoje

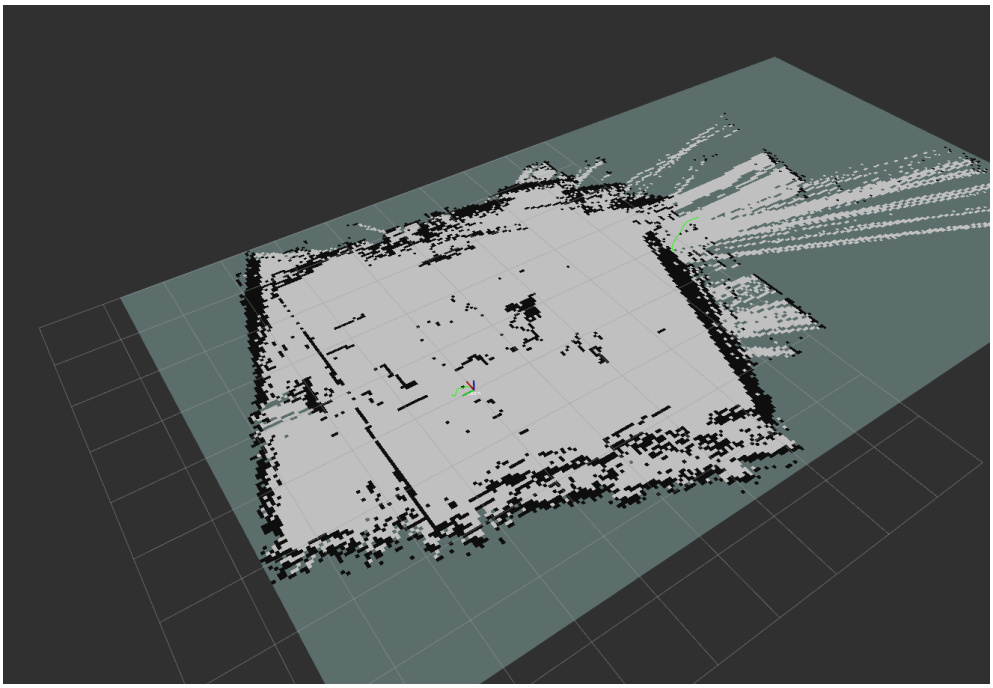
Greitis, cm/s	**Lokalizacijos paklaida, m**	**Ilgis, m**	**Plotis, m**	**CPU, %**	**RAM, MB**
5	0,10	14,19	5,87	51,21	70,51
10	0,16	14,12	5,79	54,53	60,62
15	0,14	13,94	5,85	49,41	58,71
Vidutinė reikšmė:	**0,13**	**14,08**	**5,84**	**51,72**	**63,28**

4.4. *Karto SLAM* metodas

Lokalizacijos ir žemėlapio sudarymui naudojamas lazerinis atstumo jutiklis ir roboto enkoderių duomenys odometrijai.

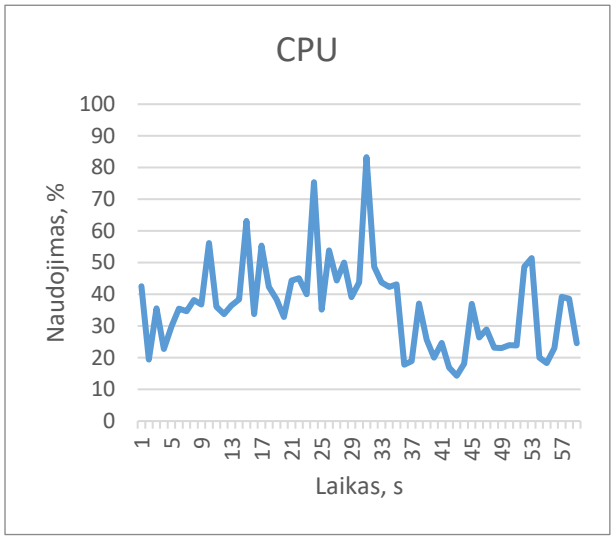
Žemėlapio sudarymas ir lokalizacija mažoje aplinkoje

4.56 pav. tikslaus žemėlapio su *Karto SLAM* algoritmu nepavyko sudaryti. Naudojant *Karto SLAM* metodą robotui sunkiai sekėsi atlikti lokalizaciją aplinkoje ypač po atlikto pasisukimo, kas lėmė neteisingą aplinkos interpretaciją net ir judant nedideliu greičiu. Sugeneruotame žemėlapyje yra daug triukšmo ir persidengimų dėl neteisingo roboto lokalizavimo. Robotui sugrįžus į pradinę poziciją nustatyta 1,12 m lokalizacijos paklaida, kuri yra itin aukšta lyginant su tirtais pastaraisiais metodais tokiu pačiu greičiu.

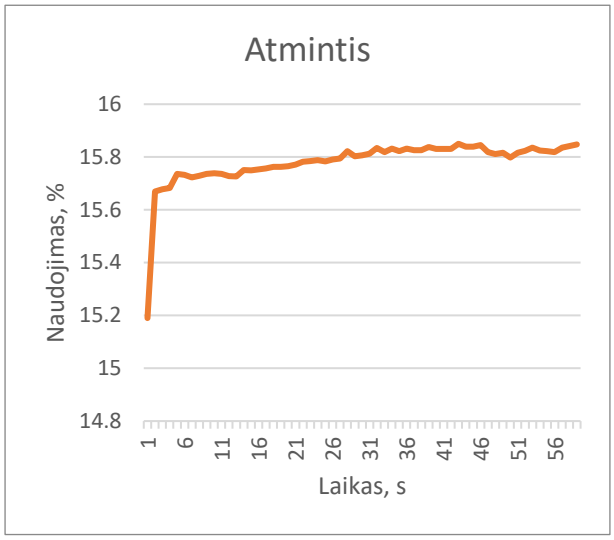


4.56 pav. *Karto SLAM* sudarytas aplinkos žemėlapis robotui judant 5 cm/s greičiu

4.57 pav. didžiausias procesoriaus naudojimas užfiksuotas 31 sekundę su 83,3 %. Vidutiniškai *CPU* naudojimas siekė 36,25 %, kuris panašus kaip ir ankstesnių matuotų *SLAM* algoritmų. 4.58 pav. reikalingas operatyviosios atminties kiekis – 26,31 MB.

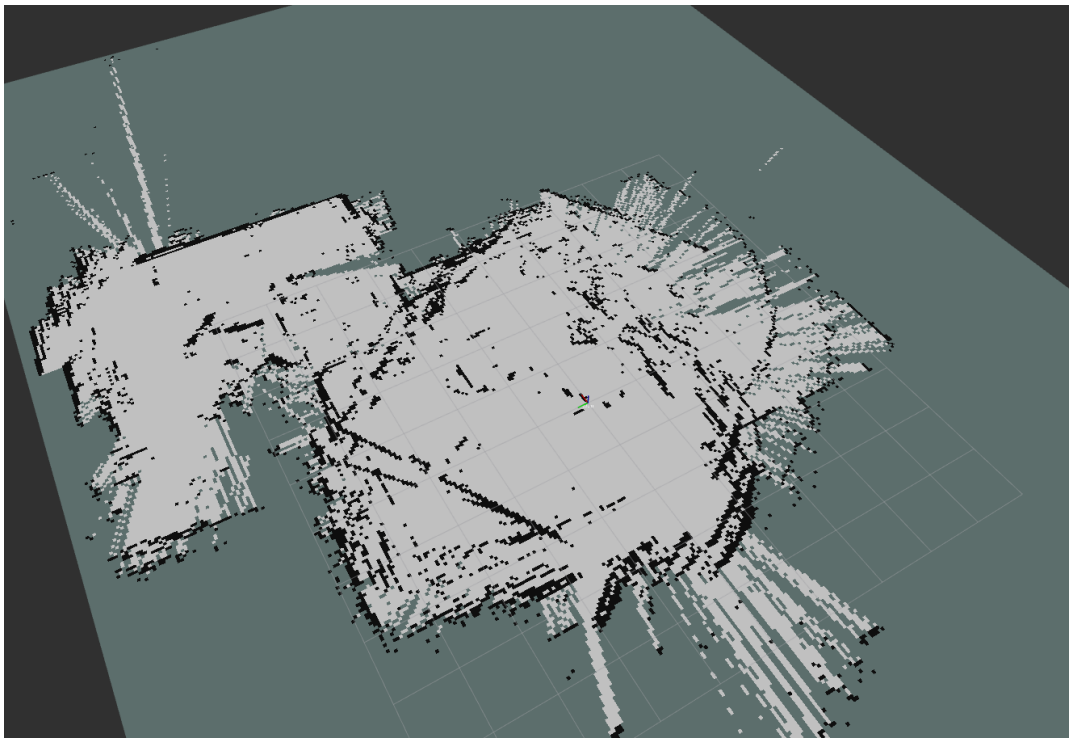


4.57 pav. *Karto SLAM CPU* naudojimas robotui judant 5 cm/s



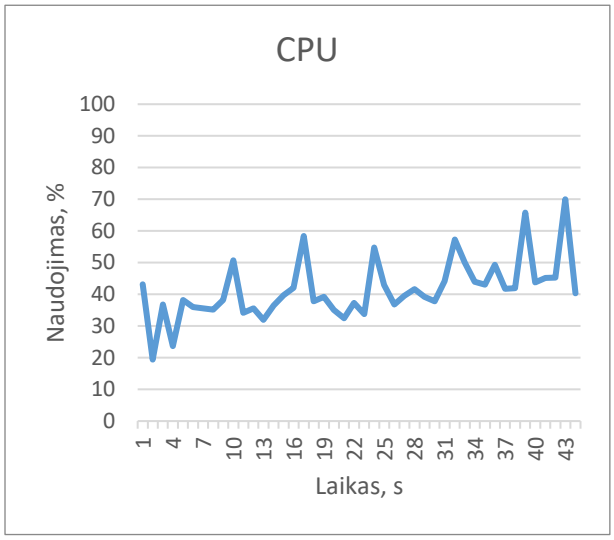
4.58 pav. *Karto SLAM* operatyviosios atminties naudojimas robotui judant 5 cm/s

4.59 pav. padidinus roboto greitį *Karto SLAM* algoritmas nebegali tiksliai įvertinti roboto pozicijos ir sparčiai nuklysta po atlikto roboto posūkio. Sudarytas žemėlapis nebeatspindi tikrovės dėl itin didelės roboto lokalizavimo paklaidos.

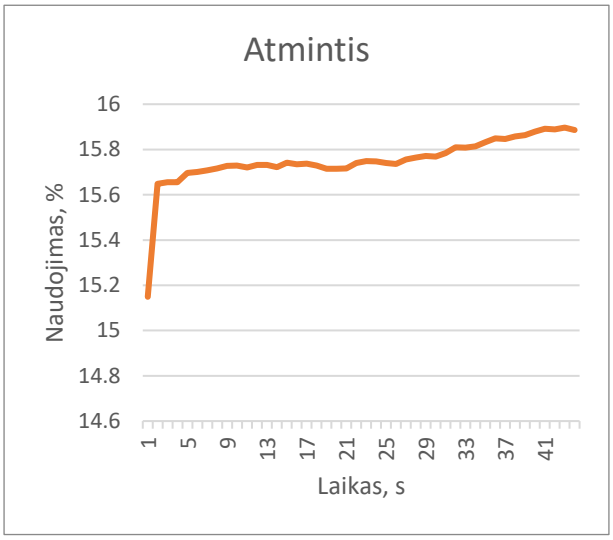


4.59 pav. *Karto SLAM* sudarytas aplinkos žemėlapis robotui judant 10 cm/s greičiu

4.60 pav. didžiausia *CPU* naudojimo vertė pasiekta 43 sekundę su 70 %. Vidutiniškai CPU naudojimas buvo 41,49 %. 4.61 pav. reikalingas *RAM* kiekis lokalizavimui ir žemėlapiui sudaryti – 29,49 MB.



4.60 pav. *Karto SLAM CPU* naudojimas robotui judant 10 cm/s



4.61 pav. *Karto SLAM* operatyviosios atminties naudojimas robotui judant 10 cm/s

Kadangi nepavyko atrasti tinkamų parametrų *Karto SLAM* algoritmo tiksliam veikimui, tolimesni eksperimentai su didesniu greičiu ir aplinka nebuvo atlikti, nes gaunama roboto lokalizacijos paklaida yra itin aukšta, kas veda prie neteisingo žemėlapio sudarymo. 4.7 lentelėje yra pateikiami rezultatai su 5 cm/s ir 10 cm/s greičiais, kur abiem atvejais atsispindi didelės lokalizacijos paklaidos, nors ir naudojami resursai yra panašus į ankstesnių tirtų algoritmų. Sudaryto žemėlapio ilgio ir pločio nepavyko išmatuoti, nes gauti žemėlapiai buvo labai netikslus ir neatspindėjo tikrosios aplinkos.

4.7 lentelė. *Karto SLAM* rodikliai skirtingais judėjimo greičiais mažoje aplinkoje

Greitis, cm/s	**Lokalizacijos paklaida, m**	**Ilgis, m**	**Plotis, m**	**CPU, %**	**RAM, MB**
5	1,12	–	–	36,25	26,31
10	5,61	–	–	41,49	29,49
15	–	–	–	–	–
	3,37	–	–	**38,87**	**27,90**

4.5. Rezultatų apibendrinimas

4.8 lentelėje pateikiami skirtingose aplinkos su tirtais algoritmais gauti lokalizacijos ir žemėlapio sudarymo eksperimentų rezultatai.

4.8 lentelė. Atliktų eksperimentų su skirtingais aplinkos dydžiais ir algoritmais gauti rezultatai

	Algoritmas	**Greitis, cm/s**	**Lokalizacijos paklaida, m**	**Ilgis, m**	**Plotis, m**	**CPU, %**	**RAM, MB**
Mažesnės aplinkos matavimai	*Google Cartographer*	5	0,07	7,04	5,89	35,64	16,05
		10	0,09	7,00	5,90	38,06	6,96
		15	0,14	7,02	5,83	34,09	7,76
		Vidurkis	**0,10**	**7,02**	**5,87**	**35,93**	**10,26**
	Hector SLAM	5	0,45	6,86	5,92	30,24	23,51
		10	1,82	6,88	5,95	35,91	25,02
		15	7,85	–	–	38,33	21,04
		Vidurkis	**3,37**	**6,87**	**5,93**	**34,83**	**23,19**
	GMapping	5	0,05	6,95	5,90	37,93	31,75
		10	0,06	6,94	5,81	35,09	30,67
		15	0,06	6,98	5,83	32,03	31,63
		Vidurkis	**0,06**	**6,96**	**5,85**	**35,02**	**31,35**
	Karto SLAM	5	1,12	–	–	36,25	26,31
		10	5,61	–	–	41,49	29,49
		15	–	–	–	–	–
		Vidurkis	**3,37**	–	–	**38,87**	**27,90**
Didesnės aplinkos matavimai	*Google Cartographer*	5	0,22	14,14	5,88	42,62	29,87
		10	0,25	14,12	5,81	39,21	30,64
		15	0,33	14,15	5,84	37,16	21,06
		Vidurkis	**0,27**	**14,14**	**5,84**	**39,66**	**27,19**
	Hector SLAM	5	0,48	14,21	5,89	26,69	25,22
		10	15,20	–	–	25,18	21,76
		15	16,50	–	–	29,39	25,02
		Vidurkis	**10,73**	**14,21**	**5,89**	**27,09**	**24,00**
	GMapping	5	0,10	14,19	5,87	51,21	70,51
		10	0,16	14,12	5,79	54,53	60,62
		15	0,14	13,94	5,85	49,41	58,71
		Vidurkis	**0,13**	**14,08**	**5,84**	**51,72**	**63,28**

Iš lentelės atsispindi, jog iš atliktų tyrimų mažesnėje aplinkoje geriausią lokalizacijos tikslumą pasiekė *GMapping* algoritmas, kuris skirtingais greičiais turėjo aukščiausią tikslumą su vidutine 0,06 m paklaida. Su *Google Cartographer* taip pat buvo pasiekti tikslūs rezultatai su 0,1 m paklaida.

Pagal nustatytą kiekvieno algoritmo žemėlapio ilgį ir plotį, tiksliausiai aplinkos dydį atspindėjo *GMapping* ir *Google Cartographer* metodai, kur realios aplinkos ilgis yra 7 m ir plotis 5,9 m.

Iš atliktų eksperimentų procesoriaus sunaudojimas buvo didžiausias su *Karto SLAM* metodu 38,87 %, kuriuo buvo atlikti 2 bandymai. Po jo didžiausią turėjo *Google Cartographer* – 35,93 %, o mažiausias *CPU* naudojimas nustatytas su *Hector SLAM* algoritmu – 34,83 %.

Daugiausia operatyviosios atminties reikėjo *GMapping* metodui – 31,35 MB, o mažiausiai *Google Cartographer* – 10,26 MB.

Didesnėje aplinkoje atliktuose eksperimentuose nebuvo įtrauktas *Karto SLAM* metodas, nes su algoritmu buvo gaunami praktiškai nepanaudojami sudaryti žemėlapiai dėl jautrios lokalizacijos paklaidos robotui atliekant pasisukimus.

Pagal lokalizacijos tikslumą *GMapping* algoritmas išliko tiksliausias su 0,13 m vidutine paklaida. Po jo atitinkamai *Google Cartographer* su 0,27 m paklaida.

Sudarytos aplinkos ilgį ir plotį geriausiai atvaizdavimo *GMapping* metodas, kur tikrosios aplinkos ilgis 14 m, o plotis 5,9 m.

CPU naudojimas buvo didžiausias su *GMapping* algoritmu – 51,72 %, o mažiausias su *Hector SLAM* – 27,09 %. *GMapping* algoritmas yra intensyvesnis, todėl jam reikalingas didesnis skaičiavimų resursų kiekis dėl odometrijos ir lazerinio atstumo jutiklio duomenų interpretavimo.

Reikalingas operatyviosios atminties kiekis taip pat buvo žymiai didesnis nei kitų algoritmų su *GMapping* – 63,28 MB. Mažiausiai *RAM* reikėjo *Hector SLAM* – 24 MB. *Google Cartographer* algoritmas nereikalavo daug atminties net ir didesnėje aplinkoje – 27,19 MB.

Išvados

1. Analizės metu iš tirtų lokalizacijos ir žemėlapio sudarymo šaltinių nustatyta, kad tiksliausi rezultatai lokalizacijos ir kartografavimo problemai spręsti gaunami su *Google Cartographer* ir *GMapping* algoritmais. Su *Hector SLAM* gaunami rezultatai yra tikslūs tik robotui judant nedideliais greičiais (iki 15 cm/s) ir atliekant nestaigius posūkius (40 laipsn/s).
2. Tyrimo metu nustatyta, jog norint gauti kuo tikslesnį žemėlapį ir roboto poziciją jame, reikia nustatyti robotą judėti mažu greičiu – ~5 cm/s, nes judėjimas didesniu greičiu dažnai grąžina netikslius rezultatus (ypač robotui atliekant pasisukimą) pasitelkiant *Hector SLAM* algoritmą. Tokią problemą lemia tai, kad grąžinami lazerinio atstumo jutiklio duomenys yra pateikiami per retu intervalu ir *Hector SLAM* algoritmas praranda padėtį aplinkoje dėl per staigaus pokyčio iš anksčiau atrastų taškų atitikmenų trūkumo.
3. Autonominiam roboto judėjimui realizuoti pasirinktas *ROS move_base* paketas. Navigacijos derinimo metu buvo nustatyta, kad tiksliausiems mobiliosios platformos navigacijos rezultatams pasiekti, atsižvelgiant į reikalaujamus įrangos skaičiavimo resursus, robotui geriausiai pavyko išvengti kliūčių su 4x4 m dydžio kaštų žemėlapių ir 0,05 m žemėlapio langelio tikslumu kliūčiai įvertinti.
4. Eksperimentinėje dalyje ištirti 4 lokalizacijos ir žemėlapio sudarymo algoritmai: *Google Cartographer, GMapping, Hector SLAM* ir *Karto SLAM*. Bandymai su kiekvienu iš metodų atlikti 3 skirtingais greičiais: 5 cm/s, 10 cm/s ir 15 cm/s. Atitinkami tyrimai atlikti mažesnėje ir didesnėje uždaroje aplinkoje. Iš gautų eksperimentinių rezultatų nustatyta, kad tiksliausius rezultatus mažesnėje ir didesnėje aplinkoje grąžino *GMapping* algoritmas. Lokalizacijos paklaida su *GMapping* buvo mažiausia (mažesnėje aplinkoje – 0,06 m, didesnėje – 0,13 m), o sudarytas žemėlapis geriausiai atspindėjo realios aplinkos išmatavimus.
5. Efektyviausias algoritmas pagal skaičiavimo resursų sąnaudas – *Hector SLAM*. Mažesnėje aplinkoje *Hector SLAM* procesoriaus naudojimas vidutiniškai siekė 34,83 %, o didesnėje – 27,09 %. Pagal sunaudojama operatyvios atminties kiekį efektyviausias *Google Cartographer* algoritmas: 10,26 MB mažesnėje aplinkoje ir 27,19 MB didesnėje. *Google Cartographer* metodą aplenkė *Hector SLAM* algoritmas su vidutiniu 24 MB reikalingu atminties dydžiu. *Google Cartographer* pasižymėjo dideliu tikslumu tiek mažoje aplinkoje (0,1 m), tiek didelėje (0,27 m) su mažesniais sunaudojamais skaičiavimo resursais (CPU – 39,66 %, RAM – 27,19 MB) nei *GMapping* metodas (CPU – 51,72 %, RAM – 63,28 MB).

Literatūros sąrašas

1. Tzafestas, S.G. *Mobile Robot Control and Navigation: A Global Overview*. J Intell Robot Syst 91, 35–58 (2018). https://doi.org/10.1007/s10846-018-0805-9
2. K. Zhu and T. Zhang, *Deep reinforcement learning based mobile robot navigation: A review*. In Tsinghua Science and Technology, vol. 26, no. 5, pp. 674-691, Oct. 2021, doi: 10.26599/TST.2021.9010012
3. Duguleana, M., & Mogan, G. (2016). *Neural networks based reinforcement learning for mobile robots obstacle avoidance*. Expert Systems with Applications, 62, 104-115. https://doi.org/10.1016/j.eswa.2016.06.021
4. Ma, J., Lu, H., Xiao, J. et al. *Multi-robot Target Encirclement Control with Collision Avoidance via Deep Reinforcement Learning*. J Intell Robot Syst 99, 371–386 (2020). https://doi.org/10.1007/s10846-019-01106-x
5. Y. F. Chen, M. Everett, M. Liu and J. P. How. *Socially aware motion planning with deep reinforcement learning*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 2017, pp. 1343-1350, doi: 10.1109/IROS.2017.8202312
6. R. S. Pol and M. Murugan. *A review on indoor human aware autonomous mobile robot navigation through a dynamic environment survey of different path planning algorithm and methods*. 2015 International Conference on Industrial Instrumentation and Control (ICIC), Pune, India, 2015, pp. 1339-1344, doi: 10.1109/IIC.2015.7150956
7. Y. Jia, X. Yan and Y. Xu. *A Survey of simultaneous localization and mapping for robot*. 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chengdu, China, 2019, pp. 857-861, doi: 10.1109/IAEAC47372.2019.8997820
8. H. Durrant-Whyte and T. Bailey. *Simultaneous localization and mapping: part I*, in IEEE Robotics & Automation Magazine. Vol. 13, no. 2, pp. 99-110, June 2006, doi: 10.1109/MRA.2006.1638022
9. J. Li, L. Cheng, H. Wu, L. Xiong and D. Wang. *An overview of the simultaneous localization and mapping on mobile robot*. 2012 Proceedings of International Conference on Modelling, Identification and Control, Wuhan, China, 2012, pp. 358-364
10. Zhang, X., Lai, J., Xu, D., Li, H., & Fu, M. (2020). *2d lidar-based slam and path planning for indoor rescue using mobile robots*. Journal of Advanced Transportation, *2020*, 1-14.
11. Pascal, A., & Kuhn, J. (2013). *Simultaneous localization and mapping (SLAM) using the extended kalman filter*. Session B11, 3140
12. Pirahansiah, F. A. R. S. H. I. D., Abdullah, S. S., & Sahran, S. H. A. H. N. O. R. B. A. N. U. N. (2013). *Simultaneous localization and mapping trends and humanoid robot linkages*. Asia-Pacific Journal of Information Technology and Multimedia, 2(2), 27-38. https://doi.org/10.17576/apjitm-2013-0202-03
13. G. Dissanayake, S. Huang, Z. Wang and R. Ranasinghe. *A review of recent developments in Simultaneous Localization and Mapping*. 2011 6th International Conference on Industrial and Information Systems, Kandy, Sri Lanka, 2011, pp. 477-482, doi: 10.1109/ICIINFS.2011.6038117
14. Zheng, B., & Zhang, Z. (2019). *An improved EKF-SLAM for Mars surface exploration*. International Journal of Aerospace Engineering, 2019. https://doi.org/10.1155/2019/7637469

15. G. Grisetti, C. Stachniss and W. Burgard. *Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters*. In IEEE Transactions on Robotics, vol. 23, no. 1, pp. 34-46, Feb. 2007, doi: 10.1109/TRO.2006.889486
16. F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers and W. Burgard. *An evaluation of the RGB-D SLAM system*. 2012 IEEE International Conference on Robotics and Automation, Saint Paul, MN, USA, 2012, pp. 1691-1696, doi: 10.1109/ICRA.2012.6225199
17. B. Hampton, A. Al-Hourani, B. Ristic and B. Moran. *RFS-SLAM robot: An experimental platform for RFS based occupancy-grid SLAM*. 2017 20th International Conference on Information Fusion (Fusion), Xi'an, China, 2017, pp. 1-8, doi: 10.23919/ICIF.2017.8009744
18. Z. Li et al. *An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration*. 2019 IEEE International Solid- State Circuits Conference - (ISSCC), San Francisco, CA, USA, 2019, pp. 134-136, doi: 10.1109/ISSCC.2019.8662397
19. Xiao, L., Wang, J., Qiu, X., Rong, Z., & Zou, X. (2019). *Dynamic-SLAM: Semantic monocular visual localization and mapping based on deep learning in dynamic environment*. Robotics and Autonomous Systems, 117, 1-16. https://doi.org/10.1016/j.robot.2019.03.012
20. Donoso, F. A., Austin, K. J., & McAree, P. R. (2017). *Three new Iterative Closest Point variant-methods that improve scan matching for surface mining terrain*. Robotics and Autonomous Systems, 95, 117-128. https://doi.org/10.1016/j.robot.2017.05.003
21. Murphy, K., Russell, S. (2001). *Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks*. In: Doucet, A., de Freitas, N., Gordon, N. (eds) Sequential Monte Carlo Methods in Practice. Statistics for Engineering and Information Science. Springer, New York, NY. https://doi.org/10.1007/978-1-4757-3437-9_24
22. OUSTER. *Building Maps Using Google Cartographer and the OS1 Lidar Sensor*. [Tinkle] https://ouster.com/insights/blog/building-maps-using-google-cartographer-and-the-os1-lidar-sensor
23. Norzam, W. A. S., H. F. Hawari, and K. Kamarudin. *Analysis of mobile robot indoor mapping using GMapping based SLAM with different parameter*. IOP Conference Series: Materials Science and Engineering. Vol. 705. No. 1. IOP Publishing, 2019
24. S. Kohlbrecher, O. von Stryk, J. Meyer and U. Klingauf. *A flexible and scalable SLAM system with full 3D motion estimation*. 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto, Japan, 2011, pp. 155-160, doi: 10.1109/SSRR.2011.6106777
25. R. Yagfarov, M. Ivanou and I. Afanasyev. *Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth*. 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), Singapore, 2018, pp. 1979-1983, doi: 10.1109/ICARCV.2018.8581131
26. Z. Xuexi, L. Guokun, F. Genping, X. Dongliang and L. Shiliu. *SLAM Algorithm Analysis of Mobile Robot Based on Lidar*. 2019 Chinese Control Conference (CCC), Guangzhou, China, 2019, pp. 4739-4745, doi: 10.23919/ChiCC.2019.8866200
27. ROS. *Coordinate Frames for Mobile platforms.* [Tinkle] https://www.ros.org/reps/rep-0105.html#id8
28. ROS. *How to set up hector_slam for your robot*. [Tinkle] https://wiki.ros.org/hector_slam/Tutorials/SettingUpForYourRobot
29. Github. *YDLIDAR ROS Driver.* [Tinkle] https://github.com/YDLIDAR/ydlidar_ros_driver

30. Zhao J, Liu S, Li J. *Research and Implementation of Autonomous Navigation for Mobile Robots Based on SLAM Algorithm under ROS*. Sensors. 2022; 22(11):4172. https://doi.org/10.3390/s22114172
31. ROS. *hector_slam.* [Tinkle] https://wiki.ros.org/hector_slam
32. ROS. *Setup and Configuration of the Navigation Stack on a Robot.* [Tinkle] http://wiki.ros.org/navigation/Tutorials/RobotSetup
33. ROS. *move_base.* [Tinkle] https://wiki.ros.org/move_base

Priedai

1 priedas. *Hector SLAM* duomenų konvertavimo kodo fragmentas

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import PoseStamped, Twist, Vector3
from nav_msgs.msg import Odometry
from tf.transformations import euler_from_quaternion

class SlamPoseToOdomConverter:
    def __init__(self):
        rospy.init_node('slam_pose_to_odom_converter')
        self.slam_pose_sub = rospy.Subscriber('/slam_out_pose', PoseStamped,
          self.slam_pose_callback)
        self.odom_pub = rospy.Publisher('/odom', Odometry, queue_size=10)
        self.last_slam_pose = None

    def slam_pose_callback(self, slam_pose_msg):
        if self.last_slam_pose is None:
            self.last_slam_pose = slam_pose_msg
            return
        dt = (slam_pose_msg.header.stamp - self.last_slam_pose.header.stamp).to_sec()
        linear_velocity = self.calculate_linear_velocity(self.last_slam_pose.pose,
          slam_pose_msg.pose, dt)
        angular_velocity = self.calculate_angular_velocity(self.last_slam_pose.pose,
          slam_pose_msg.pose, dt)
        self.last_slam_pose = slam_pose_msg
        odom_msg = Odometry()
        odom_msg.header = slam_pose_msg.header
        odom_msg.pose.pose = slam_pose_msg.pose
        odom_msg.twist.twist = Twist(
            linear=Vector3(x=linear_velocity),
            angular=Vector3(z=angular_velocity)
        )
        odom_msg.pose.covariance = [0.0] * 36
        odom_msg.twist.covariance = [0.0] * 36
        self.odom_pub.publish(odom_msg)

    def calculate_linear_velocity(self, pose1, pose2, dt):
        dx = pose2.position.x - pose1.position.x
        dy = pose2.position.y - pose1.position.y
        distance = (dx**2 + dy**2)**0.5
        linear_velocity = distance / dt if dt > 0 else 0.0
        return linear_velocity
```

```
    def calculate_angular_velocity(self, pose1, pose2, dt):
        quaternion1 = (pose1.orientation.x, pose1.orientation.y, pose1.orientation.z,
          pose1.orientation.w)
        quaternion2 = (pose2.orientation.x, pose2.orientation.y, pose2.orientation.z,
          pose2.orientation.w)
        _, _, yaw1 = euler_from_quaternion(quaternion1)
        _, _, yaw2 = euler_from_quaternion(quaternion2)
        angular_distance = abs(yaw2 - yaw1)
        angular_velocity = angular_distance / dt if dt > 0 else 0.0
        return angular_velocity

    def run(self):
        rospy.spin()

if __name__ == '__main__':
    slam_pose_to_odom_converter = SlamPoseToOdomConverter()
    slam_pose_to_odom_converter.run()
```

2 priedas. *Arduino* roboto judėjimo valdymo kodo fragmentas

robot.h:

```
#ifndef robot_h
#define robot_h
#include <Arduino.h>
#include <Adafruit_MCP4725.h>
#include <ros.h>

#define DAC_RESOLUTION (9)

class Robot {
  private:
    Adafruit_MCP4725 steerDac;
    Adafruit_MCP4725 speedDac;
    void setDacVoltage(Adafruit_MCP4725 dac, float voltage);
    float mapFloat(float x, float in_min, float in_max, float out_min, float
      out_max);

  public:
    Robot();
    void setup();
    void moveForward();
    void turnLeft();
    void turnRight();
    void moveBackwards();
    void stopRobot();
    void move(float x, float theta, ros::NodeHandle nh);
};

#endif
```

robot.cpp:

```
#include "Arduino.h"
#include "robot.h"

Robot::Robot() { }

void Robot::setup() {
  steerDac.begin(0x60);
  speedDac.begin(0x61);
}

void Robot::moveForward() {
  Robot::setDacVoltage(speedDac, 2.1);
  Robot::setDacVoltage(steerDac, 2.5);
}

void Robot::turnLeft() {
```

```
  Robot::setDacVoltage(speedDac, 1.8);
  Robot::setDacVoltage(steerDac, 3.5);
}

void Robot::turnRight() {
  Robot::setDacVoltage(speedDac, 1.8);
  Robot::setDacVoltage(steerDac, 1.4);
}

void Robot::moveBackwards() {
  Robot::setDacVoltage(speedDac, 1);
  Robot::setDacVoltage(steerDac, 2.5);
}

void Robot::stopRobot() {
  Robot::setDacVoltage(speedDac, 1.1);
  Robot::setDacVoltage(steerDac, 1.1);
}

void Robot::move(float x, float theta, ros::NodeHandle nh) {
  float abs_theta = abs(theta);
  if ((abs_theta > 0.0 && x == 0.0) || (abs_theta > 0.3 && x > 0.5)) {
    x = 0.0;
    theta = theta > 0.0 ? 1.0 : -1.0;
  }
  float val_x = mapFloat(x, 0.0, 1.0, 1.8, 2.1);
  float val_theta = mapFloat(theta, -1.0, 1.0, 1.4, 3.5);
  Robot::setDacVoltage(speedDac, val_x);
  Robot::setDacVoltage(steerDac, val_theta);
}

void Robot::setDacVoltage(Adafruit_MCP4725 dac, float voltage) {
  dac.setVoltage(voltage * 4096 / 5, false);
}

float Robot::mapFloat(float x, float in_min, float in_max, float out_min, float
out_max) {
  x = constrain(x, in_min, in_max);
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

robot_control.ino:

```
#include <Wire.h>
#include <ros.h>
#include "robot.h"
#include <geometry_msgs/Twist.h>
#include <std_msgs/Header.h>

int previous_time = 0;
float linear = 0.0;
float angular = 0.0;
Robot robot;
ros::NodeHandle nh;

void callbackFunction(const geometry_msgs::Twist &msg) {
  float lin = abs(msg.linear.x);
  float ang = msg.angular.z;
  linear = lin;
  angular = ang;
}
ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel", &callbackFunction);

void periodic_update(int interval) {
  int current_time = millis();
  if (current_time - previous_time > interval) {
    nh.spinOnce();
    previous_time = millis();
  }
}

void setup() {
  Wire.begin();
  robot.setup();
  nh.initNode();
  nh.subscribe(sub);
}

void loop() {
  robot.move(linear, angular, nh);
  periodic_update(50);
}
```