VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS
KOMPIUTERIŲ INŽINERIJOS KATEDRA

Aleksej Trofimov

SKIRTINGŲ TIPŲ DIRBTINIO INTELEKTO ALGORITMŲ PANAUDOJIMAS UŽDARŲJŲ PATALPŲ NAVIGACIJOJE (DIFFERENT TYPES OF ARTIFICIAL INTELLIGENCE ALGORITHMS USE FOR INDOOR NAVIGATION)

Baigiamasis magistro darbas

Kompiuterių inžinerijos studijų programa, valstybinis kodas 62401T211
Kompiuterių technologijų specializacija
Elektronikos inžinerijos mokslo kryptis

Vilnius, 2017

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS
KOMPIUTERIŲ INŽINERIJOS KATEDRA

TVIRTINU
Katedros vedėjas

_____
(Parašas)

Algirdas Baškys
(Vardas, pavardė)

_____
(Data)

Aleksej Trofimov

SKIRTINGŲ TIPŲ DIRBTINIO INTELEKTO ALGORITMŲ PANAUDOJIMAS UŽDARŲJŲ PATALPŲ NAVIGACIJOJE (DIFFERENT TYPES OF ARTIFICIAL INTELLIGENCE ALGORITHMS USE FOR INDOOR NAVIGATION)

Baigiamasis magistro darbas

Kompiuterių inžinerijos studijų programa, valstybinis kodas 621H69001
Kompiuterių technologijų specializacija
Elektronikos inžinerijos mokslo kryptis

Vadovas	doc. dr. Nerijus Paulauskas (Moksl. laipsnis/pedag. vardas, pavardė)	_____ (Parašas)	_____ (Data)
Lietuvių kalbos konsultantas	dr. Aušra Žemienė (Moksl. laipsnis/pedag. vardas, pavardė)	_____ (Parašas)	_____ (Data)

Vilniaus Gedimino technikos universitetas
Elektronikos fakultetas
Kompiuterių inžinerijos katedra

ISBN ISSN
Egz. sk. ……..
Data ….-….-….

Kompiuterinės inžinerijos studijų programos baigiamasis magistro darbas
Pavadinimas: **SKIRTINGŲ TIPŲ DIRBTINIO INTELEKTO ALGORITMŲ PANAUDOJIMAS UŽDARŲJŲ PATALPŲ NAVIGACIJOJE**
Autorius **Aleksej Trofimov** Vadovas**: doc. dr. Nerijus Paulauskas**

Kalba
[x] lietuvių
užsienio

Anotacija

Darbe kuriama ir nagrinėjama roboto uždarose patalpose navigacijos imitacija naudojant skirtingus dirbtinio intelekto algoritmus. Darbas tiria skirtingų parametrų, tokių, kaip radijo švyturėlio įjungimas/išjungimas ir triukšmo lygis jutikliuose, įtaką roboto pozicijos nustatymui naudojant Kalmano bei dalelių filtrus. Pirmajame skyriuje nagrinėjami pagrindiniai uždarųjų patalpų navigacijoje taikomi algoritmai, jutikliai bei radijo švyturėliai. Antrajame skyriuje yra aprašoma naudojama imitacija ir programiniai komponentai bei algoritmai imitacija atlikti. Taip pat nagrinėjami du imitacijos eksperimento modeliai naudojant ir nenaudojant radijo švyturėlių. Be to, pateikiami rezultatai esant skirtingiems parinktiems signalo/triukšmo lygiams ir jų įtaka naudojamų skirtingų algoritmų gaunamų pozicijų vertėms. Išvadose nagrinėjami gauti rezultatai, aptariami tolimesni galimi tyrimai ir pateikiami pasiūlymai. Tyrimams sukurtas programinio kodo paketas bei pozicijos nustatymo algoritmų taikymo paketas.

Darbą sudaro 5 dalys: įvadas, teorinė dalis, praktinė dalis, išvados ir siūlymai, literatūros sąrašas.

Darbo apimtis — 59 p. teksto be priedų, 29 iliustracijos, 5 lent., 18 bibliografinių šaltinių.

Prasminiai žodžiai: Dalelių filtras, Kalmano filtras, vidinių patalpų navigacija.

Vilnius Gediminas Technical University
Faculty of Electronics
Department of Computer Engineering

ISBN ISSN
Copies No. ……
Date ….-….-….

Computer Engineering study programme master thesis.

Title **DIFFERENT TYPES OF ARTIFICIAL INTELLIGENCE ALGORITHMS USE FOR INDOOR NAVIGATION**

Author **Aleksej Trofimov** Academic supervisor**: doc. dr. Nerijus Paulauskas**

Language
x lithunian
foreign

Annotation

The work focuses on the robot indoor navigations simulation framework using different artificial intelligence algorithms. This work analyzes different parameters, such as radio beacon enablement and signal/noise level influence of the robot measured position in simulation using Kalman and particle filters algorithms. First section studies currently available artificial algorithms used for indoor navigations, robot used sensors and radio beams and their protocols. Second part focuses on simulation concept of an indoor robot navigation framework, analyzes two simulation models – with and without radio beacon support. Furthermore the study is done regards signal / noise level in sensors influence to the position error. The conclusion concentrates on the results analysis and propositions. During the fork a full-scale simulation framework was created with ability to plug-in different navigation algorithms.

Thesis consists of: 59 p. text without appendixes, 29 pictures, 5 tables, 18 bibliographical entries.

Keywords: Particle filter, Kalman filter, EKF, indoor navigation

TURINYS

ILIUSTRACIJŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

ĮVADAS

Tradiciškai objekto (roboto) navigacijai ir pozicionavimui naudojami du komponentai: inercijos jutiklis ir globalinės padėties nustatymo sistemos (toliau – GPS, nuo angl. *Global Positioning System*) koordinatės. Inercijos jutiklis matuojant kampinius bei radialinius pagreičius leidžia suintegruoti būsimus parametrus, o GPS rodmenys leidžia koreguoti integravimo paklaidas. Taikant technologijas įmanoma nustatyti vartotojų dabartinę buvimo vietą ir pranešti jiems apie artimiausius viešbučius, restoranus, kavines ir pan. Toks būdas patikimai gali nustatyti objekto poziciją aplinkoje, kai nėra GPS trikdžių, tačiau metodas visiškai netinkamas tuo atveju, kai GPS signalas nėra pasiekiamas – pvz., uždarose patalpose. [1]

Dėl to navigacijai uždarose patalpose reikėtų ieškoti kitų sprendimų bei išeičių. Norint išspręsti uždarų patalpų navigacijos problemą, reikia surasti būdą, kaip išvengti arba sumažinti inercinių jutiklių paklaidas. Akivaizdu, kad išvengti paklaidų visiškai neišeitų, todėl navigacijos korekcijos metu tokios paklaidos gali kauptis, o tai, laikui bėgant, gali nulemti labai didelius pozicijos nustatymo netikslumus.

Naudodami navigaciją uždarose patalpose vartotojai be sunkumų gali surasti, pvz., artimiausią registracijos stalą oro uoste, laisvą vietą automobilio stovėjimo aikštelėje, skyrių ir lentyną parduotuvėje su reikiama preke, eksponatą muziejuje (be to, iš karto gali būti pateikiamas jo aprašymas mobiliojo telefono ekrane) ir daugelį kitų dalykų, kas sutaupo jiems nemažai laiko. Taip pat navigacija uždarose patalpose svarbi gamyklose ir statybose – krovinių, personalo, technikos vietai nustatyti.[3]

Taip pat šios technologijos nulėmė naujų rinkodaros priemonių plėtrą, pvz., praeidamas pro parduotuvę, suinteresuotas vartotojas gali gauti pranešimą mobiliajame telefone apie joje vykstančias akcijas / renginius / teikiamas paslaugas (taip vadinamas „*geo-fencing*“), o jam siūlomi sprendimai remsis jo interesais, priklausomai nuo jo pirkimo istorijos, arba jis tiesiog gali gauti aktualų pranešimą, kai priartėja prie tam tikros vietos (antra navigacijos uždarose patalpose kryptis – „*geo-aware*“). Nauda yra abipusė, nes verslų savininkai galės gauti įvairiapusę statistinę informaciją apie vartotojų judėjimą parduotuvės salėse (savotiškas nevirtualus „Google Analytics“ analogas). Dėl tolimesnės navigacijos uždarose patalpose technologijų plėtros laukiamas staigus tokios geokontekstinės reklamos rinkos (angl. LBA – *location-based advertising*) augimo šuolis. [7]

Taigi matome, kad navigacijos uždarose patalpose klausimas yra **aktualus** ir vertas dėmesio. Todėl šiame darbe siekiama apžvelgti šiuolaikinius pozicionavimo uždarose patalpose algoritmus bei naudojamus jutiklius.

Atlikus esamos literatūros analizę nebuvo aptikta tyrimų, kurie parodytų dirbtinio intelekto algoritmų parametrų (tokių, kaip triukšmas arba papildomi informacijos šaltiniai apie poziciją – radijo švyturėliai) įtaką galutiniam rezultatui. Be to, nepavyko surasti ir išsamaus palyginimo tarp egzistuojančių dirbtinio intelekto algoritmų, taikomų objektų uždarų patalpų navigacijos užduočiai spręsti. Taigi šio darbo **naujumą** parodo tai, kad šiame darbe analizuojama algoritmų parametrų (jutiklių triukšmo bei papildomo švyturėlio panaudojimo) įtaka pozicijos nustatymui ir išsamiai palyginami skirtingi dirbtinio intelekto algoritmai objekto uždarose patalpose pozicijai nustatyti.

Taigi šio darbo **tikslas** yra sukurti objekto navigacijos uždarose patalpose programinį imitacinį modelį ir ištirti objekto navigacijos tikslumą, taikant skirtingus dirbtinio intelekto algoritmus.

Norint pasiekti iškeltą tikslą, reikia atlikti šiuos **uždavinius**:

- Išanalizuoti esamus dirbtinio intelekto (toliau – DI) algoritmus, taikomus objekto navigacijai uždarose patalpose.
- Sukurti programinę įrangą, imituojančią pasirinktų DI algoritmų veikimą.
- Iširti DI algoritmų parametrų įtaką objekto navigacijos uždarose patalpose tikslumui.
- Iširti, kaip keičiasi objekto navigacijos uždarose patalpose tikslumas naudojant ir nenaudojant radijo švyturių.

Norint pasiekti iškeltus tikslus darbe bus kuriamas programinis imitacijos modelis bei atliekami skirtingi imitacijos eksperimentai.

Eksperimentas bus padalintas į dvi dalis:

1. Algoritmų tikslumo tyrimas taikant skirtingus modelius;
2. Algoritmų tikslumo priklausomybė nuo įvedamų jutiklių triukšmų.

Taip pat darbo metu bus ištirti du modeliai – kai imitacija vykdoma naudojant tik roboto viduje esančius jutiklius ir kai įvedami papildomi išoriniai švyturėliai, ir pozicija patikslinama trianguliacijos būdu.

1. TEORINĖ DALIS

1.1 Uždarų patalpų navigacijos įgyvendinimo galimybės

Kadangi palydovines sistemas uždarose patalpose pritaikyti labai sudėtinga, tenka ieškoti kitų sprendimo būdų. Jų yra keli:

1) ***Navigacija su „Wi-Fi“ technologija.*** Panaudojama jau egzistuojanti ryšio tinklų infrastruktūra – Wi-Fi taškai. Koordinačių nustatymo metodika yra tokia: įrenginys skenuoja prieinamus Wi-Fi taškus, po to siunčia informaciją apie juos serveriui, kur duomenys duomenų bazėse sutikrinami su Wi-Fi prieigos taškų koordinatėmis – taip randamos vartotojo koordinatės.

Pagrindinis šios technologijos panaudojimo privalumas yra jos kaina – tai yra mažiausiai išlaidų reikalaujantis metodas, nes naudojama jau egzistuojanti infrastruktūra. Tačiau šis metodas turi nemažai trūkumų: pirma, Wi-Fi taškų koordinatės nėra tiksliai žinomos, o tuo labiau, jos gali keistis. Antra, šio metodo paklaida yra labai didelė – net iki 25 metrų. Trečia, norint gauti mažesnę paklaidą (3–5 m), turi būti kuriama speciali Wi-Fi infrastruktūra, o tai jau yra labai brangu. Ketvirta, metodui taikyti trukdo tas faktas, kad pradedant nuo iOS 8, visų Apple įrenginių mac-adresai nuolat keičiasi, kad būtų išvengta reklaminio persekiojimo. Taigi šios technologijos panaudojimas navigacijai uždarose patalpose turi daugiau trūkumų, nei privalumų.[6]

2) ***Geomagnetinis pozicionavimas***. Metodas yra pagrįstas orientavimusi pagal Žemės magnetinį lauką. Metodo esmė yra geomagnetinių anomalijų fiksavimas ir jų žymėjimas žemėlapio teritorijoje, kurioje planuojama orientuotis. Vėliau pagal sudarytą žemėlapį specialiu įrenginiu atliekama navigacija. Praktinis metodo įgyvendinimo pavyzdys – „IndoorAtlas“ sistema, kurią sukūrė suomių Oulu universiteto komanda. Pagrindiniai metodo trūkumai – gana žemas tikslumas ir sunkus realizavimas, nes uždarose patalpose egzistuoja labai daug dinamiškai besikeičiančių magnetinių anomalijų, apsunkinančių navigaciją.[6]

3) ***Palydovinės navigacijos sistemos*** (GPS, „Glonass“ ir pan.), **inercinės navigacinės sistemos** (toliau – INS, angl. *Inertial Navigation Systems*). Šių dviejų technologijų metodas taikomas, kai laikas nuo laiko dar atsiranda palydovinės navigacijos sistemų signalas. Pvz., važiuojant tunelyje: įvažiuojant dar yra prieinamos aktualios koordinatės ir judėjimo kryptis iš GPS / „Glonass“ palydovų, o vėliau jau naudojamos INS (pagrįstos akselerometru, giroskopu ir magnetinio lauko davikliu), kurios kaip pradines sąlygas naudoja paskutinius aktualius duomenis iš palydovų, o jų aktualumą palaiko besiremdamos iš daviklių gaunamais duomenimis apie dabartinį greitį, pagreitį, judėjimo kryptį, iki pat

ryšio atnaujinimo su palydovais. Pagrindinis trūkumas – INS nuolat kaupiamos klaidos, ir po tam tikro laiko duomenys, gaunami naudojant INS, vis labiau skiriasi nuo tikrovės.

4) ***Orientavimas pagal mobiliojo ryšio operatorių stotis (GSM)***. Mobiliojo ryšio telefonas / GSM modemas nuolat fiksuoja bent vieną GSM stotį, o dažniau – kelias. Metodo privalumas tas, kad šių stočių koordinatės žinomos; trūkumas – mažas tikslumas, nes stotis gali būti nutolusi nuo vartotojo 35 km, o be to, kai kurios stotys yra mobilios ir keičia savo buvimo vietą. [11]

5) ***„Beacon“ technologija, kuri veikia „Bluetooth low energy“ ryšio pagrindu***. Metodas tuo pačiu yra ir gana tikslus, ir reikalauja visai prieinamo finansinių išlaidų lygio, be to, tai yra perspektyvi ir aktyviai tobulinama technologija, todėl **būtent ji bus detaliau nagrinėjama šiame darbe**. [6]

6) ***Navigacija, paremta sinergijos efektu***. Panaudojami visi arba dauguma aukščiau išvardintų metodų, kas padeda kompensuoti klaidas ir padidinti koordinačių nustatymo tikslumą, ir taip pasiekti didesnį efektyvumą.

1.2 „Beacon“ technologija

Kaip jau buvo minėta, šiame darbe buvo pasirinkta būtent ši technologija – dėl jos aukšto tikslumo lygio ir gana žemo reikalingų finansinių išlaidų lygio. Taigi toliau ji bus pristatoma detaliau.

Technologijos veikimo schema paprasta – turime palei visą perimetrą išdėliotus Bluetooth švyturėlius, kurių buvimo vietos koordinatės yra žinomos. Vartotojo aplikacija su nustatytu periodiškumu gauna duomenis iš švyturėlių, identifikuojančius juos, ir pagal duomenų bazę nustato švyturėlių koordinates ir besiremdama signalo stiprumu, leidžiančiu nustatyti nuotolį iki kiekvieno iš jų, nustato savo buvimo vietą.

Tokie švyturėliai dažnai vadinami „iBeacon“ švyturėliais, bet toks pavadinimas yra klaidingas. Taisyklingiau juos vadinti tiesiog „Beacon“, nes „iBeacon“ – tai „Apple“ standarto pavadinimas, kuris naudojamas iOS mobiliose aplikacijose, o „Beacon“ – tai fizinė švyturėlių realizacija. Tipinis „Beacon“ švyturėlis pavaizduotas žemiau, žr. 1 pav.:

1 pav. Tipinis „Beacon“ švyturėlis

Kalbant apie fizinę realizaciją, „Beacon“ švyturėliai – tai paprasti Bluetooth 4.0 žemos energijos (toliau – LE, angl. *Low Energy*) įrenginiai. Tai reiškia, kad jų vaidmenį gali laisvai atlikti bet koks įrenginys, turintis BLE mikrovaldiklį – pvz., išmanieji telefonai su „Android“ operacine sistema, o taip pat „iPhone“, „iPad“, paprasti nešiojamieji kompiuteriai, „Raspberry Pi“ su USB Bluetooth ir kiti, kurie turi specialią aplikaciją, atliekančią „Beacon“ švyturėlio funkciją. Tipinis „Beacon“ švyturėlis, pavaizduotas aukščiau, yra gana kompaktiško dydžio ir gali veikti vos nuo vienos baterijos iki 2 metų.[15]

Schemotechniškai švyturėlis susideda iš baterijos ir integrinio grandyno (angl. SOC – *System-On-Chip*) „Texas Instruments CC2540/2541“ (taip pat gali būti taikomas „Nordic nRF51822“), kuris yra 8051 mikrovaldiklis, turintis „Beacon“ švyturėlio realizavimo funkciją, ir periferinis modulis „Bluetooth LE“.

Švyturėlio veikimo nuotolio vidurkis – 10 m (nuotolis varijuoja nuo 15–20 cm iki 25–40 m priklausomai nuo modelio ir nustatymų). Duomenų išdavimo periodiškumas – 200 msek, bet tai irgi galima pakeisti – nustatyti tiek dažnesnį, tiek retesnį periodiškumą. Veikimo iš vienos baterijos laikas – priklausomai nuo modelio nuo kiek mažiau nei 1 metai iki 3 metų (vidutiniškai – 2 metai). Vieno švyturėlio kaina – apie 15–20 JAV dolerių. Nors pats švyturėlis yra gana paprastas įrenginys, kuris (veikdamas reklamos režimu) tik pateikia duomenis, naudodamas Bluetooth GATT profilį, tačiau dažniausiai gamintojai taip pat sukuria ir nuotolinio valdymo galimybę.

1.2.1 „Beacon“ švyturėlių saugumas

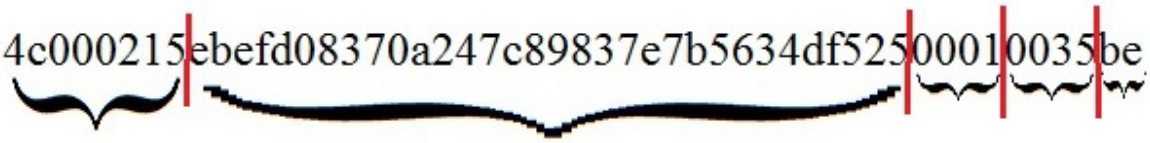
Nors „Beacon“ yra nuostabi ir perspektyvi technologija, verta nepamiršti ir apie jos panaudojimo saugumo klausimus. Pvz., buvo siūlymų naudoti „Beacon“ švyturėlį automobilyje nuotoliniu būdu užkardui atidaryti. Tačiau verta atsiminti, kad technologija nesuteikia jokių saugumo priemonių, ir niekas nedraudžia piktavaliui skenuoti realaus laiko režimu eterį ieškant švyturėlio, su kuriuo užkardas gali būti atidaromas. Toliau lieka tik įrašyti į padirbtą švyturėlį fizinį universalų unikalų įrenginio identifikavimo numerį (toliau – pUUID, angl. *physical universal unique identificator*), viršutinę bitų seką (toliau – *Major*) ir apatinę bitų seką (toliau – *Minor*), ir užkardas atidarys jam kelią į svetimą teritoriją. Taigi šiuo atveju geriau atsisakyti „Beacon“ technologijos ir panaudoti kitas – naudojančias duomenų šifravimo metodus.

Taip pat niekas nedraudžia piktavaliui įsilaužti į svetimą navigacijos sistemą, panaudojus savo padirbtą švyturėlį su sistemos, į kurią siekiama įsilaužti, pUUID/*Major*/*Minor*. Arba, jeigu krovinio buvimo vieta randama pagal jame įtaisytą švyturėlį, reikiamoje vietoje įtaisyti savo su duomenimis iš švyturėlio krovinyje ir taip pasisavinti krovinį.

Taigi naudojant „Beacon“ technologiją reikia nepamiršti apie šias grėsmes ir surasti tinkamus apsaugos būdus.

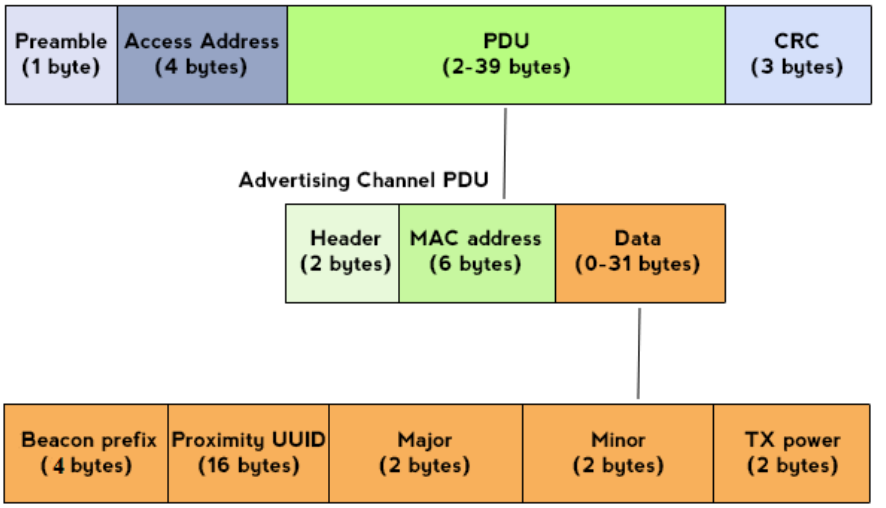
1.2.2 Duomenų formatas

Su nustatytu periodiškumu, ciklišk ai, švyturėlis išveda tą patį duomenų rinkinį:



2 pav. Švyturėlio išvedami duomenys

Bendra Bluetooth paketo struktūra yra tokia:



3 pav. Bendra Bluetooth paketo struktūra

Preambulė (4 baitai) – paketo prefiksas, leidžiantis nustatyti, kad naudojamas būtent „Beacon“ švyturėlis. Preambulė visada lygi 4c000215.
Ji susideda iš 4 laukų:

- kompanijos identifikatorius (2 baitai; aukščiau pateiktame pavyzdyje – 4c00),
- tipas (1 baitas; pavyzdyje – 0x02) ir
- duomenų ilgis (1 baitas; reikšmė – 0x15).

Atstumo identifikatorius (UUID) (16 baitų) – „Beacon“ švyturėlių grupės identifikatorius. Pvz., turime kelias prekybos sales, kuriose reikia įrengti „Beacon“ švyturėlius. Tokiu atveju visose šiose salėse švyturėliai turės vieną ir tą patį UUID, kas leis atskirti juos nuo kitų švyturėlių, pašalinių.

Major (2 baitai) – leidžia atskirti vienos grupės viduje nedidelį švyturėlių rinkinį, t. y. kiekviena didelė švyturėlių grupė, žymima vienu UUID, gali turėti kelis pogrupius, kiekvienas iš kurių identifikuojamas pagal *Major* numerį. Mūsų pavyzdyje savo *Major* numerį galėtų turėti kiekviena prekybos salė atskirai.

Minor (2 baitai) – numeris, identifikuojantis patį švyturėlį *Major* viduje. UUID + *Major* + *Minor* leidžia vienareikšmiškai identifikuoti švyturėlį ir, atitinkamai, jo koordinates.

Signalo perdavimo galia (angl. *TX Power*) (žr. 2 pav.; 2 baitai) – švyturėlio galingumo etalonine reikšmė, kas yra signalo stiprumas 1 m atstumu nuo švyturėlio. Matuojamas ir įrašomas į

švyturėlį vieną kartą, gamybos procese. Ši konstanta naudojama atstumui nustatyti nuo vartotojo iki švyturėlio. Pirmasis baitas yra ženklas (1 – „-“ 0 – „+“). Pvz., signalo perdavimo galia aukščiau pateiktame pavyzdyje, 2 pav. – 0xBE. Tai lygu 190 (dešimtainėje skaičiavimo sistemoje). Tuomet etaloninis signalo stiprumas 1 m atstumu nuo švyturėlio yra 256 - 190 = -66 dBm.

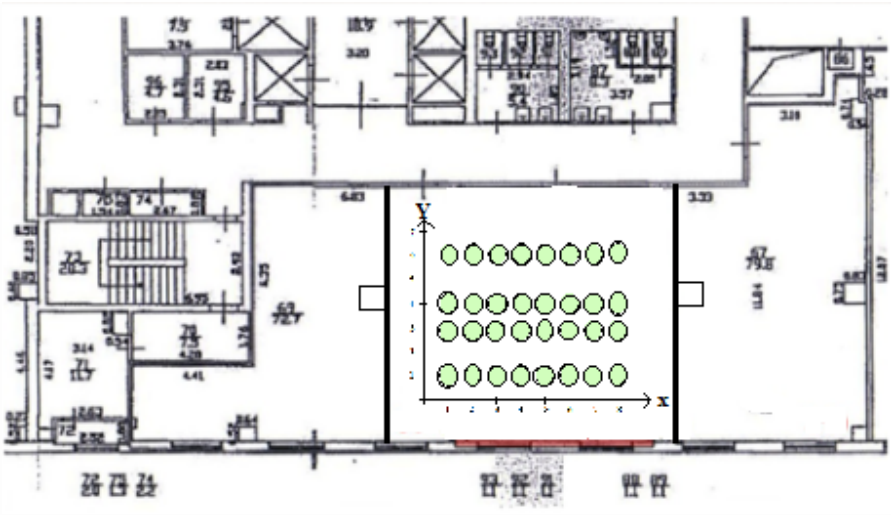
Detaliau panagrinėsime atstumo iki švyturėlių nustatymą. Norint nustatyti objekto buvimo vietą, reikia žinoti ne tik švyturėlių koordinates, bet ir atstumą iki jų. Tai leidžia padaryti gaunamo signalo stiprumo indikatoriaus (toliau – RSSI, angl. *Received Signal Strength Indicator*) parametras, išskaičiuojamas vartotojo Bluetooth imtuvu pagal priimamo signalo stiprumą. Kuo didesnė parametro reikšmė, tuo arčiau švyturėlio yra objektas. Signalo perdavimo galia – tai ir yra RSSI, tik etaloninė, išmatuota gamintojo 1 metro atstumu iki jo. Atstumui (metrais) nustatyti iki švyturėlio naudojama dabartinė RSSI reikšmė ir etaloninė signalo perdavimo galia korekcijai toliau pateikiamu būdu:

```
function get_distance(rssi, tx_power) {
if (rssi == 0) {
return-1; // Neišeina paskaičiuoti atstumo
  }
var ratio = rssi / tx_power;
if (ratio <1) {
returnMath.pow(ratio, 10);
  } else {
return0.89976 * Math.pow(ratio, 7.7095) + 0.111;
  }
}
```

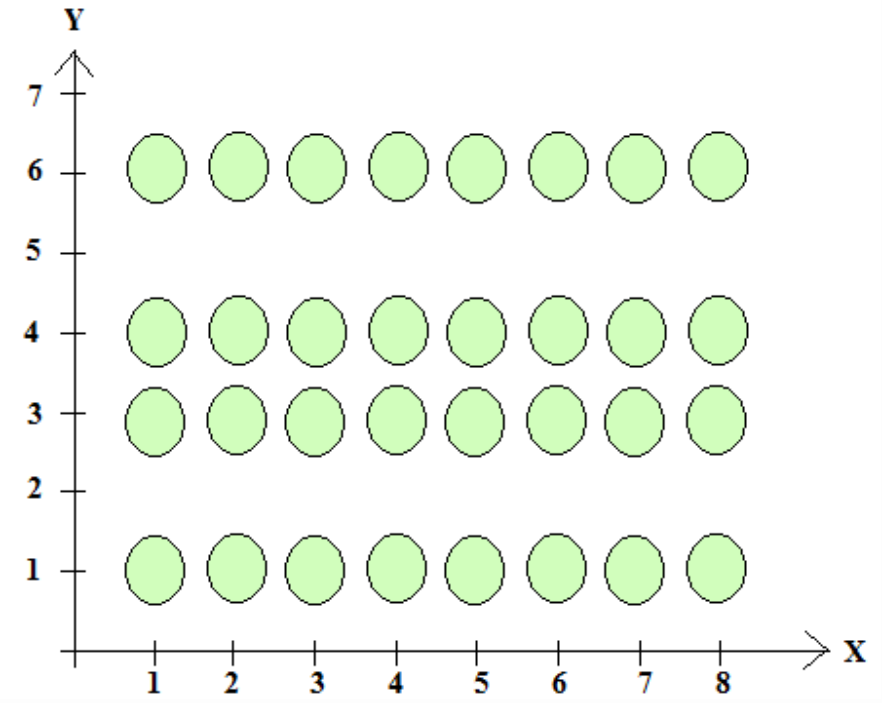
Naudojant šios funkcijos pseudo-kodą galima apskaičiuoti imitacijos roboto atstumą nuo radijo švyturėlio.

1.2.3 Koordinačių nustatymas pagal „Beacon“ švyturėlius

Pirmiausiai reikia išdėlioti patalpose „Beacon“ švyturėlius. Kuo daugiau jų bus ir kuo arčiau vieni kitų jie bus įrengti, tuo geresnis rezultatas bus gaunamas nustatant koordinates. Išdėlioti juos reikia taip, kad jie, pagal galimybes, padengtų visą patalpų plotą, ir susieti juos su koordinatėmis tinklelio patalpų žemėlapyje, atsižvelgiant į jų tarpusavio atstumą:



4 pav. „Beacon“ švyturėlių išdėstymo patalpų žemėlapyje pavyzdys



5 pav. „Beacon“ švyturėlių išdėstymo ant salės atramos stulpų pavyzdys

Geokoordinatės nėra reikalingos, nes nustatyti savo ir patalpų buvimo vietos erdvėje nesiekiama, todėl susiejama su koordinatėmis patalpų viduje.

Pavyzdyje aukščiau rutuliai – tai atramos stulpai, ant kiekvieno iš kurių sumontuota po švyturėlį. Patalpos apribotos koordinatėmis (0,0), (9,0), (9,7), (0,7).

Iš pirmo žvilgsnio, nėra nieko sudėtingo nustatant objekto buvimo vietą – tiesiog skenuojame švyturėlius, nustatome jų koordinates, pagal RSSI išskaičiuojame nuotolį ir taip gauname savo buvimo vietą. Bet būtent čia ir prasideda visi sunkumai.

Realybė atneša savo korekcijas. Esmė tame, kad netgi tiesioginio matomumo su švyturėliu sąlygomis RSSI parametras šokinėja, chaotiškai keisdamas savo reikšmę, dėl ko be matematinio aparato panaudojimo nustatyti atstumą iki švyturėlio tampa sudėtinga.

Taip atsitinka dėl šių faktorių:

- Švyturėlio ir vartotojo įrenginio antenų siuntimo ar priėmimo kryptingumo orientacija ir charakteristika;
- Stambių objektų buvimas tarp švyturėlio ir vartotojo įrenginio (prie jų taip pat priskiriamas žmogus);
- Paviršių iš medžiagų, gerai atspindinčių radijo signalą, buvimas, o taip pat didelis „Beacon“ švyturėlių susigrūdimas vienoje vietoje dėl daugelio spindulių interferencijos su pagrindiniu spinduliu.

Pirmiausiai reikia surasti kiekvieno iš švyturėlių RSSI vidurkį. Tai atliekama taip:

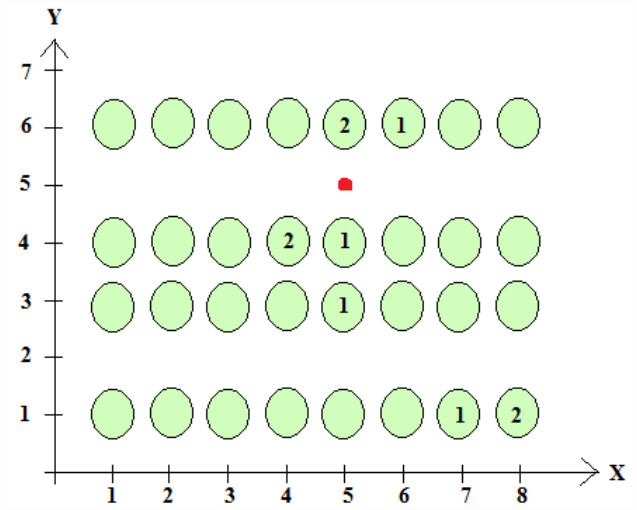
- Švyturėliai nustatomi išvesti duomenis su maksimaliu greičiu (kuo aukštesnis duomenų išvedimo periodiškumas, tuo daugiau duomenų vidurkio suradimui, taigi atitinkamai tuo didesnis ir tikslumas. Tačiau su duomenų išvedimo periodiškumo didinimu mažinamas švyturėlio baterijos resursas, taigi reikia ieškoti „auksinio vidurio“);
- Buferyje kaupiami duomenys su nustatytu periodiškumu (pvz., kartą per sekundę), remiantis buferyje sukauptais duomenimis išskaičiuojamas vidutinis RSSI kiekvienam iš švyturėlių (vėliau objekto koordinatės bus nustatomos kiekvieną sekundę remiantis šiais RSSI vidurkiais);
- Buferis išvalomas, ir kitą sekundę vėl kaupiami duomenys, ir taip ciklišai.

Toliau, po vidutinio RSSI išskaičiavimo, kiekvieną sekundę renkami trys švyturėliai su geriausiais RSSI vidurkiais ir pagal jų koordinates nustatoma objekto buvimo vieta – *trilateracijos* pagalba.

1.2.4 Rezultatų paklaidos mažinimo būdai

Taigi objekto buvimo vieta yra nustatyta. Tačiau čia ir atsiranda sunkumų. Pirmiausia, net jeigu vartotojas tiesiog stovi vietoje, jo padėtis vis tiek „šokinėja“. Praktikoje pavyksta gauti 3 metrų tikslumą.

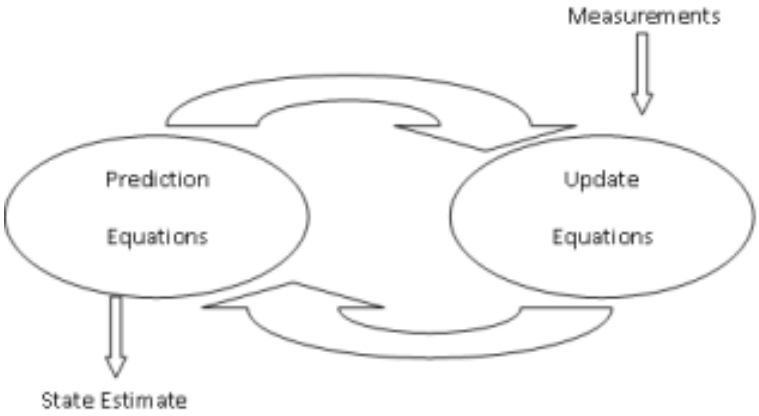
Pavyzdžiui, jeigu kiekvieną sekundę išrinkti po švyturėlį su geriausiu RSSI, tuomet po 10 matavimų bus gaunamas maždaug toks vaizdas (mastelis = 4 m):



6 pav. Švyturėlių išrinkimų skaičius pagal geriausią RSSI iš 10 bandymų

Raudonas taškas – tai įrenginio, kurio koordinatės nustatomos, buvimo vieta. Skaičiai apskritimuose – tai, kiek kartų konkretus švyturėlis turėjo geriausią RSSI iš 10 bandymų. Taigi gaunamas rezultatas vėl yra ne visai priimtinas.[7]

Tokiu atveju reikalingas tolimesnis matematinis gautų rezultatų apdorojimas. Kaip puikią priemonę galima naudoti **Kalmano filtrą** – jis panaikina matavimo triukšmus ir skaičiuoja rezultatą besiremdamas kaip dabartiniais matavimais, taip ir numatomais rezultatais, kurie gaunami pagal praeities matavimus. Filtras naudoja sistemos dinaminį modelį (judėjimo dėsnį) ir 2 besikartojančias ciklines stadijas: tai prognozavimas ir korekcija. Pirmuoju – prognozavimo etapu – išskaičiuojama sistemos būklė sekančiu laiko momentu, o antruoju – korekcijos etapu – prognozė koreguojama, naudojant eilinių matavimų rezultatus:[13]



7 pav. Kalmano filtro skaičiavimo principinė schema

Filtras turi galimybę atsižvelgti į valdantį poveikį, kuriuo gali būti, pvz., informacija iš akselerometro. Tai ženkliai pagerina rezultatą, ir aukščiau nagrinėtu atveju paklaida sudaro jau ne 3 metrus, bet 1–1,5 m.

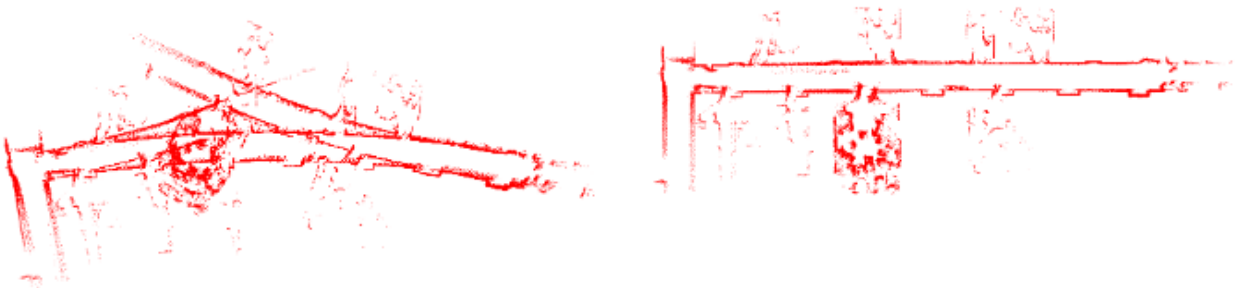
Kitas paklaidos mažinimo variantas – taikyti iš karto 2 navigacijos uždarose patalpose metodus: be „Beacon“ švyturėlių naudoti dar inercinę navigacijos sistemą (INS), kuri susideda iš akselerometro, giroskopo ir kompaso (magnetinio lauko daviklio). Akselerometras rodo veikiančių jėgų projekcijas. Giroskopas — kampinių greičių projekcijas. Magnetinio lauko daviklis – magnetinio lauko stiprumą. Tačiau duomenys iš šių įrenginių irgi reikalauja matematinio apdorojimo. Taip pat pageidautina naudoti ir temperatūros daviklį termokompensacijai, nes MEMS-giroskopai turi gana juntamą temperatūros paklaidą.

Tokiu atveju, taikant „Beacon“ ir INS metodus, koordinačių nustatymo paklaidą galima sumažinti net iki 20-30 cm.

1.3 Pozicionavimo principai

Vienas iš populiariausių sprendimų roboto navigacijai yra lygiagreti lokalizacija ir žemėlapio sudarymas (toliau – SLAM, nuo angl. *Simultenuos Localization and Mapping*) – tai yra bet kokios lokalizacijos algoritmo pagrindas (angl. *framework*). Pagrindinė SLAM algoritmo užduotis – nubraižyti alokacijos žemėlapį bei tuo pačiu metu nustatyti esamą poziciją. Teoriškai šiuo laiku SLAM uždavinys yra išspręstas, tačiau iš praktinės pusės SLAM vis dar yra sunku pritaikyti dėl naudojamų algoritmų sudėtingumo bei reikalavimų jutikliams.[9,10]

Paėmus tik roboto daviklius pozicionavimui nustatyti, dažniausiai dėl paklaidų bei pradinės pozicijos neapibrėžtumų robotas negali sukurti gero žemėlapio netgi po visų patalpų stebėjimų (vaizdas gaunamas iškraipytas). Tačiau pritaikius tam tikrą filtravimo algoritmą galima išvengti iškraipymų bei patobulinti gaunamą žemėlapį (7 pav.).

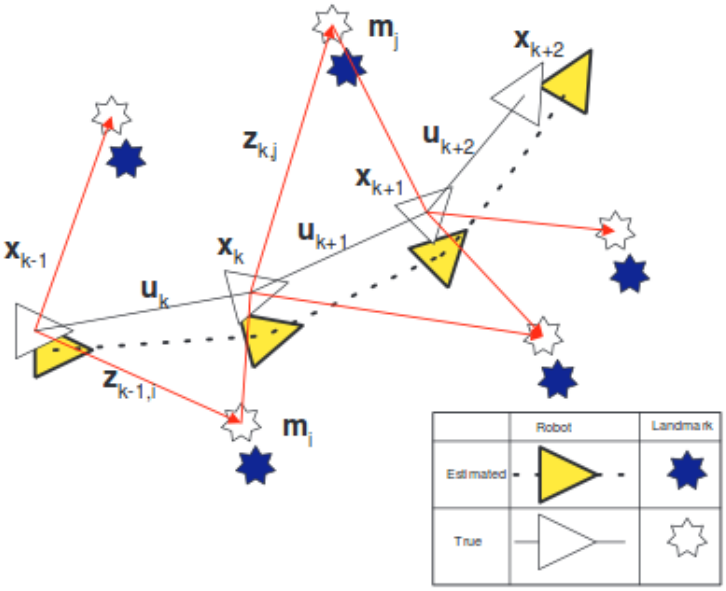


a) Žemėlapis nenaudojant filtravimo b) Žemėlapiui pritaikytas filtravimas

8 pav. (SLAM) Žemėlapis be filtravimo ir su filtravimu

1.4 SLAM uždavinio apibrėžimas

SLAM yra procesas, kurio metu objektas (robotas) nustato aplinkos žemėlapį (naudodamas jutiklių duomenis) bei pozicionuoja save toje aplinkoje (žemėlapyje) (8 pav.). SLAM metu objekto trajektorija ir kliūčių pozicija nustatomos realiu laiku, ir šios informacijos žinojimas nebūtinas prieš proceso paleidimą.



9 pav. SLAM principinė schema

Pagrindinį SLAM uždavinį sukuria sąlygos, kad kliūties bei objekto (roboto) pozicija nėra tiesiogiai nei žinoma, nei nustatoma. Įsivaizduokime, kad laiko momentu k robotas nustato, kad:

- x_k – pozicijos vektorius, apibrėžiantis objekto alokaciją ir orientaciją;
- u_k – perėjimo vektorius, apibrėžiantis objekto perėjimą iš pozicijos x_{k-1} į x_k;
- m_i – vektorius, apibrėžiantis m_i kliūties poziciją;
- z_{ik} – pastabos vektorius, apibrėžiantis kliūties i poziciją objekto (roboto) atžvilgiu momentu k;
- $X_{0:k}$ – pozicijos istorija;
- $U_{0:k}$ – kontrolinių taškų istorija;
- m – visų kliūčių pozicijos;
- $Z_{0:k}$ – visų kliūčių pastabos.

Taigi tikimybinė SLAM forma reikalauja apskaičiuoti tikimybės pasiskirstymą (1) kiekvienam k laikui.

$$P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \tag{1}$$

Toks skaičiavimas reikalauja tam tikrų modelių: judėjimo ir stebėjimo. Stebėjimo modelis apibrėžia tikimybę padaryti pastebėjimą z_k, kai objekto alokacija bei kliūties alokacija yra žinomos (2).

$$P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}). \tag{2}$$

Judėjimo modelis apibrėžia tikimybę apskaičiuoti objekto poziciją x_k naudojant tik prieš tai esančią poziciją x_{k-1} ir perėjimo vektorių u_k (3).

$$P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \tag{3}$$

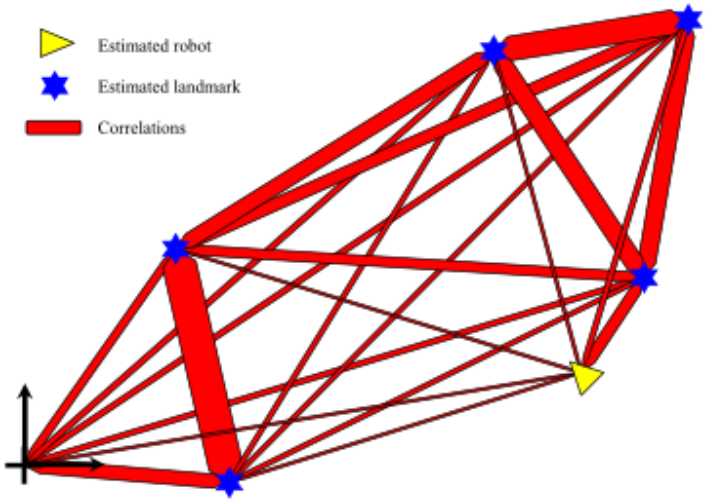
Tokiu atveju SLAM yra skaičiuojamas dviem etapais: pirmasis yra laiko atnaujinimas, o antrasis – matavimo atnaujinimas (4 - 5).[14, 16]

$$\begin{aligned} &P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0) \\ &= \int P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \\ &\times P(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0) \mathrm{d}\mathbf{x}_{k-1} \end{aligned} \tag{4}$$

$$\begin{aligned} &P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \\ &= \frac{P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}) P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}{P(\mathbf{z}_k \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})} \end{aligned} \tag{5}$$

Tai yra rekursinis metodas objekto pozicijai gauti bei žemėlapiui nustatyti laiko momentu k naudojant visų stebėjimų rezultatus bei perėjimo vektorius. Pats žemėlapis yra konstruojamas naudojant sintezę (angl. *fusion*) tarp visų pastebėjimo vektorių. Be to, laikant, kad kliūčių pozicija žemėlapyje yra žinoma, pačio objekto pozicija yra išskaičiuojama kliūčių atžvilgiu.[18]

Žiūrint iš praktinės pusės, išvardintų modelių apskaičiavimas priklauso faktiškai nuo kliūčių tarpusavio sąryšio ir atstumo tarp jų. Laikoma, kad tas atstumas visiškai nepriklauso nuo objekto judėjimo, tai yra kiekvienas kliūčių stebėjimas yra nepriklausomas įvykis, nes atstumas tarp kliūčių kiekvieno stebėjimo metu nesikeičia. Be to, kiekvieno judėjimo metu robotas koreguoja prieš tai esančių kliūčių poziciją (eliminuodamas paklaidas – stebėdamas tą patį kliūčių atstumą jau iš kitos pozicijos). Tai faktiškai sudaro koreliacijos žemėlapį tarp kliūčių (9 pav.)



10 pav. Kliūčių koreliacijos žemėlapis (SLAM)

Taigi SLAM uždavinio sprendimas yra modelių (2) ir (3) reprezentacijos paieška, kuri leidžia greitai ir tiksliai suskaičiuoti visus reikiamus vektorius. Dažniausiai modeliams apibrėžti naudojamas būklių erdvės modelis su Gauso triukšmu, kuris sprendžiamas naudojant praplėsto Kalmano filtro (angl. EKF – *Extended Kalman Filter*) algoritmą. Kita alternatyva, dažnai naudojama SLAM sprendimui – Rao-Blackwellised dalelių filtras arba greitasis lygiagrečios lokalizacijos ir žemėlapio sudarymo metodas (angl. *FastSLAM*). Tačiau skirtinguose taikymuose gali būti panaudojami ir kiti algoritmai.

1.4.1 Praplėstas Kalmano filtras (EKF)

Norint suprasti praplėsto Kalmano filtro (toliau – EKF, nuo angl. *Extended Kalman Filter*) esmę, reikėtų pradėti nuo paprasto vienos dimensijos Kalmano filtro. Tiesinės sistemos atžvilgiu objekto judėjimo modelį galima apibrėžti:

$$\mathbf{x}_{k+1} = F\mathbf{x}_k + G\mathbf{u}_k + \mathbf{w}_k, \tag{6}$$

kur x_k, x_{k+1} – sistemos būklė k ir k+1 laiko momentu, F – perėjimo matrica, G- kontrolės matrica ir w_k yra Gauso triukšmas su nuliniu vidurkiu. Tokiu atveju, tikroji sistemos būsena nėra apibrėžta, ir ją reikia numatyti kiekvieno laiko etapu naudojant proceso modelį bei stebėjimus. [13]

Stebėjimo modelis tokiu atveju atrodys taip:

$$\mathbf{z}_{k+1} = H\mathbf{x}_{k+1} + \mathbf{v}_{k+1}. \tag{7}$$

kur H yra stebėjimo matrica ir v_{k+1} yra Gauso triukšmas.

Sakykime, x būsena k laiko momentu gali būti aprašyta:

$$\mathbf{x}_k \sim N(\hat{\mathbf{x}}_k, P_k), \tag{8}$$

kur P yra tikimybinė matrica, o N – Gauso pasiskirstymo funkcija. Tuo tarpu x_{k+1} (po stebėjimo k laiko momentu) gali būti užrašyta:

$$\mathbf{x}_{k+1} \sim N(\hat{\mathbf{x}}_{k+1}, P_{k+1}) \tag{9}$$

kur būsenos įvertinimas $\hat{x}_{k+1}$ bei jos tikimybės matrica P_{k+1} gali būti apskaičiuota naudojant Kalmano filtrą:

$$\begin{aligned} \bar{\mathbf{x}}_{k+1} &= F\hat{\mathbf{x}}_k + G\mathbf{u}_k \\ \bar{P}_{k+1} &= FP_kF^T + Q \end{aligned} \tag{10}$$

Šitos formulės numato ateities reikšmes prieš stebėjimą, tuo tarpu po stebėjimo reikšmės koreguojamos:

$$\begin{aligned} \hat{\mathbf{x}}_{k+1} &= \bar{\mathbf{x}}_{k+1} + K(\mathbf{z}_{k+1} - H\bar{\mathbf{x}}_{k+1}) \\ P_{k+1} &= \bar{P}_{k+1} - KSK^T, \end{aligned} \tag{11}$$

kur inovacijos tikimybės vektorius S ($z_{k+1} - H\bar{x}_{k+1}$ vadinama inovacija – arba korekcija) ir Kalmano koeficientas K aprašomi taip:

$$S = H\bar{P}_{k+1}H^T + R$$
$$K = \bar{P}_{k+1}H^T S^{-1}. \tag{12}$$

H yra konstantų matrica, R – triukšmo tikimybė stebėjimo metu.

Priešingai Kalmano Filtro atvejui, praplėstas Kalmano filtras (EKF) taikomas nelinijinių modelių atveju.

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k, \tag{13}$$

ir

$$\mathbf{z}_{k+1} = \mathbf{h}(\mathbf{x}_{k+1}) + \mathbf{v}_{k+1}. \tag{14}$$

bei galutinė skaičiavimo formulė:

$$\hat{\mathbf{x}}_{k+1} = \bar{\mathbf{x}}_{k+1} + K(\mathbf{z}_{k+1} - \mathbf{h}(\bar{\mathbf{x}}_{k+1}))$$
$$P_{k+1} = \bar{P}_{k+1} - KSK^T, \tag{15}$$

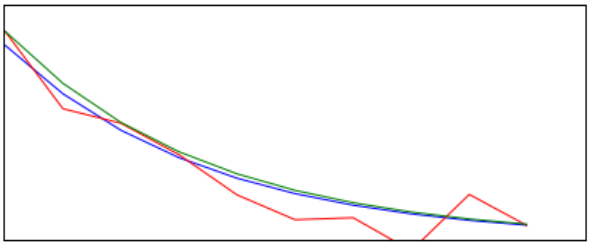
kur

$$S = \nabla\mathbf{h}\bar{P}_{k+1}\nabla\mathbf{h}^T + R$$
$$K = \bar{P}_{k+1}\nabla\mathbf{h}^T S^{-1}. \tag{16}$$

Nežiūrint į matematinių operacijų sudėtingumą, EKF galima paaiškinti paprastomis operacijomis. Norint su tam tikra apibrėžta tikimybe numatyti objekto poziciją, mums reikia žinoti du parametrus – apskaičiuotą naudojant modelį numatytą vertę bei korekcijos komponentę, su kuria būtų galimybė pakoreguoti numatymo modelį. Turint šiuos duomenis EKF ciklišk ai praeina keletą etapų: numatymą ir korekciją. Su kiekvienu etapu korekcijos bei numatymo paklaidos mažėja.

Sukuriant paprastą algoritmą galima pabandyti pritaikyti dvimačiu atveju Kalmano filtro algoritmą:

k	0	1	2	3	4	5	6	7	8	9
x_k	1000	750	563	422	316	237	178	133	100	75
z_k	1072	673	599	437	231	103	114	-56	233	72
p_k	1	0.561	0.315	0.177	0.1	0.056	0.031	0.018	0.01	0.006
$\hat{x}_k$	1072	804	603	452	339	254	191	143	107	80
g_k		0.003	0.002	0.001	0	0	0	0	0	0



11 pav. Kalmano Filtro imitacija

11 paveiksle mėlyna linija – tai tikroji trajektorija, raudona linija – triukšmingas stebėjimas, žalia linija – numatyta trajektorija.

1.4.2 Dalelių filtras

Daugeliu atvejų Kalmano filtras taikomas tuo metu, kai sistema yra tiesinė arba arti tiesinės, o triukšmas apibrėžiamas Gauso pasiskirstymu. Tačiau stipriai netiesinės sistemos arba ne Gausinio triukšmo atveju, Kalmano filtras nesugeba susidoroti su korekcijos etapais. Tokiu atveju dažniausiai naudojamas dalelių filtras (angl. *particle filtre*) arba nuoseklusis Monte-Carlo metodas. [19]

Dalelė yra objekto tikimybinė būsena (kad objektas yra toje vietoje), turinti savo svorį. Sakykime, turime dinaminę sistemą, aprašomą tokia lygtimi:

$$\mathbf{x}_k = \mathbf{f}_{k-1}\left(\mathbf{x}_{k-1}, \mathbf{v}_{k-1}\right) \tag{17}$$

kur f yra nelinijinė perėjimo funkcija ir v_k yra triukšmo funkcija. Tuo tarpu stebėjimo modelis:

$$\mathbf{z}_k = \mathbf{h}_k\left(\mathbf{x}_k, \mathbf{w}_k\right) \tag{18}$$

W_k yra triukšmo funkcija ir h_k yra stebėjimo funkcija. Taigi numatymo fazėje kiekvienam sistemos numatymui x_k apskaičiuojamas svoris w_k. Perėjus keletą ciklų artimesnės prie tikros pozicijos būsenos gauna didesnį svorį.

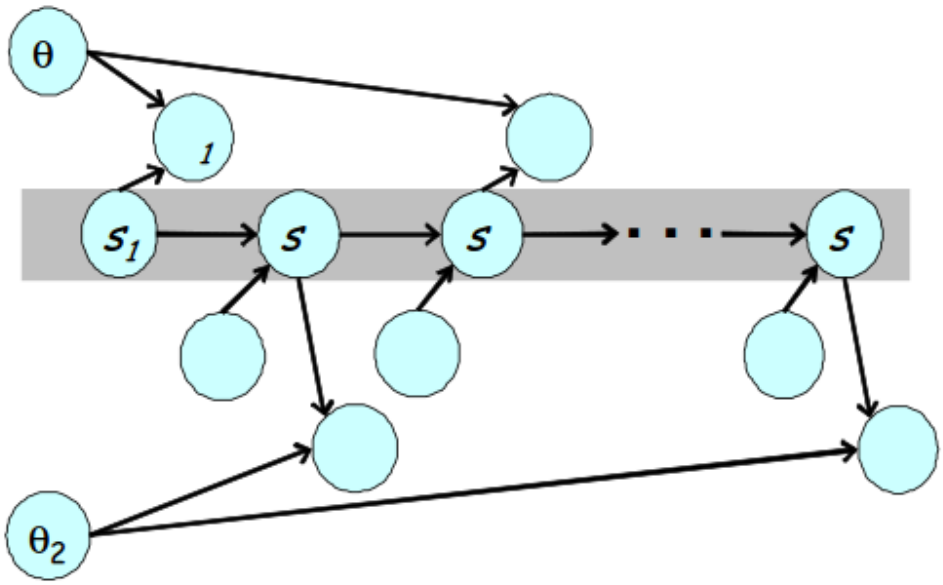
1.4.3 FastSLAM metodas

EKF naudojami SLAM uždaviniui spręsti turi keletą pagrindinių trūkumų – tai algoritmo kompleksiškumas bei didelė priklausomybė nuo klaidų asocijuojant poziciją – dėl ko šiuo metu robototechnikoje dažniausiai naudojamas greitasis lygiagrečios lokalizacijos ir žemėlapio sudarymo (angl. ir toliau – *FastSLAM*) metodas. Šio metodo pagrindas yra dalelių filtras, kurio pagrindinis algoritmas atrodytų taip:

- Gaunama nauja roboto trajektorija (naudojant prieš tai esančią poziciją ir kontrolinį vektorių);
- Kliūčių vektorius atnaujinamas po esamo stebėjimo;
- Kiekvienai daleliai priskiriamas svoris;
- Dalelės perskaičiuojamos naudojant jų svorius.

Toks algoritmas turi keletą pagrindinių privalumų, o būtent: algoritmo kompleksiškumas turi log(N) pavidalą, kas leidžia jį naudoti labai didelėse patalpose. FastSLAM dalelė taip pat neša informaciją apie roboto trajektoriją, dėl ko trajektorijos perskaičiavimas su nauja pozicija didina svorį tų dalelių, kurios teisingiausiai paskaičiuoja trajektoriją. Be to, papildomai dalelė turi informaciją apie kliūtį, kas leidžia pašalinti judančią kliūtį iš statistinio modelio esant tam tikram svoriui.

Kaip ir dalelių filtro atveju FastSLAM naudoja dinaminį Bayes tinklo apibrėžimą roboto judėjimui sumodeliuoti (11 pav.).[8,19]



12 pav. Dinaminis Bayes tinklas

Tačiau uždavinys yra performuluojamas kitaip – jeigu numanyti, kad roboto trajektorija yra žinoma, tai reiškia, kad pirmos kliūties pozicijos Θ_1 stebėjimas yra nepriklausomas nuo pozicijos Θ_2 stebėjimo, tai yra stebėjimo įvykiai yra nepriklausomi vienas nuo kito, ir, jei žinoma roboto trajektorija, tai vienos kliūties stebėjimas neduoda jokios informacijos apie kitą kliūtį. Naudojant tokį principą SLAM uždavinys gali būti padalintas į dvi dalis: pirma būtų roboto trajektorijos tikimybinė funkcija, o antra – N kliūčių pozicijos priklausomybės nuo roboto trajektorijos funkcija. Padalinus tokiu būdu algoritmą galima naudoti dalelių filtrą roboto trajektorijai gauti ir EKF kliūčių pozicijai nustatyti.

1.5 Jutikliai

Nepriklausomai nuo panaudoto algoritmo, visada egzistuoja korekcijos etapas, kurio metu numatytos vertės koreguojamos išmatuotų parametrų atžvilgiu. Norint sumažinti numatymo paklaidas reikia panaudoti jutiklinę sistemą, turinčią mažiausiai paklaidų. Šio skyriaus tikslas yra apžvelgti naudojamas jutiklines sistemas bei jų panaudojimą SLAM uždaviniui spręsti. Be to, apžvelgti jutiklių paklaidas mažinančius algoritmus, tokius kaip jutiklių sintezė (angl. *fusion*).[14,18]

Jutikliai gali būti skirstomi pagal energijos perėjimo principus:

- Aktyvūs – generuojantys kažkokį signalą bei priimantys rezultatą;
- Pasyvūs – tik priimantys signalą iš aplinkos.

Pagal kontaktą:

- Kontaktiniai – priimantys informaciją esant kontaktui;
- Bekontakčiai – gaunantys informaciją nutolę nuo aplinkos objektų.

Pagal matavimo informaciją:

- Vidiniai – matuoja vidinę roboto informaciją: greitį, poziciją, pagreitį, kampus ir t.t.;
- Išoriniai – matuoja aplinkos pokyčius: kameros, sonarai, lazeriai ir t.t.

Atstumo matavimams dažniausiai naudojami infraraudonųjų spindulių jutikliai, sonarai bei lazeriai (angl. LIDAR – *laser radar*). LIDAR leidžia vienu metu nustatyti objekto atstumą, greitį ir pagreitį ir yra vienas iš populiariausių sprendimų robotams šiuo metu.

Roboto būsenai nustatyti dažniausiai naudojami akselerometrai bei giroskopai.

1.5.1 Jutiklių sintezė (angl. *fusion*)

Jutiklių sintezė yra principas, kai kelių jutiklių duomenys naudojami norint išskaičiuoti papildomą informaciją, kurios negalima gauti naudojant tik vieną jutiklį. Pavyzdys gali būti toks: yra dvi kameros, nutolusios viena nuo kitos, abi grąžina plokščią vaizdą, tačiau panaudojus abiejų kamerų sintezę galima nustatyti atstumą iki matomų objektų. [18]

1.5.2 Radijo ryšio (Wi-Fi) pozicionavimas

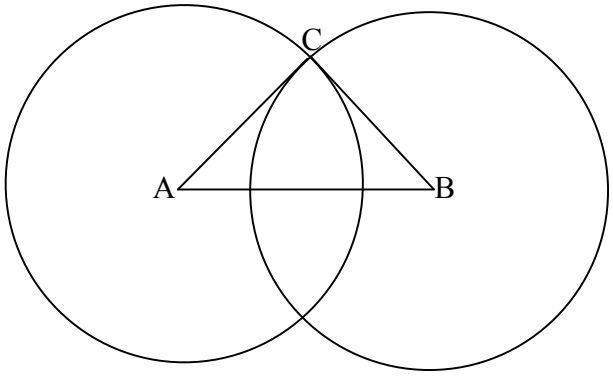
Pakankamai paprasta pozicionuoti objektą uždarose patalpose naudojant išorinius radijo švyturėlius – naudojant pelengavimo algoritmus. Tačiau tokie sprendimai reikalauja papildomos įrangos. Tuo tarpu Wi-Fi naudojamas daugumoje patalpų, dėl to kai kuriems mokslininkams kilo minčių panaudoti Wi-Fi uždarų patalpų navigacijai spręsti.

Šiuo metu egzistuoja daug skirtingų metodikų pozicijai nustatyti naudojant radijo ryšį:

- Persiuntimo laiko metodas (angl. *Cell-Identity*, TOA – *Time of Arival*),
- priėmimo laiko skirtumo metodas (angl. TDOA – *Time Difference of Arrival*),
- priėmimo kampo metodas (angl. AOA – *Angle of arrival*),
- signalo galios nustatymo būdas.

AOA metodas pozicijai nustatyti naudoja žinomų GSM stočių priėmimo kampą (žinant daugiau negu dviejų stočių pozicijas bei nuo jų atėjusio signalo kampą galima pozicionuoti imtuvą). TOA apskaičiuoja atstumą tarp imtuvo ir GSM stoties naudojant signalo sklidimo laiką.

Kaip matoma, daugelis radijo ryšio pozicijos nustatymo priemonių naudoja žinomą stoties poziciją bei atstumo iki stoties nustatymo metodą. Bendru atveju naudojamas trianguliacijos metodas (12 pav.), kurio metu žinant daugiau negu vieną atstumą iki žinomų taškų galima paskaičiuoti poziciją.



13 pav. Trianguliacijos metodo atvaizdavimas

Tokiu būdu atstumą tarp AB galima apskaičiuoti:

$$A(X_1, Y_1); B(X_2, Y_2) \rightarrow |AB| = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

2. PRAKTINĖ DALIS

Šio darbo tikslui įgyvendinti pasirinktas **tyrimo metodas** – eksperimentas (programinė imitacija). Tyrimas sudarytas iš dviejų dalių, kiekvienoje iš kurių bus naudojami du eksperimento modeliai. Pirmosios dalies **tikslas** yra nustatyti roboto poziciją naudojant virtualaus odometro, Kalmano bei dalelių filtro nufiltruotų duomenų rinkinį ir palyginti gautų pozicijos parodymų tikslumą realios roboto pozicijos atžvilgiu naudojant ir nenaudojant švyturėlius. Antroje eksperimento dalyje bus tiriama tikslumo priklausomybė nuo įvesto triukšmo jutikliuose. Tikslumas nustatomas apskaičiavus X ir Y kryptimi esančią paklaidą tarp suimituotų pozicijos rodmenų ir realios pozicijos laikui bėgant ir atsižvelgiant į visos imitacijos gautą standartinį klaidą.

2.1 Eksperimentų aprašymas

2.1.1 Imitacija

Norint įgyvendinti šio darbo uždavinius buvo pasirinktas imitacijos metodas, kuomet yra paruošiama programinė įranga, imituojanti paprasčiausio roboto veikimo principą 2D aplinkoje.

Taikant roboto imitaciją yra labai paprasta įvesti bet kokį jutiklių tipą ir jų generuojamų duomenų srautą, tuo pačiu sumažinti kaštus pačio roboto kūrimui paliekant daug laisvės roboto testavimui.

Imitacijoje sukuriamas dviejų dimensijų žemėlapis (XY) ašyse, kuris sudarytas iš laisvų ir užimtų segmentų. Kiekvienas segmentas atitinka tam tikrą žemėlapio kvadratą. Kvadratas turi apibrėžtą dydį. Žemėlapio duomenys paduodami matricos pavidalu (sudaryti iš vienetų: nulis – laisva koordinatė, vienetas – užimta, ir dvejetas – „Beacon“). Kiekvienas žemėlapio kvadratas turi koordinatę, kuri yra sveikas skaičius, tuo tarpu pozicija pačiame kvadrate gali būti nurodoma ir su skaičiais po kablelio. Kliūtis užima visą kvadratą. Imitacijos pozicijos, dydžio bei greičio dydžiai matuojami sąlyginiais vienetais, t.y. neturi dimensijos, ir pagal reikalavimą gali būti paprastai perskaičiuoti į SI sistemą. Imitacijos laikas skaičiuojamas „iteracijomis“ nuo imitacijos pradžios. Viena iteracija užima maždaug 0.23 s (priklausomai nuo kompiuterio pajėgumų).

Robotui suimituoti naudojamas automatinis algoritmas, kuris naudodamas pseudo-atsitiktinį krypties generatorių juda nustatyta kryptimi. Robotas, norint supaprastinti imitacijos algoritmą, skaitomas materialiuoju tašku. Tai yra robotas gali užimti bet kokį laisvos ląstelės tašką, išskyrus kliūties ląstelę. Robotas gali judėti bet kokia kryptimi, tačiau negali judėti pro kliūtį. Papildomai robotas gali judėti stačiai įstrižinei kliūčių sekcijai. Roboto greitis laikomas pastoviu (tai yra robotas juda be pagreičio).

Šiuo metu robotas turi keletą jutiklių tipų (triukšmingų):

- Greičio matuoklis;
- Virtualus odometras (X ir Y kryptimi);
- Kompasas;
- Kontaktinis jutiklis roboto priekyje (detektuoja užimtą ląstelę);
- Atstumo iki radijo švyturėlių matuoklį.

Kiekvienas iš jutiklių turi nepriklausomą triukšmo generatorių su užduodamu triukšmo lygiu. Triukšmas yra generuojamas naudojant kompiuterio atsitiktinių skaičių generatorių – *random.uniform* Python funkciją (4 eilutė):

```
1 import random
2
3 def add_noise(level, *coords):
4     return [x + random.uniform(-level, level) for x in coords]
```

Triukšmas yra nustatomas naudojant tolygų pasiskirstymo algoritmą. Jutikliai naudoja keletą triukšmo lygių skirtingiems jutikliams:

```
1 def add_little_noise(*coords):
2     return add_noise(0.02, *coords)
3
4 def add_some_noise(*coords):
5     return add_noise(0.1, *coords)
```

Tai yra 2 % ir 10 % nuo jutiklio paduodamo maksimalaus lygio. Vienintelis jutiklis, kuris neturi paklaidos imitacijos atveju, yra priekinis sienos detektorius.

Imitacija padalinta į dvi sritis – tai realioji ir natūralioji. Realiosios imitacijos sritis atsako už realią roboto poziciją žemėlapio atžvilgiu – tai yra neturinti jokių paklaidų. Natūralioji sritis yra roboto jutiklių duomenų sritis, kur roboto pozicija išskaičiuojama naudojant esamus jutiklių duomenis.

2.1.2 Virtualaus odometro ypatumai

Virtualus odometras iš tikrųjų įrašo ne visą praeitą maršrutą tam tikra kryptimi, o atstumą tarp roboto pradinio taško ir roboto pozicijos tam tikra kryptimi. Tai yra odometro vertė gali būti ir neigiamo dydžio. Kaip pavyzdys, vienmatis robotas statomas į X koordinatę su verte 5, pirmos iteracijos metu jis nuvažiuoja teigiama kryptimi 5 koordinates (tikro odometro vertė sutampa su virtualaus odometro verte – 5), antros iteracijos metu robotas nuvažiuoja 5 koordinates neigiama kryptimi (tikro odometro vertė yra 10, virtualaus odometro vertė yra 0).

2.2 Eksperimento eiga

Pirmojo etapo eksperimento atveju robotas pozicionuojamas atsitiktinėje vietoje, tačiau robotui žinoma nustatyta pradinė koordinatė bei žemėlapio išdėstymas. Pozicijai apskaičiuoti tokiu atveju naudojami virtualaus odometro duomenys X ir Y kryptimis. Kiekvieno iteracijos etapo pabaigoje robotas pastumiamas atsitiktine kryptimi N koordinačių (kur N yra nustatytas greitis) realiojoje srityje. Kryptis nustatoma virtualaus kompaso, kur 0 sutampa su šiaurės žemėlapio pozicija, skaičiuojama radianais. Po roboto pastūmimo virtualus odometras perskaičiuoja X ir Y prieaugį (atsižvelgdamas į kryptį).

Realiosios srities pastūmimo funkcija:

```
1 def radvance_by(self, speed, checker=None):
2     r = math.radians(self.h)
3 
4     dx = math.sin(r) * self.speed
5     dy = math.cos(r) * self.speed
6 
7     if checker is None or checker(self, dx, dy):
8         self.rmove_by(dx, dy)
9         return True
10 
11      return False
```

Kaip matoma kodo pavyzdyje, realioji sritis nenaudoja paklaidų apibrėžimo, nes imitacijos atveju į realiosios pozicijos paklaidą nėra atsižvelgiama – rezultatas skaičiuojamas realios pozicijos atžvilgiu.

Natūraliosios srities skaičiavimo principas:

```
1 def move(self, maze):
2     """
3     Robotas pastumiamas į kitą poziciją, prieš pastumiant tikrinama, ar esama kryptis nėra
```

```
užimta. Iš pradžių robotas pastumiamas realiojoje srityje, paskui paskaičiuojami natūralios
srities rodmenys.
    """
    while True:
        self.step_count += 1
        if self.radvance_by(self.speed,
                            checker=lambda r, dx, dy: maze.is_free(r.x + dx, r.y + dy)):

            #Paimami kompaso rodmenys – su paklaida. Ir išskaičiuojamas kryptinis greitis
su paklaida.
            h = self.sh
            vx, vy = add_some_noise(self.speed, self.speed)

            r = math.radians(h)
            #Nustatomos virtualaus odometro vertės
            self.ox += math.sin(r) * vx
            self.oy += math.cos(r) * vy



            break
         # Robotas pasukamas atsitiktine kryptimi
         self.chose_random_direction()
```

2.2.1 Kompiuterio ir programinio paketo konfigūracija

Imitacijai atlikti naudojami programiniai paketai:

- „Python 2.7.11“ 64 bitų ir jo standartiniai paketai (NumPY, MatplotLIB.).
    - Kalmano filtro modulis „Filterpy 0.1.2“,
    - Dalelių filtro implementacija įgyvendinta Pythono kalbos pagalba,
- „Fedora 22“ 64 bitų operacinė sistema.

Naudojami kompiuterio parametrai:

- „Dell Latitude E5440“,
    - 8GB RAM DDR3 1600Mhz,
    - Inter Core I5.

2.2.2 Vertinimo kriterijai

Norint pasiekti tikslą, reikia turėti galimybę įvertinti naudojamų algoritmų tikslumą. Bet kokios navigacijos užduoties tikslumą galima išmatuoti kaip paklaidą tarp išmatuotos ir realios pozicijos, išreikštą procentais žemėlapio ląstelės dydžio atžvilgiu (toliau – **pozicijos paklaida**). Be to, būtų

prasminga nustatyti ribą, ties kuria būtų galima teigti, kad gaunami rezultatai yra pakankamai tikslūs. Šiai problemai spręsti šiame darbe naudojamas mažiausio skirtumo koeficientas, kuris parodo, su kokia tikimybė išmatuotos koordinatės sutampa N % tikslumu su realia koordinate (darbe naudojama 1 % riba) – toliau darbe tai bus vadinama **tikslumu**:

```
1 p1 = sum(abs(a)<0.1 and abs(b)<0.1 for a,b in np.column_stack((rezpx, rezpy)))
2 m1 = sum(abs(a)<0.1 and abs(b)<0.1 for a,b in np.column_stack((rezmx, rezmy)))
3 h1 = sum(abs(a)<0.1 and abs(b)<0.1 for a,b in np.column_stack((rezhx, rezhy)))
```

Kur rez* pažymėti kintamieji yra atitinkamai virtualaus odometro, dalelių filtro bei Kalmano filtro X ir Y koordinačių skirtumas su realiomis vertėmis.

Papildomai darbe atsižvelgiama į kiekvieno algoritmo paklaidos **pasiskirstymo histogramas** ir **vidutinį kvadratinį nuokrypį.**

Taip pat norint įvertinti skirtingų algoritmų efektyvumą atsižvelgiama į algoritmo vienos iteracijos trukmę.

Laiko matavimas vykdomas mks tikslumu (10^{-6}s) naudojant *Python time()* funkciją (įeinančią į „time“ paketą). Laikas matuojamas tik tam tikrame kodo bloke – kuris vykdo algoritmo skaičiavimo funkciją. Eksperimento metu laikas matuojamas kiekvienos iteracijos metu, ir eksperimento gale apskaičiuojamas išmatuotų laikų vidurkis.

2.2.3 Eksperimentuose naudojami modeliai

Pirmasis eksperimentų modelis (be švyturėlio)

Pirmąjį eksperimento modelį sudaro dalelių bei Kalmano filtrai, ir jų apskaičiavimui bus naudojamos tik virtualaus odometro vertės be papildomų korekcijų švyturėlių atžvilgiu. Eksperimento modelio **tikslas** – trim būdais (dalelių filtro, virtualaus odometro ir Kalmano filtro) nustatyti išmatuotos roboto pozicijos tikslumą laikui bėgant (esant 100 iteracijų), išmatuoti standartinį kvadratinį nuokrypį ir palyginti rezultatus.

Pirmojo modelio dalelių filtro konfigūracija

Dalelių filtrui yra sukurta svorio funkcija, kurios pagalba bus nufiltruojamos „netikslios“ dalelių pozicijos. Svorio funkcija yra paprastas Gauso langas su pasirinkta dispersija = 0,9.

$$f(a,b) = e^{-\frac{(a-b)^2}{2\sigma^2}} \quad (19)$$

```
def w_gauss(a, b):
    error = a - b
    g = math.e ** -(error ** 2 / (2 * sigma2))
    return g
```

Kur a – atitinka roboto išmatuotą vertę, b – dalelės gautą vertę. Svorio funkcija faktiškai yra tikimybinė funkcija, parodanti, su kokia tikimybe dalelės modelis atitinka išmatuotas vertes – Gauso funkcija centruojama roboto išmatuotos pozicijos vertės atžvilgiu.

Kiekvienos iteracijos metu dalelių svoris yra perskaičiuojamas (į perskaičiavimą įtraukiamos tik tos dalelės, kurių pozicija atitinka laisvąją ląstelę).

```
for p in particles:
    if world.is_free(*p.xy):
        px,py = p.xy
        p.w = w_gauss(mx, px) * w_gauss(my, py)
    else:
        p.w = 0
```

Taigi svoriui nustatyti apskaičiuojama sąsūka tarp X ir Y tikimybių. Kuo arčiau dalelės yra X pozicijos ir Y pozicijos, tuo didesnis svoris. Akivaizdu, kad esant nepriklausomam paklaidų X ir Y virtualaus odometro pasiskirstymui, algoritmas turėtų būti subalansuotas ties realia verte.

Po svorio perskaičiavimo vyksta svorių normalizavimas:

```
nu = sum(p.w for p in particles)
if nu:
    for p in particles:
        p.w = p.w / nu
```

Toliau vyksta modelio balansavimas – procesas, kurio metu parenkamos dažniausios pagal svorį dalelės, ir sistema papildoma naujomis dalelėmis.

```
# Sukuriamas svorio pasiskirstymas
dist = WeightedDistribution(particles)

for _ in particles:
    p = dist.pick()
    if p is None:  # Tuo atveju, kai neišeina išskirti "sunkiausios" dalelės
        new_particle = Particle.create_random(1, world)[0]
    else:
        new_particle = Particle(p.x, p.y,
                    heading=robbie.sh if ROBOT_HAS_COMPASS else p.h)
     new_particles.append(new_particle)

particles = new_particles
```

Iteracijos pabaigoje dalelės yra pastumiamos ta pačia kryptimi, kaip ir robotas:

```
d_h = robbie.sh - old_heading

for p in particles:
   p.h += d_h
   p.radvance_by(robbie.get_speed())
```

Tokiu atveju, per keletą iteracijų filtras subalansuoja ir paskirsto visas daleles aukščiausios tikimybės (roboto realios pozicijos) atžvilgiu, ir pozicijai nustatyti naudojamas vidurkis tarp visų esamų dalelių:

```
def compute_mean_point(particles):
    m_x, m_y, m_count = 0, 0, 0
   for p in particles:
        m_count += p.w
        m_x += p.x * p.w
        m_y += p.y * p.w
    if m_count == 0:
        return -1, -1, False
    m_x /= m_count
    m_y /= m_count
    # Paskaičiuojama, kiek dalelių patenka į 95% ribą
    m_count = 0
    for p in particles:
        if world.distance(p.x, p.y, m_x, m_y) < 1:
            m_count += 1
    return m_x, m_y, m_count > PARTICLE_COUNT * 0.95
```

Imitacijos metu naudojama 2000 dalelių.

Pirmojo modelio Kalmano filtro konfigūracija

Kalmano filtras yra linijinių funkcijų balansavimas. Pirmojo eksperimento metu Kalmano filtras naudojamas subalansuoti filtro parametrams paprasto statinio judėjimo modelio atžvilgiu:

$$\begin{bmatrix} x \\ V_x \\ y \\ V_y \end{bmatrix}' = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ V_x \\ y \\ V_y \end{bmatrix},$$

kur

$$F = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

yra perėjimo matrica. Tuo tarpu matavimo matrica H yra pagrista tik dviem matavimo kintamaisiais – z matavimo matrica yra dydžio 2 x 1, ir kintamųjų dydis yra 4 x 1, tuo tarpu perėjimo

funkcijos dydis būtų 2 x 4. Matuojami kintamieji yra X ir Y dydžiai – taigi perėjimo matrica H yra labai paprasta:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

tai yra

$$z'_x = (1 \cdot x) + (0 \cdot V_x) + (0 \cdot y) + (0 \cdot V_y)$$

$$z'_y = (0 \cdot x) + (0 \cdot V_x) + (1 \cdot y) + (0 \cdot V_y)$$

Tuo tarpu matavimo triukšmo matricą galima sukonfigūruoti atsižvelgiant į tai, kad X ir Y matavimai yra nepriklausomi Gauso procesai – tai yra, kad X ir Y triukšmo dedamosios yra visiškai nepriklausomos tarpusavyje ir yra normaliai pasiskirsčiusios. Nustatoma pakankamai aukšta vertė (pusė žemėlapio dydžio):

$$R = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}.$$

Tuo tarpu galutinis filtro kodas atrodys taip:

```
f1 = KalmanFilter(dim_x=4, dim_z=1)
f1.F = np.array([
   [1.0, dt, 0.0, 0.0],
   [0.0, 1.0, 0.0, 0.0],
   [0.0, 0.0, 1.0, dt],
   [0.0, 0.0, 0.0, 1.0]
])
f1.R = np.eye(2) * 5.
f1.Q = np.eye(4) * .1
f1.x = np.array([[0,0,0,0]], dtype=float).T
f1.P = np.eye(4) * 500
f1.H = np.array([
   [1.0, 0.0, 0.0, 0.0],
    [0.0, 0.0, 1.0, 0.0]
])
```

Kiekvienos iteracijos metu nustatomi esamo roboto matavimo rezultatai ir bandoma perskaičiuoti modelį:

```
z = np.array([[mx], [my]])
hx, hy = f1.x[0, 0], f1.x[2, 0]
f1.predict()
f1.update(z)
```

Antrasis eksperimentų modelis (su švyturėliais)

Pagrindinis skirtumas tarp pirmojo ir antrojo eksperimento modelių yra tai, kad antruoju atveju į žemėlapį talpinami du radijo švyturėliai, ir robotas įjungia distancijos matuoklį, veikiantį trianguliacijos pagrindu tarp roboto ir švyturėlių:

```
1 class TRSensor(object):
2     def __init__(self, pos_a, pos_b, noise_factor=0.2):
3         self.A = pos_a
4         self.B = pos_b
5         self.noise_factor = noise_factor
6     def range_of (self, pos):
7         ra = math.sqrt((self.A[0] - pos[0])**2 + (self.A[1] - pos[1])**2)
8         rb = math.sqrt((self.B[0] - pos[0])**2 + (self.B[1] - pos[1])**2)
9         return add_noise(self.noise_factor, ra, rb)
```

Kaip matoma iš kodo 9 eilutės, prie jutiklio signalo pridedamas papildomas triukšmas naudojant *add_noise* funkciją.

Antrojo modelio dalelių filtro konfigūracija

Pagrindinis dalelių filtro skirtumas šiame eksperimente yra svorio funkcijos apskaičiavimas:

```
1 px,py = p.xy
2 pra, prb = d.range_of((px,py))
3 p.w = w_gauss(mx, px) * w_gauss(my, py) * w_gauss(ra, pra) * w_gauss(rb, prb)
```

Į svorio skaičiavimą papildomai įvedamas koeficientas, priklausomas nuo roboto ir dalelės išskaičiuotos distancijos (3 eilutė).

Antrojo modelio Kalmano filtro konfigūracija

Kalmano filtro atveju tenka atlikti keletą pakeitimų, nes Kalmano filtras labiau tinka tiesinėms problemoms spręsti, tuo tarpu atstumo iki švyturėlių apskaičiavimas nebus tiesinis uždavinys. Perėjimo funkcija lieka ta pati:

$$\begin{bmatrix} x \\ V_x \\ y \\ V_y \end{bmatrix}' = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ V_x \\ y \\ V_y \end{bmatrix}.$$

Tačiau matavimo funkcija gali būti gauta naudojant Pitagoro teoremą:

$$z_A = \sqrt{(x - x_A)^2 + (y - y_A)^2} + v_A$$
$$z_B = \sqrt{(x - x_B)^2 + (y - y_B)^2} + v_B,$$

kur x_A, y_A ir x_B, y_B atitinkamai yra pirmo ir antro švyturėlio koordinatės, o v_A ir v_B – kiekvieno matavimo triukšmas.

Tokiu būdu matoma, kad matavimo funkcijos nėra tiesinės, ir jas būtų sunku panaudoti Kalmano filtro atveju. Taigi esant sąlygai, kad apytikslė roboto pozicija iš tikrųjų žinoma, galima suvesti matavimo funkcijas į tiesinį pavidalą roboto koordinačių atžvilgiu ir apskaičiuoti dalinę H išvestinę koordinačių atžvilgiu:

$$\frac{\partial H}{\partial x} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \dots \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

arba

$$\frac{\partial h_1}{\partial x} = \frac{\partial}{\partial x}\sqrt{(x - x_A)^2 + (y - y_A)^2} = \frac{1}{2} \cdot 2(x - x_A) \cdot ((x - x_A)^2 + (y - y_A)^2)^{-\frac{1}{2}}$$

$$\frac{\partial h_1}{\partial x} = \frac{(x - x_A)}{\sqrt{(x - x_A)^2 + (y - y_A)^2}}$$

Arba, atsižvelgiant į tai, kad $(x - x_A)$ yra x ašies atstumas nuo roboto iki švyturėlio A, o $\sqrt{(x - x_A)^2 + (y - y_A)^2}$ yra tiesinis atstumas nuo roboto iki švyturėlio A, galima performuoti formulę į trigonometrinį pavidalą:

$$\frac{\partial h_1}{\partial x} = -cos\theta_A$$

kur θ_A yra kampas tarp roboto buvimo vietos x ašies ir tiesinės linijos tarp roboto ir švyturėlio.

Tęsiant dalinių išvestinių skaičiavimą x, y, dx, ir dy atžvilgiu, galima nustatyti tokią matavimo matricą:

$$\frac{\partial H}{\partial x} = \begin{bmatrix} -cos\theta_A & 0 & -sin\theta_A & 0 \\ -cos\theta_B & 0 & -sin\theta_B & 0 \end{bmatrix}$$

Kur 0 atitinka greičio komponentės matavimą, tačiau informacija apie atstumą iki švyturėlių neduoda mums jokios informacijos apie greitį.

Be to, šiuo atveju mes skaičiuojame ne tikrą matavimo funkciją, bet jos pokytį, kuris nėra pastovus ir keičiasi su roboto judėjimu. Tuo tarpu matavimo funkcija neduoda informacijos apie roboto poziciją, bet tik apie roboto greitį švyturėlių atžvilgiu. Taigi reikia pertvarkyti perėjimo funkciją, kad ji atitiktų ne tikros pozicijos matavimą, bet pozicijos pokytį:

$$\begin{bmatrix} \Delta x \\ \Delta V_x \\ \Delta y \\ \Delta V_y \end{bmatrix}' = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta V_x \\ \Delta y \\ \Delta V_y \end{bmatrix}.$$

Taip pat kiekvienos iteracijos metu reikės apskaičiuoti dalinę išvestinę *H:*

```
def H_of (pos, pos_a, pos_b):
    theta_a = atan2(pos_a[1]-pos[1], pos_a[0] - pos[0])
    theta_b = atan2(pos_b[1]-pos[1], pos_b[0] - pos[0])
    return np.array([[0, -cos(theta_a), 0, -sin(theta_a)],
                     [0, -cos(theta_b), 0, -sin(theta_b)]])
```

Pačio filtro programinis kodas:

```
ra, rb = d.range_of((mx, my))
hx, hy = mx + f1.x[0, 0], my + f1.x[2, 0]
rx, ry = d.range_of((hx, hy))
z = np.array([[ra - rx], [rb - ry]])
# paskaičiuojame H esamu momentu
f1.H = H_of((mx, my), pos_a, pos_b)
f1.predict()
f1.update(z)
```

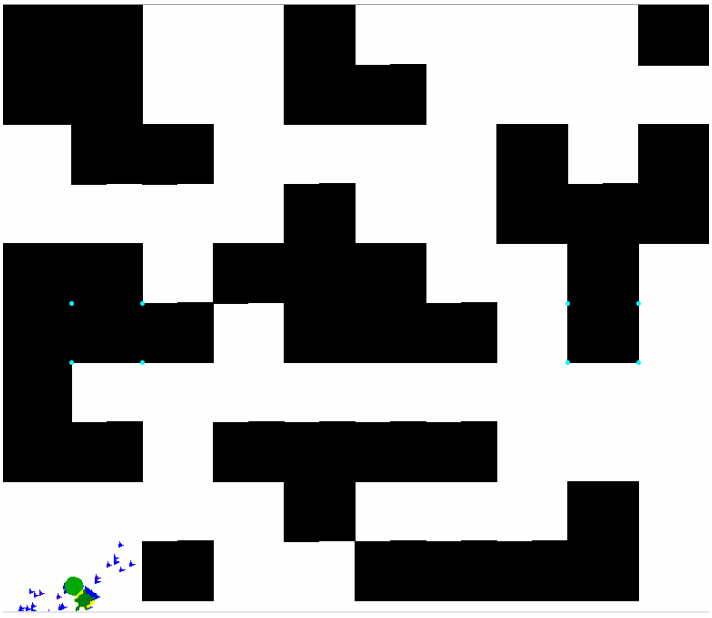
Tokiu būdu kiekvienos iteracijos metu koreguojama roboto pozicija švyturėlių atžvilgiu.

2.3 Eksperimentų rezultatai ir jų aptarimas

Eksperimento metu naudojamas žemėlapis:

```
maze_data = ( ( 1, 1, 0, 0, 1, 0, 0, 0, 0, 1 ),
              ( 1, 1, 0, 0, 1, 1, 0, 0, 0, 0 ),
              ( 0, 1, 1, 0, 0, 0, 0, 1, 0, 1 ),
              ( 0, 0, 0, 0, 1, 0, 0, 1, 1, 1 ),
              ( 1, 1, 0, 1, 1, 1, 0, 0, 1, 0 ),
              ( 1, 2, 1, 0, 1, 1, 1, 0, 2, 0 ),
              ( 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
              ( 1, 1, 0, 1, 1, 1, 1, 0, 0, 0 ),
              ( 0, 0, 0, 0, 1, 0, 0, 0, 1, 0 ),
              ( 0, 0, 1, 0, 0, 1, 1, 1, 1, 0 ))
```

Grafinis žemėlapio atvaizdavimas:



14 pav. Grafinis imitacijos žemėlapio atvaizdavimas

Žemėlapis sudarytas iš matricos: 10 x 10 ląstelių su 0, 1 ir 2 reikšmėmis, kur:

0 – laisva ląstelė,

1 – siena,

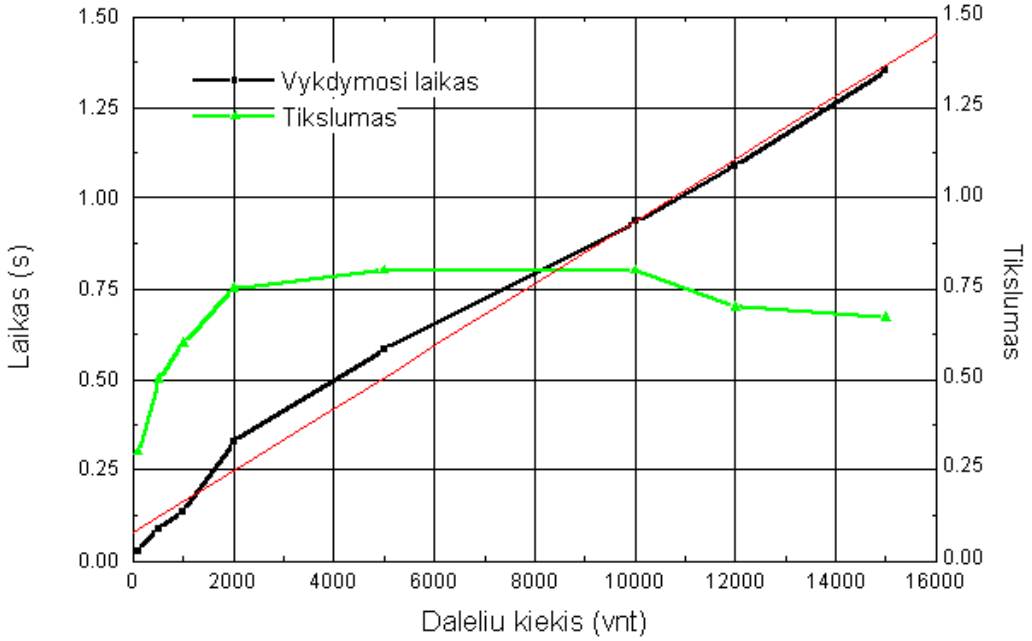
2 – švyturėlis (kartu su siena).

1 lentelė. Pagrindiniai žemėlapio žymėjimai

Žymėjimas	Reikšmė
Žalias vėžlys	Realioji roboto padėtis
Pilkas/Žalias rutulys	Dalelių filtro vidurkio padėtis (žalias – kai 95 % dalelių pataiko į tą patį kvadratą, pilkas – priešingu atveju)
Mėlynas trikampis	Natūralioji roboto padėtis (gaunama iš virtualaus odometro)
Geltonas rutulys	Kalmano filtro pakoreguota padėtis

2.3.1 Eksperimento pirmoji dalis

Eksperimento pirmojoje dalyje bus tiriama, kaip skirtingų algoritmų tikslumas ir pozicijos paklaida priklauso nuo eksperimente naudojamų švyturėlių įjungimo/išjungimo, esant vienam ir tam pačiam signalo ir triukšmo santykiui visuose jutiklių rodmenyse.



15 pav. Dalelių filtro tikslumo bei vykdymo laiko priklausomybė nuo dalelių kiekio.

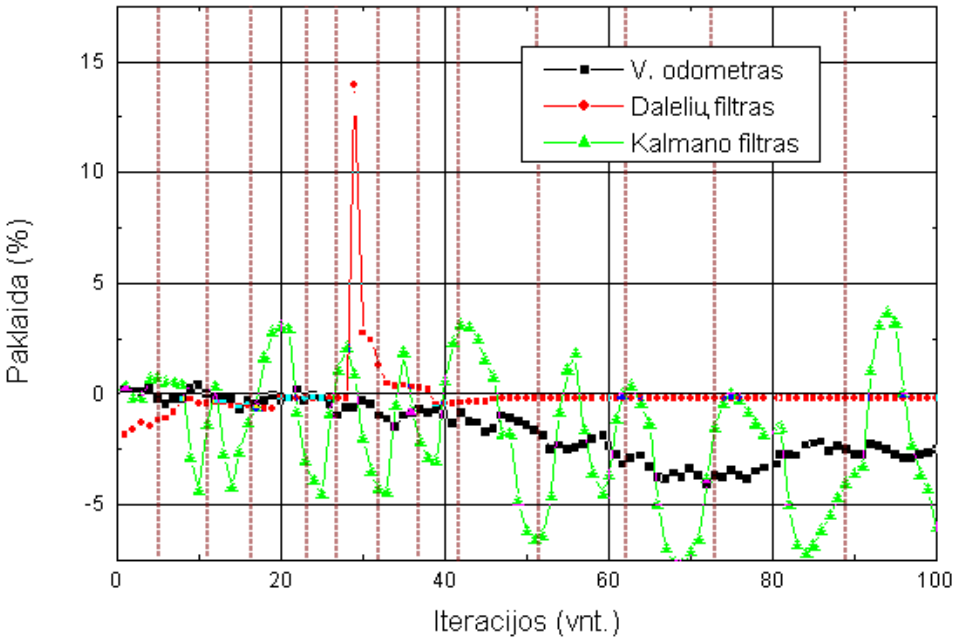
Norint tinkamai ir optimaliai parinkti dalelių kiekį dalelių filtrui (norint išvengti nereikalingo uždelsimo), buvo ištirta dalelių filtro tikslumo bei vykdymo laiko priklausomybė nuo dalelių kiekio, taikant pirmąjį eksperimento modelį. Kaip matoma iš pav. 15, vykdymo laikas tiesiškai priklauso nuo dalelių kiekio (su statumo koeficientu $0{,}07*10^{-3}$), tuo tarpu tikslumas stipriai didėja atkarpoje nuo 100 iki 2000 dalelių. Atsižvelgiant į šiuos rezultatus visiems eksperimentams atlikti buvo parinktas 2000 dalelių skaičius, nes didinant dalelių kieki, tiesiškai didėja imitacijos laikas, tačiau tikslumo vertė išlieka maždaug ta pati.

Pirmosios eksperimento dalies rezultatai taikant pirmąjį modelį (be švyturėlių)

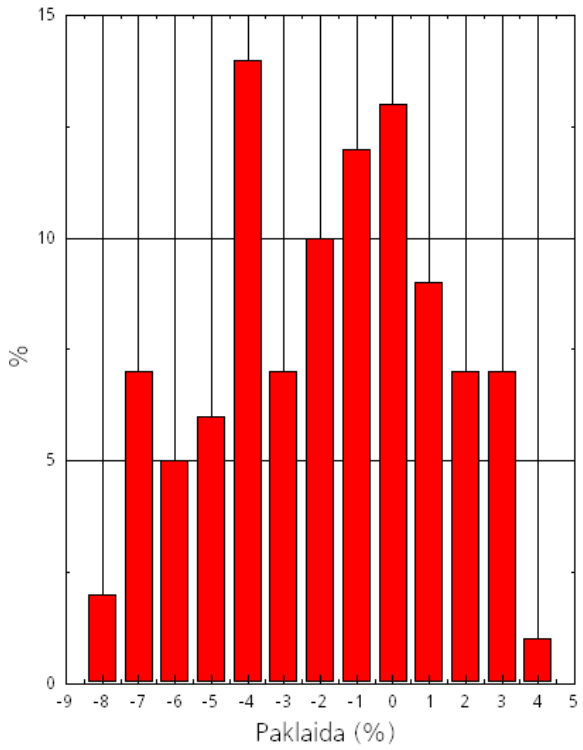
Pirmoji eksperimento dalis buvo atlikta naudojant 20dB signalo/triukšmo parinktą santykį ir atliktą imitaciją su 100 iteracijų skaičiumi. Buvo išmatuotos paklaidos, dispersijos bei tikslumas. Esminės pozicijos paklaidos priklausomybės nuo iteracijų skaičiaus pavaizduotos 16 pav., kur „Normal“ atitinka virtualaus odometro vertes, „Particle“ – dalelių filtro vertes bei „Kalman“ – Kalmano filtro vertes. Be to, pav. 15-17 pateiktos paklaidų histogramos, o 2. lentelėje – vidutinės kvadratinės paklaidos ir tikslumo palyginimas.

Kaip matoma iš pirmosios eksperimento dalies rezultatų, Kalmano filtro pozicijos paklaidos vertė stipriai varijuoja esant roboto trajektorijos pokyčiams (raudona brūkšninė linija). Tai galima

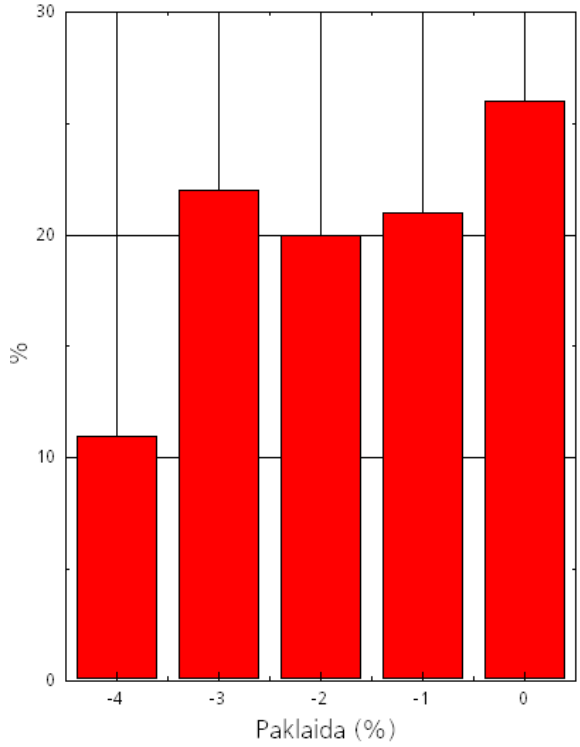
paaiškinti tuo, kad eksperimento metu buvo panaudotas linijinis Kalmano filtro modelis, kuris netinkamas esant chaotiškiems roboto krypties pasikeitimams (nepaisant to, kad linijinis roboto greitis buvo pastovus). Tuo tarpu matoma, kad Kalmano tikslumo reikšmė taip pat yra gana mažos vertės – 0,07 palyginus su dalelių filtro vertėmis – 0,71. Be to, iš eksperimento rezultatų matoma sąlyginai didelė paklaida (dispersija) dalelių filtro atžvilgiu bei „išsišokimai“ pozicijos paklaidų verčių grafike (pav.16). Tai gali būti paaiškinta tuo, kad dalelių filtro atžvilgiu, filtro pozicija skaičiuojama kaip vidurkis tarp visų dalelių pozicijų. Tuo metu, kai dalelių pozicija perskaičiavimo metu patenka į sienos vietą, algoritmas patalpina dalelę į atsitiktinę vietą žemėlapyje. Išskirtiniu atveju, kai dauguma dalelių robotui chaotiškai judant „atsitrenka“ į sieną bei atsitiktinai paskirstomos žemėlapyje, visų dalelių pozicijos vidurkis nukrypsta nuo realios vertės.



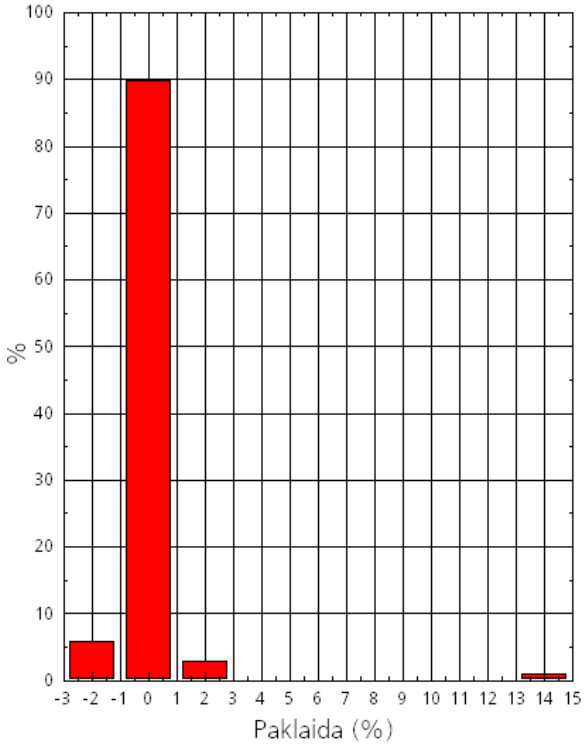
16 pav. Pozicijos paklaidų verčių priklausomybė nuo iteracijos kiekio taikant modelį be švyturėlių.



17 pav. Kalmano filtro paklaidos (%) histograma taikant modelį su švyturėliais



18 pav. Virtualaus odometro paklaidos (%) histograma taikant modelį su švyturėliais



19 pav. Dalelių filtro paklaidos (%) histograma taikant modelį be švyturėlių

Kaip matoma iš 17-19 pav., beveik visos dalelių filtro pozicijos paklaidų vertės patenka į [-1, 1] verčių diapazoną. Tuo tarpu Kalmano bei virtualaus odometro paklaidos verčių histograma yra žymiai platesnė.

Iš eksperimento rezultatų taip pat matoma, kad esant pakankamai dideliam (20 dB) signalo ir triukšmo santykiui virtualaus odometro tikslumo vertė yra net didesnė nei linijinio Kalmano filtro atveju.

2 lentelė. Tikslumo bei vidutinės kvadratinės klaidos palyginimas taikant modelį be švyturėlių.

	Virtualus odometras	Dalelių filtras	Kalmano filtras
Vidutinė kvadratinė klaida	0,24	1,29	0,88
Tikslumas	0,21	0,71	0,07

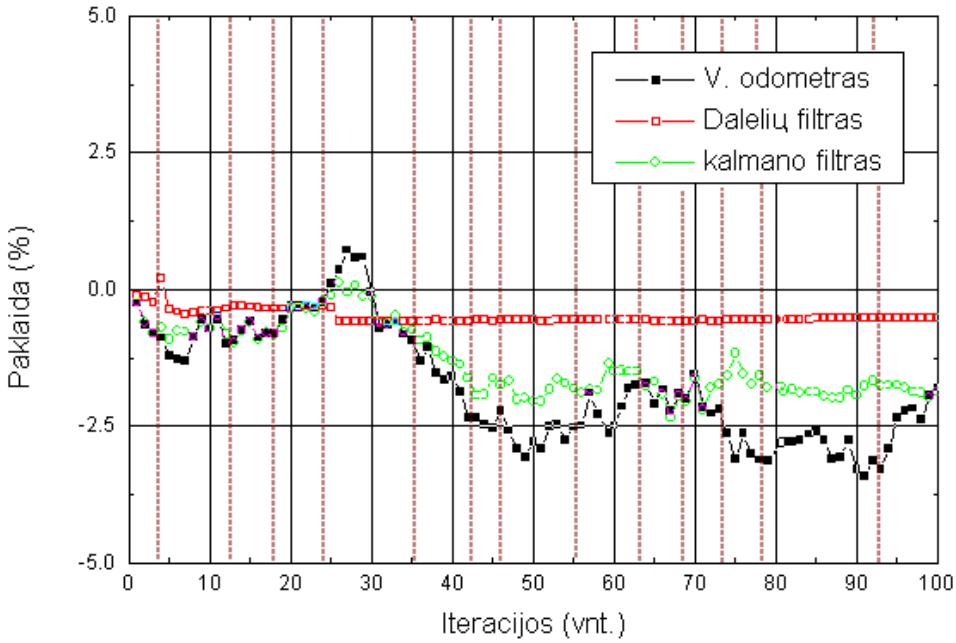
Pirmosios eksperimento dalies rezultatai taikant antrąjį modelį (su švyturėliais)

Eksperimento metu taikant antrąjį modelį, signalo ir triukšmo santykis taip pat buvo nustatytas 20 dB, ir imitacija buvo atlikta su 100 iteracijų skaičiumi su įjungtais švyturėliais.

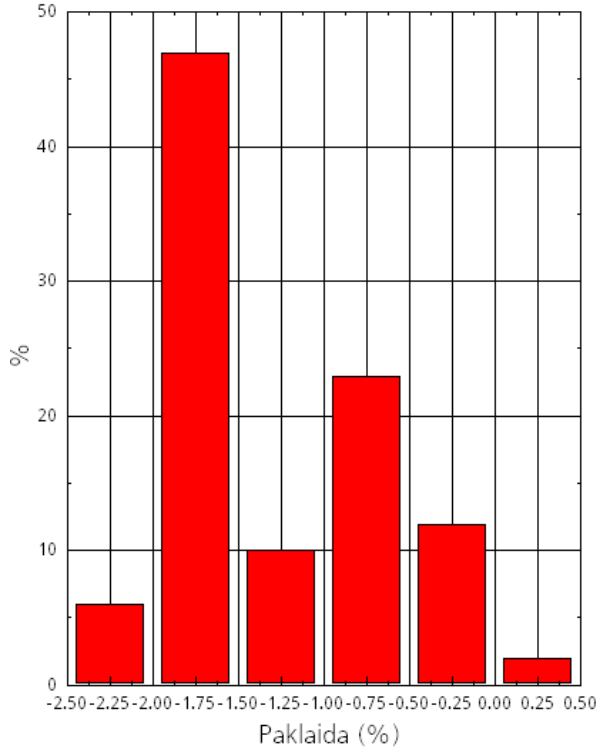
Kaip matoma iš rezultatų, švyturėlių įjungimas padidino abiejų algoritmų tikslumą. Kalmano filtro atveju, perskaičiavus modelį naudojant netiesinius algoritmus (EKF), fluktuacijos esant roboto krypties pokyčiams buvo panaikintos. Iš išmatuotų tikslumo verčių matoma, kad dalelių filtro bei Kalmano filtro vidutinės paklaidos sumažėjo: Kalmano – ~8 kartus, dalelių filtro – ~100 kartų. Tuo tarpu tikslumas išaugo: Kalmano ~700 % ir Dalelių filtro ~26 %. Be to virtualaus odometro vertės beveik nepasikeitė.

Taip pat iš pateiktų histogramų matoma, kad Kalmano filtro bei dalelių paklaidos verčių histogramos plotis sumažėjo. Nepaisant pagerėjusių rezultatų, dalelių filtro tikslumas lieka žymiai geresnis, negu Kalmano filtro atveju.

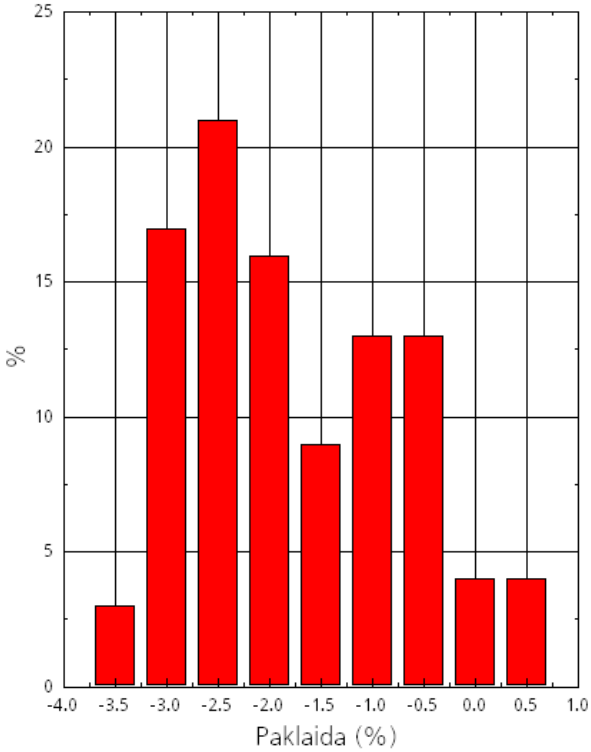
Be to, eksperimento metu matoma, kad virtualaus odometro tikslumas nukrypsta nuo nulinės vertės laikui bėgant (didėjant iteracijų skaičiui). Tai galima paaiškinti virtualaus odometro ypatumais: su kiekviena iteracija odometras „kaupia“ prieš tai esamas klaidas.



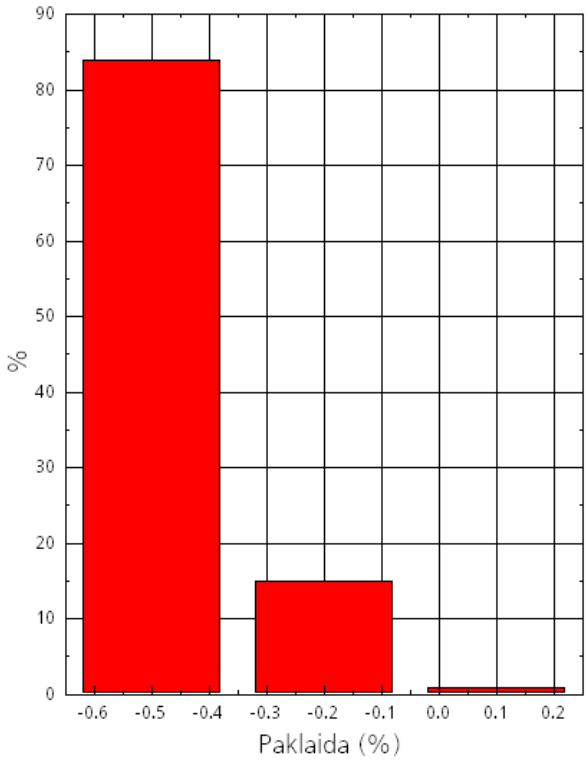
20 pav. Pozicijos paklaidų verčių priklausomybė nuo iteracijos kiekio taikant modelį su švyturėliais



21 pav. Kalmano filtro paklaidos (%) histograma taikant modelį su švyturėliais



22 pav. Virtualaus odometro paklaidos (%) histograma taikant modelį su švyturėliais



23 pav. Dalelių filtro paklaidos (%) histograma taikant modelį su švyturėliais

3 lentelė. Tikslumo bei vidutinės kvadratinės klaidos palyginimas taikant modelį su švyturėliais.

	Virtualus odometras	Dalelių filtras	Kalmano filtras
Vidutinė kvadratinė klaida	0,18	0,009	0,10
Tikslumas	0,2	0,90	0,61

2.3.2 Eksperimento antroji dalis

Antrosios eksperimento dalies metu bus nustatoma kiekvieno eksperimento modelio priklausomybė nuo signalo/triukšmo santykio, esančio jutiklių kanaluose. Signalo/triukšmo santykis perskaičiuojamas pagal standartinę formulę:

$$SNR(dB) = 20log_{10}\left(\frac{A_s}{A_t}\right),$$

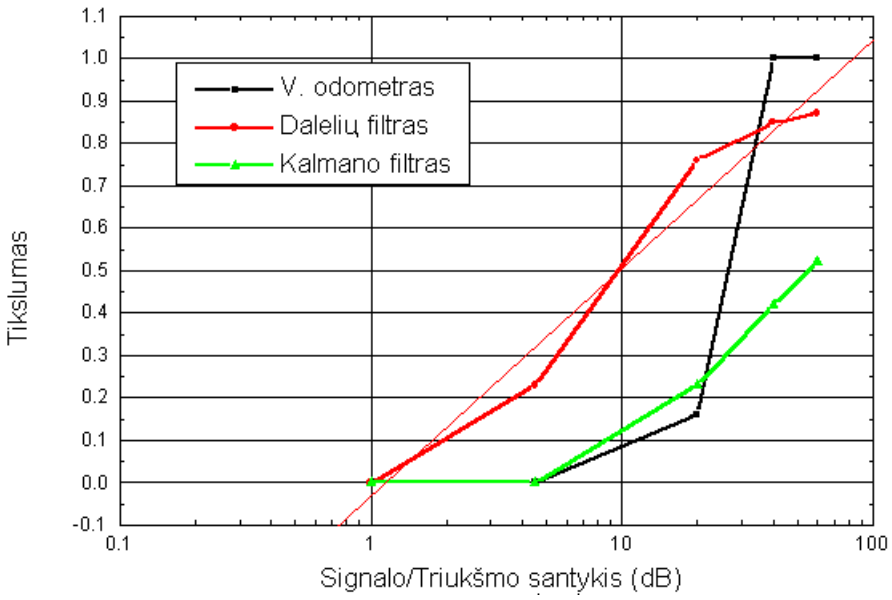
kur A_s – signalo amplitudė (imitacijos atveju signalo lygis matuojamas santykiniais vienetais), ir A_t – triukšmo lygis.

Eksperimentas buvo atliktas parinkus 60, 20, 10, 4.5, 1 dB signalo ir triukšmo santykius. Jo metu buvo išmatuotos tikslumo bei vidutinės kvadratinės paklaidos vertės taikant aukščiau aprašytus eksperimentų modelius.

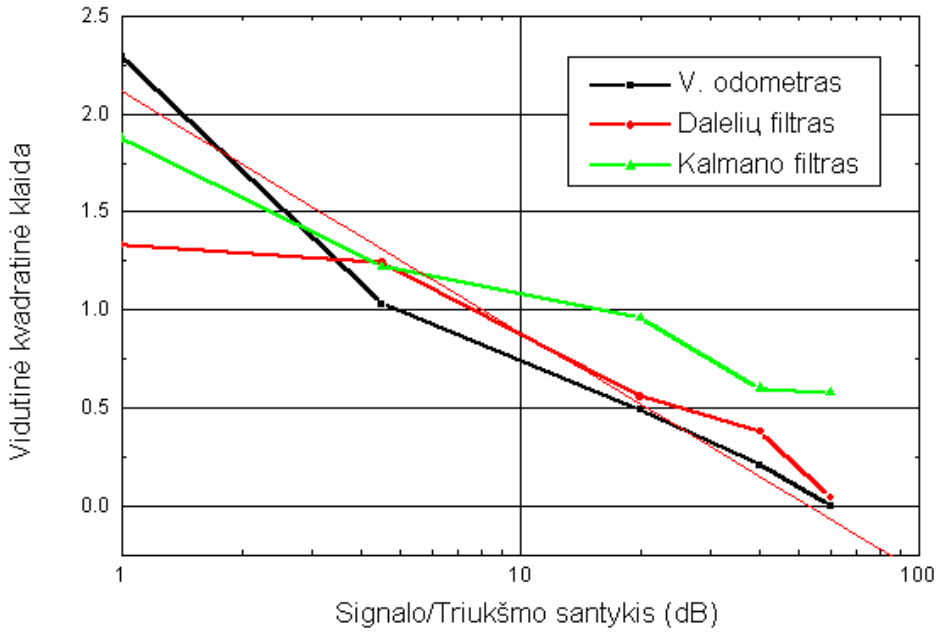
Antrosios eksperimento dalies rezultatai taikant pirmąjį modelį (be švyturėlių)

Taikant pirmąjį modelį tikslumo vertės leidžia pasakyti, kad virtualaus odometro bei Kalmano filtro atveju tikslumo vertės greitai slopinamos signalo/triukšmo santykiui mažėjant, kas savo ruožtu parodo, jog Kalmano filtras (linijinių modelių atveju) stipriai priklauso nuo esamo triukšmo lygio.

23 paveikslėlyje nagrinėjant vidutinės kvadratinės paklaidos priklausomybės nuo signalo/triukšmo lygio eksponentinį grafiką, galima teigti, kad Kalmano filtras šiek tiek sumažina paklaidų dydžio priklausomybę nuo signalo/triukšmo lygio paprasto virtualaus odometro atžvilgiu, tačiau tiesinė priklausomybė išlieka. Tuo tarpu dalelių filtro atžvilgiu matoma netiesinė vidutinės kvadratinės paklaidos priklausomybė nuo triukšmo lygio, tai yra mažėjant signalo / triukšmo lygio santykiui (didėjant triukšmui), paklaidos vertė keičiasi mažiau bei tikslumas mažėja liečiau. Tai reiškia, kad dalelių filtras geriau išfiltruoja nepriklausomus triukšmo šaltinius. Papildomai dalelių filtro tikslumo priklausomybė nuo signalo/triukšmo kreivė (24 pav.) buvo aproksimuota naudojant tiesinę aproksimaciją su statumo koeficientu 0,53.



24 pav. Tikslumo nuo signalo/triukšmo lygio (dB) priklausomybės eksponentinė išraiška taikant modelį be švyturėlių

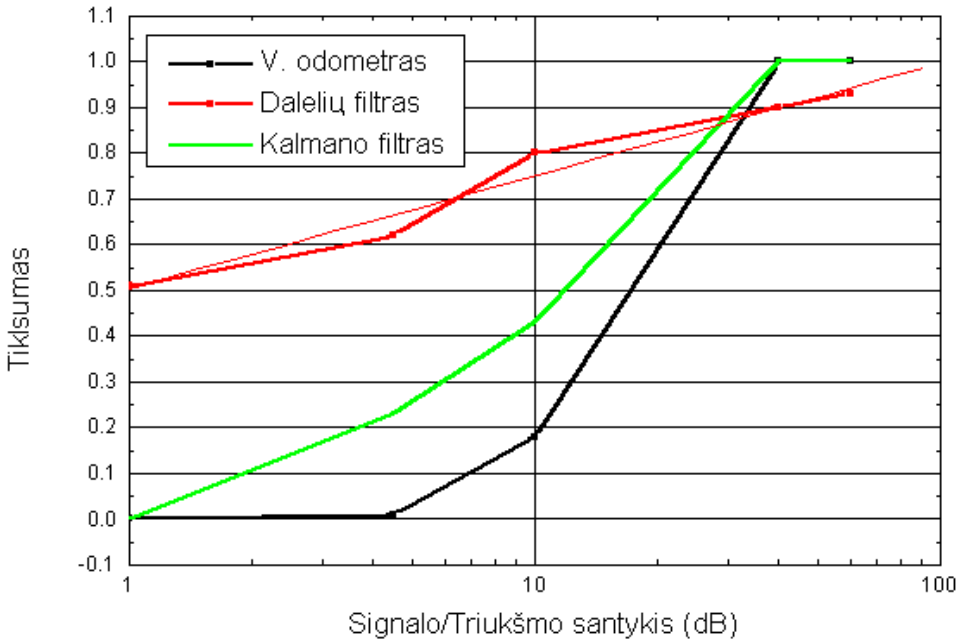


25 pav. Vidutinės kvadratinės klaidos nuo signalo / triukšmo santykio (dB) eksponentinė priklausomybė taikant modelį be švyturėlių

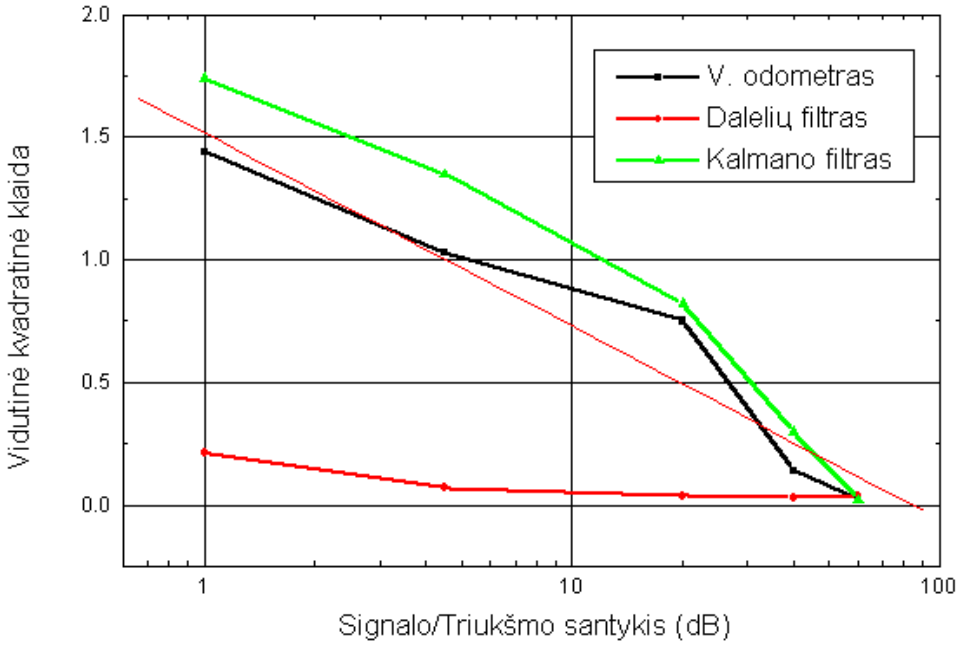
Antrosios eksperimento dalies rezultatai taikant antrąjį modelį (su švyturėliais)

Taikant antrąjį modelį buvo įjungiami švyturėliai bei tiriama tikslumo priklausomybė nuo įvesto triukšmo.

Kaip matoma iš eksperimento rezultatų, Kalmano filtro tikslumo vertės padidėjo, tačiau vis tiek išlieka tiesinė vidutinių kvadratinių paklaidų nuo signalo/triukšmo lygio priklausomybė. Tuo tarpu dalelių filtro rezultatai tampa stabilesni – tikslumo rezultatų tiesinės aproksimacijos statumo koeficientas mažėja – 0,28. Tai yra pridedant papildomus pozicijos matavimo šaltinius, dalelių filtro bei Kalmano filtro atveju pozicija nustatoma tiksliau. Tačiau dalelių filtras labiau reaguoja į papildomus informacijos šaltinius – iš 25 pav. matoma, kad dalelių filtro vidutinės kvadratinės paklaidos lyginant su pirmu modeliu stipriai sumažėjo ir yra labai arti nulinės ašies su labai mažu linijinės aproksimacijos statumo koeficientu ~0,06.



26 pav. Tikslumo nuo signalo/triukšmo lygio (dB) priklausomybės eksponentinė priklausomybė taikant modelį su švyturėliais

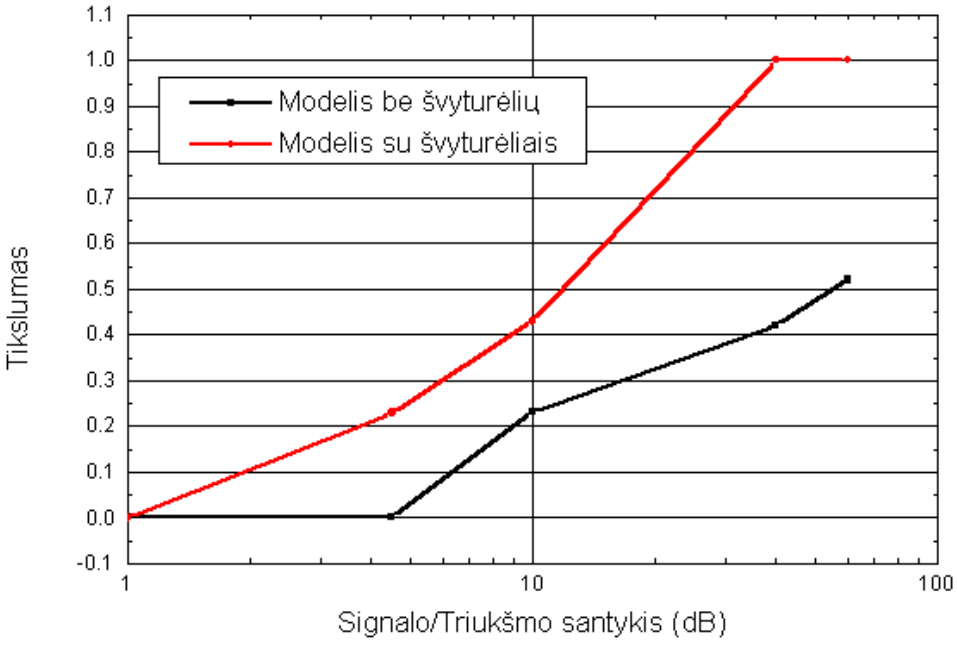


27 pav. Vidutinės kvadratinės klaidos nuo signalo/triukšmo santykio (dB) eksponentinė priklausomybė taikant modelį su švyturėliais

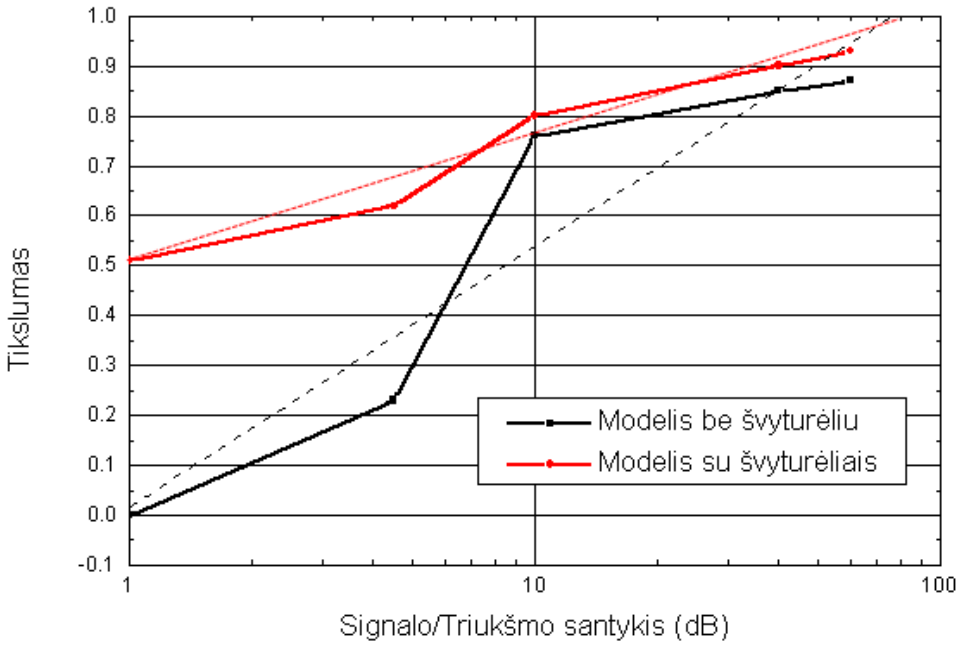
Nepaisant papildomo pozicijos matavimo šaltinio (švyturėlių įjungimo), pozicijos matavimo tikslumas yra gana mažas esant labai stipriam triukšmui (signalo / triukšmo santykis – apie 1dB): dalelių filtro atveju – apie 0,5, o Kalmano filtro – apie 0.

Antros eksperimento dalies skirtingų modelių rezultatų palyginimas

Iš rezultatų duomenų buvo išrinkti Kalmano filtro bei dalelių filtro tikslumo vertės eksperimentų modelių be ir su švyturėliais palyginimui. Iš pav. 26 -27 matoma, kad švyturėlių įjungimas pozityviai įtakoja abiejų algoritmų tikslumą Dalelių filtro atveju tikslumą galimą tiesiškai aproksimuoti logaritminėje skalėje, o šios tiesinės aproksimacijos statumo koeficientai lygūs 0,53 ir 0,28 taikant modelį be ir su švyturėliais atitinkamai. Kalmano filtro atveju tikslumo vertės netiesiškai priklauso nuo triukšmo lygio, kas apsunkina tiesinį modelių palyginimą. Tačiau lyginant absoliutines vertes esant atitinkamoms signalo/triukšmo santykių vertėms galima pastebėti ~2 kartus didesnę tikimybę tiksliai nustatyti koordinatę.



28 pav. Abiejų eksperimento modelių tikslumo priklausomybės nuo signalo/triukšmo (dB) palyginimo grafikas (Kalmano filtras)



29 pav. Abiejų eksperimento modelių tikslumo priklausomybės nuo signalo / triukšmo (dB) palyginimo grafikas (Dalelių filtras)

Papildomai norint įvertinti skirtingų algoritmų efektyvumą buvo išmatuoti algoritmų vykdymo laikai esant 20 dB signalo/triukšmo lygiui. Jie pateikti 4 lentelėje. Eksperimente laikoma, kad virtualaus odometro apskaičiavimo laikų galima nepaisyti (dėl algoritmo paprastumo) Kalmano filtro ir dalelių filtro vykdymo laiko atžvilgiu.

4 lentelė. Skirtingų algoritmų vykdymo laikų vidurkis ir sunormuoti laikai taikant eksperimento modelį be ir su švyturėliais.

	Dalelių filtras	Kalmano filtras
Pirmasis eksperimento modelis (be švyturėlių)	0,037452 s	0,001838 s
Antrasis eksperimento modelis (su švyturėliais)	0,058144 s	0,001949 s

5 lentelė. Skirtingų algoritmų apskaičiuotas tikslumas

	Dalelių filtras	Kalmano filtras
Pirmasis eksperimento modelis (be švyturėlių)	0,71	0,07
Antrasis eksperimento modelis (su švyturėliais)	0,9	0,61

Kaip matoma 4-5 lentelėse, nors dalelių filtras su didele tikimybe nustato tikslią poziciją, tačiau jo vykdymo laikas yra 30 kartų didesnis, negu Kalmano filtro atveju. Tai yra dalelių filtras 1.5 kartų tiksliau nustato koordinatę, tačiau 30 kartu lėčiau veikia. Šis rezultatas parodo, kad taikymuose, kur reikalaujama labai greito pozicijos koregavimo (esant dideliems judėjimo greičiams), esamo dalelių filtro greičio gali neužtekti, ir pozicijos vertės gali būti vėluojančios realaus laiko atžvilgiu. Tuo tarpu Kalmano netiesinis modelis gali duoti didesnį tikslumą neužlėtintos pozicijos vertės dėka.

Taip pat matoma, kad dalelių filtro atveju, norint pridėti papildomų pozicijos duomenų šaltinių (švyturėlių), vykdymo laikas didėja ~40 %, kas savo ruožtu gali riboti papildomų duomenų šaltinių kiekį. O Kalmano filtro atveju didelio skirtumo tarp tiesinio ir netiesinio modelio apskaičiavimo nepastebėta.

IŠVADOS IR PASIŪLYMAI

Atlikus pirmąją ir antrąją eksperimento dalis, kuriose buvo ištirti dviejų skirtingų modelių gaunami rezultatai, įvertintos tikslumo vertės, pozicijos paklaida, išmatuotos vidutinė kvadratinė paklaida ir vykdymo laikas, buvo pastebėti skirtingų parametrų pokyčiai.

Buvo ištirta dalelių filtro tikslumo bei vykdymo laiko priklausomybė nuo dalelių kiekio siekiant nustatyti optimalią dalelių kiekio vertę. Rezultatai parodo, kad optimali dalelių kiekio vertė yra 2000, nes viršijus šį kiekį algoritmo vykdymo laikas tiesiškai didėja, tačiau tikslumo vertės keičiasi labai siaurame diapazone ($\pm$0,1).

Kalmano filtro atveju linijinis Kalmano modelis (modelis be švyturėlių) stipriai varijuoja robotui chaotiškai keičiant kryptį, tačiau papildžius modelį švyturėliais (perskaičiavus modelį) pavyko išvengti fluktuacijų. Vis dėlto nepaisant fluktuacijų, pirmosios eksperimento dalies atveju Kalmano filtro pozicijos paklaida neviršija 10 % taikant modelį be švyturėlių ir 2,5 % taikant modelį su švyturėliais, esant 20 dB signalo / triukšmo lygiui. Be to, imitacijos metu Kalmano filtro tikslumas padidėjo nuo 0,07 pirmojo modelio atveju iki 0,61 antrojo modelio atveju. Tai yra prijungiant papildomus informacijos apie lokaciją šaltinius (švyturėlius), Kalmano filtro atveju tikslumas padidėjo beveik devynis kartus, tačiau vykdymo laikas padidėjo vos 6 %. Atsižvelgus į vidutinę kvadratinę paklaidą taip pat pastebima, jog modelio su švyturėliais metu gaunamas žymiai siauresnis pozicijos paklaidos pasiskirstymo spektras (vidutinis kvadratinis nuokrypis sumažėjo nuo 0,88 iki 0,10 vertės – ~9 kartus).

Dalelių filtro atveju taip pat pastebimas tikslumo padidėjimas pridedant švyturėlius – tikslumo vertė padidėjo nuo 0,7 iki 0,9, tuo pačiu pozicijos vidutinis kvadratinis nuokrypis sumažėjo nuo 1,29 iki 0,009. Tačiau buvo pastebėta, jog dalelių filtras turi pasitaikančių „išsišokimų“ (pikų) tuo metu, kai dauguma esamų dalelių lokalizuojasi uždaruose žemėlapio regionuose (sienoje). O antrasis eksperimentų modelis dėl padidėjusio tikslumo (pozicijos paklaida neviršija ~1 %) šio reiškinio nėra paveiktas. Vertinant algoritmų vykdymo laiką, pastebima, jog dalelių filtro vykdymo laikas stipriai priklauso nuo papildomų pozicijos informacijos šaltinių (švyturėlių) ir didėja beveik 56 % (milisekundžių eilės) palyginus modelius be ir su švyturėliais. Tačiau Kalmano filtro vykdymo laikas yra 30 kartų trumpesnis (mikrosekundžių eilės) esant tom pačiom sąlygom.

Antrojoje eksperimento dalyje išanalizavus tikslumo priklausomybę nuo triukšmo verčių galima teigti, kad Kalmano filtro atveju (taikant tiek pirmąjį, tiek antrąjį eksperimento modelius) pastebima

eksponentinė priklausomybė nuo triukšmo lygio, t. y. algoritmo gautos pozicijos tikslumas stipriai krenta didėjant triukšmui. Dalelių filtro atveju ši priklausomybė yra tiesinė, kas reiškia, kad tikslumas krenta žymiai lėčiau negu Kalmano filtro atveju. Be to, dalelių filtro atveju tikslumas, taikant modelį su švyturėliais, krenta žymiai lėčiau negu taikant modelį be švyturėlių – apskaičiuotas statumo koeficientas esant tiesinei aproksimacijai lygus 0,53 modelio be švyturėlių atveju ir 0,28 – modelio su švyturėliais atveju.

Naudojant nurodytą vertinimo kriterijų galima teigti, kad atliktoje imitacijoje dalelių filtro tikslumas yra žymiai didesnis: pirmojo eksperimento atveju tikimybė, kad paklaida (skirtumas) sudaro mažiau negu 1 % yra 0,71, ir antrojo eksperimento – 0,9. Kalmano filtro atveju – 0,07 ir 0,61 atitinkamai.

Rekomendacijos ir pasiūlymai

Apibendrinus eksperimento rezultatus, galima daryti išvadą, jog Kalmano filtrą dėl žymiai mažesnio vykdymo laiko geriau naudoti netiesinio modelio atvejį (tikslumas ~61 %, esant 20 dB signalo / triukšmo lygiui), kai nėra keliami labai dideli reikalavimai tikslumui (paklaida mažesnė ~10 %). O dalelių filtras visais imitacijos patikrintais būdais yra tikslesnis, bet jis sunaudoja žymiai daugiau skaičiavimo laiko, dėl ko šis algoritmas reikalauja žymiai didesnių skaičiavimo pajėgumų, be to, turi būti atsižvelgta į galimą algoritmo uždelsimo paklaidą.

Tolimesni tyrimai

Tolimesnių tyrimų metu gali būti kuriamas taikymo algoritmas (*SLAM*), kuris gali panaudoti skirtingų algoritmų tipus, esant realiu laiku išmatuotiems paklaidų ir greitaveikos parametrams, bei gali būti kuriamas atitinkamas įvairių platformų palaikomas programinis paketas.

LITERATŪRA

1. Bachrach, A.; *et al.* 2011. RANGE–Robust autonomous navigation in GPS-denied environment, *Journal of Field Robotics* 28(5): 644–666 [interaktyvus], [žiūrėta 2017-01-10]. Prieiga per internetą: <https://people.csail.mit.edu/prentice/papers/jfr11-mav.pdf>.
2. *Blösch, M.; et al.* 2010. Vision based MAV navigation in unknown and unstructured environments. In *Robotics and Automation (ICRA), 2010 IEEE International Conference* 1050-4729. [interaktyvus], [žiūrėta 2016-11-11]. Prieiga per internetą: <https://www.ifi.uzh.ch/dam/jcr:3d76fbb1-1d38-4b5f-8689-34b36952af8c/ICRA10_bloesch.pdf>.
3. Brockers, R.; *et al.*. 2014. Towards Autonomous Navigation of Miniature. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on* 23-28 June 2014. ISSN: 2160-7516. [interaktyvus], [žiūrėta 2016-05-24]. Prieiga per internetą: <https://pdfs.semanticscholar.org/fe79/90c6b0fa7bbb2c611652fad5a3a8a1935905.pdf>.
4. Bry, A.; Bachrach, A.; Roy, N. 2012. State Estimation for Aggressive Flight in GPS-Denied Environments Using Onboard Sensing. In *IEEE International Conference on Robotics and Automation* 2012. ISSN: 1050-4729. [interaktyvus], [žiūrėta 2016-10-26]. Prieiga per internetą: <https://pdfs.semanticscholar.org/95fc/4e78a08c2782c5988097025c5843aef8b618.pdf>
5. Chen, Z.. 2013. *Bayesian Filtering: From Kalman Filters to Particle Filters, and Beyond*. [interaktyvus], [žiūrėta 2016-09-28]. Prieiga per internet: <http://www.dsi.unifi.it/users/chisci/idfric/Nonlinear_filtering_Chen.pdf>
6. Chirakkal, V.; Park, M.; Han, D. S. 2015. Indoor navigation using WiFi for smartphones: An improved Kalman filter based approach. In *Consumer Electronics (ICCE), 2015 IEEE International Conference on* 9-12 Jan. 2015. ISSN: 2158-3994. [interaktyvus], [žiūrėta 2016-12-12]. Prieiga per internetą: <http://ieeexplore.ieee.org/document/7066328/>.
7. Chowdhary, G.; e*t al.*. 2013. GPS-denied Indoor and Outdoor Monocular Vision Aided Navigation and Control of Unmanned Aircraft, *Journal of Field Robotics*. 30(3):415–438 . [interaktyvus], [žiūrėta 2016-11-26]. Prieiga per internetą: <http://onlinelibrary.wiley.com/doi/10.1002/rob.21454/abstract>.
8. Feng, J.; Liu, Y. 2012. Wifi-based Indoor Navigation with Mobile GIS and Speech Recognition, *JCSI International Journal of Computer Science Issues* 9(6) No 2. ISSN: 1694-

0814 [interaktyvus], [žiūrėta 2016-06-06]. Prieiga per internetą: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.468.3685>.

9. Huang, S. 2010. *Understanding Extended Kalman Filter*. [interaktyvus], [žiūrėta 2016-10-26]. Prieiga per internetą: <http://services.eng.uts.edu.au/~sdhuang/Extended%20Kalman%20Filter_Shoudong.pdf>
10. Ito, A.; *et al.* 2016. A Service Model using Bluetooth Low Energy Beacons. In *The Eighth International Conferences on Advanced Service Computing*. [interaktyvus], [žiūrėta 2016-02-22]. Prieiga per internetą: <http://www.thinkmind.org/download.php?articleid=service_computation_2016_1_30_10022>
11. Layh, T.; *et al*. 2014. *GPS-Denied Navigator for Small UAVs*. [interaktyvus], [žiūrėta 2017-02-28]. Prieiga per internetą: <http://borders.arizona.edu/cms/sites/default/files/GPS-Denied%20Navigator%20for%20Small%20UAVs%20%28Compressed%29.pdf>
12. Lynen, S. 2013. A robust and modular multi-sensor fusion approach applied to MAV navigation. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on* 3-7 Nov. 2013. ISSN: 2153-0858. [interaktyvus], [žiūrėta 2016-12-11]. Prieiga per internetą: <http://www.margaritachli.com/papers/IROS2013fusion_paper.pdf>.
13. Miller, M. *et al.* 2010. Navigation in GPS Denied Environments: Feature-Aided Inertial Systems. [interaktyvus], [žiūrėta 2016-04-26]. Prieiga per internetą: <www.dtic.mil/get-tr-doc/pdf?AD=ADA581023>.
14. Pieh Wen, L.; *et al.* 2011. Application of WiFi-based Indoor Positioning System in Handheld Directory System. In *ECC'11 Proceedings of the 5th European conference on European computing conference*. 21-27. ISBN: 978-960-474-297-4. [interaktyvus], [žiūrėta 2017-01-11]. Prieiga per internetą: <http://www.wseas.us/e-library/conferences/2011/Paris/ECC/ECC-01.pdf>
15. Razeen Ridhwan, U.; *et al.* 2015. Non GPS navigation usingsimultaneous localization andmapping in UAV, *International Journal of Application or Innovation in Engineering & Management* 4(1) . ISSN 2319 – 4847. [interaktyvus], [žiūrėta 2016-12-06]. Prieiga per internetą: <http://www.ijaiem.org/Volume4Issue1/IJAIEM-2015-01-31-68.pdf>.
16. Schön, T. B.; Karlsson, R.; Gustaffson, F. 2006. The Marginalized Particle Filter in Practice. In *Aerospace Conference, IEEE*. ISSN: 1095-323X. [interaktyvus], [žiūrėta 2016-11-05]. Prieiga per internetą: <http://user.it.uu.se/~thosc112/pubpdf/schonkg2006.pdf>.

17. Turner, L. 2013. *An Introduction to Particle Filterin*. [interaktyvus], [žiūrėta 2016-11-06]. Prieiga per internetą: < http://www.lancaster.ac.uk/pg/turnerl/PartileFiltering.pdf>
18. Wu, A.; *et al.* 2013. Autonomous Flight in GPS-Denied Environments Using Monocular Vision and Inertial Sensors., *Journal of Aerospace Information Systems* 10(4): 172–186. EISSN: 2327-3097. [interaktyvus], [žiūrėta 2016-10-02]. Prieiga per internetą: <https://pdfs.semanticscholar.org/a7e9/5cb296f3a1178c05b024d9375c6f994d5364.pdf>.

PRIEDAI