



**KAUNO TECHNOLOGIJOS UNIVERSITETAS
ELEKTROS IR ELEKTRONIKOS FAKULTETAS**

Vytautas Senvaitis

**ADAPTYVAUS 2D POZICIONAVIMO METODO
AUTONOMINIAM ROBOTUI TYRIMAS**

Baigiamasis magistro projektas

Vadovas

Prof. dr. Vytautas Deksnys

KAUNAS, 2016

KAUNO TECHNOLOGIJOS UNIVERSITETAS
ELEKTROS IR ELEKTRONIKOS FAKULTETAS
ELEKTRONIKOS INŽINERIJOS KATEDRA

ADAPTYVAUS 2D POZICIONAVIMO METODO
AUTONOMINIAM ROBOTUI TYRIMAS

Baigiamasis magistro projektas
Įterptinės sistemos (621H61004)

Vadovas

(parašas) Prof. dr. Vytautas Deksnys
(data)

Recenzentas

(parašas) Doc. dr. Ramūnas Ramanauskas
(data)

Projektą atliko

(parašas) Vytautas Senvaitis
(data)

KAUNAS, 2016



KAUNO TECHNOLOGIJOS UNIVERSITETAS

Elektros ir elektronikos fakultetas

(Fakultetas)

Vytautas Senvaitis

(Studento vardas, pavardė)

Įterptinės sistemos (621H61004)

(Studijų programos pavadinimas, kodas)

Adaptyvaus 2d pozicionavimo metodo autonominiam robotui tyrimas

AKADEMINIO SAŽININGUMO DEKLARACIJA

20 16 m. birželio 3 d.
Kaunas

Patvirtinu, kad mano **Vytauto Senvaičio** baigiamasis projektas tema „Adaptyvaus 2d pozicionavimo metodo autonominiam robotui tyrimas“ yra parašytas visiškai savarankiškai, o visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka)

(parašas)

Senvaitis, Vytautas. Adaptyvaus 2d pozicionavimo metodo autonominiam robotui tyrimas. *Magistro laipsnio* baigiamasis projektas / vadovas Prof. dr. Vytautas Deksnys; Kauno technologijos universitetas, Elektros ir elektronikos fakultetas, Elektronikos inžinerijos katedra. Kaunas, 2016. – 69 p.

SANTRAUKA

Analitinėje dalyje apžvelgiamas SLAM algoritmas, lazerinio atstumo skenerio veikimas, EKF ir UKF filtrai. Pritaikomi EKF matematiniai modeliai autonominiam robotui su dviem varomaisiais ratais, bei lazerinio atstumo skenerio lygtys EKF filtrui. Palyginami EKF ir UKF filtrai. Modeliavimo dalyje sumodeliuojamas EKF filtras 2D roboto pozicionavimui MATLAB programa bei sumodeliuojamas EKF algoritmas STM32 mikrovaldikliu naudojantis DSP biblioteką, palyginama greitisveika STM32 ir MATLAB. Analizuojamas EKF filtro veikimas. Projektinėje dalyje suprojektuojama ir sukonstruojama eksperimentinė įranga: autonominis robotas, ir „Lidar“ testavimo įranga. Robotui integruojamas STM32F407 mikrovaldiklis, „Lidar“ jutiklis, žingsniniai varikliai. Sukuriama programinė įranga STM32F407 mikrovaldikliui, naudojantis DMA, „TIMER“, DSP bibliotekas. Eksperimentinėje dalyje tiriamas roboto pozicionavimo tikslumas su EKF ir be EKF filtro, rezultatai išanalizuojami.

Reikšminiai žodžiai: Autonominis robotas, EKF filtras, STM32 mikrovaldiklis, 2D roboto pozicionavimas

Senvaitis, Vytautas. Research of adaptive 2D positioning method for autonomous robot. Final project of *master degree*/ supervisor Prof. dr. Vytautas Deksnys; Kaunas University of Technology, Faculty of Electrical and Electronics Engineering, department of Electronic engineering. Kaunas, 2016. – 69 p.

SUMMARY

Overview SLAM algorithm, laser distance scanner working principle, EKF and UKF filters in analytical part. EKF mathematical models are implemented for autonomous robot with two-wheel drive and for laser distance scanner. EKF and UKF filters are compared. 2D robot positioning with EKF filter are modeled and simulated in MATLAB and STM32 microcontroller with DSP library. MATLAB and STM32 are compared in speed test. Analyzing EKF filter working. Design and construct autonomous robot experimental equipment, Lidar testing equipment. Robot is integrated with STM32F407 microcontroller, Lidar sensor, stepper motors. Software is created for STM32F407 microcontroller using DMA, TIMER, DSP libraries. Robot positioning accuracy with EKF and without EKF are analyzed in experimental part.

Keywords: Autonomous robot, EKF filter, STM32 microcontroller, 2D robot positioning

TURINYS

SANTRUMPŲ IR ŽENKLŲ AIŠKINIMO ŽODYNAS	9
ĮVADAS	10
1. ANALITINĖS LITERATŪROS APŽVALGA.....	11
1.1 SLAM procesas	11
1.1.1 Žymės	12
1.2 Lazerinis atstumu skeneris (LiDar)	13
1.3 EKF SLAM	17
1.3.1 Mobilaus roboto modelis.....	17
1.3.2 LiDar jutiklio modelis	19
1.3.3 EKF pritaikymas	21
1.4 UKF SLAM.....	21
1.5 UKF ir EKF palyginimas	24
1.6 Apibendrinimas	26
2. KALMAN FILTRO ANALITINIS MODELIAVIMAS MATLAB	27
2.1 Kalman filtras eksperimentas su stm32f429	28
2.2 Apibendrinimas	30
3. EKSPERIMENTINĖS ĮRANGOS KONSTRAVIMAS	31
3.1 Roboto geometrija.....	34
3.2 Roboto pavaros.....	35
3.3 Neato xv-11 lazerinis atstumo skeneris.....	36
3.4 XV-11 jutiklio versijos nustatymas ir parametrai	36
3.5 XV-11 jutiklio duomenų formatas	37
3.6 Apibendrinimas	39
4. PROGRAMINĖS ĮRANGOS SUKŪRIMAS IR INTEGRAVIMAS	40
4.1 Žingsninių variklių valdymas.....	40
4.2 Koordinačių į poslinkį skaičiavimas	41
4.3 Lidar duomenų nuskaitymas	42
4.4 Apibendrinimas	43
5. EKSPERIMENTINIAI TYRIMAI	44
5.1 Pavaros pozicionavimo tikslumo tyrimas	44
APIBENDRINIMAS IR IŠVADOS	47
LITERATŪRA	48
PRIEDAI.....	50
1 PRIEDAS. STM32F407 PROGRAMOS KODAS.....	50
1.1 Kalman.c programos kodas	50
1.2 Lidar.c programos kodas	53
1.3 Zingsniai_varikliai.c programos kodas	57
2 PRIEDAS. MATLAB KODAS	65
3 PRIEDAS. KTU „TECHNORAMA 2016“ SERTIFIKATAS.....	68
4 PRIEDAS. „ROBOTŲ INTELEKTAS 2016“ SERTIFIKATAS	69

PAVEIKSLAI

1 pav. SLAM schema [1].....	11
2 pav. SLAM sudaryti žemėlapiai [2].....	12
3 pav. RANSAC algoritmu išskirta linija [3].....	13
4 pav. Hough transformacija.....	13
5 pav. TOF matavimo metodo blokinė diagrama [4].....	14
6 pav. „Triangulation“ matavimo principas.....	14
7 pav. Atstumo skaičiavimas	15
8 pav. Hokuyo UTM-30LX lazerinis atstumo skeneris ir su juo atliktas matavimas[5]	16
9 pav. Roboto modelio schema.....	17
10 pav. Roboto modelis su omni ratų važiuokle	19
11 pav. LiDar matavimo modelis.....	20
12 pav. Funkcijų tiesinimas UKF (b), EKF (c), tikras (a) [13]	22
13 pav. Žemėlapiai sukurti pagal EKF algoritmą [15]	24
14 pav. Žemėlapiai sukurti pagal UKF algoritmą [15].....	24
15 pav. RMSE pozicijos klaidos priklausomybė nuo atliktų skaičiavimų skaičiaus [15].....	25
16 pav. Spėtos pozicijos klaidos (m) priklausomybė nuo kontūro dydžio (m)[15]	25
17 pav. EKF filtras su didėjančia spėjimo paklaida(a) ir su pastovia spėjimo paklaida(b).....	27
18 pav. EKF filtras su pastovia spėjimo paklaida, ir 20% matavimo triukšmų	28
19 pav. MCU greičio matavimas su Kalman filtru.....	28
20 pav. Skaičiavimo greitaveikos palyginimas	29
21 pav. RC tanko modelis originalus (a) ir modifikuotas(b).....	31
24 pav. Roboto elektronikos sujungimo schema	33
25 pav. Roboto PCB	34
26 pav. Roboto geometrija.....	34
27 pav. Roboto judėjimas apskritimu	35
29 pav. XV-11 LiDar testavimo įranga	37
30 pav. CVI su LiDar sujungimo schema.....	38
31 pav. LiDar nuskenotas kambario vaizdas	38
32 pav. Roboto valdymo algoritmas	40
33 pav. Užduotų X Y koordinačių keitimas į poslinkį algoritmas	41
34 pav. Žingsninių periodo suskaičiavimo algoritmas	42
35 pav. Lidar duomenų apdorojimo algoritmo supaprastinta schema	42
36 pav. Suskaitmenizuota nuvažiuota roboto kreivė	44

37 pav. 200mm kvadratas valdant tik poslinkiu(a) ir su Kalman filtru(b).....	45
38 pav. 100mm kvadratas valdant tik poslinkiu(a) ir su Kalman filtru(b).....	46
39 pav. Pozicionavimo paklaidos palyginimas	46

LENTELĖS

1 lentelė. Hokuyo UTM-30LX parametrai[5]	16
2 lentelė. Roboto sudedamosios dalys	32
3 lentelė. Važiavimas tiese.....	44
4 lentelė. Sukimo paklaida.....	45

SANTRUMPŲ IR ŽENKLŲ AIŠKINIMO ŽODYNAS

- SLAM – (angl. *Simultaneous localization and mapping*) nuolatinis pozicijos ir žemėlapių sudarymas
- LiDar – Lazerinis atstumo skeneris
- RANSAC – (angl. *Random Sampling Consensus*) atsitiktines imties konsensusas
- TOF – (angl. *Time of flight*) skrydžio laikas
- CMOS – (angl. *Complementary metal–oxide–semiconductor*)
- EKF – (angl. *Extended Kalman filter*) išplėstinis Kalman filtras
- UKF – (angl. *Unscented Kalman filter*)
- UT – (angl. *Unscented transform*)
- DSP – (angl. *digital signal processing*) skaitmeninis signalų apdorojimas
- RC – (angl. *radio-controlled model*) radijo bangomis valdomas modelis
- DC – (angl. *Direct current*) nuolatinė srovė
- GPS – (angl. *Global Positioning System*) – Globali padėties nustatymo sistema
- OFS – (angl. *optical flow sensor*) optinis poslinkio jutiklis
- UART – (angl. *Universal Asynchronous Receiver – Transmitter*) Universalus Asinchroninis Imtuvas-Siųstuvas
- SPI – (angl. *Serial Peripheral Interface*) Nuosekli duomenų sąsaja
- SRAM – (angl. *Static random-access memory*) statinė operatyvioji atmintis
- API – (angl. *application programming interface*)
- PCB – (angl. *Printed circuit board*) spausdinta schema
- PWM – (angl. *Pulse Width Modulation*) Impulso pločio moduliacija
- DMA – (angl. *Direct memory access*) Tiesioginė kreiptis į atmintį
- ŽEV – Žingsninis elektros variklis
- MCU – (angl. *Microcontroller unit*) Mikrovaldiklis
- ARM – (angl. *Advanced RISC Machine*) Mikrovaldiklio architektūra

ĮVADAS

Technologijos tapo neatsiejama mūsų gyvenimo dalis. Žmogus nuolat kuria technologijas, kurios palengvina gyvenimo ir aplinkos sąlygas, efektyviau išnaudoja gamtos resursus. Robotika yra viena iš technologinių sričių, kuri plačiai taikoma įvairiose srityse, t. y. gamybos sektoriuje, logistikoje, automobilių pramonėje, medicinoje, agronomijoje ir t.t. Robotikos kryptis autonominiai robotai yra viena iš aktualiausių bei perspektyviausių. Aktuali autonominių robotų tyrinėjimo sritis yra autonominiai automobiliai, galintys važiuoti be žmogaus pagalbos. Tyrinėjimo sritis yra labai kompleksiška ir kol kas nėra iki galo išnagrinėta. Vienas iš pagrindinių klausimų yra tikslus pozicijos nustatymas nežinomojo aplinkoje. Robotai dar daro įvairiais klaidas, kurios lieka dėl netikslaus pozicijos nustatymo. Tai jutiklių paklaidos, įvairūs aplinkos veiksniai (saulės spinduliavimas, lietus, sniegas), tai pat roboto dinamika, atsitiktinės valdymo klaidos ir t.t. Tyrimo tikslas iširti būdą autonominio roboto adaptyviam pozicionavimui aplinkoje.

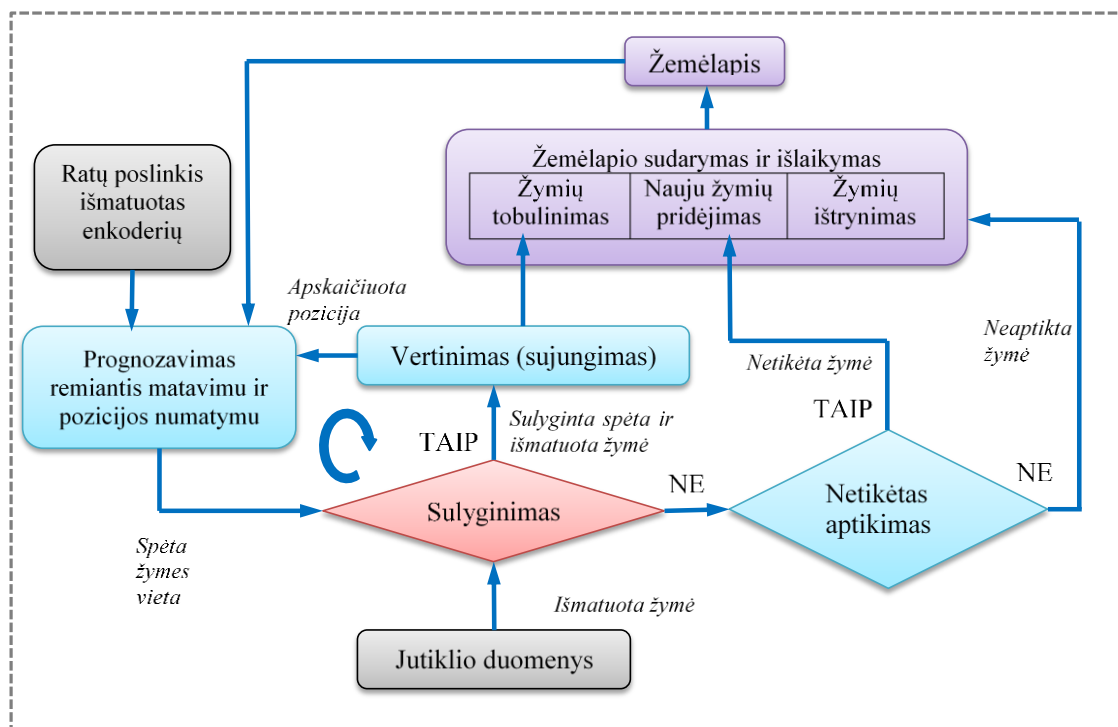
Darbe analizuojami pozicijos nustatymo būdas su 2D lazeriniu skeneriu ir roboto judesio kinematika, konstruojama eksperimentinė įranga, sudaromas roboto valdymo algoritmas ir programa.

1. ANALITINĖS LITERATŪROS APŽVALGA

Peržvelgus straipsnius ir patentus, susijusius su roboto pozicijos nustatymu, plačiausiai yra taikomas Kalman filtras ir jo modifikacijos. Roboto pozicija nustatoma naudojantis greitaveikiu lazeriniu atstumo matavimo skeneriu, nuolat atnaujinant nežinomos aplinkos žemėlapi (SLAM angl. *Simultaneous localization and mapping*). Jį sudarant atsiranda atsitiktinių ir sistematiųjų paklaidų ir tenka naudoti metodus, patikslinančius iš SLAM gaunamus duomenis. Populiariausi yra „*Extended Kalman filter SLAM*“, „*Unscented Kalman Filter SLAM*“ Šie metodai ir bus apžvelgiami analitinėje dalyje.

1.1 SLAM procesas

SLAM procesas apima keletą žingsnių. Pagrindinis tikslas yra iš aplinkos ir roboto kinematikos (iš roboto ratų greičių, geometrijos ir laiko skirtumų suskaičiuojamas roboto poslinkis, o iš jo - globalios koordinatės) nustatyti roboto poziciją. Kadangi iš roboto kinematikos modelio gautos koordinatės nėra labai tikslios, todėl negalime vien jomis pasikliauti. Patikslinimui tenka naudoti jutiklius, kurie stebi aplinką. Tam tinka lazerinis atstumo matavimo skeneris (LiDar). Jis leidžia aplinkoje išskirti požymius ir surasti kelią, kuriuo robotas turi judėti. Filto panaudojimas yra labai svarbus SLAM procese, jis apjungia roboto kinematikos modelį, jutiklius, įvertina pastarųjų paklaidas ir patikslina judesio trajektoriją.



1 pav. SLAM schema [1]

Schemoje (1 pav.) pavaizduotas pozicionavimo algoritmas. Atsiradus paklaidoms bei palyginus išmatuotą ir prognozuotą padėtį, algoritmas pereina prie naujos padėties arba jei tai

buvo numatyta padėtis, bet su paklaida, algoritmas žymė ištrina. Matavimas atliekamas su lazerinių atstumo jutiklių, kurio išmatuotos reikšmės sulyginamos su žemėlapiu sudarytomis žymėmis bei išmatuotų roboto poslinkių enkoderiais pasitelkiant EKF filtrą.

2 pav. pateikti sudarytų SLAM žemėlapių pavyzdžiai, naudojantis LiDar ir FastSLAM filtru. Violetine spalva pažymėta neaptiktos žemėlapiu vietos arba nesama erdvė. Balta spalva pažymėta roboto atidengta erdvė. Juoda linija: sienos ar aptiktos kliūtis.



2 pav. SLAM sudaryti žemėlapiai [2]

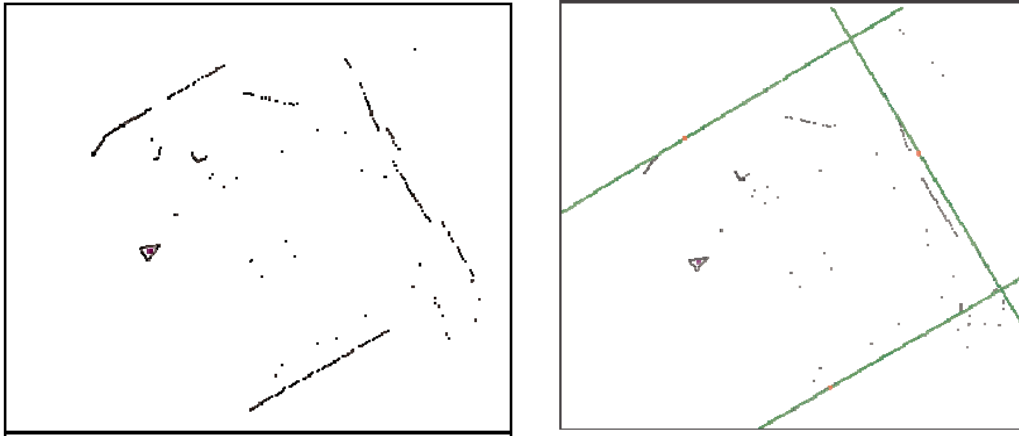
1.1.1 Žymės

Roboto pozicijos nustatymui svarbios yra žymės. Jų darbo pradžioje robotas neturi nei vienos, nes yra nežinomoje aplinkoje. Žymės turi būti aptinkamos skirtingose pozicijose skirtingais kampais. Jos turėtų būti unikalios ir lengvai identifikuojamos. Jeigu žymė nebus teisingai identifikuota, galima jas sumaišyti, o tai paveiktų roboto pozicijos nustatymo tikslumą. Žemėlapyje reikia turėti kuo didesnę žymių kiekį, bet jos turėtų būti stacionarios.

Žymėmis gali būti:

Taškas, kuris yra sudaromas iš unikalios taško. Unikali taško žymė gaunama naudojantis keletą taškų, nes vieną žemėlapiu tašką galima lengvai sumaišyti su kitais taškais žemėlapyje. Taigi sukuriant taško žymę yra geriau remtis 2 ar daugiau pavienių gretimų taškų derinių kurie turėtų individualų atstumą tarp savų taškų, tokiu atveju kiti taškų deriniai nebus sumaišyti su sudaryta žyme iš taškų.

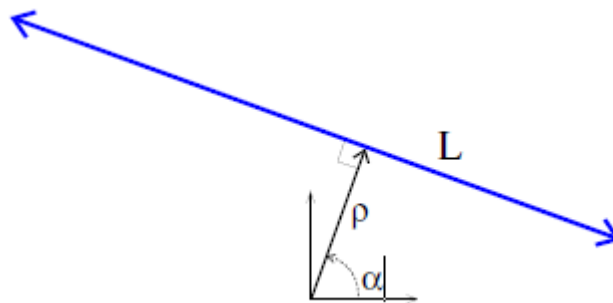
Linija gali būti išskiriama iš matavimo duomenų, naudojantis tokiais algoritmais kaip RANSAC (angl. *Random Sampling Consensus*). Uždaroje aplinkoje linijos gali būti aptinkamos dažnai, nes čia yra sienos. RANSAC metodas suranda linijas atsitiktinai paimdamas lazerio duomenis naudojantis aproksimavimu (3 pav.).



3 pav. RANSAC algoritmu išskirta linija [3]

Kitas linijų išskyrimo būdas yra *Hough* transformacija (4 pav.). Čia linija yra nustatoma Dekarto koordinatinių sistemoje pasitelkus parametrus (ρ, α) , kur:

$$\rho = x \cos(\alpha) + y \sin(\alpha) \quad (1)$$



4 pav. Hough transformacija

Čia, ρ – atstumas iki linijos [mm]

α – spindulio kampas pagal X ašį [rad]

L – linija

1.2 Lazerinis atstumu skeneris (LiDar)

Lazerinis atstumo matavimo skeneris yra bekontaktis. Jį naudojant atstumas gali būti nustatytas dviem būdais, t. y. skaičiuojant vėlinimo laiką tarp išsiūsto ir grįžusio lazerio spindulio (*TOF- angl. time of flight*). Antras būdas - atsispindėjusio lazerio kampo pokytis priklausomai nuo objekto padėties.

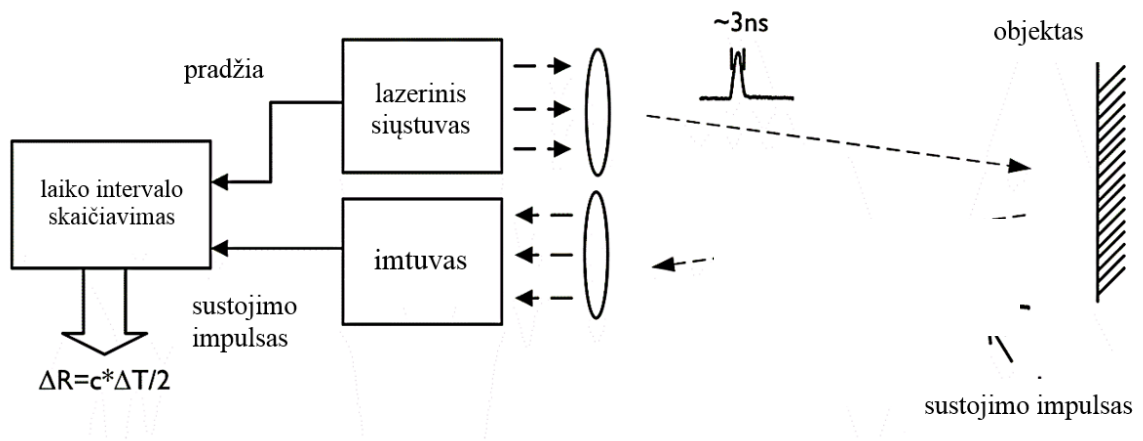
Kadangi šviesos greitis yra žinomas, tai atstumas gali būti apskaičiuotas iš formulės:

$$R = c \cdot \frac{\Delta T}{2} \quad (2)$$

, čia c - šviesos greitis $3 \cdot 10^8$ m/s

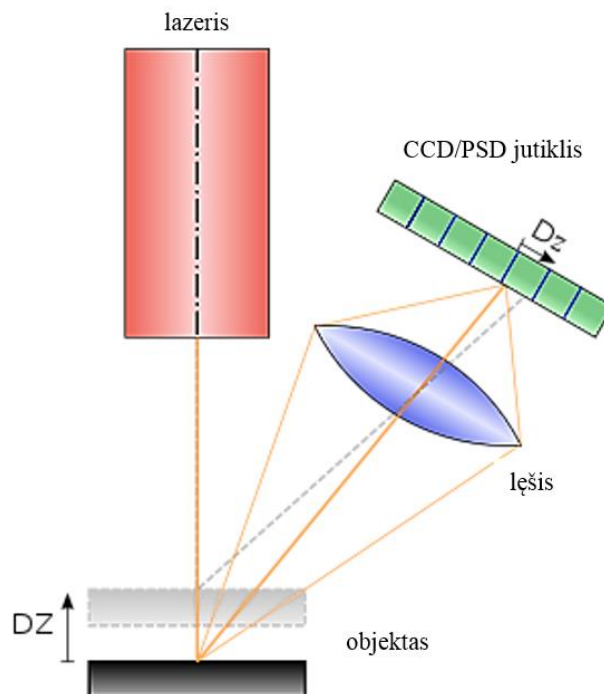
ΔT -laiko intervalas tarp išsiūsto ir grįžusio lazerinio šviesos impulso [s]

R - atstumas iki objekto [m]



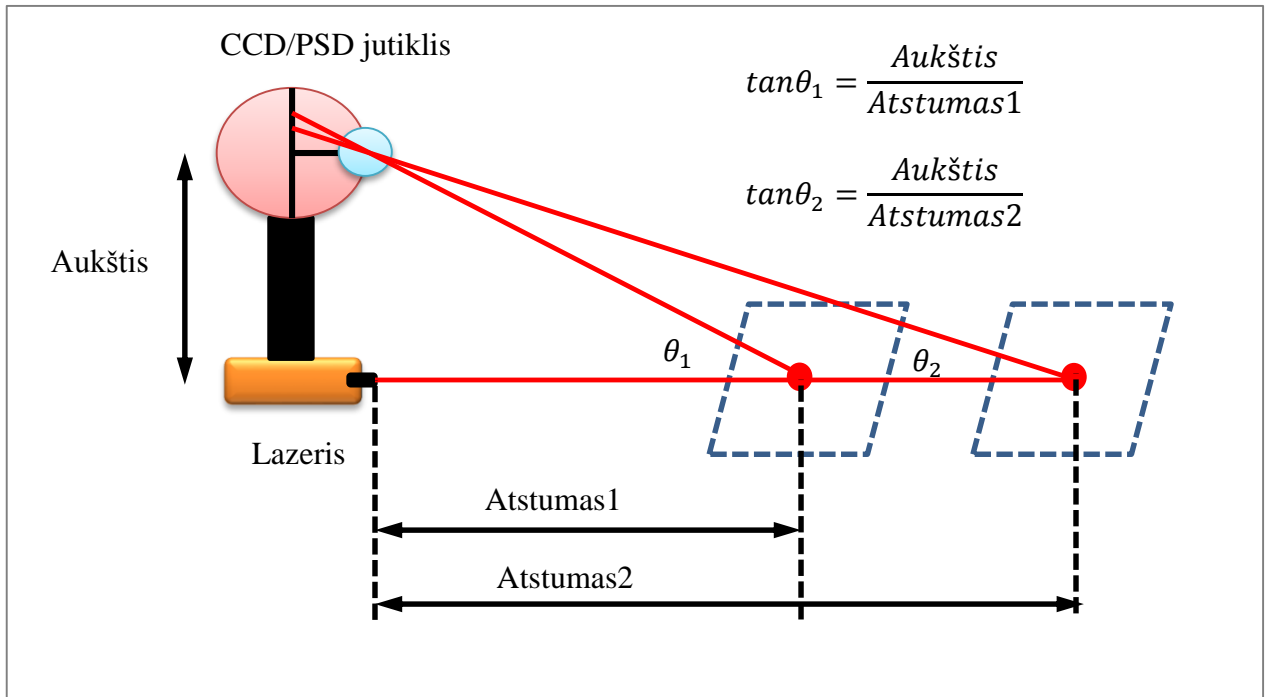
5 pav. TOF matavimo metodo blokinė diagrama [4]

5 pav. yra pavaizduota TOF matavimo (5 pav.), schema. Čia lazerinis siūstuvus išsiunčia 3-4 ns trukmės šviesos impulsą. Imtuvas priima atsispindėjusį signalą su CMOS jutikliu. Metodas užtikrina platų matavimo diapazoną nuo kelių milimetrų iki kelių tūkstančių metrų [4]. Kitas metodas yra vadinamas trianguliacija (angl. *triangulation*) čia kamera ir lazeris tarpusavyje sudaro statų trikampį.



6 pav. „Triangulation“ matavimo principas

Kadangi tarp lazerio ir jutiklio atstumas yra žinomas, o kampas išmatuojamas, galima apskaičiuoti atstumą iki objekto. Iš paveikslėlio matome, kad taško padėtis optiniame jutiklyje keičiasi priklausomai nuo objekto padėties (6 pav.).



7 pav. Atstumo skaičiavimas

Atstumą galime apskaičiuoti pagal tokia formulę(7 pav.):

$$\text{Atstumas} = \frac{\text{Aukštis}}{\tan\theta} \quad (3)$$

,čia *Atstumas*- atstumas iki objekto [m].

Aukštis- atstumas tarp jutiklio ir lazerio [m]

θ - kampas projektuojamas jutiklyje [rad]

Palyginus abu metodus galima pasakyti, kad (TOF) metodas yra greitas, galimas didelis matavimo atstumas, bet galimas tikslumo praradimas dėl didelio šviesos greičio, matavimo neapibrėžtis būna keletas milimetrų. „Triangulation“ metodas yra labai tikslus galima matavimo neapibrėžtis kelių mikrometrų, bet ribotas matavimo atstumas (iki keleto šimtų metrų).

Lazerinių skenerių gamyboje populiariausias ir plačiausių asortimentą turi kompanija Hokuyo. Hokuyo UTM-30LX (8 pav.) skenerio parametrai pateikti 1 lentelėje:

1 lentelė. Hokuyo UTM-30LX parametrai[5]

Produkto pavadinimas:	Lazerinis atstumo skeneris
Modelis	UTM-30LX
šviesos šaltinis	Lazeris $\lambda = 870\text{nm}$, Pirma lazerio klase
Maitinimo šaltinis	12VDC $\pm 10\%$
Maitinimo srovė	Max: 1A, Normal : 0.7A
Naudojama galia	mažiau negu 8W
apskaičiuojamas atstumas ir nustatomas objektas	Patikrintas atstumas: 0.1 ~ 30m (baltas popieriaus lapas) Maksimalus atstumas : 0.1 ~ 60m minimalus plotis iki 10m : 130mm (Keičiasi su atstumu)
Tikslumas	iki 3000lx : baltas popieriaus lapas: $\pm 30\text{mm}^{*1}$ (0.1m to 10m) iki 100000lx : baltas popieriaus lapas : $\pm 50\text{mm}^{*1}$ (0.1m to 10m)
Matavimo rezoliucija ir Atkartojamo matavimo tikslumas	1mm iki 3000lx : $\sigma = 10\text{mm}^{*1}$ iki 100000lx : $\sigma = 30\text{mm}^{*1}$
Skenavimo kampas	270°
Kampo rezoliucija	0.25° (360°/1440)
Skenavimo greitis	25ms (variklio sukimosi greitis : 2400rpm)
Šąsaja	USB Ver2.0 Full Speed (12Mbps)
Išėjimas	<i>Synchronous Output I- Point</i>
Aplinkos reikalavimai (temperatūra, drėgnumas)	-10°C ~ +50°C Mažiau negu 85%
Saugojimo temperatūra	-25~75°C
Aplinkos poveikis	Matavimo atstumas mažesnis esant snigimui, lietai, ar tiesioginiams saulės spinduliams
Vibravimo atsparumas	10 ~ 55Hz amplitudę 1.5mm į X, Y, Z ašis 2 val. 55 ~ 200Hz 98m/s ² į X, Y, Z ašis 1 val.
Smūgio atsparumas	196m/s ² į X, Y, Z ašis 10 kartu.
Apsauga	Optics: IP64
Izoliavimo varža	10MΩ DC500V Megger
Svoris	210g
Korpusas	Polycarbonate
Išoriniai matmenys (W×D×H)	60mm×60mm×85mm MC-40-3127

Matome, kad lazerinis atstumo skeneris praranda tikslumą didėjant apšvietimui, kai sninga, lyja ar veikia tiesioginis saulės spinduliavimas.



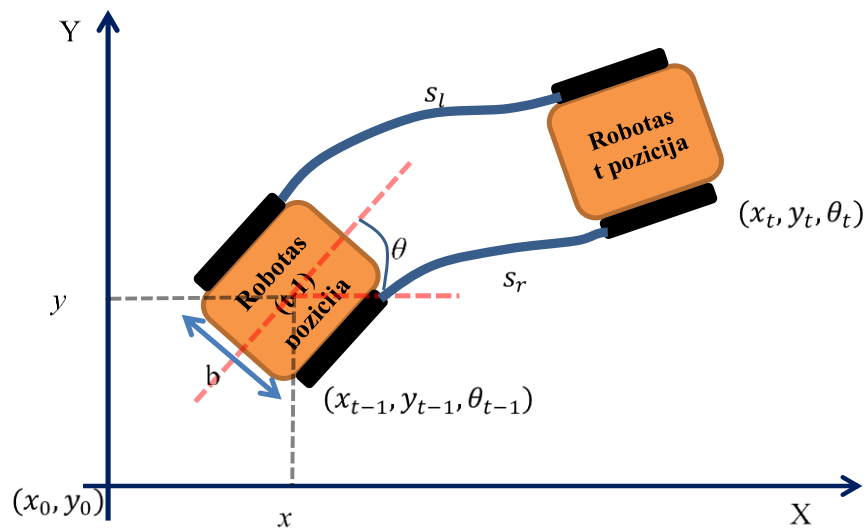
8 pav. Hokuyo UTM-30LX lazerinis atstumo skeneris ir su juo atliktas matavimas[5]

1.3 EKF SLAM

1.3.1 Mobilais roboto modelis

Mobilais roboto modelis aprašomas kinematinėmis (greičiais ir poslinkiais) lygtimis. Iš poslinkio ar greičio apskaičiuojama globali roboto pozicijos koordinatė kintanti laike.. Roboto pozicija gali būti aprašyta trimis parametrais (9 pav.) (x, y, θ) , kur x, y - Dekarto koordinatės (roboto pozicija pagal globalias koordinates), θ – roboto pakrypimo kampas nuo globalios x ašies.

Roboto su dviem varomaisiais ratais kinematinis modelis yra toks:



9 pav. Roboto modelio schema

Roboto valdymo komanda (transponuota matrica):

$$u_t = [s_l \ s_r]^T \quad (4)$$

, čia s_l – roboto kairio rato poslinkis [m];

s_r – roboto dešinio rato poslinkis [m].

Tada koordinatės t laiko momentu gali būti randamos taip:

$$x'_t = f(x_{t-1}, u_t) = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \Delta s_t \cdot \cos(\theta_{t-1} + \Delta\theta_t/2) \\ \Delta s_t \cdot \sin(\theta_{t-1} + \Delta\theta_t/2) \\ \Delta\theta_t \end{bmatrix} \quad (5)$$

, čia

$$\Delta s_t = (s_r + s_l)/2 \quad (6)$$

$$\Delta\theta_t = (s_r - s_l)/b \quad (7)$$

Čia b – atstumas tarp ratų [m](9 pav.);

s_l – roboto kairiojo rato poslinkis [m];

s_r – roboto dešiniojo rato poslinkis [m].

Kadangi funkcija yra netiesinė, dėl to mes ją ištiesiname su pirmos eilės aproksimacija.

$$x'_t = f(x_{t-1}, u_t) \approx F_{x,t}x_{t-1} + F_{u,t}u_t \quad (8)$$

,čia

$F_{x,t}$ - f funkcijos išvestinė pagal būseną (x, y, θ) ;

$F_{u,t}$ - f funkcijos išvestinė pagal valdymą (s_l, s_r) .

Pritaikius *Jacobian* [6] transformaciją sistemos būseną laike kis taip:

$$F_{x,t} = \begin{bmatrix} \frac{\partial fx}{\partial x} & \frac{\partial fx}{\partial y} & \frac{\partial fx}{\partial \theta} \\ \frac{\partial fy}{\partial x} & \frac{\partial fy}{\partial y} & \frac{\partial fy}{\partial \theta} \\ \frac{\partial f\theta}{\partial x} & \frac{\partial f\theta}{\partial y} & \frac{\partial f\theta}{\partial \theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta s_t \cdot \sin(\theta_{t-1} + \Delta\theta_t/2) \\ 0 & 1 & \Delta s_t \cdot \cos(\theta_{t-1} + \Delta\theta_t/2) \\ 0 & 0 & 1 \end{bmatrix} \quad (9)$$

$$F_{u,t} = \begin{bmatrix} \frac{\partial fx}{\partial s_r} & \frac{\partial fx}{\partial s_l} \\ \frac{\partial fy}{\partial s_r} & \frac{\partial fy}{\partial s_l} \\ \frac{\partial f\theta}{\partial s_r} & \frac{\partial f\theta}{\partial s_l} \end{bmatrix} = \begin{bmatrix} 0,5\cos(\theta_{t-1} + \Delta\theta_t/2) & 0,5\sin(\theta_{t-1} + \Delta\theta_t/2) \\ 0,5\cos(\theta_{t-1} + \Delta\theta_t/2) & 0,5\sin(\theta_{t-1} + \Delta\theta_t/2) \\ \frac{1}{b} & \frac{1}{b} \end{bmatrix} \quad (10)$$

Valdymo klaidos koreliacijos matrica, koreliuoja su $F_{u,t}$, sistemos būsenos matrica. Kuri naudojama EKF filtre skaičiuojant klaidos kovariacijos matricą P.

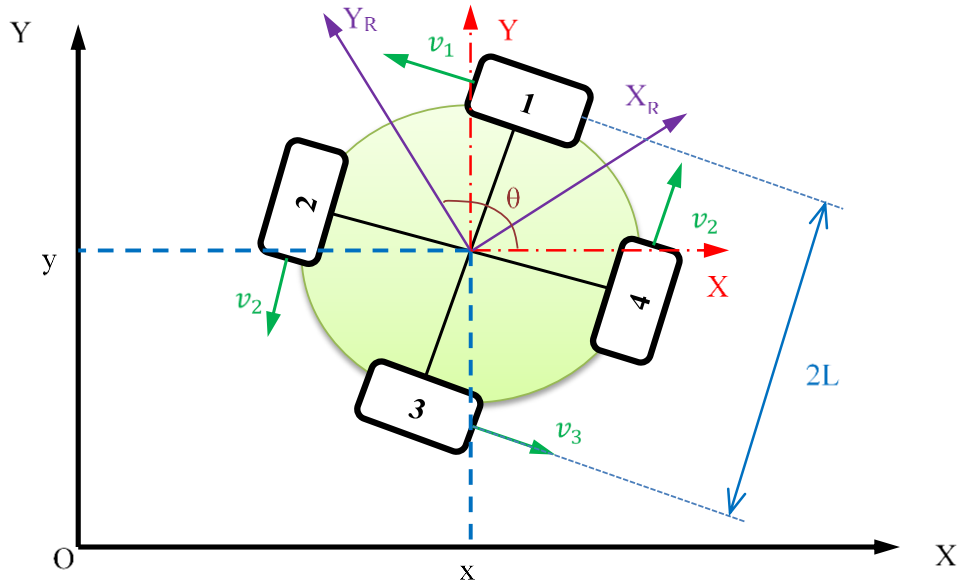
$$Q = \begin{bmatrix} k \cdot |s_r| & 0 \\ 0 & k \cdot |s_l| \end{bmatrix} \quad (11)$$

Klaidos kovariacijos matrica su sistemos būseną.

$$P'_t = F_{x,t} \cdot P_{t-1} \cdot F_{x,t}^T + F_{u,t} \cdot Q_t \cdot F_{u,t}^T \quad (12)$$

Valdymo klaidos matrica sudaroma remiantis poslinkio paklaida, kuri šiuo atveju yra $k \cdot |s_r|$. Klaidos koreliacijos matricoje P'_t yra įvertinama paklaida, atsirandanti sistemos būsenoje, kuri skaičiuojama remiantis ankstesne sistemos klaidos matrica P_{t-1} ir poslinkio paklaida Q.

Roboto su keturių ratų *omni* važiuokle modelis pateiktas 10 pav.



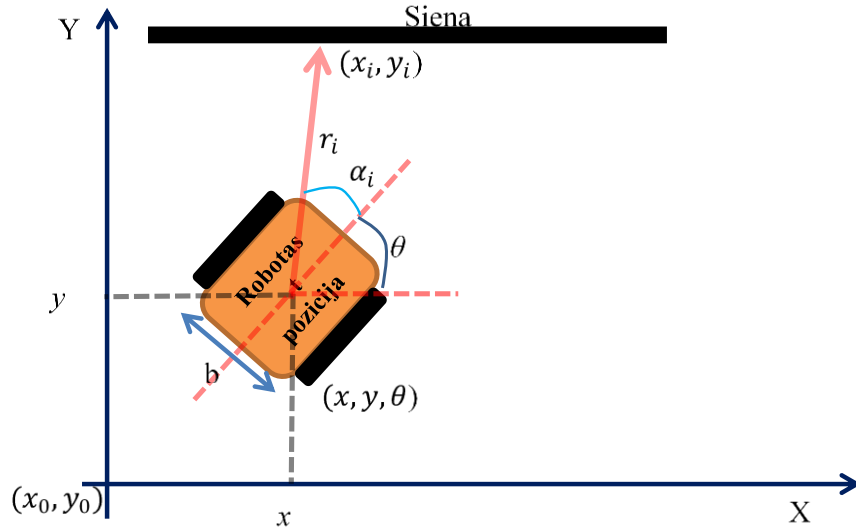
10 pav. Roboto modelis su omni ratų važiuokle

Roboto pozicija aprašoma trimis parametrais (10 pav.) (x, y, θ) . Laikomasi nuostatos, kad roboto ratų greičiai v_1 - v_4 , kada robotas keičia poziciją iš laiko momento $t-1$ į t . Roboto globalios koordinatės apskaičiuojamos formule[4]:

$$x'_t = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} 0,5 \cdot \left[-\sin\left(-\frac{\pi}{4} + \theta\right) v_1 - \sin\left(\frac{3\pi}{4} - \theta\right) v_2 + \sin\left(-\frac{\pi}{4} + \theta\right) v_3 + \sin\left(\frac{3\pi}{4} - \theta\right) v_4 \right] \Delta t \\ 0,5 \cdot \left[\cos\left(-\frac{\pi}{4} + \theta\right) v_1 - \cos\left(\frac{3\pi}{4} - \theta\right) v_2 - \cos\left(-\frac{\pi}{4} + \theta\right) v_3 + \cos\left(\frac{3\pi}{4} - \theta\right) v_4 \right] \Delta t \\ \frac{1}{4L} (v_1 + v_2 + v_3 + v_4) \Delta t \end{bmatrix} \quad (13)$$

1.3.2 LiDar jutiklio modelis

Globalią roboto poziciją apskaičiuoti galima iš atstumo nuo žymės, kuris yra išmatuojamas su jutikliu. Populiariausi jutikliai yra lazerinis atstumo skeneris (LiDar), ultragarsinis atstumo matuoklis, video kamera, Kinect. Kinect yra žaidimų konsolės Xbox jutiklis kuris gali sekti objektų judesį ir padėti. Geriausiai tinkantis globaliai koordinatei nustatyti yra LiDar, nes jis išmatuoja atstumą pagal kryptį. Kinect metodas tai pat matuoja atstumą ir krypties kampą (r_i ir α_i) (11 pav.)



11 pav. LiDar matavimo modelis

Roboto pozicijos ryšys su matavimo rezultatais [7, 8]:

$$h(x) = \begin{bmatrix} r_i \\ \alpha_i \end{bmatrix} = \begin{bmatrix} \sqrt{(x - x_i)^2 + (y - y_i)^2} + \omega_r \\ \tan^{-1} \frac{y - y_i}{x - x_i} - \theta + \omega_\alpha \end{bmatrix} \quad (14)$$

, čia ω_r, ω_α - matavimo triukšmai

Padėties prognozė pagal žymės [7]:

$$\hat{z}_t = \begin{bmatrix} \sqrt{(x - x_i)^2 + (y - y_i)^2} \\ \tan^{-1} \frac{y - y_i}{x - x_i} - \theta \end{bmatrix} \quad (15)$$

Paklaida tarp išmatuoto ir nuspėto matavimo[7]:

$$v_t = z_t - \hat{z}_t \quad (16)$$

Koreliacija tarp Lidar sistemos modelio H funkcijos ir klaidos kovariacijos matricos P sudarytos kinematinio modelio :

$$S_t = H_{x,t} \cdot P'_t \cdot H_{x,t}^T + R_t \quad (17)$$

,čia $H_{x,t}$ - h funkcijos išvestinė pagal roboto būseną (x, y, θ)

$$H_{x,t} = \begin{bmatrix} \frac{\partial hr}{\partial x} & \frac{\partial hr}{\partial y} & \frac{\partial hr}{\partial \theta} \\ \frac{\partial h\alpha}{\partial x} & \frac{\partial h\alpha}{\partial y} & \frac{\partial h\alpha}{\partial \theta} \end{bmatrix} = \begin{bmatrix} \frac{2x - 2x_i}{2\sqrt{(x - x_i)^2 + (y - y_i)^2}} & \frac{2y - 2y_i}{2\sqrt{(x - x_i)^2 + (y - y_i)^2}} & 0 \\ \frac{y - y_i}{(x - x_i)^2 \cdot \left(\frac{(y - y_i)^2}{(x - x_i)^2} + 1\right)} & \frac{1}{(x - x_i)^2 \cdot \left(\frac{(y - y_i)^2}{(x - x_i)^2} + 1\right)} & -1 \end{bmatrix} \quad (18)$$

$$R_t = \begin{bmatrix} \sigma_{\alpha\alpha,t}^i & \sigma_{\alpha r,t}^i \\ \sigma_{r\alpha,t}^i & \sigma_{rr,t}^i \end{bmatrix} - \text{matavimo neapibrėžtis}$$

1.3.3 EKF pritaikymas

1) Roboto pozicijos prognozavimas t laiko momentu[9, 10]:

$$x'_t = f(x_{t-1}, u_t) \approx F_{x,t}x_{t-1} + F_{u,t}u_t \quad (19)$$

$$P'_t = F_{x,t} \cdot P_{t-1} \cdot F_{x,t}^T + F_{u,t} \cdot Q_t \cdot F_{u,t}^T \quad (20)$$

2) Matavimas ir jo prognozavimas[9, 10]

$$\hat{z}_t = \begin{bmatrix} \sqrt{(x - x_i)^2 + (y - y_i)^2} \\ \tan^{-1} \frac{y - y_i}{x - x_i} - \theta \end{bmatrix} \quad (21)$$

$$v_t = z_t - \hat{z}_t \quad (22)$$

$$S_t = H_{x,t} \cdot P'_t \cdot H_{x,t}^T \quad (23)$$

3) Būsenos atnaujinimas[9, 10]

$$x_t = x'_t + K_t \cdot v_t \quad (24)$$

čia, K_t – Kalmano koeficientas

$$P_t = P'_t - K_t \cdot S_t \cdot K_t^T \quad (25)$$

Kalmano stiprinimo matrica:

$$K_t = P'_t \cdot H_{x,y}^T \cdot P_t^{-1} \quad (26)$$

1.4 UKF SLAM

„Unscented“ Kalmano filtravimas yra Kalmano filtro pritaikymas netiesinei sistemai. Kadangi sistema yra netiesinė, blogėja išplėstinio Kalmano filtro savybės, nes funkcijos ištiesinimui naudojama pirmos eilės aproksimacija kuri esant stipriai netiesinei sistemai sudaro dideles paklaidas. Tam tikslui naudojamas UKF. UKF sugeneruoja baigtinį skaičių atrinktų taškų (sigma taškai) (12 pav.). Sistemos klaidų koreliacijos matrica ir sigma taškai yra paremtos *sigma* vidurkių ir koreliacijų reikšmėmis. UKF algoritmas :

UT (angl. *unscented transformation*) transformacija[11]:

UT transformacija sukuria 2n+1 sigma taškų χ [12] :

$$\chi_0 = \bar{x} \quad (27)$$

$$\chi_i = \bar{x} + \left(\sqrt{(n + \lambda)P_x} \right)_i, i = 1, \dots, n \quad (28)$$

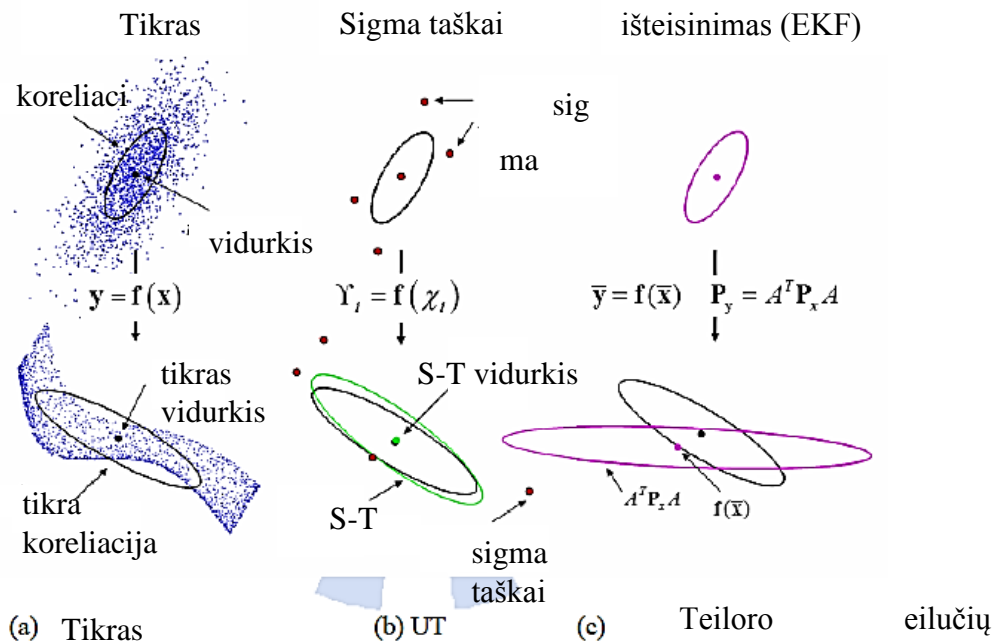
$$\chi_i = \bar{x} - \left(\sqrt{(n + \lambda)P_x} \right)_i, i = n + 1, \dots, 2n \quad (29)$$

čia \bar{x} – vidurkis

P_x - x kintamųjų koreliacija

$$\lambda = \alpha^2(n + k) - n$$

α – sigma taškų pasiskirstymas aplink x vidurkį



12 pav. Funkcijų tiesinimas UKF (b), EKF (c), tikras (a) [13]

UKF algoritmas:

1) Suskaičiuojami būsenos vidurkis ir koreliacijos matrica[14]:

$$\hat{x}_0 = E[x_0] \quad P_0 = E[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T] \quad (30)$$

2) Suskaičiuojami sigma taškai pagal 27,28,29 formulės[14]

3) Prognozė

$$\zeta_{i,k|k-1} = f(\chi_{i,k-1}), i = 0, \dots, 2n \quad (31)$$

$$\hat{x}_k^- = \sum_{i=0}^{2n} W_i^m \zeta_{i,k|k-1} \quad (32)$$

$$P_k^- = \sum_{i=0}^{2n} W_i^m (\zeta_{i,k|k-1} - \hat{x}_k^-)(\zeta_{i,k|k-1} - \hat{x}_k^-)^T + Q_k \quad (33)$$

$$\eta_{i,k|k-1} = h(\zeta_{i,k|k-1}), i = 0, \dots, 2n \quad (34)$$

$$\hat{z}_k^- = \sum_{i=0}^{2n} W_i^m \eta_{i,k|k-1} \quad (35)$$

4) Matavimo atnaujinimas (korekcija):

$$P_{\hat{z}_k \hat{z}_k} = \sum_{i=0}^{2n} W_i^c (\eta_{i,k|k-1} - \hat{z}_k^-)(\eta_{i,k|k-1} - \hat{z}_k^-)^T + R_k \quad (36)$$

$$P_{\hat{x}_k \hat{z}_k} = \sum_{i=0}^{2n} W_i^c (\zeta_{i,k|k-1} - \hat{x}_k^-)(\eta_{i,k|k-1} - \hat{z}_k^-)^T + R_k \quad (37)$$

$$K_k = P_{\hat{x}_k \hat{z}_k} P_{\hat{z}_k \hat{z}_k}^{-1} \quad (38)$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - \hat{z}_k^-) \quad (39)$$

$$P_k = P_k^- - K_k P_{\hat{z}_k \hat{z}_k} K_k^T \quad (40)$$

,čia

Q_k - proceso triukšmų koreliacijos matrica

R_k - matavimo triukšmų koreliacijos matrica

$P_{\hat{z}_k \hat{z}_k}$ - matavimo koreliacijos matrica

$P_{\hat{x}_k \hat{z}_k}$ - kryžmine koreliacijos matrica

K_k - Kalmano stiprinimas

\hat{x}_k - atnaujinta būseną

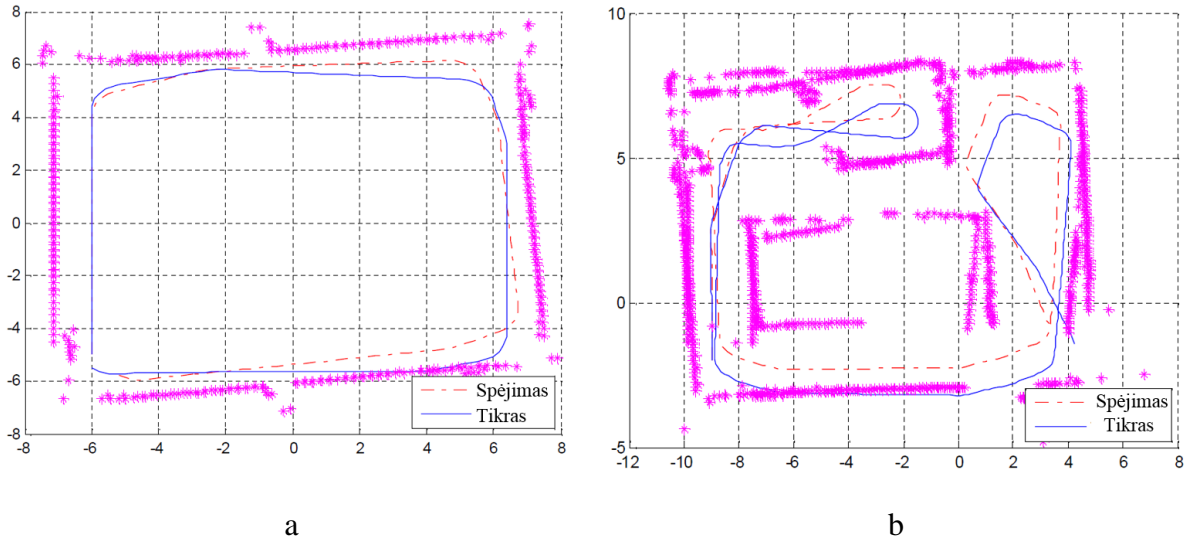
P_k - atnaujintos būsenos koreliacijos matrica

z_k - matavimas

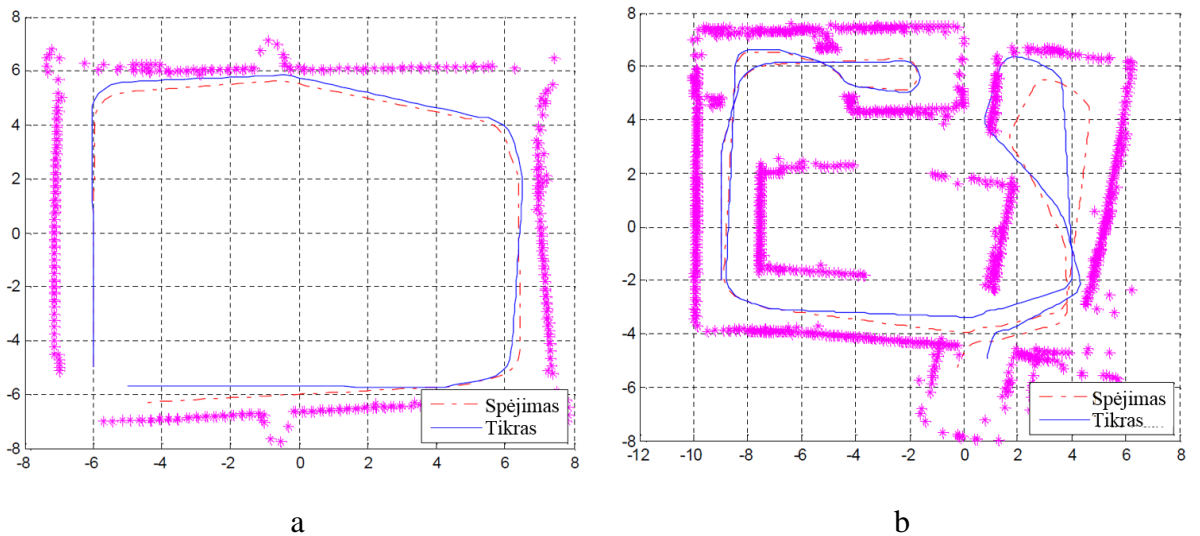
Kadangi UKF yra labai kompleksiškas savo sudėtingumu eksperimentinėje dalyje netaikomas.

1.5 UKF ir EKF palyginimas

Žemėlapiai sukurti pagal UKF ir EKF algoritmus pavaizduoti (13,14 pav.). Iš paveikslėlių matome, kad EKF algoritmas roboto kelią nuspėjo blogiau negu UKF algoritmas.

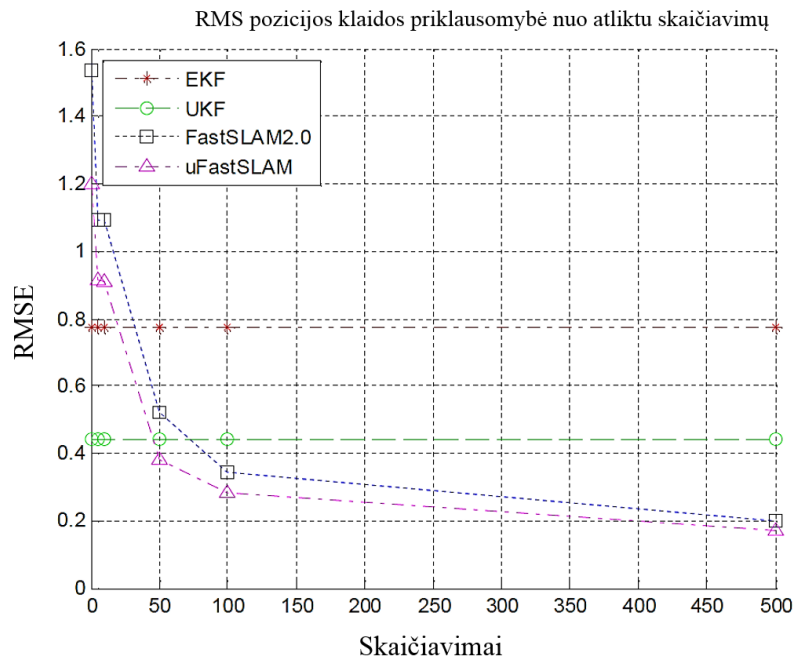


13 pav. Žemėlapiai sukurti pagal EKF algoritmą [15]

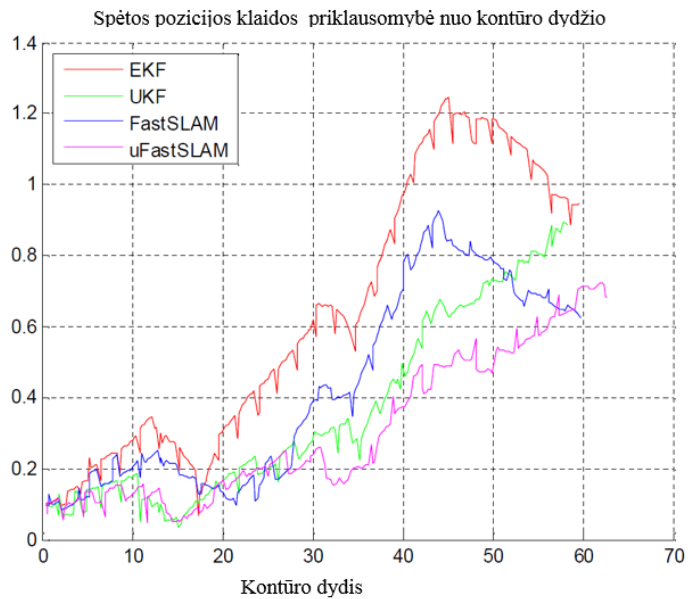


14 pav. Žemėlapiai sukurti pagal UKF algoritmą [15]

Tai pat UKF b žemėlapis lyginant su EKF b yra sukurtas labiau netikslesnis, nes matomas sienų(violetinių taškų linija) pakrypimas bei tikra roboto pozicija(mėlyna linija) UKF žemėlapyje labiau nukrypusi nuo spėtos pozicijos(14 pav. b).



15 pav. RMSE pozicijos klaidos priklausomybė nuo atliktų skaičiavimų skaičiaus [15]
 Lyginant EKF ir UKF (15 pav.), pozicijos klaida gauta mažesnė UKF algoritme. Tai pat abiejuose filtruose paklaida pastovi.



16 pav. Spėtos pozicijos klaidos (m) priklausomybė nuo kontūro dydžio (m)[15]

Iš (16 pav.) matome, kad UKF turi mažesnę pozicijos nukrypimą didėjant kontūrai negu EKF.

1.6 Apibendrinimas

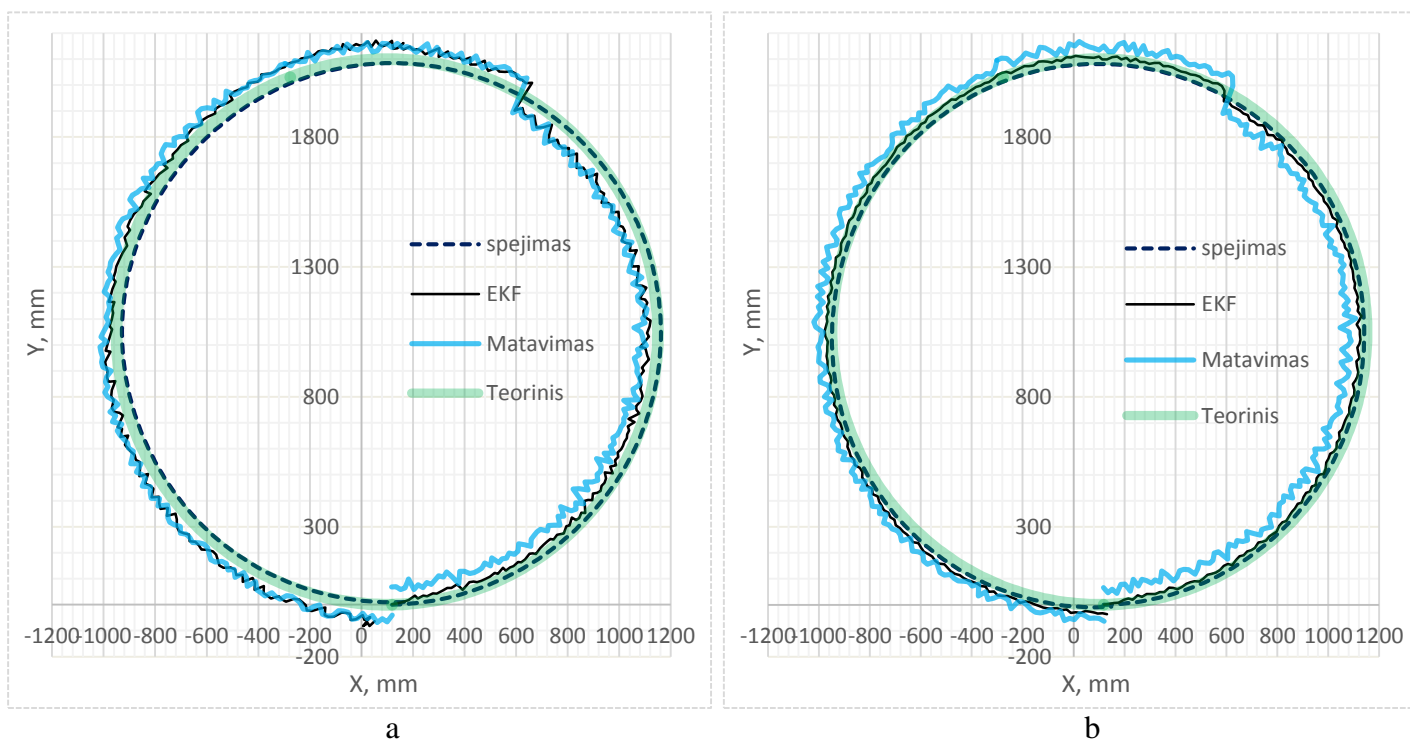
Pasitelkus SLAM galima nepertraukiamai sekti roboto poziciją ir sudarinėti žemėlapi. Sudarytos individualios žymės gali būti sumaišomos, todėl gali atsirasti klaidų. Globalios koordinatės roboto yra suskaičiuojamos iš roboto kinematikos lygčių, įvertinus roboto greitį ir poslinkį. Globalią koordinatę galima suskaičiuoti ir išmatavus atstumus iki turimos žymės. Lazerinis atstumo matavimo skeneris (LiDar) praranda tikslumą didėjant foniniam apšvietimui, kai sninga ar lyja. Norint tiksliau pozicionuoti robotą reikia naudoti EKF ir UKF tipų filtrus.

2. KALMAN FILTRO ANALITINIS MODELIAVIMAS MATLAB

Kalman filtro modeliavimo Matlab tikslas sukurti programinį modelį EKF filtrui. Iširti kaip kalman filtras elgiasi su matavimo bei spėjimo (robotui užduodamo poslinkio) paklaidomis.

Tyrimo metodika yra sudaryti roboto judėjimo apskritimu modelį. Žymė sukuriama stacionari apskritimo centre, lidar duomenys sudaromi pagal žymę. Modelyje skaičiuojama EKF filtro 360 taškų. Suvedami „Lidar“ parametrai, matavimo neapibrėžtis atstumui R matricioje $\sigma_{rr,t}^i = 1\text{mm}$, kampo neapibrėžtis $\sigma_{\alpha\alpha,t}^i = 1^\circ$, likę R matricos parametrai 0. Suvedami kinematinio modelio Q parametrai pirmu atveju kintantis nuo k atskaitos $k \cdot |s_r|$ ir $k \cdot |s_l|$ o kitu atveju pastovus kuris lygus valdymo signalo poslinkiui $1 \cdot |s_r|$ ir $1 \cdot |s_l|$.

Modeliuosime roboto judėjimą apskritimu vertindami ratų poslinkį ir matavimo triukšmus. Žemėlapyje uždedama viena žymė apskritimo centre. Imituojamas 5% matavimo triukšmas ir roboto poslinkių 7% netikslumas. Pirmu atveju imituojamas mažas triukšmas norint įvertinti kai EKF elgiasi esant mažiems triukšmams. Gauti rezultatai pateikti (17 pav. ir 18 pav.)

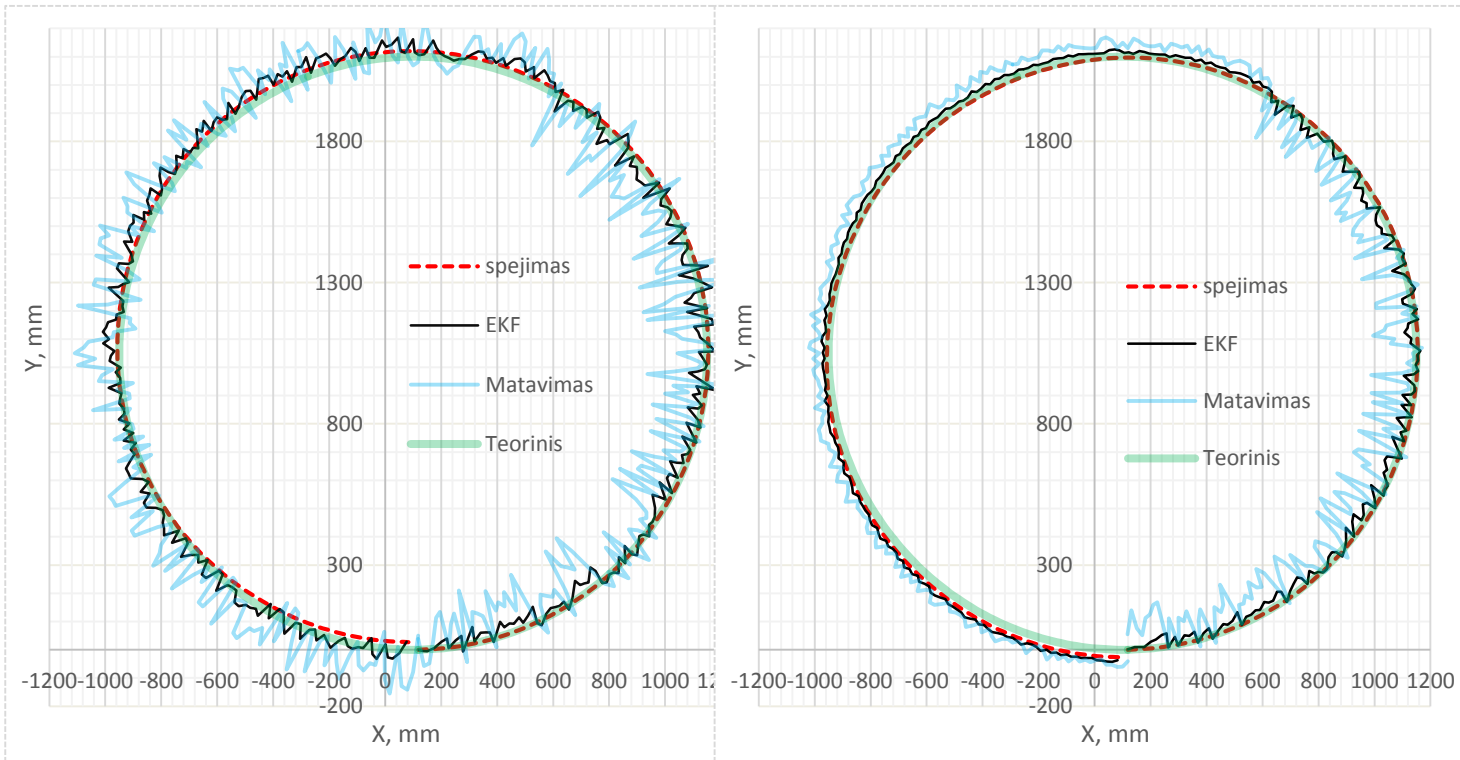


17 pav. EKF filtras su didėjančia spėjimo paklaida(a) ir su pastovia spėjimo paklaida(b)

Su didėjančia važiavimo poslinkio paklaida, kai paklaida dauginama iš k atskaitos, (17 pav. matome, kad pradžioje filtras remiasi abiem matavimais, bet vis didėjant važiavimo poslinkio paklaidai pasikliauna tik matavimų(17 pav. a).

Su pastovia važiavimo poslinkio paklaida Kalman filtras sudaro bendrą roboto pozicijos koordinatę(17 pav. b) pažymėta juodai. Ir persikėlus matavimo netikslumui į apskritimo išorę nuo galimai tikros pozicijos filtras elgiasi teisingai mažindamas nuklydimą.

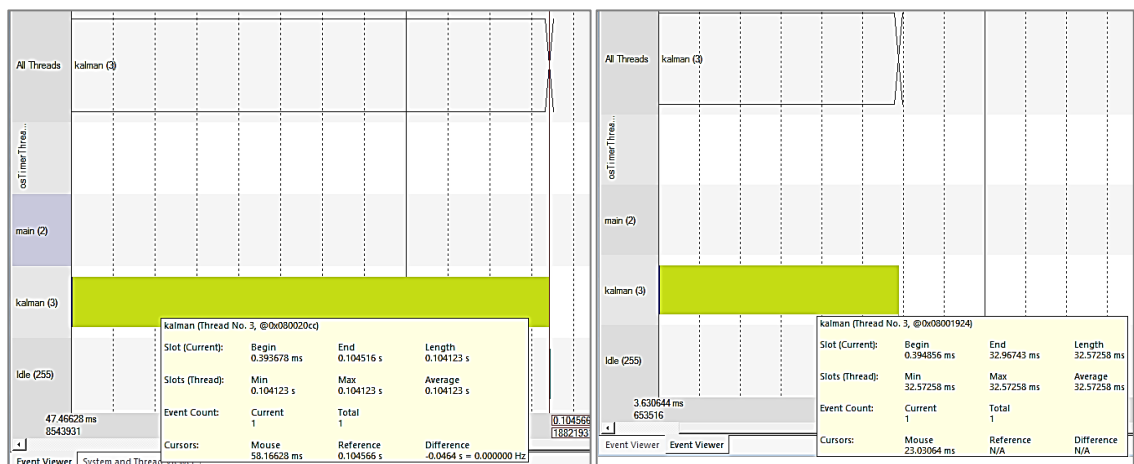
Esant dideliam LiDar matavimo triukšmui EKF nepasikliauna LiDar matavimų ir jį nufiltruoja iki patikimo matavimo dėl to robotas pozicionuojamas tiksliai net ir po labai didelio triukšmingo piko, kuris kartais net gali drastiškai pabloginti roboto poziciją (18 pav.).



18 pav. EKF filtras su pastovia spėjimo paklaida, ir 20% matavimo triukšmų

2.1 Kalman filtras eksperimentas su stm32f429

STM32F429 mikroprocesoriui Kalman filtro programa parašyta su C programavimo kalba. Atlikti 2 bandymai. Pirmas bandymas atliktas naudojantis DSP tik matricos skaičiavimo funkcijomis, o sin ir cos reikšmėms skaičiuoti buvo taikytos standartinės C kalbos funkcijos iš bibliotekos *math.h*. Antras bandymas atliktas taikant ir DSP funkcijas skaičiuojant sin ir cos reikšmes. MCU skaičiavimo rezultatai sutapo su gautais Matlab.



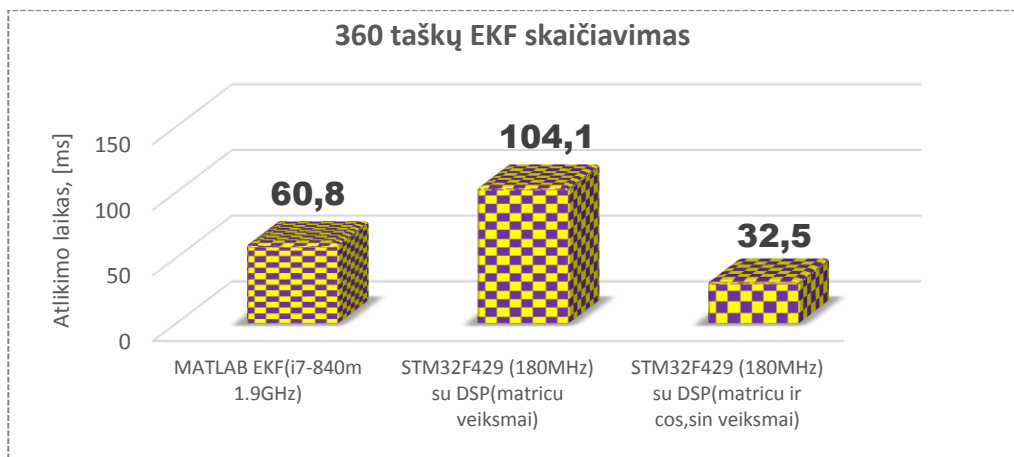
19 pav. MCU greičio matavimas su Kalman filtru

MCU matavimas atliktas Keil programos „Debug“ režime, „System and thread viewer“ analizėje. Filtras paleistas su RTOS(angl. *real-time operating system*) sistema suteikiant EKF filtro skaičiavimo procesui aukščiausia prioritetą. Pirmas MCU greitaveikos bandymo rezultatai pateikti (19 pav. a) antro bandymo (19 pav. b) čia žalias stulpelis proceso atlikimo laikas pirmo bandymo laikas „Lenght“ (0,104123s), antro bandymo atlikimo laikas „32.57258ms“.

Pakeistos eilutes MCU EKF filtro skaičiavimo algoritme :

Pirmas bandymas	Antras bandymas
Xmat[k]=zyme_f32[0]+zt[k]*sin((k*3.14)/180); Ymat[k]=zyme_f32[1]+zt[k]*cos((k*3.14)/180);	Xmat[k]=zyme_f32[0]+zt[k]*arm_sin_f32((k*3.14)/180); Ymat[k]=zyme_f32[1]+zt[k]*arm_cos_f32((k*3.14)/180);
trikst=(u1[k]+u2[k])/2; trikkam=(u1[k]-u2[k])/b; xt[0]=xt2[0]+ trikst*cos(xt2[2]+trikkam/2); xt[1] = xt2[1] + trikst*sin(xt2[2]+trikkam/2); xt[2] = xt2[2] + trikkam; Fx_f32[0] = 1.0; Fx_f32[1] = 0.0; Fx_f32[2] = -trikst * sin(xt2[2] + trikkam / 2); Fx_f32[3] = 0.0; Fx_f32[4] = 1.0; Fx_f32[5] = trikst * cos(xt2[2] + trikkam / 2); Fx_f32[6] = 0.0; Fx_f32[7] = 0.0; Fx_f32[8] = 1.0; Fu_f32[0] = 0.5 * cos(xt2[2] + trikkam / 2); Fu_f32[1] = 0.5 * sin(xt2[2] + trikkam / 2); Fu_f32[2] = 0.5 * cos(xt2[2] + trikkam / 2); Fu_f32[3] = 0.5 * sin(xt2[2] + trikkam / 2); Fu_f32[4] = 1/b; Fu_f32[5] = 1/b;	trikst=(u1[k]+u2[k])/2; trikkam=(u1[k]-u2[k])/b; cosr=arm_cos_f32(xt2[2]+trikkam/2); sinr=arm_sin_f32(xt2[2]+trikkam/2); xt[0]=xt2[0]+ trikst*cosr; xt[1] = xt2[1] + trikst*sinr; xt[2] = xt2[2] + trikkam; Fx_f32[0] = 1.0; Fx_f32[1] = 0.0; Fx_f32[2] = -trikst * sinr; Fx_f32[3] = 0.0; Fx_f32[4] = 1.0; Fx_f32[5] = trikst * cosr; Fx_f32[6] = 0.0; Fx_f32[7] = 0.0; Fx_f32[8] = 1.0; Fu_f32[0] = 0.5 * cosr; Fu_f32[1] = 0.5 * sinr; Fu_f32[2] = 0.5 * cosr; Fu_f32[3] = 0.5 * sinr; Fu_f32[4] = 1/b; Fu_f32[5] = 1/b;

Pakeistos ne tik eilutės algoritme, bet ir pasikartojimai kurie skaičiuoja tą pačią reikšmę.



20 pav. Skaičiavimo greitaveikos palyginimas

Iš (20 pav. matome, kad STM32F429 su 180MHz sinchronizavimo dažniu 360 skaičiavimų atlieka per 104,1ms naudodamas tik matricų skaičiavimo DSP funkcijas, kai tuo tarpu MATLAB su i7-840m 1,9GHz procesorių atlieka per 60,8ms kas beveik 2 kartus greičiau, nors procesorius 10 kartų greitis iš to matome, kad bendros paskirties procesoriai nepritaikyti DSP skaičiavimams ir juos atlieka sunaudodami žymiai daugiau resursų. Taip pat pritaikius STM32F4 DSP komandas ir sin bei cos reikšmių skaičiavimą STM32F4 su užduotimi „susidorojo“ 2 kartus greičiau negu MATLAB su i7-840m procesoriumi, iš to galime spręsti, kad STM32F4 su DSP komandomis skaičiavimus atlieka xx kartų greičiau ir su daug mažesniu energijos kiekiu.

2.2 Apibendrinimas

Globali pozicija roboto išmatuojama žymiai tiksliau kai naudojamas išplėstinis Kalmano filtras, kuris apjungia matavimo iš LiDar ir ratų poslinkio skaičiavimo duomenis. Esant dideliems „Lidar“ matavimo ar poslinkio triukšmams EKF pasikliauna labiau netriukšmingesnių matavimų ir roboto pozicija nustato arčiau netriukšmingesnio. Esant dideliam triukšmui robotas nepasimeta ir savo poziciją koreguoja pakankamai tiksliai, remdamasis judesiu. STM32F429 (180MHz) su DSP skaičiuojant matricas bei sin, cos reikšmes Kalman filtro 360 taškų suskaičiavimo per 32,5ms, o tai 2 kartus greičiau negu MATLAB su i7-840m 1,9GHz procesoriumi. STM32F429 su standartinės bibliotekos sin ir cos funkcijomis skaičiavimus atliko 3 kartus lėčiau negu naudojant DSP komandas.

3. EKSPERIMENTINĖS ĮRANGOS KONSTRAVIMAS

Roboto platformai buvo panaudotas RC 1/16 tanko modelis (21 pav. a). Platforma pasirinkta todėl, kad ji atitinka praeitame darbo etape sudarytam Kalman filtro lygčių sistemos funkcijoms. RC tankas modifikuotas panaikinant viršutinę dalį ir pridendant laikančią konstrukciją (21 pav. b). Pavaros DC varikliai pakeisti į žingsninius variklius tam, kad būtų galima suformuoti tikslesnį poslinkį.

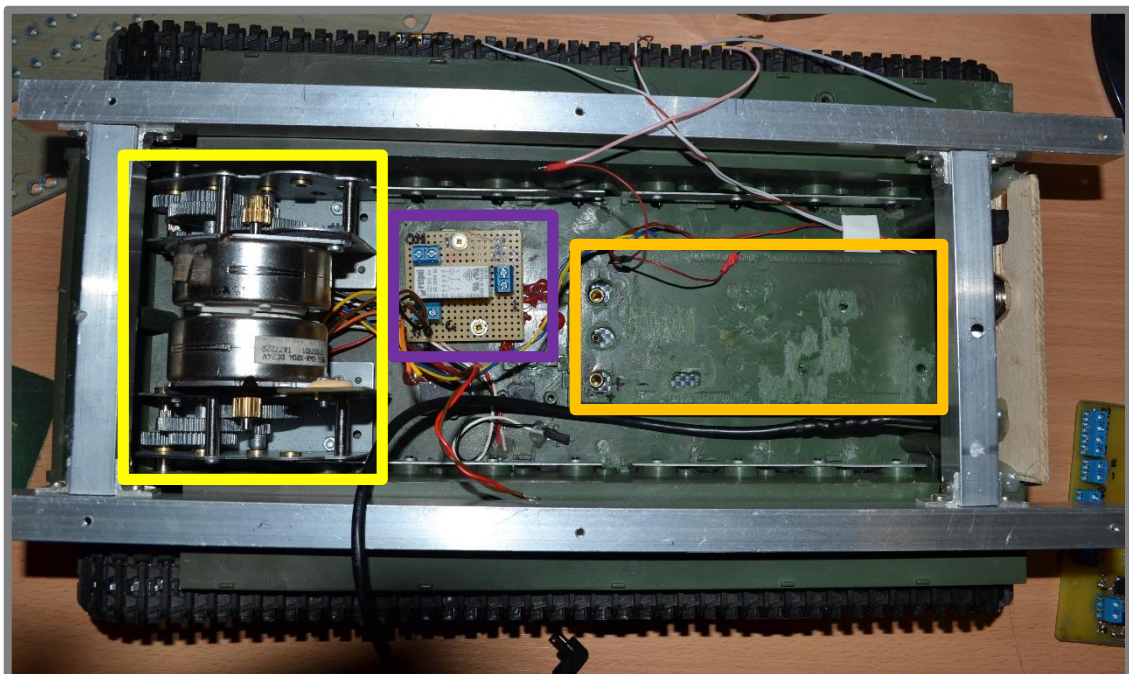


a

b

21 pav. RC tanko modelis originalus (a) ir modifikuotas(b)

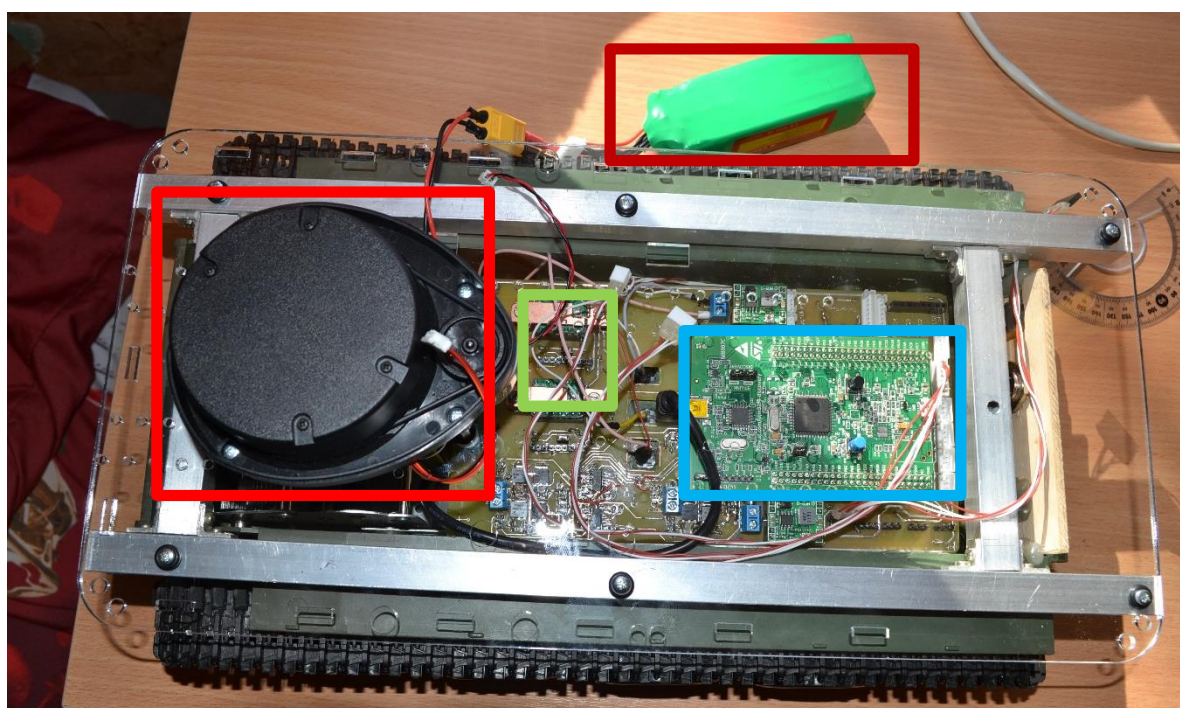
Mobilią roboto elektroniką sudaro 2 žingsninių variklių kontroleriai, 2 DC/DC keitikliai („buck“ tipo). Roboto sudedamųjų dalių schema pateikta (22 pav. . ir 23 pav.).



22 pav. Mobilios roboto platformos sudedamosios dalys

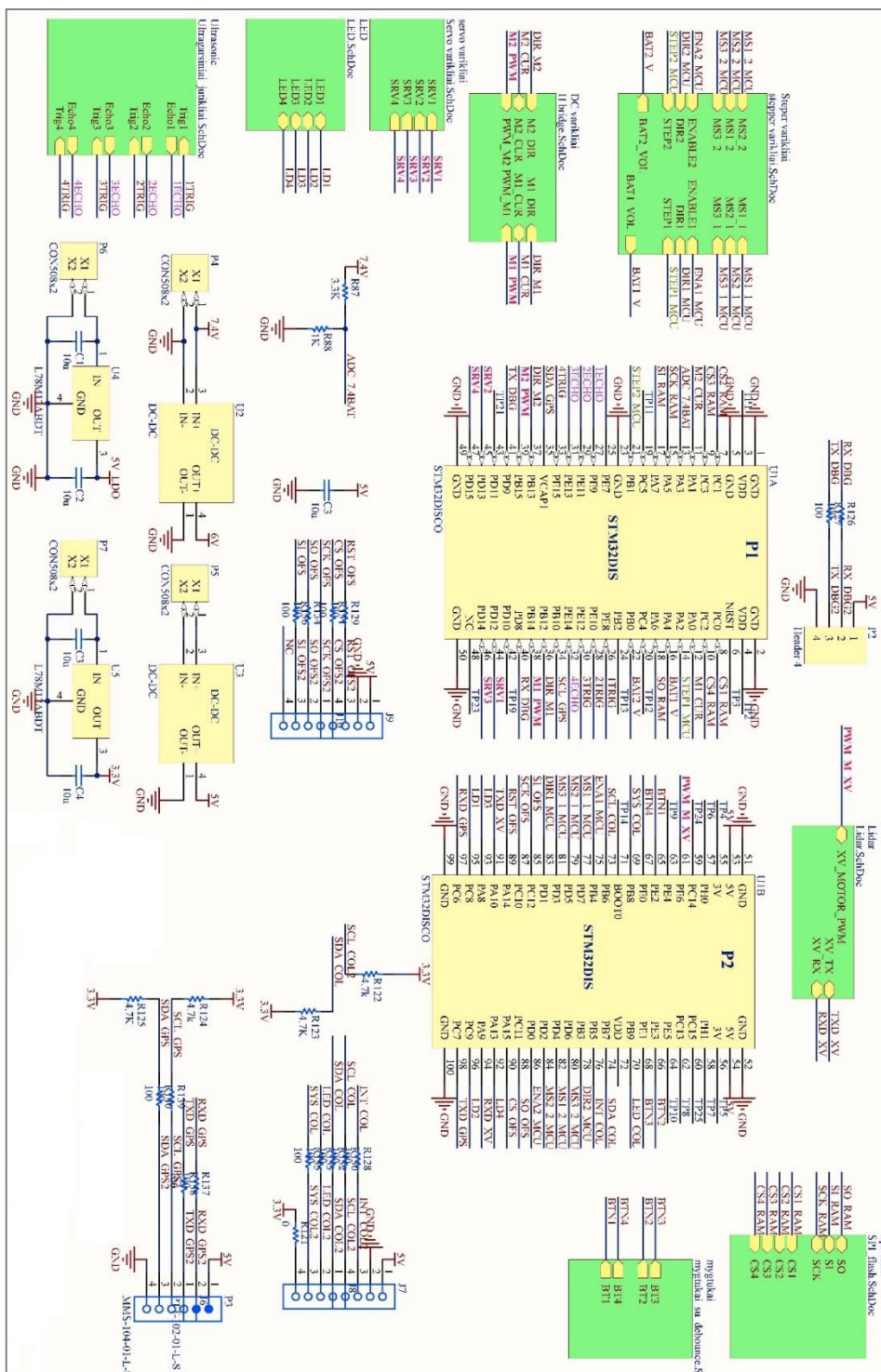
2 lentelė. Roboto sudedamosios dalys

	Reduktoriai su žingsniniais varikliais
	2 kontaktų relė sujungti akumulatoriams su elektronika
	7,4V 5000mah Ličio jonų akumulatorius elektronikos schemai
	XV-11 „Lidar“ jutiklis
	Žingsniniu variklių valdikliai A3988
	„STM32F407 Discovery“ plokštė
	18,4V 2200mah Ličio jonų akumulatorius ŽEV valdikliams



23 pav. Mobilaus roboto platforma

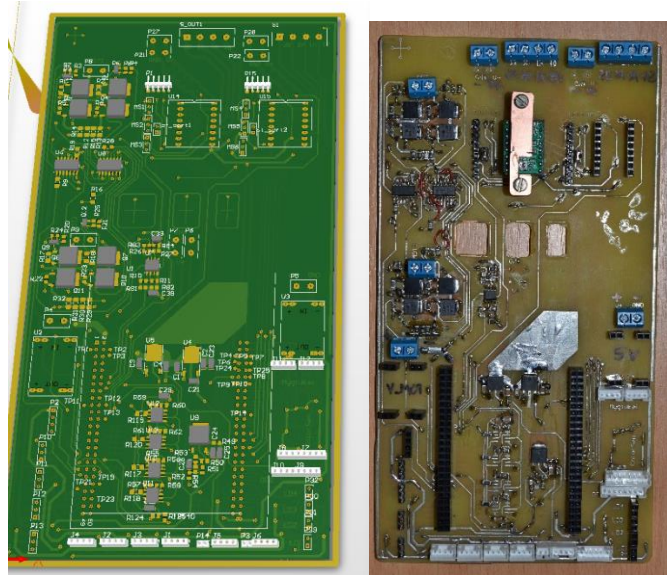
Ličio jonų 18,4V (23 pav.) akumulatorius panaudotas žingsninių variklių valdiklių (A3988) maitinimui. MP1584(23 pav.) „Buck“ keitiklis naudojamas maitinti STM32 Cortex mikroprocesorių bei XV-11 jutiklį (+5 V), o kitas papildomai elektronikai. MP1584 įėjimo įtampa nuo 4,5V iki 28V išėjimo nustatoma nuo 0,8V iki 20V, maksimali srovė 3A. Impulsiniai maitinimo šaltiniai pasirinkti todėl, kad roboto platforma maitinama iš 8,4V 5000mAh Li-Po baterijos ir norint kuo efektyviau išnaudoti energiją, reikia aukšto efektyvumo keitiklių. Kompensacinių keitiklių efektyvumas siekia tik 50-70%, tuo tarpu impulsinių virš 90%.



24 pav. Roboto elektronikos sujungimo schema

A3977 žingsninių variklių valdiklis gali būti nustatytas į „FULL“, „HALF“, „1/4 microstepping“, „1/8 microstepping“ ir „1/16 microstepping“ darbo režimus. „FULL“ darbo režimas kai žingsninio variklio veikia abi fazės kai pirmą fazę išjungiamo, įjungiamo antra fazė ir ŽEV atlieka vieną žingsnį. „HALF“ - jungiamos kelios fazės, kurio vienos fazės atjungiamos, o kitos lieka įjungtos, todėl posūkio kampas sumažėja. Mikrožingsnio režimo atveju srovė apvijoje keičiama dalimis, dėl to gauname 1/4, 1/8, 1/16, 1/32 mikrožingsnio darbo režimus. Mes naudojame 1/8 mikrožingsnio režimą. Tai pat gali būti nustatoma variklio srovė, kuri

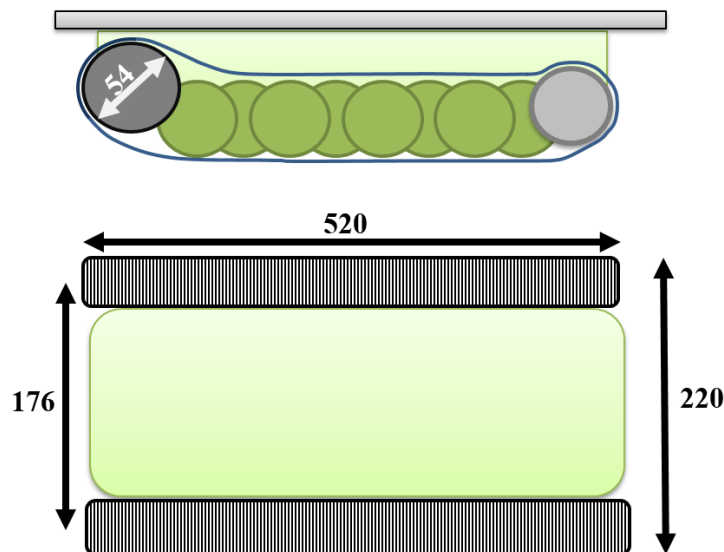
Žingsniniam varikliui PM55L-048[16] parinkta iki 600mA. Žingsnio valdymo signalas (stačiakampis impulsas) yra paduodamas i STEP įėjimą, posūkio kryptis keičiama su signalu DIR. Schema suprojektuota taip, kad būtų galima prijungti papildomus roboto mazgus. Prie roboto galima prijungti dar 2 papildomus DC variklius, 4 ultragarsinius atstumo jutiklius, GPS, OFS („Optical flow sensor“), UART sąsaja, Lidar jutiklį, SPI SRAM atmintis, 4 RC servo variklius (24 pav.). Suprojektuota dvipusė PCB plokštė taip, kad tilptų į RC tanko korpusą(25 pav.).



25 pav. Roboto PCB

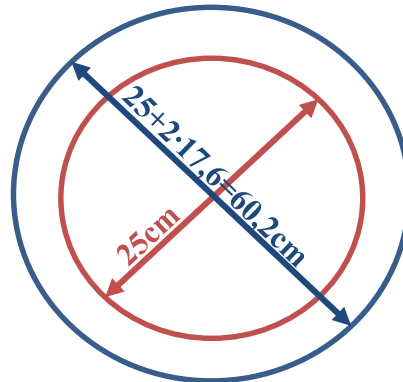
3.1 Roboto geometrija

Pagal rato skersmenį yra suskaičiuojamas vieno apsisukimo poslinkis, kuris lygus apskritimo perimetrai. Vienu apsisukimu robotas nuvažiuoja $\pi \cdot d = 3,14 \cdot 54 = 170\text{mm}$. Atstumas tarp vikšrų centrų yra 176mm.



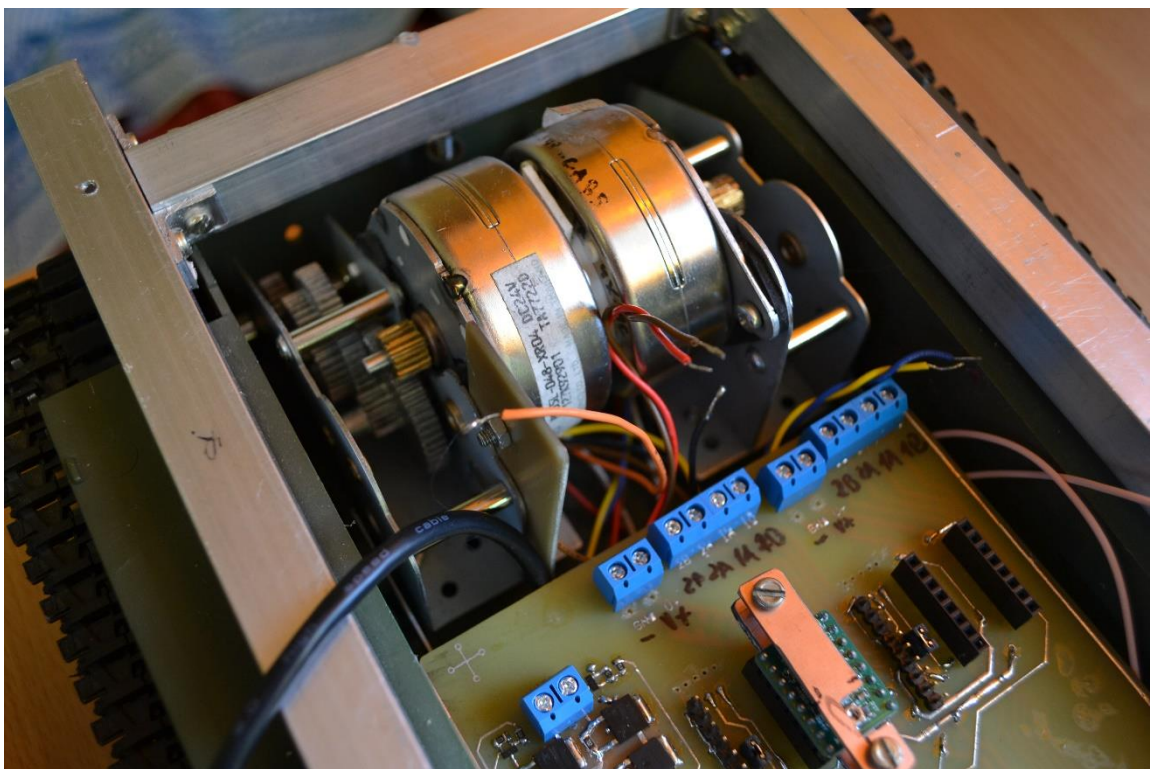
26 pav. Roboto geometrija

Sudaromas pradinis roboto judesio apskritimu modelis t.y. vienas ratas turi pasislinkti didesniu apskritimu - kitas mažesniu, tarp apskritimų turi būti ratų bazės skirtumas 176mm. Vienam ratui reikia padaryti poslinkį $25 \cdot 3,14 = 78,5\text{cm}$, o kitam $60,2 \cdot 3,14 = 189\text{cm}$ (27 pav.).



27 pav. Roboto judėjimas apskritimu

3.2 Roboto pavaros



28 pav. Roboto pavaros

PM55L-048[16] žingsninio variklio žingsnio kampas yra $7,5^\circ$. Kad apsisuktų viena kartą reikia atlikti $360/7,5=48$ žingsnius. Reduktoriaus santykis $1/18$, taigi vienam apsisukimui reikia $18 \cdot 48 \cdot 8 = 6912$ žingsnių. Vienas roboto rato žingsnis atitinka $170/6912=0,0245\text{mm}$ poslinkį, bet praktiškai robotas dėl praslydimo su paviršiumi gali teorinio tikslumo nepasiekti.

3.3 Neato xv-11 lazerinis atstumo skeneris

Kalman filtrui realizuoti reikalingi 2 matavimai, vienas iš matavimų tai atstumo matavimas iki objektų. Tam panaudotas „LiDar“ (lazerinis atstumo skeneris).

3.4 XV-11 jutiklio versijos nustatymas ir parametrai

Jutiklio versija galima sužinoti per UART serial lidar jungtį, per kurią nusiuntus <ESCAPE> API komandą jutiklis nustoja skenuoti aplinką ir gali būti konfigūruojamas ir kalibruojamas su komandomis:

- Help – gaunamas esamų komandų sąrašas;
- Log – komanda nupiešia širdį su simboliais;
- GetVersion- gaunami jutiklio duomenis;
- SaveCal;
- SetBaud;
- SetSerial;
- Upload;
- Wanderer
- Calibrate b16 b8 SunBlind loop2AA loop155;
- GetCal A B C LPT LFL LFT LFH IMX IB LPI LCH LPD ANG;
- SetCal A B C LPT LFL LFT LFH IMX IB LPI LCH LPD ANG;
- Spin Fake DotX DotI Text Hash Timing Foto RPS Pac;
- TestEncoder;

Gauti turimo jutiklio duomenys yra:

```
GetVersion...3 ESCs or BREAK to abort...:)
Piccolo Laser Distance Scanner
Copyright (c) 2009-2011 Neato Robotics, Inc.
All Rights Reserved

Loader      V2.5.14010
CPU         F2802x/c001
Serial      WTD19411AA-0018950
LastCal     [5371726C]
Runtime     V2.6.14138
OK
#
```

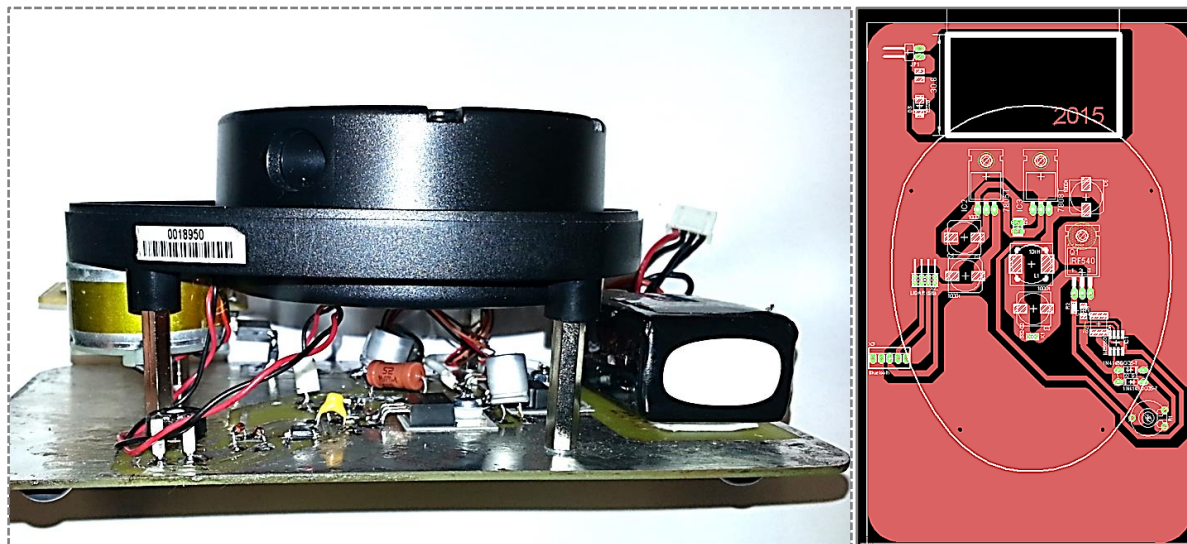
Taigi nusistačius XV-11 lazerinio atstumo skenerio versiją, galima nustatyti visus jo parametrus. Serial sąsaja 8N1 115200bps[17]. Maitinimas +5V suvartojama srovė lazerio ~135mA, tai pat veikia ir su +3,3V.

Jungties sujungimai:

Raudonas	+5V
Rudas	LDS_RX
Oranžinis	LDS_TX
Juodas	GND

XV-11 LiDar variklis gali būti prijungtas prie 3,3V kas sukutų variklį ir lidar „galva“ 240rpm. Jutiklis turi būti sukamas tarp 180 - 349 rpm. Pagal atlikta analizė [17] virš 349rpm greičio duomenų paketai būna prarandami.

Pagal tai suprojektuota jutiklio testavimo įranga (29 pav.) kurią sudaro duomenų sąsaja Bluetooth, atskiras maitinimas iš Li-Po baterijos, variklio greičio reguliavimas su PWM.



29 pav. XV-11 LiDar testavimo įranga

3.5 XV-11 jutiklio duomenų formatas

Per 360 laipsnių apsisukimą yra perduodama 90 paketų, kurių kiekvieną sudaro 4 atstumo matavimo rezultatai. Paketo ilgis 22 baitai, iš viso vienas apsisukimas pateikia 1980 baitų duomenų[17].

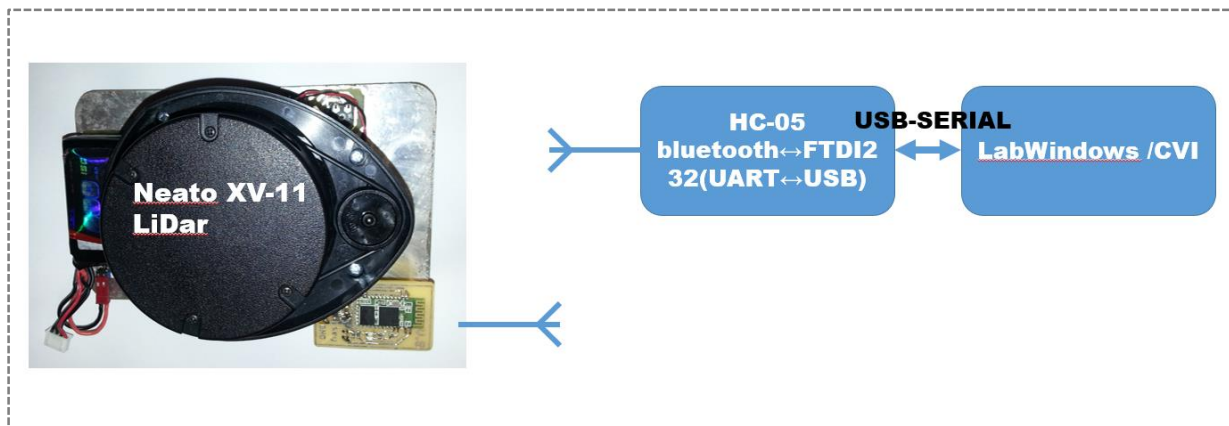
Kiekvienas paketas organizuotas taip[17]:

```
<start> <index> <speed_L> <speed_H> [Data 0] [Data 1] [Data 2] [Data 3] <checksum_L>
<checksum_H>
```

čia,

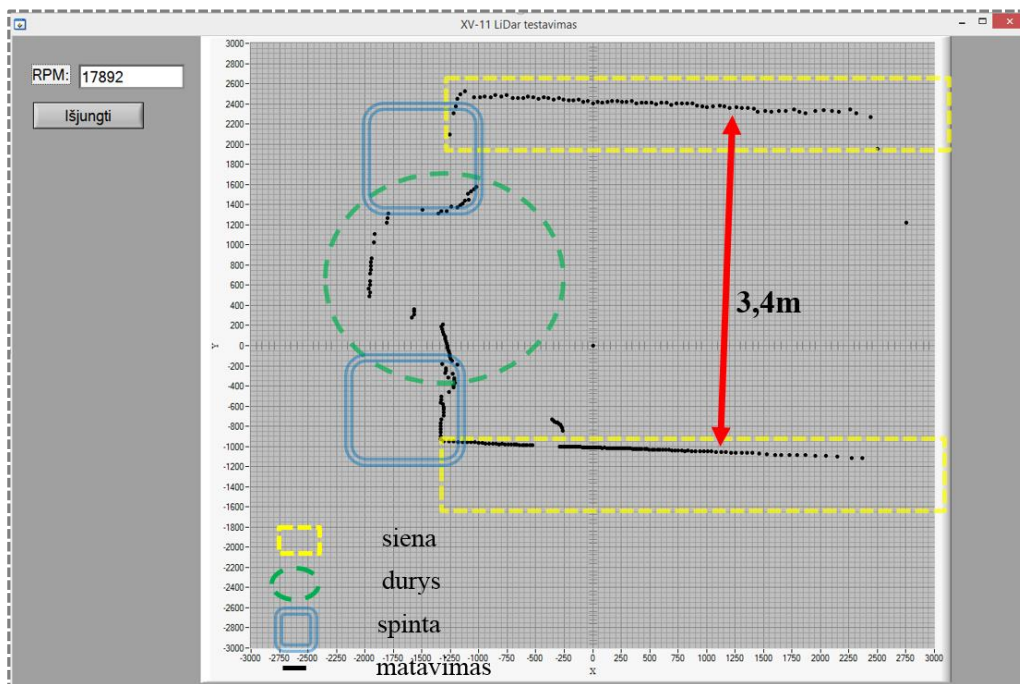
- start visada 0xFA
- index paketo numeris prasideda nuo 0xA0 (0 paketas, 4 atstumo matavimai 0 iki 3) iki 0xF9 (89 paketai, matavimas kampas 356-359).
- speed paketas sudarytas iš 2 byte, išduodantis jutiklio „galvos“ sukimosi greiti RPM formatu.
- [data 0] - [Data 3] kiekvienas turi 4 baitus:
 - `byte 0 : <atstumas 7:0>`
 - `byte 1 : <"invalid data" flag> <"strength warning" flag> <atstumas 13:8>`
 - `byte 2 : <signal stiprumas 7:0>`
 - `byte 3 : <signal stiprumas 15:8>`

Atstumas matuojamas milimetrais 14bitų tikslumu. Kai 7bitas iš pirmo baido yra 1, tai atstumas negali būti nuskaitymas. Kai 6 bitas iš pirmo baido yra vienetas, tai signalas negali būti nuskaitymas, nes lazerio spindulys neatsispindi nuo juodo paviršiaus, arba kai atspindėtas spindulys nukrypsta nuo šaltinio.



30 pav. CVI su LiDar sujungimo schema

Pagal nuskenuotą kambario planą (31 pav.) matome, kad jutiklis veikia gerai, pastebi sienas, spintas, duris, bet yra ir nepastebimų dalykų kaip langas ar į šviesą nukreiptas spindulys, jutiklis nustatytas sukis 179rpm greičiu. Duomenys siunčiami tokiu greičiu be klaidų (31 pav.). Prie didesnio greičio jutiklis turi disbalansą ir vibruoja. Dėl to tyrimams pasirinktas mažesnis greitis.



31 pav. LiDar nuskenuotas kambario vaizdas

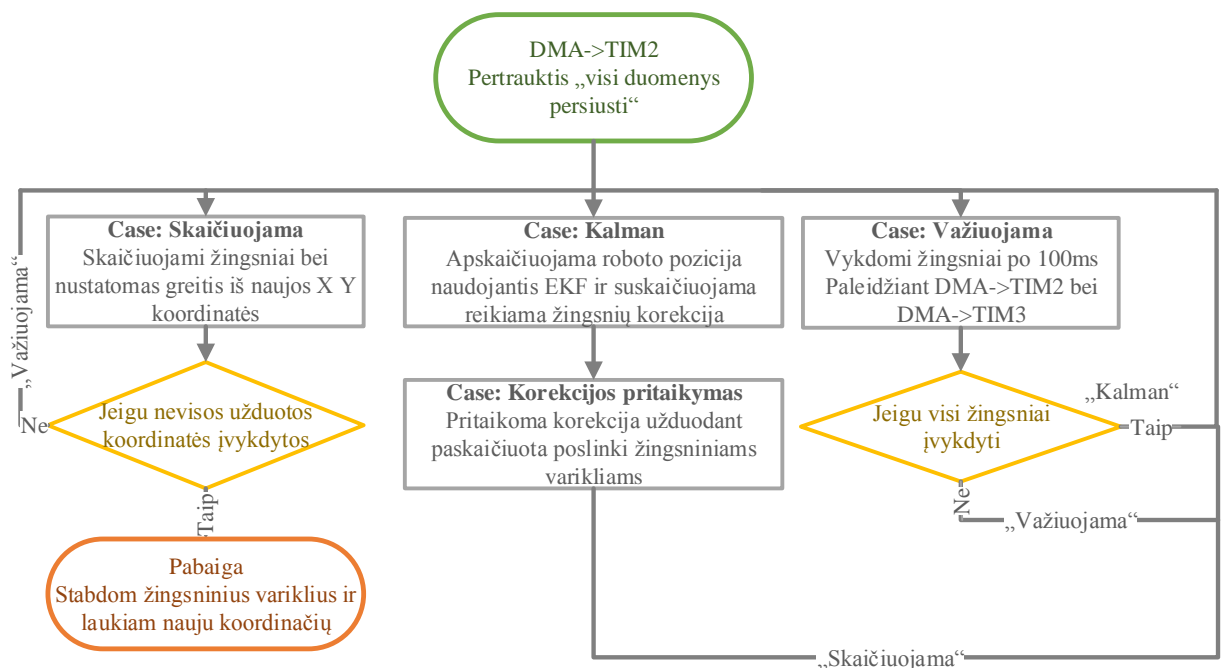
3.6 Apibendrinimas

Sukonstruota eksperimentinė įranga paremta Neato XV-11 lazeriniu atstumo skeneriu, roboto pavaras sudaro žingsniniai varikliai, važiuoklė vikšrinė kuri atitinka sudarytoms Kalman filtro matematinio modelio lygtims. Žingsniniai varikliai valdomi A3988 valdikliu kuris nustatytas 1/8 mikrožingsnio režimu. Suskaičiuotas vieno žingsnio poslinkis robotui lygus 0,0245mm. Suprojektuota PCB plokštė kuri integruojama į roboto korpusą. Robotas valdomas STM32F407 bendros paskirties ARM mikrovaldikliu. Suprojektuota plokštė gali prisijungti papildomus roboto mazgus: 2 DC variklius, 4 ultragarsinius jutiklius, 4 RC servo variklius, GPS, OFS, UART bei SPI SRAM. Sukonstruota „Lidar“ testavimo įranga ir ištestuotas veikimas. Nustatomas lazerinio atstumo skenerio sukimosi greitis 180 rpm, nes prie didesnių greičių jutiklis turi disbalansą ir vibruoja.

4. PROGRAMINĖS ĮRANGOS SUKŪRIMAS IR INTEGRAVIMAS

4.1 Žingsninių variklių valdymas

Sudaroma programa žingsniniams varikliams valdyti su kintamu greičio bei poslinkio reguliavimu kiekvienam varikliui. Žingsninių variklių valdymo signalas formuojamas STM32F407 „Timer 2“ ir „Timer 3“ kurie nustatyti veikti „Output Compare“ režimu ir nustatytas „TIM_OCMODE_TOGGLE“ išėjimas, išėjimo signalas keičia būseną iš 0 į 1 kai „Timer“ periodas persipildo. „Timer“ periodas nustatytas, kad 1 atitiktų 250ns periodą, maksimali užduodama reikšmė 32-bit. Periodas užkraunamas tiesiogiai iš atminties masyvo kuris suformuojamas pagal koordinatės X Y ir norimą važiavimo greitį supaprastinta algoritmo schema pateikta (**Error! Reference source not found.**). Kalman filtras pritaikomas po X Y koordinatėjų nuvažiavimo. Suskaičiuojamos koordinatės pagal Kalman filtrą pritaikius Lidar ir poslinkio duomenis. Toliau suskaičiuojama koordinatės paklaida ir pritaikoma korekcija pasiekti užduota koordinatė.

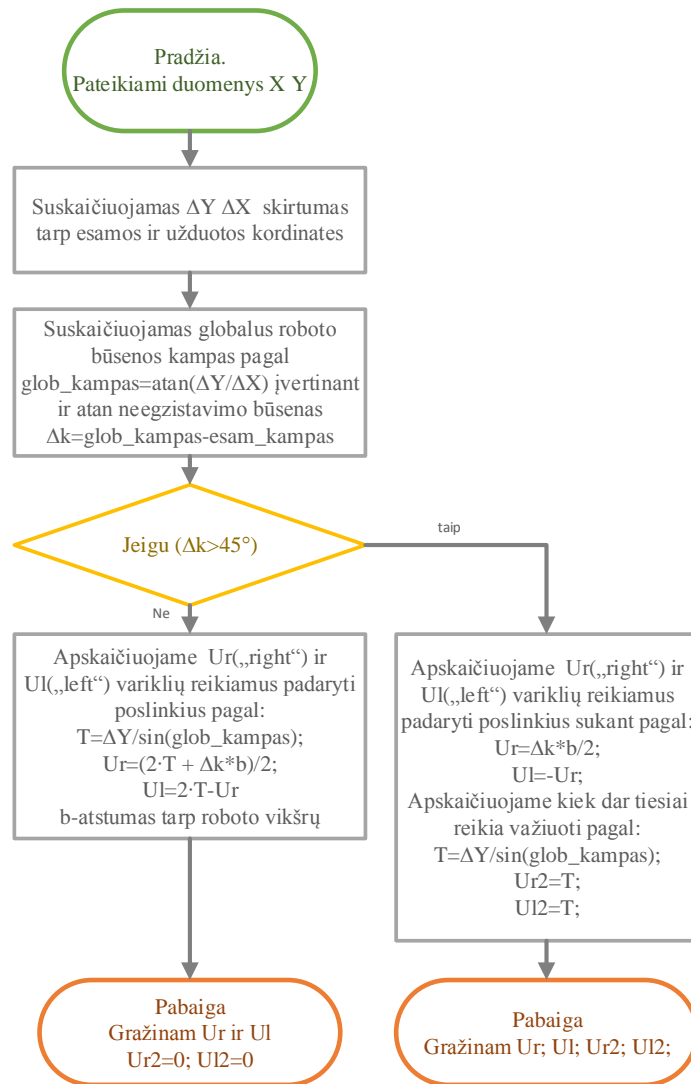


32 pav. Roboto valdymo algoritmas

ŽEV didžiausias greitis pagal variklio duomenis 500Hz kas atitinka 2ms periodą arba MCU 8000 „timer“ periodo reikšmę, bet kadangi užduodamas signalas yra nustatytas „TIM_OCMODE_TOGGLE“ režimu tai reikšmė dalinama pusiau, kad atitiktų reikiama signalo dažnį, tai pat ŽEV valdomas 1/8 „microstepping“ režimu tai pat dalinama iš 8 taigi MCU reikšmė esant didžiausiam greičiui lygi 500. Nuo didžiausios variklio greičio periodo reikšmės

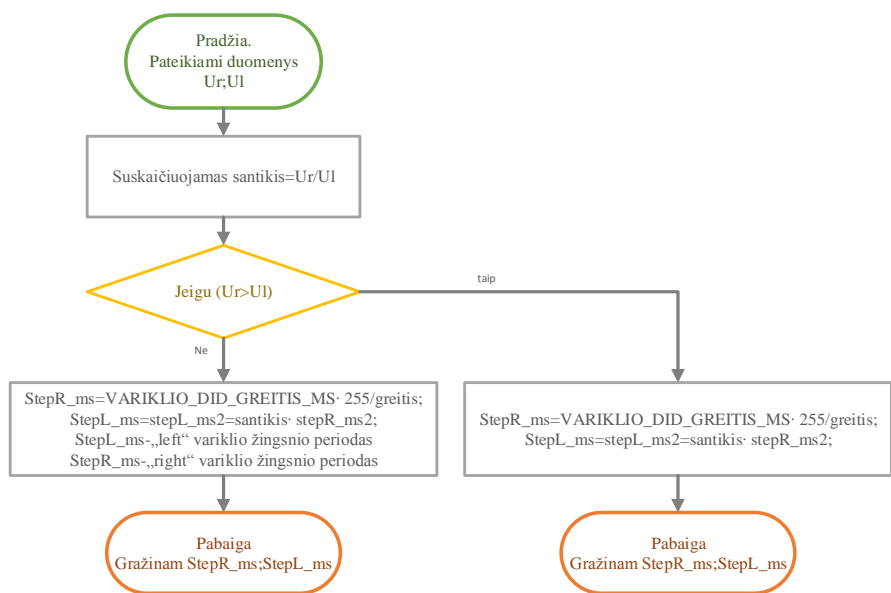
atliekami skaičiavimai kaip bus suformuotas „right“ ir „left“ variklių signalai, kad būtų valdoma pagal X ir Y ir greičio duomenis. Valdymas su DMA ir „Timer“ neapkrauna procesoriaus, procesorius apkraunamas mažiausiai lyginant su kitais valdymo metodais[18].

4.2 Koordinatų į poslinkį skaičiavimas



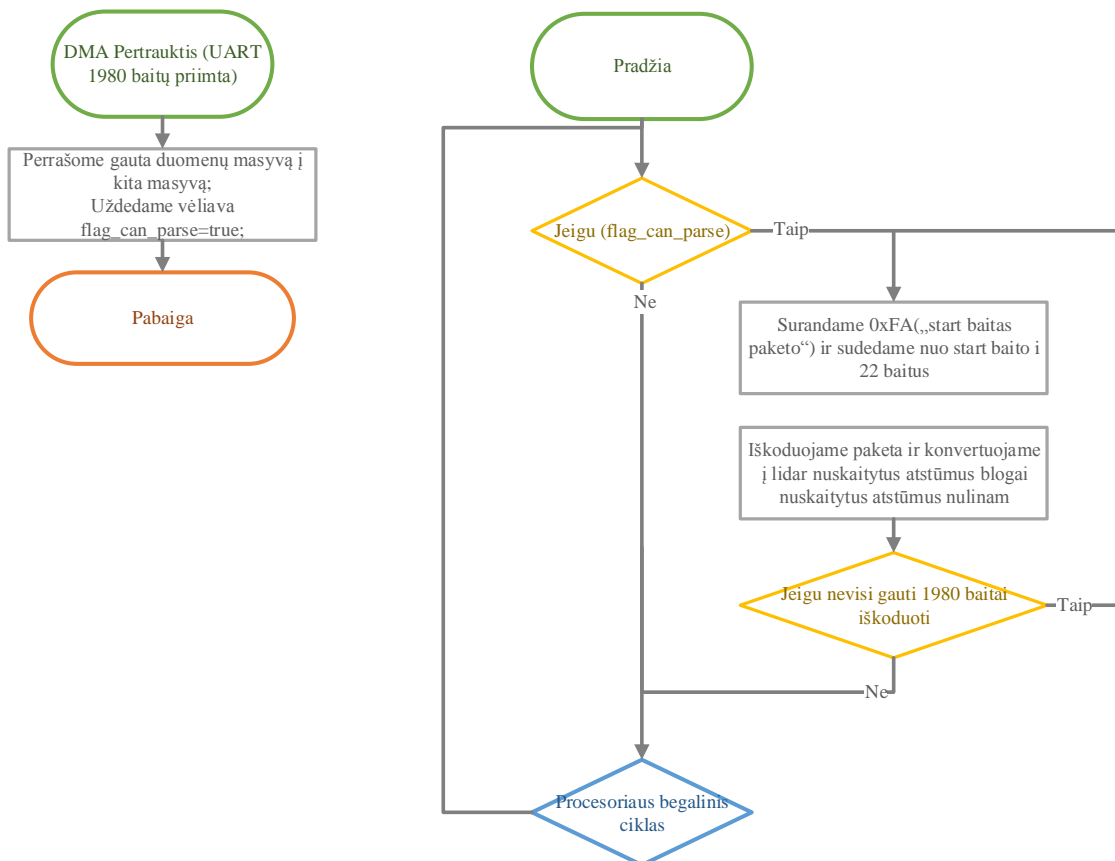
33 pav. Užduotų X Y koordinatų keitimas į poslinkį algoritmas

Robotas valdomas užduodant koordinatas kurias jis turi pasiekti, koordinatas keičiamos į reikiamus padaryti poslinkius kiekvienam varikliui pagal (33 pav.) pateikta algoritmą. Skaičiavimo algoritmas taikomas (**Error! Reference source not found.**) algoritmo „case: skaičiuojama“ būsenoje. Po to seka kitas skaičiavimo algoritmas(34 pav.), kuris suskaičiuoja kokių periodu turi veikti ŽEV. Bei iš turimų suskaičiuotų duomenų sudaromas žingsnių masyvas, kuris paskui per DMA perduodamas į „Timer 2“ ir „Timer 3“ periodų reikšmės.



34 pav. Žingsninių periodo suskaiciavimo algoritmas

4.3 Lidar duomenų nuskaitymas



35 pav. Lidar duomenų apdorojimo algoritmo supaprastinta schema

Lazerinio atstumo skenerio duomenys gaunami per UART sąsają duomenys kaupiami tiesiogiai į atmintį(DMA) kai gaunamas 90 paketu po 22baitus kas atitinka 1980 baitų gaunama pertrauktis ir duomenys perrašomi į kitą duomenų masyvą bei uždėdama vėliavėlė, kad galima

duomenys dekoduoti. Duomenys iškoduojami ir surašomi į 360 atstumo duomenų masyvą, indeksas atitinka išmatuoto atstumo kampą. Iš atstumo duomenų masyvo surandama priskirtą žymė ir gauti žymės duomenys naudojami EKF skaičiavime. Žymė priskiriama pirmu skenavimų surandant arčiausią objektą.

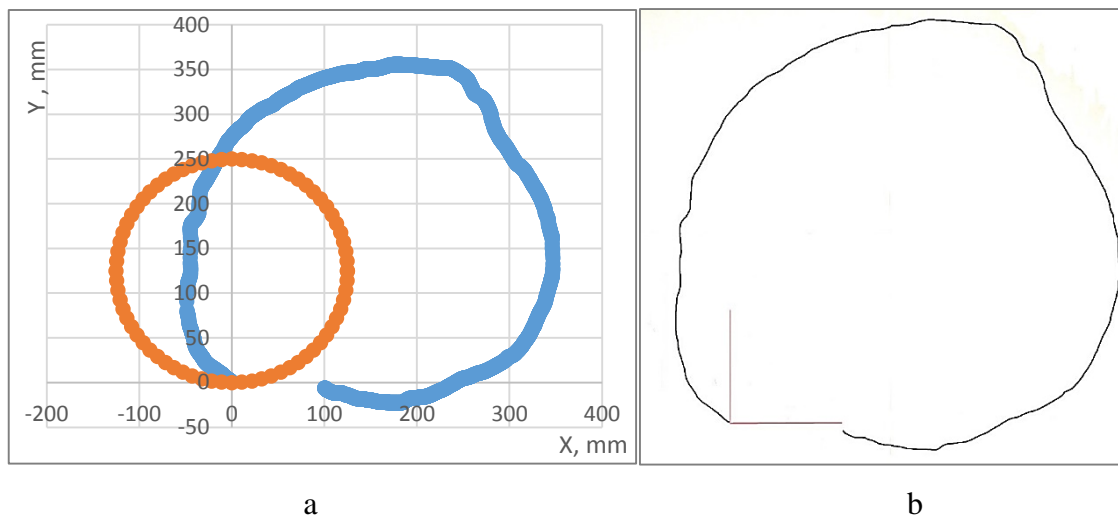
4.4 Apibendrinimas

Sudaryti Lidar duomenų nuskaitymo ir duomenų apdorojimo algoritmai, naudojantis DMA, taip neapkraunant mikroprocesoriaus. Parašytas žingsnių variklių valdymo algoritmas užduodant robotui norimas koordinatas X Y. Žingsniniai varikliai valdomi „Timer 2“ ir „Timer 3“ naudojantis DMA „Timer“ periodo perkėlimu, toks valdymas neapkrauna mikroprocesoriaus. Procesorius apkraunamas mažiausiai lyginant su kitais valdymo metodais[18]. Sudarytas algoritmas integruojantis Kalman filtrą, daromoms roboto paklaidoms sumažinti.

5. EKSPERIMENTINIAI TYRIMAI

5.1 Pavaros pozicionavimo tikslumo tyrimas

Pavaros pozicionavimo tyrimas, buvo atliktas su užduotu 25cm apskritimo poslinkiais pavaroms. Robotui buvo pritvirtintas pieštukas kuris ant balto popieriaus išbrėžė realia nuvažiuota kreivę (36 pav. b). Važiavimo paviršius tiesus, popierius. Kreive buvo suskaitmenizuota su Im2graph programa ir palyginta su realia užduota (36 pav. a).



36 pav. Suskaitmenizuota nuvažiuota roboto kreive

Kaip matome pagal rezultatus (36 pav.) robotas smarkiai nuklydo nuo užduotos kreives. Paklaida iki 20cm, tokia paklaida atsirado dėl vikšrų praslydimio bei plačių vikšrų, dėl kuriu negalima tiksliai nustatyti vikšro poslinkio kreives(36 pav.) taip pat dėl to kad vikšrai plastikiniai ir neturi reikiamo sukibymo su paviršiumi.

3 lentelė. Važiavimas tiese

žingsnių skaičius	Teorinis atstumas(mm)	Nuvažiuota	Paklaida
1020	25mm	26mm	1mm
2040	50mm	50mm	0mm
4080	100mm	102mm	2mm

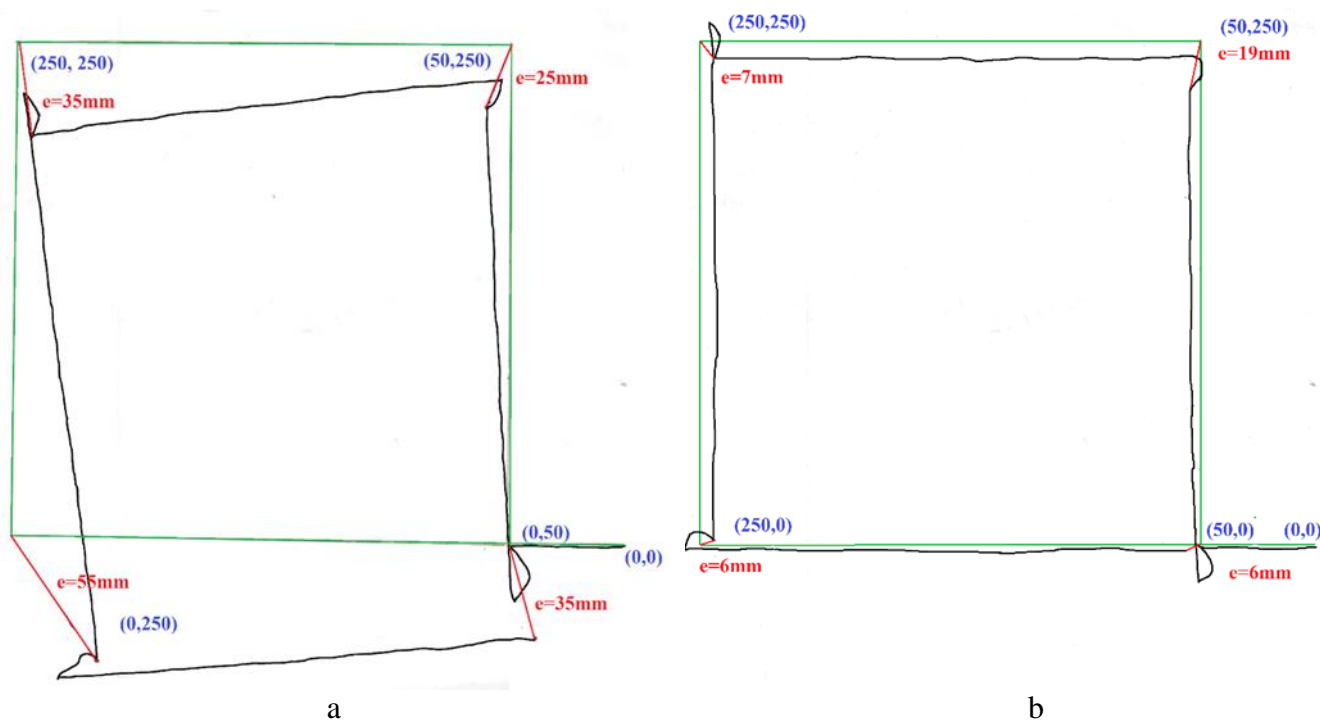
Buvo atliktas važiavimo tiesė tyrimas nupiešiant tiese, o paskui išmatuojant su slankmačiu paklaida, robotas padarė 1-2mm pozicionavimo paklaidą. Teorinis tikslumas 0,07mm buvo nepasiektas dėl reduktorių ir praslydimio paklaidų.

Atliktas kampo pasisukimo tyrimas užduodant laipsnius ir išmatuojant slankmačiu kiek truko atstumo milimetrais iki reikiamos pozicijos. Gauti rezultatai pateikti (4 lentelė.), matome kad paklaida keletą kartų didesnė lyginant su paklaida važiuojant tiese.

4 lentelė. Sukimo paklaida

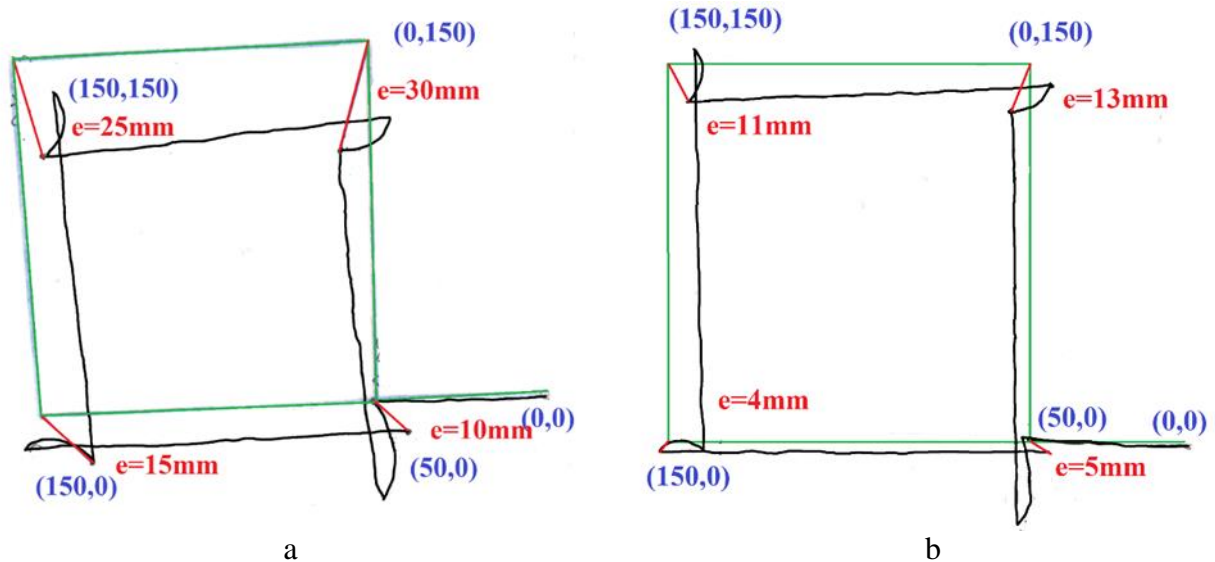
Užduotas kampas °	Pasisukta °	Paklaida [mm]
45	40	16mm
90	98	25mm
135	122	41mm

Lyginat važiavimą tiese su apskritimu (36 pav.), matome kad važiavimas tiese yra pakankamai tikslus, nes neįtakoja slydimas kuris reikalingas, kai robotas daro posūkį, arba važiuoja kreive. Pridėjus ant vikšrų guminių padelių ir valdant vien tik su poslinkių (37 pav. a) rezultatai gauti žymai geresni.

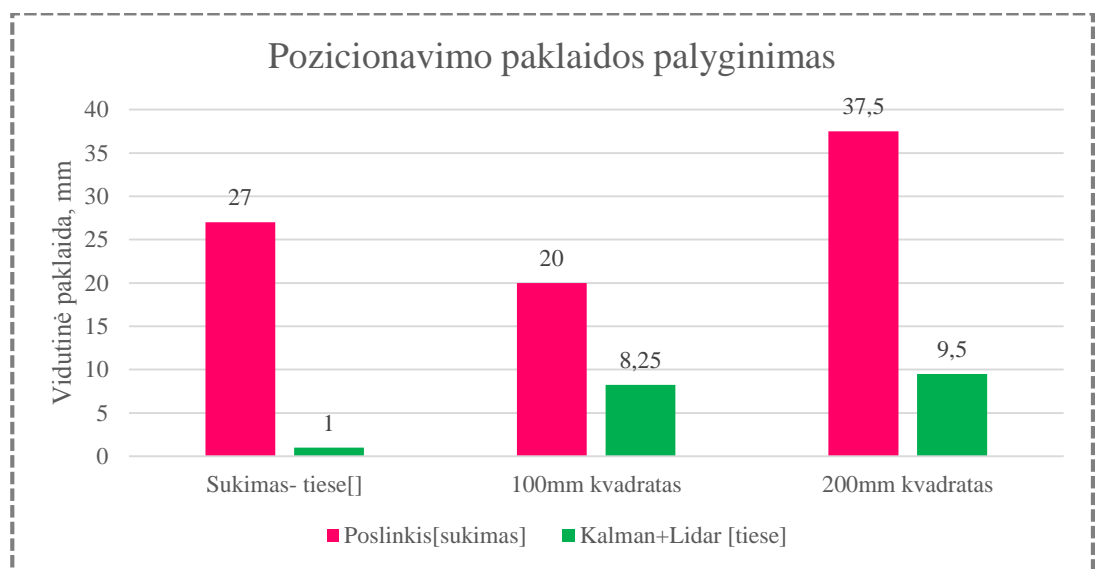


37 pav. 200mm kvadratas valdant tik poslinkiu(a) ir su Kalman filtru(b)

Grafikai sudaromi pridėdant robotui pieštuką po roboto centru ir piešiant kelia ant balto popieriaus (37 pav. ir 38 pav.). Tikslus kelias nubraižomas su liniuote ir išmatuojama paklaida kampuose. Kaip matome iš (37 pav. ir 38 pav.) roboto kelias valdant tik poslinkių(37 pav. a ir 38 pav. a) turi pakirpusias linijas kas lemia dideles poslinkio paklaidas. Taip pat matome kampuose linijų nukrypimų, mažus apskritimo brėžimus kurie atsirado dėl pieštuko pritvirtinimo centre paklaidos ir roboto sukimosi. Pritaikius Kalman filtrą su Lidar kampas ištaisomas ir paklaidos sumažinamos (37 pav. b ir 38 pav. b).



38 pav. 100mm kvadratas valdant tik poslinkiu(a) ir su Kalman filtru(b)



39 pav. Pozicionavimo paklaidos palyginimas

Lyginant gautus rezultatus pagal (39 pav.) matome, kad roboto paklaida sukantis yra 27 kartus didesnė negu važiuojant tiese, nes vikšrinė važiuoklė turi dideli slydimą sukant[19, 20]. Robotui užduodant judėjimą kvadratu be EKF iš (37 pav. 38 pav. a) matome sukimo paklaidas ir didelės pozicionavimo paklaidas. Vidutinė paklaida važiuojant 100mm kvadratu gauta 20mm, o važiuojant 200mm kvadratu 37,5mm. Pritaikius EKF paklaida sumažinama apie 60% robotui važiuojant 200mm kvadratu ir apie 75% robotui važiuojant 200mm kvadratu.

APIBENDRINIMAS IR IŠVADOS

Lazerinis atstumo matavimo skeneris (LiDar) praranda tikslumą didėjant foniniam apšvietimui, kai sniega ar lyja. Norint tiksliau pozicionuoti robotą reikia naudoti EKF ir UKF tipų filtrus.

Globali roboto pozicija išmatuojama žymiai tiksliau kai naudojamas išplėstinis Kalmano filtras, kuris apjungia matavimo iš LiDar ir ratų poslinkio skaičiavimo duomenis. Esant dideliems „Lidar“ matavimo ar poslinkio triukšmams EKF pasikliauna labiau netriukšmingesnių matavimų ir roboto poziciją nustato arčiau netriukšmingesnio. STM32F429 (180MHz) su DSP skaičiuojant matricas bei sin, cos reikšmes Kalman filtro 360 taškų suskaičiavimo per 32,5ms, o tai 2 kartus greičiau negu MATLAB su i7-840m 1,9GHz procesoriumi. STM32F429 su standartinės bibliotekos sin ir cos funkcijomis skaičiavimus atliko 3 kartus lėčiau negu naudojant DSP funkcijas.

Lyginant gautus rezultatus galime teigti, kad roboto paklaida sukantis yra 27 kartus didesnė negu važiuojant tiese, nes vikšrinė važiuoklė turi dideli slydimą sukant. Robotui užduodant judėjimą kvadratu be EKF atsiranda sukimo paklaidos ir didelės pozicionavimo paklaidos. Vidutinė paklaida važiuojant 100mm kvadratu gauta 20mm, o važiuojant 200mm kvadratu 37,5mm. Pritaikius EKF paklaida sumažinama apie 60% robotui važiuojant 200mm kvadratu ir apie 75% robotui važiuojant 200mm kvadratu.

Sudaryti Lidar duomenų nuskaitymo ir duomenų apdorojimo algoritmai, naudojantis DMA, taip neapkraunant mikroprocesoriaus. Parašytas žingsninių variklių valdymo algoritmas užduodant robotui norimas koordinatas X Y. Žingsniniai varikliai valdomi „Timer 2“ ir „Timer 3“, naudojantis DMA periodo perkėlimu, toks valdymas neapkrauna mikroprocesoriaus. Procesorius apkraunamas mažiausiai lyginant su kitais valdymo metodais. Sudarytas algoritmas integruojantis Kalman filtrą, daromoms roboto paklaidoms mažinti.

Sukonstruota eksperimentinė įranga paremta Neato XV-11 lazeriniu atstumo skeneriu, roboto pavaras sudaro žingsniniai varikliai, važiuoklė vikšrinė kuri atitinka sudarytomis Kalman filtro matematinio modelio lygtims. Suskaičiuotas vieno žingsnio poslinkis robotui lygus 0,0245mm. Robotas valdomas STM32F407 bendros paskirties ARM mikrovaldikliu. Suprojektuota plokštė gali prisijungti papildomus roboto mazgus: 2 DC variklius, 4 ultragarsinius jutiklius, 4 RC servo variklius, GPS, OFS, UART bei SPI SRAM.

LITERATŪRA

1. SIEGWART, R. and NOURBAKHSI, I.R. *Introduction to Autonomous Mobile Robots*. Cambridge, Mass.: MIT Press, 2004 ISBN 026219502X.
2. LEÓN, A., et al. *Multi-Robot SLAM and Map Merging*. , 2008.
3. RIISGAARD, S. and BLAS, M.R. *SLAM for Dummies*. , 2003 [viewed 2015-11-11]. Available from: http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf.
4. KOSTAMOVAARA, J., KURTTI, S. and JANSSON, J. A receiver–TDC Chip Set for Accurate Pulsed Time-of-Flight Laser Ranging, 2012.
5. *Scanning Laser Range Finder UTM-30LX/LN Specification*. [viewed 2016-04-25]. Available from: http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/UTM-30LX_spec_en.pdf.
6. RIISGAARD, S. and BLAS, M.R. *SLAM for Dummies. A Tutorial Approach to Simultaneous Localization and Mapping*, 2003, vol. 22, no. 1-127. pp. 126.
7. R. Havangi, M. A. Nekoui and M. Teshnehlab. *Adaptive Neuro-Fuzzy Extended Kalman Filtering for Robot Localization*. , 2010 DOI 10.1109/EPEPMC.2010.5606833.
8. KONG, F., et al. *Mobile Robot Localization Based on Extended Kalman Filter*. IEEE, 2006.
9. L. Chen, H. Hu and K. McDonald-Maier. *EKF Based Mobile Robot Localization*. , 2012 DOI 10.1109/EST.2012.19.
10. SULIMAN, C., CRUCERU, C. and MOLDOVEANU, F. Mobile Robot Position Estimation using the Kalman Filter. *Scientific Bulletin of the "Petru Maior" University of Targu Mures*, 2009, vol. 6. pp. 75.
11. JULIER, S.J. and UHLMANN, J.K. *New Extension of the Kalman Filter to Nonlinear Systems*. International Society for Optics and Photonics, 1997.
12. IVANJKO, E., KITANOV, A. and PETROVIC, I. *Model Based Kalman Filter Mobile Robot Self-Localization*. INTECH Open Access Publisher, 2010.
13. HAYKIN, S.S., HAYKIN, S.S. and HAYKIN, S.S. *Kalman Filtering and Neural Networks*. Wiley Online Library, 2001.
14. HOUSHANGI, N. and AZIZI, F. *Accurate Mobile Robot Position Determination using Unscented Kalman Filter*. IEEE, 2005.
15. KURT-YAVUZ, Z. and YAVUZ, S. *A Comparison of EKF, UKF, FastSLAM2. 0, and UKF-Based FastSLAM Algorithms*. IEEE, 2012 Available from: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6249866&abstractAccess=no&userType=inst.

16. *Standard PM Step Motors PM55L-048- HHD0*. [viewed 2016-04-25]. Available from: <http://www.nmbtc.com/pdf/motors/PM55L-048-HHD0.pdf>.
17. *Hacking the Neato XV-11*. Wikispaces: [viewed 2016-01-15]. Available from: <https://xv11hacking.wikispaces.com/>.
18. Driving Bipolar Stepper Motors using a Medium-Density STM32F103xx Microcontroller [viewed 2016-04-25]. Available from: http://www.st.com/st-web-ui/static/active/jp/resource/technical/document/application_note/CD00207733.pdf.
19. IOSSAQUI, J.G., CAMINO, J.F. and ZAMPIERI, D.E. *A Nonlinear Control Design for Tracked Robots with Longitudinal Slip*. , 2011.
20. FAN, Z., BORENSTEIN, J., WEHE, D. and KOREN, Y. *Experimental Evaluation of an Encoder Trailer for Dead-Reckoning in Tracked Mobile Robots*. , 1995.

PRIEDAI

1 PRIEDAS. STM32F407 PROGRAMOS KODAS

1.1 Kalman.c programos kodas

```
/* Includes -----*/
#include "main.h"
#include "arm_math.h"
#include "stdio.h"
#include "stdlib.h"

//kintamieji matavimo
float32_t skirx,skiry,skirxk,skiryk;
float32_t zz[2];

//kintamieji
float32_t xt[3] =
{100, 0, 0};

float32_t xt2[3] =
{100, 0, 0};

float32_t PP_f32[9] =
{ // Const, numTaps, blockSize, numTaps*blockSize
  0, 0, 0,
  0, 0, 0,
  0, 0, 0, };
float32_t zyme_f32[2] =
{100,1050,};

float32_t R_f32[4] =
{200, 0,
 0, 0.5};
float32_t Q_f32[4] =
{0, 0,
 0, 0};

float32_t XKalman[361];
float32_t YKalman[361];
float32_t kamKalman[361];
float32_t nnn;
/* -----
 * Temporary buffers for storing intermediate values
 * ----- */
//float32_t Xmat[361];
//float32_t Ymat[361];
//int i=0;
//int j=0;
//matricu nustatimai

void Kalman_Init(void)
{
//spejimo matricos
uint32_t srcRows, srcColumns; /* Temporary variables */
srcRows = 3;
srcColumns = 3;
arm_mat_init_f32(&Fx, srcRows, srcColumns, Fx_f32);
arm_mat_init_f32(&FxT, srcRows, srcColumns, FxT_f32);
arm_mat_init_f32(&P, srcRows, srcColumns, P_f32);
arm_mat_init_f32(&PP, srcRows, srcColumns, PP_f32);
arm_mat_init_f32(&BB, srcRows, srcColumns, BB_f32);
arm_mat_init_f32(&A, srcRows, srcColumns, A_f32);
arm_mat_init_f32(&AA, srcRows, srcColumns, AA_f32);
arm_mat_init_f32(&WW, srcRows, srcColumns, WW_f32);
```

```

srcRows = 3;
srcColumns = 2;
arm_mat_init_f32(&Fu, srcRows, srcColumns, Fu_f32);
arm_mat_init_f32(&B, srcRows, srcColumns, B_f32);
//matavimo matrica
arm_mat_init_f32(&HT, srcRows, srcColumns, HT_f32);
arm_mat_init_f32(&L, srcRows, srcColumns, L_f32);
arm_mat_init_f32(&K, srcRows, srcColumns, K_f32);
arm_mat_init_f32(&W, srcRows, srcColumns, W_f32);

srcRows = 2;
srcColumns = 3;
arm_mat_init_f32(&FuT, srcRows, srcColumns, FuT_f32);
//matavimo matricos
arm_mat_init_f32(&H, srcRows, srcColumns, H_f32);
arm_mat_init_f32(&N, srcRows, srcColumns, N_f32);
arm_mat_init_f32(&KT, srcRows, srcColumns, KT_f32);

srcRows = 2;
srcColumns = 2;
arm_mat_init_f32(&Q, srcRows, srcColumns, Q_f32);
//matavimo matricos
arm_mat_init_f32(&V, srcRows, srcColumns, V_f32);
arm_mat_init_f32(&VTr, srcRows, srcColumns, VTr_f32);
arm_mat_init_f32(&M, srcRows, srcColumns, M_f32);
arm_mat_init_f32(&MM, srcRows, srcColumns, MM_f32);
arm_mat_init_f32(&NN, srcRows, srcColumns, NN_f32);
arm_mat_init_f32(&R, srcRows, srcColumns, R_f32);
arm_mat_init_f32(&S, srcRows, srcColumns, S_f32);
arm_mat_init_f32(&SS, srcRows, srcColumns, SS_f32);
arm_mat_init_f32(&SINV, srcRows, srcColumns, SINV_f32);
//matavimo matricos
srcRows = 3;
srcColumns = 1;
arm_mat_init_f32(&J, srcRows, srcColumns, J_f32);
srcRows = 2;
srcColumns = 1;
arm_mat_init_f32(&vt, srcRows, srcColumns, vt_f32);

xt2[0]=0;
xt2[1]=0;
xt2[2]=1.5708;
//k=0;
}

void Kalman_calc(int32_t Urs, int32_t Uls, uint32_t lidar_l, uint32_t lidar_kampas)
{
Q_f32[0]=Urs;
Q_f32[3]=Uls;
//spejimas
/* xt2- matrica roboto busenos t-1 momentu (x,y,kam) */
/* Pt2- klaidos koreliacijos matrica t-1 momentu */
/* u- valdimas sr ir sl */
/* Q- valdimo klaidos matrica */
/* roboto tarp ratu atstumas {mm} */
trikst=(Urs+Uls)/2;
trikkam=(Urs-Uls)/b;

xt[0]=xt2[0]+ trikst*cosf(xt2[2]+trikkam);
xt[1] = xt2[1] + trikst*sinf(xt2[2]+trikkam);
xt[2] = xt2[2] + trikkam;

Fx_f32[0] = 1.0;
Fx_f32[1] = 0.0;
Fx_f32[2] = -trikst * sinf(xt2[2] + trikkam );

Fx_f32[3] = 0.0;
Fx_f32[4] = 1.0;
Fx_f32[5] = trikst * cosf(xt2[2] + trikkam );

Fx_f32[6] = 0.0;

```

```

Fx_f32[7] = 0.0;
Fx_f32[8] = 1.0;

Fu_f32[0] = 0.5 * cosf(xt2[2] + trikkam );
Fu_f32[1] = 0.5 * sinf(xt2[2] + trikkam );
Fu_f32[2] = 0.5 * cosf(xt2[2] + trikkam );
Fu_f32[3] = 0.5 * sinf(xt2[2] + trikkam );
Fu_f32[4] = 1/b;
Fu_f32[5] = 1/b;

//FxT=Fx'
arm_mat_trans_f32(&Fx, &FxT);
//FuT=Fu'
arm_mat_trans_f32(&Fu, &FuT);
//A=Fx*PP
arm_mat_mult_f32(&Fx, &PP, &A);
//AA=Fx'*A
arm_mat_mult_f32(&A, &FxT, &AA);
//B=Fu*Q
arm_mat_mult_f32(&Fu, &Q, &B);
//BB=Fu'*B
arm_mat_mult_f32(&B, &FuT, &BB);
//P=Fx*Pt2*Fx'+Fu*Q*Fu';
//P=AA+BB;
arm_mat_add_f32(&AA, &BB, &P);

// matavimas
skirx=-zyme_f32[0]+xt[0];
skiry=-zyme_f32[1]+xt[1];
skirxk=skirx*skirx;
skiryk=skiry*skiry;
nnn=skiry/skirx;

zz[0]=sqrt(skirxk+skiryk);
zz[1]=atan(nnn)-xt[2];
//zz[1]=0;

H_f32[0]=2*skirx/(2*sqrt(skirxk + skiryk));
H_f32[1]=2*skiry/(2*sqrt(skirxk + skiryk));
H_f32[2]=0;
H_f32[3]=skiry/(skirxk*((skiryk/skirxk)+1));
H_f32[4]=1/(skirxk*((skiryk/skirxk)+1));
H_f32[5]=-1;

V_f32[0]=1;
V_f32[1]=0;
V_f32[2]=0;
V_f32[3]=1;

//HT=H'
arm_mat_trans_f32(&H, &HT);
//VT=V'
arm_mat_trans_f32(&V, &VTr);
//N=H*P
arm_mat_mult_f32(&H, &P, &N);
//NN=N*H'
arm_mat_mult_f32(&N, &HT, &NN);
//M=V*R
arm_mat_mult_f32(&V, &R, &M);
//MM=M*V'
arm_mat_mult_f32(&M, &VTr, &MM);
//S=NN+MM;
arm_mat_add_f32(&MM, &NN, &S);
arm_mat_add_f32(&MM, &NN, &SS);

vt_f32[0]=lidar_l-zz[0];
vt_f32[1]=lidar_kampas-zz[1];

//L=P*H'
arm_mat_mult_f32(&P, &HT, &L);
//inv(S)

```

```

arm_mat_inverse_f32(&S, &SINV);
//K=L*Sinv
arm_mat_mult_f32(&L, &SINV, &K);

//J=K*vt;
arm_mat_mult_f32(&K, &vt, &J);

xxx[0]=xt[0]+J_f32[0];
XKalman[kk]=xxx[0];
xxx[1]=xt[1]+J_f32[1];
YKalman[kk]=xxx[1];
xxx[2]=xt[2]+J_f32[2];
kamKalman[kk]=xxx[2];

xt2[0]=xt[0];
xt2[1]=xt[1];
xt2[2]=xt[2];

//K'
arm_mat_trans_f32(&K, &KT);
//W=K*S
arm_mat_mult_f32(&K, &SS, &W);
//WW=Q*K'
arm_mat_mult_f32(&W, &KT, &WW);
//PP=P-QQ
arm_mat_sub_f32(&P, &WW, &PP);

kk++;
}
//

float get_kalman_X(void)
{return xt[0];}

float get_kalman_Y(void)
{return xt[1];}

float get_kalman_kamp(void)
{return xt[2];}

```

1.2 Lidar.c programos kodas

```

#include "main.h"
#include "Lidar.h"
#include "stm32f4xx_it.h"
#include "debuguart.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include "Kalman.h"
//timerio 9 struktura PE6 PWM isejimas
TIM_HandleTypeDef htim9;
TIM_OC_InitTypeDef hOC9;
GPIO_InitTypeDef GPIO_E_st;
UART_HandleTypeDef Uart_lidar;

//lidaras
#define XV11_PACKAGE_LENGTH 22
#define XV11_START_BYTE 0xFA

uint8_t XV11_Fpaketas[2100];
uint8_t XV11_Fpaketas2[2100];
uint8_t XV11_Package[22];
uint16_t Distance[360];
uint16_t PackageChecksum(uint8_t * packagePointer);
//const float RAD = (M_TWOPI / 360.0f);
uint16_t deg;
float radians, displayDistance;

```

```

uint16_t x[360], y[360];
uint16_t degSelected[2];
uint8_t pointSelect = 0;
uint8_t syn_indeksas=22;
bool flag_nepraleisti=true;
bool FA_rado=true;
bool flag_syms=true;
bool flag_can_parse=false;
bool flag_get_min_object=true;
extern DMA_HandleTypeDef hdma_tx;
extern DMA_HandleTypeDef hdma_rx;

extern int32_t Ur_pos, Ul_pos;
void Lidar_init(void)
{
    /*##-1- paleidziamė periferijos clk #####*/
    /* Enable GPIOE Channels Clock */
    GPIOE_CLK_ENABLE();
    GPIO_E_st.Mode = GPIO_MODE_AF_PP;
    GPIO_E_st.Pull = GPIO_PULLUP;
    GPIO_E_st.Speed = GPIO_SPEED_HIGH;
    GPIO_E_st.Alternate = GPIO_AF3_TIM9;
    GPIO_E_st.Pin = TIM_9_GPIO_PIN_CHANNEL2;
    HAL_GPIO_Init(TIM_9_GPIO_CHANNEL2_PORT, &GPIO_E_st);

    //sukonfiguruojuame pwm
    htim9.Instance = TIM_9;
    htim9.Init.Prescaler = MOTOR_PWM_PRESCALER;
    //prescaiu kokiu greiciu countins 7/84/ =50khz
    htim9.Init.Period = MOTOR_PWM_PERIOD; // Top reiksme 237hz
    htim9.Init.ClockDivision = 0;
    htim9.Init.CounterMode = TIM_COUNTERMODE_UP;
    if(HAL_TIM_PWM_Init(&htim9) != HAL_OK)
    {Error_Handler(); }

    hOC9.OCMode = TIM_OCMode_PWM1;
    hOC9.OCpolarity = TIM_OCPolarity_HIGH;
    hOC9.OCFastMode = TIM_OCFAST_DISABLE;
    hOC9.Pulse = 400;
    if(HAL_TIM_PWM_ConfigChannel(&htim9, &hOC9, TIM_CHANNEL_2) != HAL_OK)
    {Error_Handler();}
    /* paleidziamė 2 kanala */
    if(HAL_TIM_PWM_Start(&htim9, TIM_CHANNEL_2) != HAL_OK)
    { Error_Handler();}
}
//
void Lidar_uart_init(void)
{
    //uart innitinimas sukonfiguravimas
    Uart_lidar.Instance = USARTx;
    Uart_lidar.Init.BaudRate = 115200;
    Uart_lidar.Init.WordLength = UART_WORDLENGTH_8B;
    Uart_lidar.Init.StopBits = UART_STOPBITS_1;
    Uart_lidar.Init.Parity = UART_PARITY_NONE;
    Uart_lidar.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    Uart_lidar.Init.Mode = UART_MODE_RX;
    Uart_lidar.Init.OverSampling = UART_OVERSAMPLING_16;
    if(HAL_UART_Init(&Uart_lidar) != HAL_OK)
    {Error_Handler();}

    //Isvalome buferi
    __HAL_UART_FLUSH_DRREGISTER(&Uart_lidar);

    //Paleidziamė duomenų kaupimą
    if(HAL_UART_Receive_DMA(&Uart_lidar, (uint8_t *)XV11_Fpaketas, 2000) != HAL_OK)
    {Error_Handler();}
}
//0-100
void Lidar_motor_power(uint16_t power)
{
    //nusatatomė lidar galvos sukimosi greiti

```

```

        power=(power*164)>>4;
        if (power>=MOTOR_PWM_PERIOD) {power=MOTOR_PWM_PERIOD;}
        __HAL_TIM_SET_COMPARE(&htim9, TIM_CHANNEL_2, power);
    }

uint16_t Speed;
uint16_t GoodReadings = 0, BadReadings = 0;
uint16_t AnglesCovered = 0;
void ParsePackage(uint8_t * packagePointer)
{
    uint16_t i;
    uint16_t Index;
    uint8_t InvalidFlag[4];
    uint8_t WarningFlag[4];
    uint16_t Checksum, ChecksumCalculated;
    Checksum = ((uint16_t)packagePointer[21] << 8) | packagePointer[20];
    ChecksumCalculated=PackageChecksum(packagePointer);
    if (Checksum != ChecksumCalculated) {BadReadings += 4;}

    Index = (packagePointer[1] - 0xA0) * 4;
    Speed = ((uint16_t)packagePointer[3] << 8) | packagePointer[2];
    InvalidFlag[0] = (packagePointer[5] & 0x80) >> 7;
    InvalidFlag[1] = (packagePointer[9] & 0x80) >> 7;
    InvalidFlag[2] = (packagePointer[13] & 0x80) >> 7;
    InvalidFlag[3] = (packagePointer[17] & 0x80) >> 7;
    WarningFlag[0] = (packagePointer[5] & 0x40) >> 6;
    WarningFlag[1] = (packagePointer[9] & 0x40) >> 6;
    WarningFlag[2] = (packagePointer[13] & 0x40) >> 6;
    WarningFlag[3] = (packagePointer[17] & 0x40) >> 6;

    if (Index == 0) {AnglesCovered = 0;
    for (i = 0; i < 360; i++) {
        if (Distance[i] > 0) AnglesCovered++;
    }
    GoodReadings = 0;
    BadReadings = 0;
    }

    for (i = 0; i < 4; i++) {
        if (!InvalidFlag[i])
        {
            Distance[Index+i] = packagePointer[4+(i*4)] |
            ((uint16_t)packagePointer[5+(i*4)] & 0x3F) << 8);
            GoodReadings++;
        } else {
            Distance[Index+i] = 0;
            BadReadings++;
        }
    }
}

uint16_t PackageChecksum(uint8_t * packagePointer)
{
    uint8_t i;
    uint16_t data[10];
    uint16_t checksum;
    uint32_t chk32;

    // group the data by word, little-endian
    for (i = 0; i < 10; i++) {
        data[i] = packagePointer[2*i] | (((uint16_t)packagePointer[2*i+1]) << 8);}

    // compute the checksum on 32 bits
    chk32 = 0;
    for (i = 0; i < 10; i++) {
        chk32 = (chk32 << 1) + data[i];
    }
    // return a value wrapped around on 15bits, and truncated to still fit into 15 bits
    checksum = (chk32 & 0x7FFF) + ( chk32 >> 15 ); // wrap around to fit into 15
bits
    return checksum = checksum & 0x7FFF; // truncate to 15 bits
}

```

```

}

uint16_t k;
uint8_t j;
void Lidar_handleris(void)
{
    k=0;
    j=0;
    //isparsinam visa pilna paketa
    //while (k < 2000) {
    for (k=0;k<2000;k++) {
if (XV11_Fpaketas2[k] == XV11_START_BYTE) {
    //syn_indeksas=k;
    XV11_Package[0]=0xFA;
    XV11_Package[1]=XV11_Fpaketas2[k+1];
    XV11_Package[2]=XV11_Fpaketas2[k+2];
    XV11_Package[3]=XV11_Fpaketas2[k+3];
    XV11_Package[4]=XV11_Fpaketas2[k+4];
    XV11_Package[5]=XV11_Fpaketas2[k+5];
    XV11_Package[6]=XV11_Fpaketas2[k+6];
    XV11_Package[7]=XV11_Fpaketas2[k+7];
    XV11_Package[8]=XV11_Fpaketas2[k+8];
    XV11_Package[9]=XV11_Fpaketas2[k+9];
    XV11_Package[10]=XV11_Fpaketas2[k+10];
    XV11_Package[11]=XV11_Fpaketas2[k+11];
    XV11_Package[12]=XV11_Fpaketas2[k+12];
    XV11_Package[13]=XV11_Fpaketas2[k+13];
    XV11_Package[14]=XV11_Fpaketas2[k+14];
    XV11_Package[15]=XV11_Fpaketas2[k+15];
    XV11_Package[16]=XV11_Fpaketas2[k+16];
    XV11_Package[17]=XV11_Fpaketas2[k+17];
    XV11_Package[18]=XV11_Fpaketas2[k+18];
    XV11_Package[19]=XV11_Fpaketas2[k+19];
    XV11_Package[20]=XV11_Fpaketas2[k+20];
    XV11_Package[21]=XV11_Fpaketas2[k+21];
}
if ((k+21)<2000) {ParsePackage(XV11_Package);}
k+=21;}
}
flag_can_parse=false;
}

void packetrewrite(void)
{
    if (flag_can_parse==false)
    {k=0;
    for (k=0;k<2000;k++) {
    XV11_Fpaketas2[k]=XV11_Fpaketas[k];
    }
    flag_can_parse=true;
    flag_get_min_object=true;
}
}

bool get_flag_can_parse(void)
{
    return flag_can_parse;
}

bool get_flag_find_min_object(void)
{
    return flag_get_min_object;
}

uint16_t minimumas;
uint16_t minimumo_kampas;
float min_kampas_rad;
float rad_todeg=0.017453;
float Xzym,Yzym;
float esamas_kampas=1.5708;

void objekto_find_handleris(void)

```



```

{
    uint16_t kampas;
    minimumas=0xFFFF;
    for (kampas = 0; kamps < 360; kamps++) {
        //min paieska atmetan 0;
        if (Distance[kampas] > 0)
        {
            if (Distance[kampas]<minimumas)
            {
                minimumas=Distance[kampas];
                minimumo_kampas=kampas;
            }
        }
        minimumo_kampas+=80;
        if (minimumo_kampas>=360)
        {minimumo_kampas-=360;}
        min_kampas_rad=(float)minimumo_kampas*rad_todeg;
        //jeigu nera zymes ja pridedame ir islaikome
        if (flag_nera_zymes)
        {
            flag_nera_zymes=false;
            Yzym=minimumas*sinf(min_kampas_rad);
            Xzym=minimumas*cosf(min_kampas_rad);
            flag_get_min_object=false;}else {
            //kvieciame kalman filtra ir ji apdorojame
            Kalman_calc(Ur_pos, Ul_pos, minimumas,minimumo_kampas);
        }
    }
}

```

1.3 Zingsniai_varikliai.c programos kodas

```

#include "main.h"
//#include "stm32f4xx_hal.h"
#include "zingsniniai_varikliai.h"
//#include "arm_math.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#include "debuguart.h"
#include "Kalman.h"

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
/* Timer Output Compare Configuration Structure declaration */
TIM_OC_InitTypeDef hOC2; //output compare timer2
TIM_OC_InitTypeDef hOC3; //output compare timer3

/* zingzniu buferis timer2 */ //eight
static uint32_t zingsniu_buferisR[buffer_dydis] = {600,600,600,600}; //vazuoja
500 apie 19cm/s
static uint32_t zingsniu_buferisL[buffer_dydis] = {600,600,600,600};
//testas2
uint32_t X_bufferis[200] = {0,200, 200, 0 , 0};
uint32_t Y_bufferis[200] = {50,50, 250, 250, 50};
uint8_t XX_vyg=5;

//dma struktura timer2
DMA_HandleTypeDef hdma_tim2;
//dma struktura timer3
DMA_HandleTypeDef hdma_tim3;

//isejimu struktura
static GPIO_InitTypeDef GPIO_D_st;
static GPIO_InitTypeDef GPIO_B_st;
//kokiu rezimu nustatyti
//SET_ZV_QUATER_STEP
static uint8_t STEPER_VARIKLIU_REZOLIUCIJA=SET_ZV_EIGHT_STEP;
//static uint8_t STEPER_VARIKLIU_REZOLIUCIJA=SET_ZV_QUATER_STEP;
int32_t X=0,Y=0;

```

```

float kampas=1.570;
float glob_kampas, trik_kampas;
int32_t Ur_pos, Ul_pos;
int32_t Ur_pos2, Ul_pos2;
//float Ur_pos, Ul_pos;
int32_t Xrobot, Yrobot;
uint16_t xx=0;
uint32_t Ur_zingsniai=0, Ul_zingsniai=0;
uint32_t Ur_zingsniai2=0, Ul_zingsniai2=0;

bool flag_nevygdoma=true;
uint16_t i=0;
uint16_t stepR_vyg=0, stepL_vyg=0;
uint16_t stepR_vyg2=0, stepL_vyg2=0;
float stepR_ms, stepL_ms; //milisekundes kiek nuo vieno prie kito zingnio
float stepR_ms2, stepL_ms2; //milisekundes kiek nuo vieno prie kito zingnio
uint32_t stepR_msu, stepL_msu; //milisekundes kiek nuo vieno prie kito zingnio
uint32_t stepR_ms2u, stepL_ms2u; //milisekundes kiek nuo vieno prie kito zingnio
uint8_t zingsniu_greitis=255; //150
uint8_t status_varikliai=ST_SKAICIUOJAMA;

uint32_t DEAD_TIME[20]={20000,20000,20000,20000,20000,20000,20000,20000,20000,20000,20000,20000,20000,20000,20000,20000,20000,20000,20000,20000};
bool flag_kampas;

void stepper_init(void){
zv_gpio_init();
//initinam timer2 OC kanala 1
htim2.Instance = TIM_2;
htim2.Init.Period = 0xFFFF;
htim2.Init.RepetitionCounter = 0;
htim2.Init.Prescaler = TIMER_REZOLIUCIJA;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1 ;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;

if(HAL_TIM_OC_Init(&htim2) != HAL_OK)
{Error_Handler();}

hOC2.OCMode = TIM_OCMODE_TOGGLE;
hOC2.OCPolarity = TIM_OCPOLARITY_HIGH;
hOC2.Pulse = 0x0;
if(HAL_TIM_OC_ConfigChannel(&htim2, &hOC2, TIM_CHANNEL_3) != HAL_OK)
{ Error_Handler();}
/*##-3- Start PWM signal generation in DMA mode #####*/
if( HAL_TIM_OC_Start(&htim2, TIM_CHANNEL_3) != HAL_OK)
{ Error_Handler();}

//initinam timer3 kanala 4
htim3.Instance = TIM_3;
htim3.Init.Period = 0xFFFF;
htim3.Init.RepetitionCounter = 0;
htim3.Init.Prescaler = TIMER_REZOLIUCIJA; //nustatomas 1mhz
htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
if(HAL_TIM_OC_Init(&htim3) != HAL_OK)
{Error_Handler();}
hOC3.OCMode = TIM_OCMODE_TOGGLE;
hOC3.OCPolarity = TIM_OCPOLARITY_HIGH;
hOC3.Pulse = 0x0;

if(HAL_TIM_OC_ConfigChannel(&htim3, &hOC3, TIM_CHANNEL_4) != HAL_OK)
{Error_Handler();}

if( HAL_TIM_OC_Start(&htim3, TIM_CHANNEL_4) != HAL_OK)
{Error_Handler();}
//pradedame ARR reiksmes perkelimus is DMA
HAL_TIM_Base_Start_DMA(&htim2, (uint32_t*)zingsniu_buferisR, 4);
HAL_TIM_Base_Start_DMA(&htim3, (uint32_t*)zingsniu_buferisL, 4);
}
//

```

```

float trik_kampas_deg;
uint16_t corect_zings;
//dma interuptas nauji skaiciavimai ir apdorojimas
void interuptas_set_next_st(void)
{
    switch (status_varikliai)
    {
        case ST_SKAICIUOJAMA:
            zv_disable();
            set_steps_is_XY(X_bufferis[xx], Y_bufferis[xx]);
            //zingsniu skaiciavimas
            //jeigu poslingis maziau negu nulis keiciam dir
            if (Ur_pos<0) {
                //sukimas
                set_R_dir_Rev();
            }
            else {
                //zingsniu greitis=190;
                set_R_dir_For();
            }

            if (Ul_pos<0) {set_L_dir_Rev();}
            else {set_L_dir_For();}
            if (Ul_pos<0 | Ur_pos<0)
            {zingsniu_greitis=150;
              flag_kampas=true; }
            else
            {zingsniu_greitis=255;
              flag_kampas=false;
            }

            Ur_zingsniai=abs(Ur_pos)*ZINGSNIU_MM_MAZ;
            Ul_zingsniai=abs(Ul_pos)*ZINGSNIU_MM_MAZ;
            set_steps(Ur_zingsniai, Ul_zingsniai,zingsniu_greitis);
            stepR_vyg=150000/((uint16_t)stepR_ms); //kiek zingniu ivygydyti vienu 150ms periodu
            stepL_vyg=150000/((uint16_t)stepL_ms); //kiek zingniu ivygydyti vienu 150ms periodu
            Ur_zingsniai2=abs(Ur_pos2)*ZINGSNIU_MM;
            Ul_zingsniai2=abs(Ul_pos2)*ZINGSNIU_MM;
            zingsniu_greitis=255;
            set_steps2(Ur_zingsniai2, Ul_zingsniai2,zingsniu_greitis);
            stepR_vyg2=150000/((uint16_t)stepR_ms2);//kiek zingniu ivygydyti vienu 250ms periodu
            stepL_vyg2=150000/((uint16_t)stepL_ms2);//kiek zingniu ivygydyti vienu 250ms periodu

            xx++;
            flag_nevygdoma=false;
            set_status(ST_VAZIUOJAMA1);
            interuptas_set_next_st();

            break;

            case ST_VAZIUOJAMA1:
                //enablinam variklius
                zv_enable();
                flag_nevygdoma=false;
                //sudedam buverius
                if (Ur_zingsniai>0)
                {
                    if (Ur_zingsniai>stepR_vyg)
                    {
                        for (i=0;i<stepR_vyg;i++){zingsniu_buferisR[2*i]=stepR_msu;
                        zingsniu_buferisR[2*i+1]=stepR_msu;
                        }
                        for (i=0;i<stepL_vyg;i++){zingsniu_buferisL[2*i]=stepL_msu;
                        zingsniu_buferisL[2*i+1]=stepL_msu;
                        }
                        Ur_zingsniai-=stepR_vyg;
                        Ul_zingsniai-=stepL_vyg;

                        HAL_TIM_Base_Start_DMA(&htim2, (uint32_t*)zingsniu_buferisR, 2*stepR_vyg);
                        HAL_TIM_Base_Start_DMA(&htim3, (uint32_t*)zingsniu_buferisL, 2*stepL_vyg);
                    }
                }
    }
}

```

```

    }

    else
    {
        For(i=0;i<Ur_zingsniai;i++){zingsniu_buferisR[2*i]=stepR_msu;
zingsniu_buferisR[2*i+1]=stepR_msu;}
        for (i=0;i<Ul_zingsniai;i++){zingsniu_buferisL[2*i]=stepL_msu;
zingsniu_buferisL[2*i+1]=stepL_msu; }

        HAL_TIM_Base_Start_DMA(&htim2, (uint32_t*)zingsniu_buferisR, 2*Ur_zingsniai);
        HAL_TIM_Base_Start_DMA(&htim3, (uint32_t*)zingsniu_buferisL, 2*Ul_zingsniai);
        Ur_zingsniai=0;
        Ul_zingsniai=0;

    }

}

else
{set_status(ST_DEAT_TIME_KALMAN);
interuptas_set_next_st();
}
break;

case ST_VAZIUOJAMA2:
    zv_enable();
if (Ur_zingsniai2>0){
    if (Ur_pos2<0) {set_R_dir_Rev();}
    else {set_R_dir_For();}

    if (Ul_pos2<0) {set_L_dir_Rev();}
    else {set_L_dir_For();}

if (Ur_zingsniai2>stepR_vyg2)
    {
for (i=0;i<stepR_vyg2;i++){zingsniu_buferisR[2*i]=stepR_ms2u;
zingsniu_buferisR[2*i+1]=stepR_ms2u; }
for (i=0;i<stepL_vyg2;i++){zingsniu_buferisL[2*i]=stepL_ms2u;
zingsniu_buferisL[2*i+1]=stepL_ms2u; }

        HAL_TIM_Base_Start_DMA(&htim2, (uint32_t*)zingsniu_buferisR, 2*stepR_vyg2);
        HAL_TIM_Base_Start_DMA(&htim3, (uint32_t*)zingsniu_buferisL, 2*stepL_vyg2);
        Ur_zingsniai2-=stepR_vyg2;
        Ul_zingsniai2-=stepL_vyg2;

    }

    else
    {
for (i=0;i<Ur_zingsniai2;i++){zingsniu_buferisR[2*i]=stepR_ms2u;
zingsniu_buferisR[2*i+1]=stepR_ms2u; }
for (i=0;i<Ul_zingsniai2;i++){zingsniu_buferisL[2*i]=stepL_ms2u;
zingsniu_buferisL[2*i+1]=stepL_ms2u; }
        HAL_TIM_Base_Start_DMA(&htim2, (uint32_t*)zingsniu_buferisR, 2*Ur_zingsniai2);
        HAL_TIM_Base_Start_DMA(&htim3, (uint32_t*)zingsniu_buferisL, 2*Ul_zingsniai2);
        Ur_zingsniai2=0;
        Ul_zingsniai2=0;

    }
}
else
{
flag_kampas=false;
set_status(ST_DEAT_TIME_KALMAN);
interuptas_set_next_st();
}

break;

case ST_STOP_NEXT_CALC:
flag_nevygdoma=true;
zv_disable();
if (xx<XX_vyg) {set_status(ST_SKAICIUOJAMA);}

```

```

else {set_status(ST_STOP_ALL);}
interuptas_set_next_st();
break;

case ST_STOP_ALL:
zv_disable();
break;

case ST_DEAD_TIME1:
zv_disable();
set_status(ST_VAZIUOJAMA2);
HAL_TIM_Base_Start_DMA(&htim2, (uint32_t*)DEAD_TIME, 20);
HAL_TIM_Base_Start_DMA(&htim3, (uint32_t*)DEAD_TIME, 20);
break;

case ST_DEAD_TIME2:
zv_disable();
set_status(ST_STOP_NEXT_CALC);
HAL_TIM_Base_Start_DMA(&htim2, (uint32_t*)DEAD_TIME, 20);
HAL_TIM_Base_Start_DMA(&htim3, (uint32_t*)DEAD_TIME, 20);
break;

case ST_CORRECT_KAMPAS:
    zv_enable();
    if (corect_zings>0)
    {
        if (corect_zings>stepR_vyg)
        {
            for (i=0;i<stepR_vyg;i++){zingsniu_buferisR[2*i]=stepR_msu;
zingsniu_buferisR[2*i+1]=stepR_msu; }
            for (i=0;i<stepL_vyg;i++){zingsniu_buferisL[2*i]=stepL_msu;
zingsniu_buferisL[2*i+1]=stepL_msu; }
            corect_zings-=stepR_vyg;
            HAL_TIM_Base_Start_DMA(&htim2, (uint32_t*)zingsniu_buferisR, 2*stepR_vyg);
            HAL_TIM_Base_Start_DMA(&htim3, (uint32_t*)zingsniu_buferisL, 2*stepL_vyg);

        }
        else
        {
            for (i=0;i<corect_zings;i++){zingsniu_buferisR[2*i]=stepR_msu;
zingsniu_buferisR[2*i+1]=stepR_msu; }
            for (i=0;i<corect_zings;i++){zingsniu_buferisL[2*i]=stepL_msu;
zingsniu_buferisL[2*i+1]=stepL_msu; }
            HAL_TIM_Base_Start_DMA(&htim2, (uint32_t*)zingsniu_buferisR, 2*corect_zings);
            HAL_TIM_Base_Start_DMA(&htim3, (uint32_t*)zingsniu_buferisL, 2*corect_zings);
            corect_zings=0;
            corect_zings=0;
        }
    }
else
{
set_status(ST_DEAD_TIME1);
interuptas_set_next_st();}
break;

case ST_CORRECT_TIESE:

break;

case ST_CHECK_KAMPAS:
//tikrinam trik kampas ir ismatuota kampa su KALMAN
set_status(ST_CORRECT_KAMPAS);
PRINTF("check kampas\r\n");
trik_kampas_deg=(trik_kampas*180)/3.1416;
PRINTF("%f\r\n",trik_kampas_deg);
kalman_deg_e=trik_kampas_deg-get_kalman_kamp();
corect_zings=(uint16_t)(abs(kalman_deg_e)*3.15)*ZINGSNIU_MM;
//corect_zings=0;
//jeigu kampas minusas valdom viena puse jei pliusas kita
if (kalman_deg_e>0)

```

```

{set_R_dir_For();
set_L_dir_Rev();}
else
{//atimti
set_R_dir_Rev();
set_L_dir_For();}
interuptas_set_next_st();
break;

case ST_CHECK_TIESE:
set_status(ST_DEAD_TIME2);
PRINTF("check tiesė\r\n");
if (Ur_pos2!=0){PRINTF("poslinkis %i %i \r\n",Ur_pos2,Ul_pos2);}
else{PRINTF("poslinkis %i %i\r\n",Ur_pos,Ul_pos);}
interuptas_set_next_st();
break;

case ST_DEAT_TIME_KALMAN:
zv_disable();
if (flag_kampas) {set_status(ST_CHECK_KAMPAS); interuptas_set_next_st();}
else {set_status(ST_CHECK_TIESE);}
HAL_TIM_Base_Start_DMA(&htim2, (uint32_t*)DEAD_TIME, 20);
HAL_TIM_Base_Start_DMA(&htim3, (uint32_t*)DEAD_TIME, 20);
break;

default:
break;
}
}

void zv_gpio_init(void)
{
    //enable isejimus D ir B portai
    GPIOD_CLK_ENABLE();
    GPIOB_CLK_ENABLE();
    //visi ant porto D
    GPIO_D_st.Pin = (R_VARIKLIS_MS1_GPIO_PIN | L_VARIKLIS_MS1_GPIO_PIN
| R_VARIKLIS_MS2_GPIO_PIN | L_VARIKLIS_MS2_GPIO_PIN
| R_VARIKLIS_MS2_GPIO_PIN | L_VARIKLIS_MS2_GPIO_PIN
| L_VARIKLIS_ENABLE_PIN | R_VARIKLIS_DIR_PIN );
    //initinam D portus
    GPIO_D_st.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_D_st.Pull = GPIO_NOPULL;
    GPIO_D_st.Speed = GPIO_SPEED_MEDIUM;
    HAL_GPIO_Init(GPIOD, &GPIO_D_st);
    GPIO_B_st.Pin = (R_VARIKLIS_ENABLE_PIN | L_VARIKLIS_DIR_PIN);
    GPIO_B_st.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_B_st.Pull = GPIO_NOPULL;
    GPIO_B_st.Speed = GPIO_SPEED_MEDIUM;
    HAL_GPIO_Init(GPIOB, &GPIO_B_st);

switch (STEPPER_VARIKLIU_REZOLIUCIJA){

case SET_ZV_EIGHT_STEP:
//setinam MS1 H ir MS2 H MS3 L
//pirmas variklis
HAL_GPIO_WritePin(R_VARIKLIS_MS1_GPIO_PORT ,R_VARIKLIS_MS1_GPIO_PIN,GPIO_PIN_SET);
HAL_GPIO_WritePin(R_VARIKLIS_MS2_GPIO_PORT ,R_VARIKLIS_MS2_GPIO_PIN,GPIO_PIN_SET);
HAL_GPIO_WritePin(R_VARIKLIS_MS3_GPIO_PORT,R_VARIKLIS_MS3_GPIO_PIN,GPIO_PIN_RESET);
//antras variklis
HAL_GPIO_WritePin(L_VARIKLIS_MS1_GPIO_PORT ,L_VARIKLIS_MS1_GPIO_PIN,GPIO_PIN_SET);
HAL_GPIO_WritePin(L_VARIKLIS_MS2_GPIO_PORT ,L_VARIKLIS_MS2_GPIO_PIN,GPIO_PIN_SET);
HAL_GPIO_WritePin(L_VARIKLIS_MS3_GPIO_PORT,L_VARIKLIS_MS3_GPIO_PIN,GPIO_PIN_RESET);
break;

default:
//full step default
//pirmas variklis
HAL_GPIO_WritePin(R_VARIKLIS_MS1_GPIO_PORT,R_VARIKLIS_MS1_GPIO_PIN,GPIO_PIN_RESET);
HAL_GPIO_WritePin(R_VARIKLIS_MS2_GPIO_PORT,R_VARIKLIS_MS2_GPIO_PIN,GPIO_PIN_RESET);
HAL_GPIO_WritePin(R_VARIKLIS_MS3_GPIO_PORT,R_VARIKLIS_MS3_GPIO_PIN,GPIO_PIN_RESET);
}
}

```

```

//antras variklis
HAL_GPIO_WritePin(L_VARIKLIS_MS1_GPIO_PORT,L_VARIKLIS_MS1_GPIO_PIN,GPIO_PIN_RESET);
HAL_GPIO_WritePin(L_VARIKLIS_MS2_GPIO_PORT,L_VARIKLIS_MS2_GPIO_PIN,GPIO_PIN_RESET);
HAL_GPIO_WritePin(L_VARIKLIS_MS3_GPIO_PORT,L_VARIKLIS_MS3_GPIO_PIN,GPIO_PIN_RESET);
break;
}
//pradine DIR reiksme
HAL_GPIO_WritePin(R_VARIKLIS_DIR_PORT,R_VARIKLIS_DIR_PIN ,GPIO_PIN_RESET);
HAL_GPIO_WritePin(L_VARIKLIS_DIR_PORT ,L_VARIKLIS_DIR_PIN ,GPIO_PIN_SET);
//pradine disable reiksme
HAL_GPIO_WritePin(R_VARIKLIS_ENABLE_PORT,R_VARIKLIS_ENABLE_PIN,GPIO_PIN_SET);
HAL_GPIO_WritePin(L_VARIKLIS_ENABLE_PORT,L_VARIKLIS_ENABLE_PIN,GPIO_PIN_SET);
}
//
void zv_enable(void){
HAL_GPIO_WritePin(R_VARIKLIS_ENABLE_PORT,R_VARIKLIS_ENABLE_PIN,GPIO_PIN_RESET);
HAL_GPIO_WritePin(L_VARIKLIS_ENABLE_PORT,L_VARIKLIS_ENABLE_PIN,GPIO_PIN_RESET);}

void zv_disable(void){
HAL_GPIO_WritePin(R_VARIKLIS_ENABLE_PORT,R_VARIKLIS_ENABLE_PIN,GPIO_PIN_SET);
HAL_GPIO_WritePin(L_VARIKLIS_ENABLE_PORT,L_VARIKLIS_ENABLE_PIN,GPIO_PIN_SET);}

void set_R_dir_For(void)
{ HAL_GPIO_WritePin(R_VARIKLIS_DIR_PORT,R_VARIKLIS_DIR_PIN ,GPIO_PIN_RESET);}

void set_R_dir_Rev(void)
{HAL_GPIO_WritePin(R_VARIKLIS_DIR_PORT,R_VARIKLIS_DIR_PIN ,GPIO_PIN_SET);}

void set_L_dir_For(void)
{HAL_GPIO_WritePin(L_VARIKLIS_DIR_PORT,L_VARIKLIS_DIR_PIN ,GPIO_PIN_SET);}

void set_L_dir_Rev(void)
{HAL_GPIO_WritePin(L_VARIKLIS_DIR_PORT,L_VARIKLIS_DIR_PIN ,GPIO_PIN_RESET);}
//
void set_status(uint8_t st)
{if (status_varikliai!=st)
{status_varikliai=st;}
}
//step pasiruosimas
void set_steps(uint32_t Ur_zing, uint32_t Us_zing, uint8_t greitis)
{
uint8_t i;
float santikis;
    if (Ur_zing>Us_zing)
    {
        santikis=(float)Ur_zing/(float)Us_zing;
        stepR_ms=((float)VARIKLIO_DID_GREITIS_MS*255)/((float)greitis);
        stepL_ms=santikis*stepR_ms;
    }
    else
    {
        santikis=(float)Us_zing/(float)Ur_zing;
        stepL_ms=((float)VARIKLIO_DID_GREITIS_MS*255)/((float)greitis);
        stepR_ms=santikis*stepL_ms;
    }
    stepL_msu=(uint32_t) stepL_ms;
    stepR_msu=(uint32_t) stepR_ms;
}

void set_steps2(uint32_t Ur_zing, uint32_t Us_zing, uint8_t greitis)
{
uint8_t i;
float santikis;
if (Ur_zing>Us_zing)
{
    santikis=(float)Ur_zing/(float)Us_zing;
    stepR_ms2=((float) VARIKLIO_DID_GREITIS_MS*255)/((float)greitis);
    stepL_ms2=santikis*stepR_ms2;
}
else

```

```

    {
        santikis=(float)Ur_zing/(float)Us_zing;
        stepL_ms2=((float)VARIKLIO_DID_GREITIS_MS*255)/((float)greitis);
        stepR_ms2=santikis*stepL_ms2;
    }

    stepL_ms2u=(uint32_t) stepL_ms2;
    stepR_ms2u=(uint32_t) stepR_ms2;
}

void set_steps_is_XY(int32_t Xin, int32_t Yin)
{
    //ivedamas mm;
    float Ykam,Xkam;
    int32_t temp;
    Ur_pos=0;
    Ul_pos=0;
    Ur_pos2=0;
    Ul_pos2=0;

    glob_kampas=0;
    //apskaiciuojamas poslinkis
    Xrobot=Xin;
    Yrobot=Yin;
    Ykam=(Yin-Y);
    Xkam=(Xin-X);
    if (Xkam==0) {Xkam=0.01;}
    if (Ykam==0) {Ykam=0.01;}

    glob_kampas=atan((float)((Ykam)/(Xkam)));
    if (Ykam<0 && Xkam<0) {glob_kampas+=(float)3.1416;}
    else if (Ykam<0) {glob_kampas+=(float)6.2832;}
    else if (Xkam<0) {glob_kampas+=(float)3.1416;}

    trik_kampas=glob_kampas-kampas;
    if (fabs(trik_kampas)>0.785)
    //sukimas
        if (fabs(trik_kampas)>3.1416) {
            if ((trik_kampas)>3.1416) {trik_kampas-=6.2832;}
            else if (trik_kampas<-3.1416) {trik_kampas+=6.2832;}
        }

    }

    Ur_pos=trik_kampas*b_atstumas/2;
    Ul_pos=-Ur_pos;
    // kampas=glob_kampas;

        if (abs(Xkam)>10 ) {temp=Xkam/(cos(glob_kampas));}
        else {temp=Ykam/(sin(glob_kampas));}
        Ur_pos2=temp;
        Ul_pos2=temp;
    }
    else
    {
        if (abs(Xkam)>10) {temp=Xkam/(cos(glob_kampas));}
        else {temp=Ykam/(sin(glob_kampas));}
        Ur_pos=(2*temp + trik_kampas*b_atstumas)/2;
        Ul_pos=2*temp-Ur_pos;
    }
    X=Xrobot;
    Y=Yrobot;
    kampas=glob_kampas;
}

```


2 PRIEDAS. MATLAB KODAS

Matlab analitinio modeliavimo kodas:

```
clear all
close all;
clc;
xt=[100 0 0];

PP=[0 0 0
     0 0 0
     0 0 0];

zyme=[100 1050];

R=[2 0
   0 1];
i=0;
j=0;
for k=1:361
u(1)=19.1986;
u(2)=17.4533;

j=j+1;
if (j>150)
zt=[1100; 0];
else
zt=[980; 0];
end

zt2=zt;
i=i+1;
if (i==1)
zt(1)=(0.98 + (1.02-0.98)*rand(1))*zt(1);
i=0;
end

Xmat(k)=zyme(1)+zt(1)*sin((k*pi)/180);
Ymat(k)=zyme(2)-zt(1)*cos((k*pi)/180);

u(1)=(0.99 + (1.01-0.99)*rand(1))*u(1);
u(2)=(0.99 + (1.01-0.99)*rand(1))*u(2);

Q=[0.1 0
   0 2];

[xt , P] = spejimas(xt, PP, u, Q);
[ x, PP, vt, S ] = Matavimas( xt, P,zt, R, zyme );
vtt(k)=vt(1);
vttt(k)=vt(2);

xx(k)=xt(1);
yy(k)=xt(2);

x1(k)=x(1);
y1(k)=x(2);

end

figure(1)
plot(xx,yy,'b--');
```

```

title('Išplestinis Kalman filtras');
hold on
plot(x1,y1, 'r');
plot(Xmat, Ymat, 'g');
plot(100,1050,'c*');
xlim([-1200 1200])
ylim([-100 2300])
xlabel('X, mm');
ylabel('Y, mm');

legend('spejimas', 'EKF', 'Matavimas', 'Žyme');
legend('boxoff')
hold off
set(gca, 'FontSize', 8, 'LineWidth', 0.75); %<- Set properties

width = 10;
height= 9;

set(gcf, 'InvertHardcopy', 'on');
set(gcf, 'PaperUnits', 'centimeters');
myfiguresize = [0, 0, width, height];
set(gcf, 'PaperPosition', myfiguresize);

print('kalmanband', '-dpng', '-r600');

```

```

function [xt , P] = spejimas(xt2, Pt2, u, Q)

% xt2- matrica roboto būsenos t-1 momentu (x,y,kam)
% Pt2- klaidos koreliacijos matrica t-1 momentu
% u- valdymas sr ir sl
% Q- valdymo klaidos matrica

%roboto tarp ratu atstumas {mm}
b=100;

trikst=(u(1)+u(2))/2;
trikkam=(u(1)-u(2))/b;

xt=[xt2(1)+ trikst*cos(xt2(3)+trikkam/2)
    xt2(2)+ trikst*sin(xt2(3)+trikkam/2)
    xt2(3)+ trikkam];

Fx=[1 0 -trikst*sin(xt2(3)+trikkam/2)
    0 1 trikst*cos(xt2(3)+trikkam/2)
    0 0 1
    ];

Fu=[0.5*cos(xt2(3)+trikkam/2) 0.5*sin(xt2(3)+trikkam/2)
    0.5*cos(xt2(3)+trikkam/2) 0.5*sin(xt2(3)+trikkam/2)
    1/b 1/b
    ];

P=Fx*Pt2*Fx'+Fu*Q*Fu';

end

```

```

function [ x, PP, vt, S] = Matavimas( xt, P,zt, R, zyme )
%xt- spėjimo roboto būseną
% P- klaidos koreliacijos matrica t-1 momentu
% zt- sensoriaus matavimas r ir alfa
% R- sensoriaus matavimo neapibrėžtis
% žemėlapių žymė

%išėjimas x atnaujinta roboto būseną
%PP- klaidos koreliacijos matrica t momentu
%vt-klaida tarp išmatuoto ir nuspėto matavimo
%S- Koreliacija
skirx=-zyme(1)+xt(1);
skiry=-zyme(2)+xt(2);
skirxk=skirx^2;
skiryk=skiry^2;

zz=[sqrt(skirxk + skiryk); atan(skiry/skirx)-xt(3)];

A1=(2*skirx)/(2*sqrt(skirxk + skiryk));
A2=(2*skiry)/(2*sqrt(skirxk + skiryk));
A3=skiry/(skirx*((skiry/skirx)+1));
A4=1/(skirx*((skiry/skirx)+1));

H=[ A1 A2 0
    A3 A4 -1];

V=[1 0
   0 1];

S=H*P*H'+V*R*V';
vt=zt-zz;

K=P*H'*inv(S);
x=xt+K*vt;

PP=P-K*S*K';

end

```

3 PRIEDAS. KTU „TECHNORAMA 2016“ SERTIFIKATAS



ktu nacionalinis inovacijų ir verslo centras
1922

JAUNŲJŲ MOKSLININKŲ
DARBŲ PARODA-KONKURSAS

SERTIFIKATAS

Šiuo sertifikatu pažymima, kad

Vytautas Senvaitis

Dalyvavo jaunųjų mokslininkų parodoje-konkurse „Technorama '16“, kurioje pristatė darbą

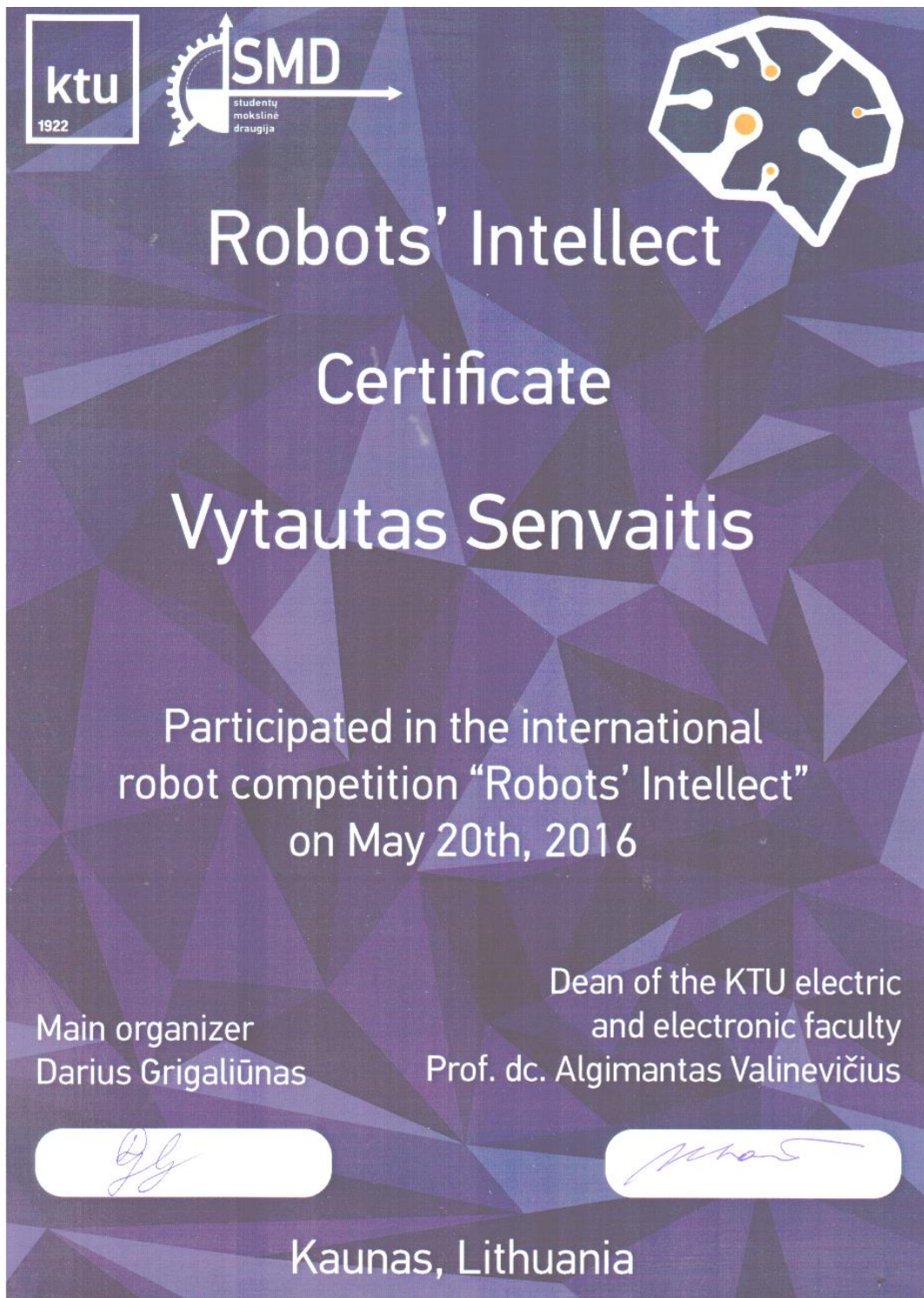
Adaptivus 2d pozicionavimo metodo autonominiam robotui sukūrimas ir tyrimas

„Technorama 2016“ darbų vertinimo komisijos pirmininkė
KTU mokslo prorektorė
Asta Pundzienė

TECH
NO
RA 2016
MA

KAU
NAU
LIETUVOS SVEIKATOS
MOKSLŲ UNIVERSITETAS
teo intermedix TeleSoftas Practica Capital DIZREGA REKLAMOS DAMYBA
VYTAUTO DIDŽIOJO
UNIVERSITETAS seo Helis Kauno mokslo ir technologijų parkas CSC Fima ktu

4 PRIEDAS. „ROBOTŲ INTELEKTAS 2016“ SERTIFIKATAS



The certificate features a dark purple background with a geometric, low-poly pattern. In the top left corner, there is a logo for 'ktu' (Kaunas University of Technology) with the year '1922' below it. To its right is the 'SMD' logo, which includes a gear icon and the text 'studentų mokslinė draugija'. In the top right corner, there is a stylized white brain icon with orange dots representing neurons. The main text is centered and reads 'Robots' Intellect Certificate Vytautas Senvaitis'. Below this, it states 'Participated in the international robot competition "Robots' Intellect" on May 20th, 2016'. At the bottom, there are two white rounded rectangular boxes containing signatures. The left signature is for Darius Grigaliūnas, and the right signature is for Prof. dr. Algimantas Valinevičius. The location 'Kaunas, Lithuania' is printed at the very bottom.

ktu
1922

SMD
studentų
mokslinė
draugija

Robots' Intellect
Certificate
Vytautas Senvaitis

Participated in the international
robot competition "Robots' Intellect"
on May 20th, 2016

Main organizer
Darius Grigaliūnas

Dean of the KTU electric
and electronic faculty
Prof. dr. Algimantas Valinevičius

Darius Grigaliūnas

Algimantas Valinevičius

Kaunas, Lithuania