Autonominis robotas - tai automatinė mašina, sugebanti judėti aplinkoje [2]. Jie sugeba valyti patalpas, skalbti, ruošti valgį, ieškoti nurodytų daiktų, juos transportuoti į nurodytas vietas, orentuotis pastovioje arba kintančioje aplinkoje, sąveikauti su aplinka, padėti žmogui valdyti sudėtingą techniką, atlikti pavojingas užduotis, atlikti užduotis aplinkoje, kur negali patekti žmogus, būti tam tikros teritorijos sargais ir netgi tarnauja žmoniems, turintiems negalią. Pastariesiems elektrinis vežimėlis, kuris vadovaujasi jam parašytu algoritmu, leidžiančiu autonominiam dviračiui robotui orentuotis vienoje ar kitoje aplinkoje, išvengti kliūčių bei jas apvažiuoti, labai palengvina gyvenimą.

Autonominiai robotai turi galimybę judėti aplinkoje ir nėra įtvirtinti vienoje fizinėje vietoje.Pramoninius robotus dažniausiai sudaro lanksti ranka (kelių jungčių manipuliatorius) irgriebtuvas, pritvirtintų prie nejudamo paviršiaus [2]. Jie populiarūs šiuodieniniuose tyrimuose ir beveik kiekvienas didelisuniversitetas turi vieną ar kelias laboratorijas, kurios skiria visą dėmesį savaeigių robotųtyrimams. Šių robotų taip pat galima rasti pramoninėje, karinėje ir saugumo aplinkose. Jietaip pat pasirodo ir kaip vartotojų produktai pramogoms ar tam tikroms funkcijoms, tokioms kaip dulkių siurbimas ar žolės pjovimas atlikti [2].

# METODŲ APŽVALGA

Autonominiai robotai gali būti klasifikuojami:

● Aplinka, kurioje jie dirba:

◦ Lauko arba namų robotai. Jie dažniausiai būna ratuoti, bet pasitaiko ir žingsinuojančių robotų. turinčių dvi ar daugiau kojų (humanoidiniai ar panašūs į gyvūnus ar vabzdžius).

◦ Skraidantys robotai dar dažnai vadinami nepilotuojamomis aeromašinomis (NAM, angl. UAV – unmanned aerial vehicles).

◦ Povandeniniai robotai dar vadinami autonominėmis povandeninėmis mašinomis (APM, angl. AUV – autonomous underwater vehicles).

● Įtaisas, kurį jie naudoja judėjimui:

◦ Žingsniuojantys robotai: žmogiškos kojos (pvz. androidas) arba gyvūniškos kojos.

◦ Ratiniai robotai.

◦ Vikšriniai robotai [2].

Išskirtinos yra dvi pagrindinės ir dažniausiai pasitaikančios autonominių robotų rūšys, t.y. važinėjantys robotai ir žingsniuojantys robotai. Šio darbo tikslas yra aptarti ratuotų robotų judėjimą, pagal jiems užduotą trajektoriją.

## Ratuoto roboto linijos sekimas

Šiuo metu tai labiausiai paplitusi robotų rūšis, turinti gana nemažą taikomąją sritį. Jų populiarumą nulėmė labai paprastos konstrukcijos ir nesudėtingos valdymo schemos bei algoritmai. Ratai įgalina juos judėti gana greitai, kas suteikia jiems dar vieną privalumą. Dėl šių priežasčių juos yra labai lengva valdyti, o savikaina labai maža, todėl jie plačiai taikomi kosminiuose bei kitokiuose tyrimuose, gelbėjimo bei policijos operacijose, gamyklose gaminiams vežti ir kt.

Linijos sekimo metodas plačiai naudojamas autonominėse transporto priemonėse, kuris leidžia mašinoms orientuotis įvairiose aplinkose bei sekti jiems užduotą liniją. Jie dažnai naudojami gamyklose arba sandėliuose nenutrūkstamai pernešinėti įvairius daiktus, vykdyti nenutrūkstamą darbą be jokio vairuotojo [3].

## Sudėtingų trajektorijų sekimas

Visame pasaulyje mokslininkai nagrinėja sudėtingas roboto trajektorijos, tokios kaip tiesi linija, vingis, status posūkis, T- formos sandūra ir kryžkelė, sekimo metodus naudojant T formos infraraudonuosius daviklius. T formos daviklių sekimo signalas suprojektuotas įrengiant papildomą infraraudonąjį sensorių roboto judėjimo kryptimi patalpinant jį ant vienos eilės sensorių [7]. Eksperimentų rezultatai rodo, kad T- formos sensorius gali sėkmingai įvykdyti sudėtingą trajektorijos sekimą. Ketvirtame paveiksle matome roboto prototipą, kuris seka jam nurodyta trajektorija.



4 pav. Sudėtingos trajketorijos sekimo roboto prototipas [7]

Šiuo metu pažangiausi robotai trajektorijos sekimui naudoja vienaeilius arba dvieilius infraraudonuosius daviklius. Vienos eilės infraraudonieji davikliai suteikia mažiau informacijos apie kelią ir įvykdyti sekimą yra sudėtinga. Dviejų eilių pirmyn nukreiptas infraraudonasis daviklis yra alternatyva kamerai. Jis galėtų priimti daugiau informacijos apie kelią ir padaryti sprendimą anksčiau, kas padėtų stabilizuoti roboto kontrolę ir pagreitinti trajektorijos sekimą. Ypatingai posūkiuose, priekinio ir galinio vaizdo sekimas vienu metu, praplėstų roboto atpažinimo galimybes.

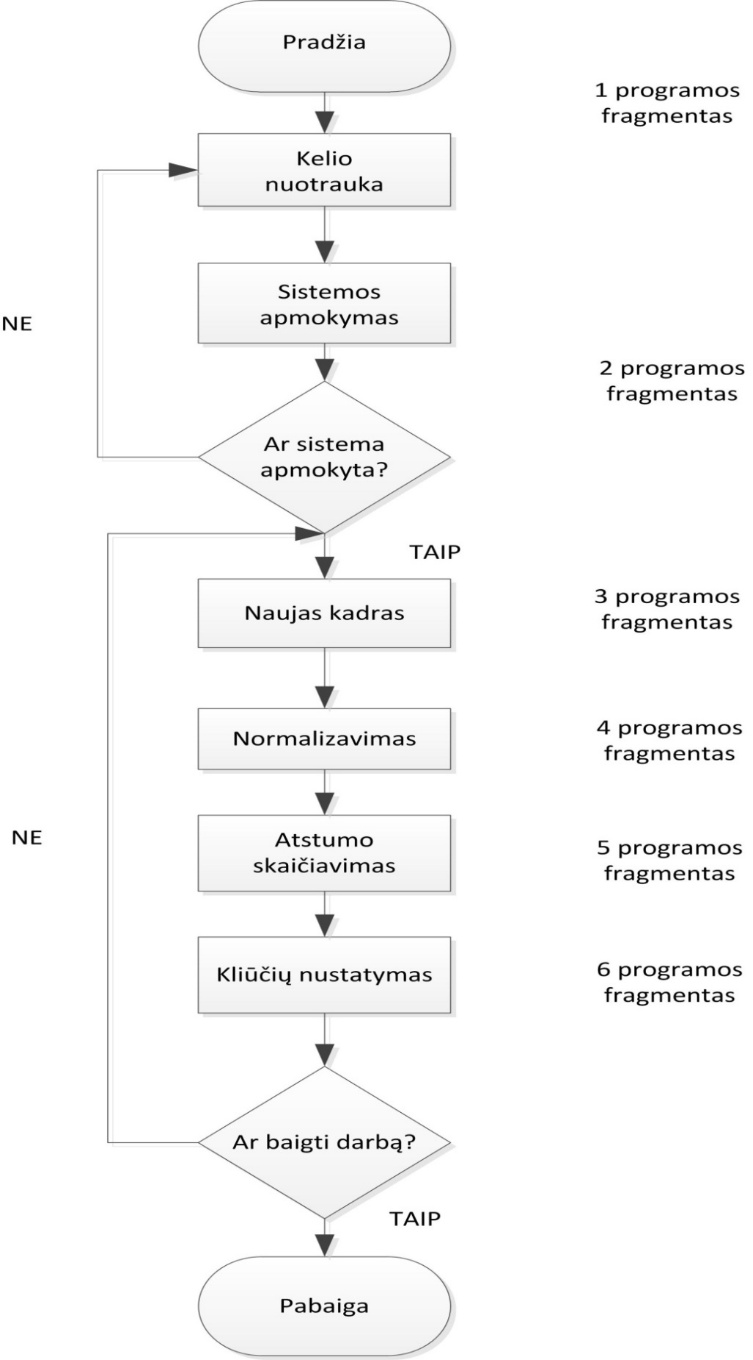
## Kompiuterinė rega

Atpažįstant objektus, buvo žinoma, kad žmogaus smegenys apdoroja regimąją informaciją daugiausia semantiniu lygmeniu, t.y. išskiriant semantiškai reikšmingus bruožus, tokius kaip kontūro segmentai, ribos, forma ir taip toliau. Bet šiandieninės informacijos apdorojimo technologijos neleidžia šių bruožų nustatyti aiškiai, todėl kompiuterinis regėjimas vis dar negali apdoroti regimosios informacijos taip kaip žmogaus smegenys [8]. Kompiuteriai turi apdoroti regimąją informaciją duomenų plotmėje, suformuotoje lengvai nustatomų, bet mažiau reikšmingų bruožų, tokių kaip spalvos, faktūros ir t.t. Taigi apdorojimo proceso metodika kompiuteriuose gerokai skiriasi nuo žmonių.

# SIŪLOMO METODO TEORINIS PAGRINDIMAS

## Apmokymo ir judėjimo algoritmas

Autonominis vėžimėlis turi galimybę judėti aplinkoje ir nėra įtvirtintas vienoje fizinėje vietoje. Norint, kad vėžimėlis važiuotų tam tikru keliu, jis turi būti nufilmuotas bei kruopščiai išanalizuotas. Tam naudojamas roboto apmokymo ir judėjimo algoritmas bei rašomas kodas MATLAB aplinkoje. Toliau pateiktas visas algoritmas darytas su Microsoft Visio 2010 bei išanalizuoti atskiri jo fragmentai.



Pirmasis programos fragmentas nurodo, jog paimama tiriamosios trajektorijos nuotrauka iš kurios yra iškerpamas tam tikro dydžio regionas. Aktyvavus iškirptąjį regioną, pradedami analizuoti pavieniai kadrai: nuo pirmojo iki paskutiniojo. Sistemos apmokymas paremtas vidutinių rgb (angl. k. red - raudona, green - žalia, blue - mėlyna) reikšmių skaičiavimu. Sistemą apmoko ekspertas (tyrėjas), kuris patikrina ar algoritmas tinkamas paleidimui ir ar visi parametrai tinkamai nustatyti. Nustatoma vidutinė spalvų kanalų matrica, vidutinė spalvinio kanalo spalva bei apskaičiuojamas standartinis spalvos nuokrypis nuo vidurkio. Trečiasis programos fragmentas orientuojasi į kadrų keitimą. Šio fragmento tikslas išanalizuoti visus programoje nustatytus kadrus. Kiekvienas imamas kadras turi būti normalizuotas, taigi ketvirtojo fragmento tikslas yra normalizavimas. Tikrinama vidutinė spalvinio kanalo spalva bei standartinis spalvos nuokrypis nuo vidurkio. Penktasis fragmentas apskaičiuoja atstumą tarp rgb atspalvių. Tam naudojama euklido atstumo formulė, kuri skaičiuoja atstumą tarp dviejų spalvinių taškų vektorių. Šeštojo fragmento tikslas – nustatyti trajektorijose sutinkamas kliūtis. Kadangi filmuotos trajektorijos yra skirtingų atspalvių, atspindžių bei tekstūrų, autonominis robotas apmokomas, kaip jas aptikti bei jų išvengti.

## Sudedamosios algoritmo dalys

**Kadrų skaičiaus bei vaizdo formato parinkimas**

Bandymai buvo atliekami filmuojant tris skirtingus tiriamuosius kelius. Filmavimo laiko trukmė kiekvienam keliui buvo 40 sekundžių. Per tokį laiką matlab terpėje buvo užfiksuoti 936 skirtingi kadrai. Filmuotų vaizdų skiriamosios gebos formatas buvo 640 x 480 pikselių.

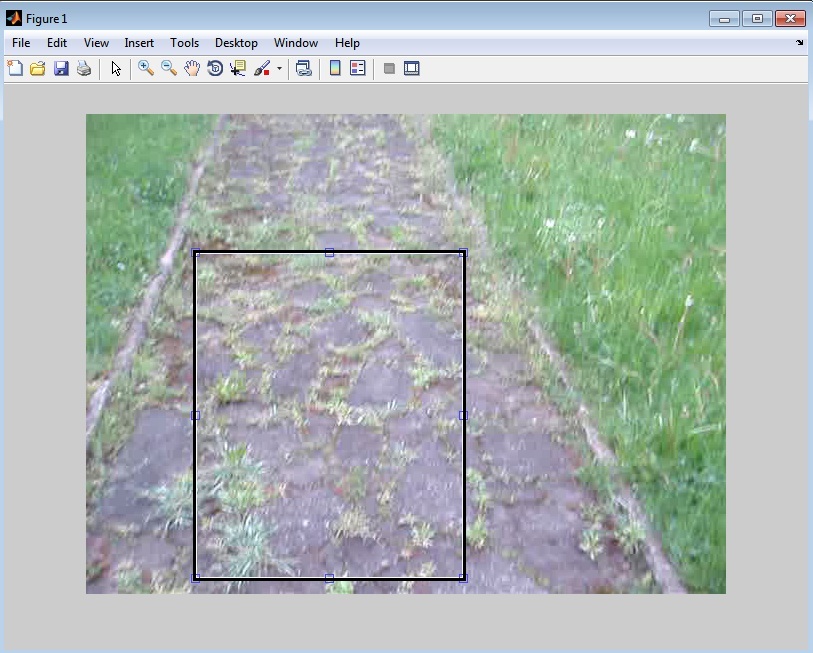
|  |
| --- |
| Programos fragmentas Nr.1 Kadro formatai |
| nFrames = 936;  vidHeight = 480;  vidWidth = 640; |

Funkcija *nFrames* parodo kiek kadrų buvo fiksuota kiekviename tiriamajame kelyje. Funkcijos *vidHeight* ir *vidWith* parodo video skiriamosios gebos formatą.

**Kelio nuotraukos paėmimas**

|  |
| --- |
| Programos fragmentas Nr. 2 Apmokymo duomenų parinkimas |
| Road = double(imcrop(imresize(read(xyloObj, 100),[480 640])));  Kelio nuotrauka |

*Imcrop* funkcija iškerpa tam tikro dydžio regioną iš nuotraukos. Iškirptąjį regioną aktyvavus programos kodas pradeda tikrinti kiekvieną einamąjį kadrą. Funkcija *imresize* pakeičia nenustatytą kadro formatą, į nustatytą, šiuo atveju 640 x 480 pikselių. Fragmente skaičius 100 reiškia, kad mokymui skirti duomenys imami iš kadro numeris šimtas.

****

6 pav. Tiriamosios trajektorijos fragmento iškirpimas

Paveikslėlyje juodas stačiakampis yra pažymėtoji nuotraukos dalis (funkcija *imcrop*), kurį aktyvavus programos kodas pradeda skaičiuoti kadrus. Stačiakampio dydį galima keisti rankiniu būdu.

**Sistemos apmokymas**

|  |
| --- |
| Programos fragmentas Nr. 3 Vidutinių rgb reikšmių nustatymas  Sistemos apmokymas |
| sumRoad = ((Road(:,:,1)+ Road(:,:,2)+ Road(:,:,3))./3)+1;  normRoad = zeros(size(Road));  for i=1:NumberOfChannels  Ar sistema apmokyta?  normRoad(:,:,i) = Road(:,:,i)./sumRoad;  end      for i=1:NumberOfChannels  MeanChannel = [MeanChannel; median(mean(normRoad(:,:,i)))];  stdChannel = [stdChannel; mean(std(normRoad(:,:,i)))]; |

Kintamasis *sumRoad* apibūdina vidutinę spalvų kanalų matricą, apskaičiuotą pagal formulę:

*Γ = [R G B]* (1)

*Γ1 =R Γ2=G Γ3 =B* (2)

(3)

Formulėje (3) kintamasis *U* reiškia eilučių skaičių, o kintamasis *V* – stulpelių skaičių. Vidurkis gaunamas susumavus visus rgb eilučių ir stulpelių narius bei padalinus juos iš trijų.

Kintamasis *normRoad* yra nuotraukos normavimas pagal šviesį, jis užrašomas formulėmis:

(4)

(5)

(6)

Kintamasis *Meanchannel* – apibūdina vidutinę spalvinio kanalo spalvą.

Kintamasis *stdchannel* -apskaičiuoja standartinį spalvos nuokrypį nuo vidurkio.

**Kadrų atnaujinimas**

|  |
| --- |
| Programos fragmentas Nr. 4 Kadrų tikrinimas  Naujas  kadras |
| for k = 1 : nFrames  rIm = imresize(read(xyloObj, k),[480 640]); |

Trajektorijos sekimui įgyvendinimui reikia tikrinti kiekvieno kadro formatą. Tam naudojamas ciklas nuo pirmojo iki paskutiniojo kadro. Šiuo atveju naudojami 936 skirtingi atvaizdai, kurie yra kruopščiai patikrinami. Funkcija *imresize* reiškia, kad gautas kadro formatas paverčiamas į 640 ant 480 pikselių.

**Normalizavimas**

|  |
| --- |
| Programos fragmentas Nr. 5 Kadrų normalizavimas |
| Road = double(Image);  Normalizavimas  sumRoad = ((Road(:,:,1)+ Road(:,:,2)+ Road(:,:,3))./3)+1;  normRoad = zeros(size(Road));  for i=1:NumberOfChannels  normRoad(:,:,i) = Road(:,:,i)./sumRoad;  end |

Normalizavimas atliekamas kiekvienam naujam kadrui. Normalizavimo principas paremtas formulėmis (žr. (1)-(6) formules).

**Atstumo skaičiavimas**

|  |
| --- |
| Programos fragmentas Nr. 6 Euklido atstumas |
| for i=1:NumberOfChannels  colormap = double(normRoad(:,:,i));  Atstumo skaičiavimas  bw(:,i) = colormap(:);  end    rbw = dist(bw,MeanChannel);  rbw = reshape(rbw,vidHeight,vidWidth); |

Kintamasis *colormap* yra tarpinis kintamasis, kuris reikalingas atlikti tam tikrą programinį veiksmą, t.y. matricą paversti stulpeliu. Rezultatas yra išsaugomas kintamajame *bw*, kurio dydis yra 640 x 480 eilučių ir stulpelių. Taip pat taikoma atstumo skaičiavimo formulė (euklido atsumas):

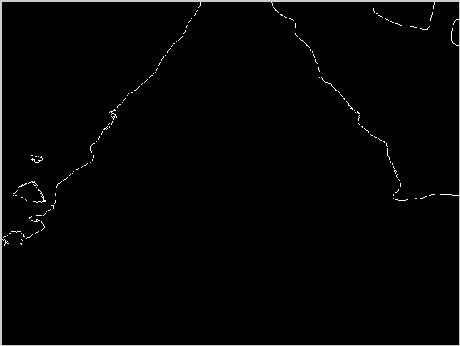
(7)

; ; - tai kintamieji nusakantys spalvinių (red, green, blue) vektorių padėtis. Gautasis rezultatas naudojant funkciją *reshape* yra paverčiamas į matricą.

**Kliūčių nustatymas**

|  |
| --- |
| Programos fragmentas Nr. 7 Dispersijos koeficientas |
| Kliūčių nustatymas  ind = find(rbw(:)>mean(rbw(:))+std(rbw(:))\*0.1);  Obstacle = sum(ForBumper(VertOffSetMidl:end,:),1);  ObstacleINDleft = find(Obstacle(1:HorMidlPoint)>0);  ObstacleINDright = find(Obstacle(HorMidlPoint:end)>0); |

Funkcijos *find* pagalba skaičiuojamas atstumas iki kliūčių. Jei atstumas viršija vidurkį, programos lange tie pikseliai yra pažaliuojami (žr. 8 pav.) bei vertinami kaip kliūtys. Kintamojo *ind* tikslas parodyti matricoje vertes tenkinančias sąlygas. Fragmente skaičius 0.1 reiškia dispersijos koeficientą. Darbe atliktiems tyrimams jis buvo keičiamas nuo 0.1 iki 1 (žingsniu 0,1). Kuo dispersijos koeficientas didesnis tuo funkcija fiksuodavo mažiau aptinkamų kliūčių. Obstacle yra kintamasis, kuriame nurodyti regionai žymintys kliūtis paveiksle. Tai yra juodai baltas paveikslas, kuriame kliūtys nudažomos balta spalva (žr. 7 pav). Kintamieji *ObstacleINDleft* ir *ObstacleINDright* naudojami šoninių kraštų riboms nustatyti.

7 pav. Juodai baltas kliūčių nustatymo paveikslas

# EKSPERIMENTINIAI TYRIMAI

## Eksperimentinės nuostatos

Eksperimentinio tyrimo metu buvo kuriama kelio atpažinimo bei jo sekimo, išvengiant kliūčių, sistema. Sistemos tikslas, leisti autonominiam robotui savarankiškai judėti nufilmuotu paviršiumi ir išvengti kelyje pasitaikiusių kliūčių. Iš pradžių, pasitelkus vaizdo stebėjimo kamerą buvo filmuojamos skirtingos dangos ir skirtingų matmenų kelio atkarpos. Kameros atstumas nuo filmuojamo kelio paviršiaus buvo 0,4 metro, filmavimo trukmė 40 sekundžių. Per nustatytą laiką sistema fiksavo 936 skirtingus kadrus, šie gauti vaizdai apdoroti Matlab paketu. Apačioje pateiktose lentelėse matyti tyrimo metu naudoti prietaisai.

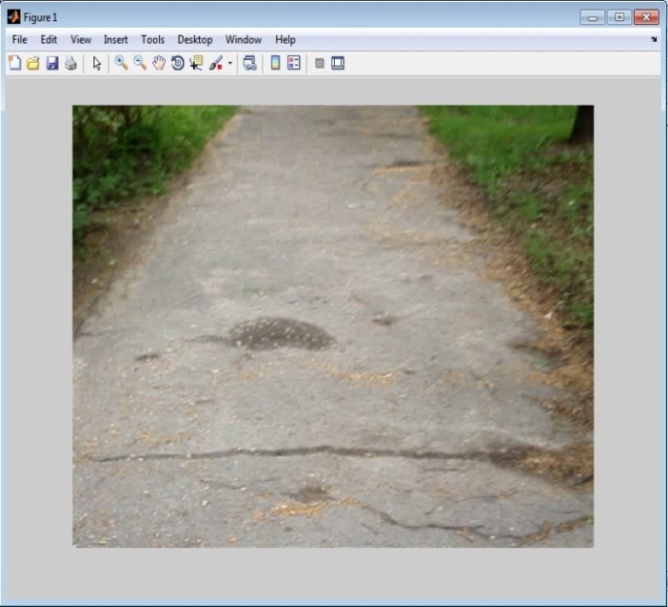
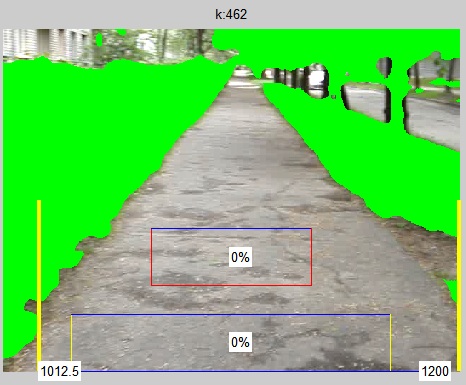
1 lentelė. Vaizdo kameros parametrai

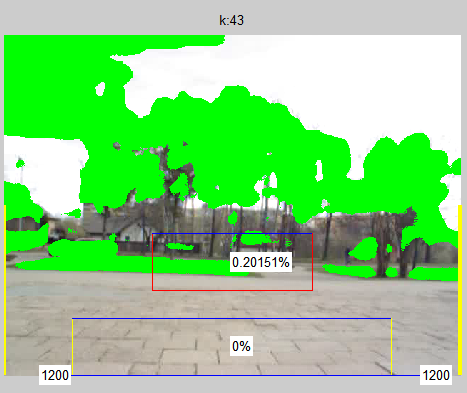
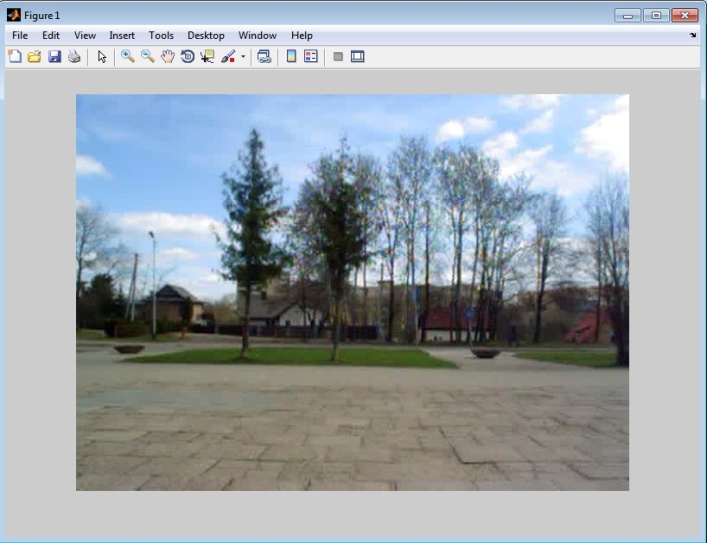
|  |  |
| --- | --- |
| **Vaizdo stebėjimo kamera:** | |
| Gamintojas: | Sony DSC-W55 |
| Objektyvo raiška: | 7.2 mln efektyvių taškų |
| Optikos tipas: | Carl-Zeiss Vario-Tessar |
| Vidinė atmintis: | 56mb |
| Optinis priartinimas: | 3x |
| Ekranas: | 2.5“ LCD |

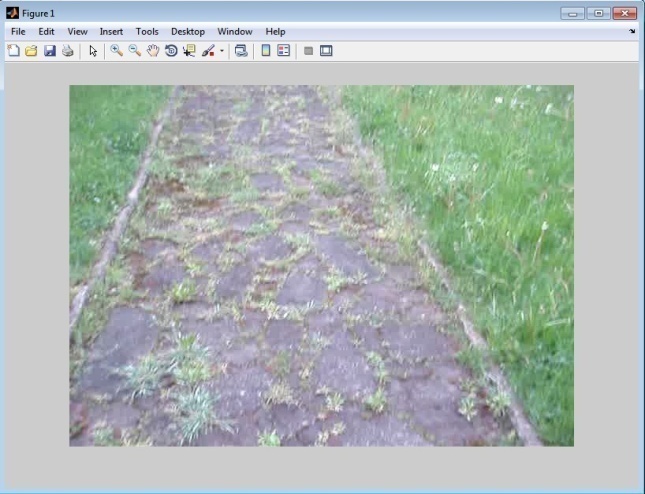
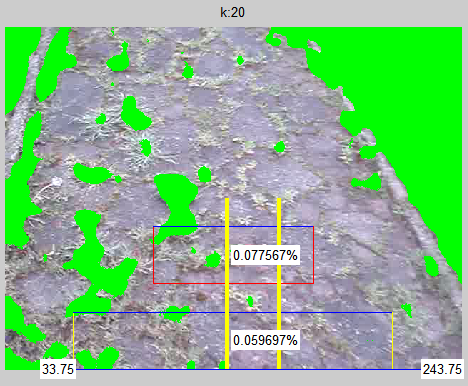
2 lentelė. Tyrimams naudotas kompiuteris

|  |  |
| --- | --- |
| **Kompiuteris:** | |
| Gamintojas: | Acer Aspire 6920G |
| Ekranas: | 16” LCD |
| Procesorius: | Intel Core Duo 2,2 Ghz |
| Operatyvioji atmintis: | 4GB ddr2 |
| Kietasis diskas: | 250GB |
| Vaizdo plokštė: | NVIDIA GeForce 9500M GS 512MB |
| Operacinė sistema: | Windows 7 Home Premium |

Norint įvertinti skirtingų paviršių variaciją, eksperimento metu buvo filmuojamos trys visiškai skirtingos kelio dangos, su įvairialypiais paviršiais. Dėl šešėlių, atspindžių, kameros judėjimo bei paviršių išskirtinumo reikėjo keisti dispersijos koeficientą, kad surasti idealiausias sąlygas roboto judėjimui. Tiriamųjų kelių nuotraukos pateiktos (8-10) paveiksluose. Kairysis paveikslas tai pirmojo kadro nuotrauka, o dešinysis matlab aplinkoje veikiantis algoritmas, įvertinantis kiekvieną kadrą ir ieškantis kliūčių. Nuotraukos darytos kai dispersijos koeficientas buvo lygus 0,1. Didinant dispersijos koeficientą, žaliųjų pikselių kiekis mažėdavo, t.y. tų pačių kliūčių, kurios buvo aptiktos prie mažesnio dispersijos koeficiento, neaptikdavo.

8 pav. Pirmoji tiriama trajektorija

9 pav. Antroji tiriama trajektorija

10 pav. Trečioji tiriama trajektorija

**Kelių pločio nustatymas**

Eksperimento metu pasirinktųjų kelių plotis buvo išmatuotas rankiniu būdu. Gauti rezultatai buvo įvedami Matlab aplinkoje ir paleidus algoritmą, programa skaičiuodavo kelio plotį kiekviename atskirame kadre. Kadangi filmuojant kelio paviršius buvo trikdžių, tokių kaip kelio dangos pokyčiai, šešėlių ar atspindžių buvimas, atstumas kiekviename kadre labai skirtingas. Matlab aplinkoje atstumas verčiamas iš metrų į milimetrus, tam, kad būtų labai tikslūs skaičiavimai.

3 lentelė. Rankiniu būdu išmatuoti tiriamųjų kelių pločiai

|  |  |
| --- | --- |
| Pirmoji tiriama trajektorija: | 2,05 m |
| Antroji tiriama trajektorija: | 2,5 m |
| Trečioji tiriama trajektorija: | 1,8 m |

Suvedus realius kelių pločius į programos kodą, algoritmas pats perskaičiuodavo atstumus. Grafikuose (žr. 11,12,13 pav.) pateikta tiriamųjų trajektorijų priklausomybė nuo pavienių kadrų.

11 pav. Pirmosios trajektorijos pločio priklausomybė nuo pavienio kadro

12 pav. Antrosios trajektorijos pločio priklausomybė nuo pavienio kadro



13 pav. Trečiosios trajektorijos pločio priklausomybė nuo pavienio kadro

Iš gautų grafikų matyti, jog tiriamųjų kelių paviršius turi didelę įtaką, kelio pločio nustatymui atskirame kadre. Programa mažiausiai kliūčių bei triukšmų aptiko antrojoje trajektorijoje (žr. 9 pav.), nes paviršius yra lygiausias ir be didesnių šalutinių veiksnių, tokių kaip šešėliai ar dideli kelio dangos pokyčiai. Didžiausi kelio pločio svyravimai pasireiškė trečiajame tiriamame kelyje, nes jo paviršius buvo netolygus ir su daug kliūčių. Kiekviename atskirame kadre kelio plotis kito labai intensyviai ir praktiškai nebuvo tokios atkarpos, kad atstumo pokytis būtų nusistovėjęs.

## Kadrų apdorojimo trukmė

Eksperimento metu kiekvieno tiriamojo kelio filmavimo trukmė buvo 40 sekundžių. Per tiek laiko, buvo fiksuojami 936 skirtingi kadrai. Tyrimams naudoti prietaisai buvo keturių ir daugiau metų senumo, taigi rezultatai, atliekant tyrimus modernesniais prietaisais būtų greičiau apdorojami. Dėl kadrų skirtingumo, jų apdorojimo trukmė kito labai intensyviai. Grafikuose (žr. 14-16 pav.) matyti kiekvieno kadro apdorojimo trukmė Matlab aplinkoje. Ketvirtoje lentelėje pateikta atskirų trajektorijų video kadrų apdorojimo trukmės vidurkis.

4 lentelė. Vidutinė video kadrų apdorojimo trukmė

|  |  |
| --- | --- |
| Pirmosios tiriamos trajektorijos: | 0,236 s |
| Antrosios tiriamos trajektorijos: | 0,247 s |
| Trečiosios tiriamos trajektorijos: | 0,233 s |

14 pav. Pirmosios trajektorijos video kadrų apdorojimo trukmės kitimo grafikas



15 pav. Antrosios trajektorijos video kadrų apdorojimo trukmės kitimo grafikas



16 pav. Trečiosios trajektorijos video kadrų apdorojimo trukmės kitimo grafikas

Iš pateiktųjų grafikų matosi, jog kadro apdorojimo trukmei turi įtakos kadrų skirtingumas. Kadangi tiriamųjų trajektorijų paviršiai tarpusavyje buvo skirtingi, tai ir laiko grafikuose svyravimai yra skirtingi kiekvienu atveju. Intensyviausiai kito pirmosios trajektorijos kadrų apdorojimo laikas, nes kelio paviršiuje buvo aptikta daug įvairių atspindžių bei šešėlių. Antrosios trajektorijos kadrų apdorojimo laikas buvo pastoviai kintantis, vietomis su didesniais šuoliais. Mažiausi šuoliai užfiksuoti trečiajame grafike (žr. 16 pav.), nepaisant to, kad paviršius buvo labai nelygus bei su didelėmis kliūtimis, tai įtakos apdorojimo trukmei nesudarė, nes kadrai tarpusavyje buvo mažai skirtingi.

## Dispersijos koeficiento įtaka

Viso tiriamojo darbo tikslas buvo sukurti elektrinio vežimėlio valdymo algoritmą, kuris galėtų orientuotis pagal jam nustatyta važiuojamąją dalį. Svarbiausias uždavinys buvo iš nufilmuoto paviršiaus nustatyti kliūtis bei padaryti taip, jog važiuojant, jų būtų išvengta. Didžiausią įtaką tam turi dispersijos koeficientas. Jį keičiant, programa reaguodavo skirtingai į tą patį filmuotą kelią. Dispersijos koeficientas, kaip buvo minėta antroje darbo dalyje (žr. 21psl.), buvo keičiamas nuo 0.1 iki 1 (žingsniu 0,1). Kuo didesnis dispersijos koeficientas, tuo mažiau kliūčių algoritmas aptikdavo. Vidutinė absoliutinė paklaida gauta pagal formules:

(8)

Kintamasis sk žymi skirtumą tarp pločio *p(i)*, kurį randa algoritmas Matlab aplinkoje vykdant procesą, ir realaus, ranka išmatuoto pločio, kuris pažymėtas kintamuoju .

(9)

Matlab aplinkoje vektoriai prasideda nuo skaičiau 1, todėl *i=1.* Kintmasis *N* yra kadrų skaičius, šiuo atveju – 936.

17 pav. Pirmosios trajektorijos vidutinė absoliutinės paklaidos priklausomybė nuo dispersijos koeficiento

18 pav. Antrosios trajektorijos vidutinė absoliutinės paklaidos priklausomybė nuo dispersijos koeficiento

19 pav. Trečiosios trajektorijos vidutinė absoliutinės paklaidos priklausomybė nuo dispersijos koeficiento

Kadangi pirmosios ir antrosios trajektorijos paviršiai, lyginant su trečiąja, buvo gan tolygūs ir be didesnių žymių pokyčių, tad vidutinė absoliutinė paklaida augo tol kol buvo didinamas dispersijos koeficientas.

# LITERATŪRA

[1]. Tarptautinių žodžių žodynas. <http://www.zodynas.lt/tarptautiniu-zodziu/R/robotas>, 2012-05-14.

[2]. Wikipedia. Mobile robots. <http://en.wikipedia.org/wiki/Mobile_robot>, 2012-05-12.

[3]. L. Armesto, J. Tornero. “Automation of Industrial Vehicles: A Visio-based Line Tracking Application”. Technical University of Valencia, Camino de Vera. September 2009, p. 646-652.

[4]. Norhashim Mohd Arshad, Mohamed Farid Misnan, Noorfadzli Abdul Razak. „Single Infra-Red Sensor Technique for Line-Tracking Autonomous Mobile Vehicle“. Faculty of Electrical Engineering UiTM, Malaysia. March 2011, p. 159-162

[5]. Alexandra Bartolo, Kenneth P. Camilleri, Simon G. Fabri, Jonathan C. Borg. “Line Tracking Algorithm for Scribbled Drawings”. Department of Systems and Control Engineering University of Malta. March 2008, p. 554-559.

[6]. Kekuan Wang, Jianhu Zhang, Bingjie Zhao, Deyu Tang. “Research on Control Technology of Trajectory Tracking for Robotic Welding”. Computational Intelligence and Software Engineering (CiSE), Research Institute of Engineering Technology, China. December 2010, p. 1-4.

[7]. Wang Yue, Jin Xin, Li Qi-an, Li Ping. “T-shaped infrared sensor design of the robot and complex trajectory tracking method”. School of Information and Control Engineering, Liaoning Shihua University, Fushun, China. September 2009, p. 303-306.

[8]. Gregor Miller, Sidney Fels, Steve Oldridge. “A Conceptual Structure for Computer Vision”. Human Communication Technologies Laboratory ECE, University of British Columbia Vancouver, Canada. January 2011, p. 168-174.

[9]. Bo Zhang. “Computer Vision vs. Human Vision”. College of Information Science & Technology Dept. of Computer Science and Technology, Tsinghua University, China. July 2010, p. 1-3.

# PRIEDAI

**1 priedas**

clear all

close all

clc

VideoFile = ('C:\Users\dolfino\Desktop\vezimelio\kelias2.MPG');

xyloObj = VideoReader(VideoFile);

%load frames.mat;

nFrames = 936; %xyloObj.NumberOfFrames;

vidHeight = 480; %xyloObj.Height;

vidWidth = 640; %xyloObj.Width;

% nFrames = size(Frames,2);

% [vidHeight vidWidth NumberOfChannels] = size(Frames{1});

h = fspecial('disk',10);

%========Bumperis==================

HorMidlPoint = round(vidWidth/2);

VerTMidlPoint = round(vidHeight/2);

HorzOffSetLow = round(HorMidlPoint/1.2);

HorzOffSetMidle = round(HorzOffSetLow/1.2);

HorzOffSetHigh = round(HorzOffSetMidle/2);

VertOffSetLow = vidHeight;

VertOffSetMidl = VertOffSetLow-round(VerTMidlPoint/3);

VertOffSetHigh = VertOffSetMidl-round(VerTMidlPoint/2);

Bumper = [%Apatine linija

HorMidlPoint-HorzOffSetLow VertOffSetLow;

HorMidlPoint+HorzOffSetLow VertOffSetLow;

%Vidurine linija

HorMidlPoint-HorzOffSetMidle VertOffSetMidl;

HorMidlPoint+HorzOffSetMidle VertOffSetMidl;

%Virsutine linija

HorMidlPoint-HorzOffSetHigh VertOffSetHigh;

HorMidlPoint+HorzOffSetHigh VertOffSetHigh;

];

Bumper1 = [HorMidlPoint-HorzOffSetMidle VertOffSetMidl HorzOffSetMidle\*2 round(VerTMidlPoint/3)];

Bumper2 = [HorMidlPoint-HorzOffSetHigh VertOffSetHigh HorzOffSetHigh\*2 round(VerTMidlPoint/3)];

%=========Bumperis=================

%=======Zona plento atnaujinimui======

PlentasZone = zeros(vidHeight,vidWidth);

zone1 = zeros(vidHeight,vidWidth);

%=====================================

%%

%Road = double(imcrop(read(xyloObj, 500)));

%Kelio pikseliu atnaujinimui

a = .5;

b = 1-a;

%Plyteles ir pikseliu santykis

Mastelis = 3.75; %mm

%realus plotis

real\_plot = 2000 %mm

Road = double(imcrop(imresize(read(xyloObj, 100),[480 640])));

%Road = double(imcrop(Frames{340},Bumper1));

MeanChannel = [];

stdChannel = [];

NumberOfChannels = 3;

OfffSet = 3;

plotis = [];

% Normalization of the lumminance

sumRoad = ((Road(:,:,1)+ Road(:,:,2)+ Road(:,:,3))./3)+1; % normalizuoja paveikslo sviesuma

normRoad = zeros(size(Road));

for i=1:NumberOfChannels

normRoad(:,:,i) = Road(:,:,i)./sumRoad;

end

for i=1:NumberOfChannels

MeanChannel = [MeanChannel; median(mean(normRoad(:,:,i)))]; %vidutine spalva rgb

stdChannel = [stdChannel; mean(std(normRoad(:,:,i)))];

end

%Frames={};

% Read one frame at a time.

t = [];

for k = 1 : nFrames

rIm = imresize(read(xyloObj, k),[480 640]);

tic

%Frames{k} = Image;

%rIm = Frames{k};

Image = imfilter(rIm,h,'symmetric');

%Image = rIm;

Road = double(Image);

sumRoad = ((Road(:,:,1)+ Road(:,:,2)+ Road(:,:,3))./3)+1;

normRoad = zeros(size(Road));

for i=1:NumberOfChannels

normRoad(:,:,i) = Road(:,:,i)./sumRoad;

end

bw = zeros(vidHeight\*vidWidth,NumberOfChannels);

rbw = zeros(vidHeight\*vidWidth,1);

ForBumper = zeros(vidHeight\*vidWidth,1);

for i=1:NumberOfChannels

colormap = double(normRoad(:,:,i));

bw(:,i) = colormap(:);

end

rbw =dist(bw,MeanChannel); % euklido atstumas

rbw = reshape(rbw,vidHeight,vidWidth);

%bwdiff = abs(diff(rbw,1));

ind = find(rbw(:)>mean(rbw(:))+std(rbw(:))\*.1);

ff = [0 255 0]; % cia koki skaiciu vietoj 255 irasai, tokia spalva parodo kliuti (siuo atveju zalia)

for i =1:NumberOfChannels

colormap = rIm(:,:,i);

colormap = colormap(:);

colormap(ind) = ff(i);

rIm(:,:,i) = reshape(colormap,vidHeight,vidWidth);

end

%Plento pikseliu atnaujinimas

ForBumper(ind) = 1;

ForBumper = reshape(ForBumper,vidHeight,vidWidth);

Bumper1Mask = imcrop(ForBumper,Bumper1);

BumperInd = find(Bumper1Mask);

Bumper1Area = imcrop(normRoad,Bumper1);

for i = 1:NumberOfChannels

colormap = Bumper1Area(:,:,i);

colormap = colormap(:);

colormap(BumperInd) = [];

MeanChannel(i) = MeanChannel(i)\*a + median(colormap)\*b;

end

%Artimiausiu krastu paieska

Obstacle = sum(ForBumper(VertOffSetMidl:end,:),1);

ObstacleINDleft = find(Obstacle(1:HorMidlPoint)>0);

ObstacleINDright = find(Obstacle(HorMidlPoint:end)>0);

if ~isempty(ObstacleINDright)

RightLimitLine = [HorMidlPoint+ObstacleINDright(1) vidHeight;

HorMidlPoint+ObstacleINDright(1) VerTMidlPoint];

else

RightLimitLine = [vidWidth vidHeight;

vidWidth VerTMidlPoint];

end

if ~isempty(ObstacleINDleft)

LeftLimitLine = [ObstacleINDleft(end) vidHeight;

ObstacleINDleft(end) VerTMidlPoint];

else

LeftLimitLine = [1 vidHeight;

1 VerTMidlPoint];

end

% Kliuties atstumas iki vaizdo jutiklio centro

LeftObstacle = (HorMidlPoint-LeftLimitLine(1,1)+1)\*Mastelis;

RigthObstacle = (RightLimitLine(1,1)-HorMidlPoint)\*Mastelis;

%Kliutis bumperio zonoje Procentaliai

Bumper2Mask = imcrop(ForBumper,Bumper2); %Virsutinis

Bumper1Mask; %Apatinis

%Procentai

Bumper2Proc = length(find(Bumper2Mask))/(Bumper2(3)\*Bumper2(4));

Bumper1Proc = length(find(Bumper1Mask))/(Bumper1(3)\*Bumper1(4));

%Elgsenos busenos

%STOP

if Bumper1Proc >= 80

end

%Goto rigth

if Bumper2Proc >=90 & Flag == 0

end

t = [t toc];

subplot(1,2,1)

imshow(rIm)

hold on

rectangle('position',Bumper1,'EdgeColor','y')

rectangle('position',Bumper2,'EdgeColor','r')

%apatine linija

plot(Bumper(1:2,1),Bumper(1:2,2),'b-')

%vidurine linija

plot(Bumper(3:4,1),Bumper(3:4,2),'b-')

%virsutine linija

plot(Bumper(5:end,1),Bumper(5:end,2),'b-')

%Soniniai barjerai

%Right

line(RightLimitLine(:,1),RightLimitLine(:,2),'color','y','LineWidth',3)

%Left

line(LeftLimitLine(:,1),LeftLimitLine(:,2),'color','y','LineWidth',3)

plotis = [plotis (RigthObstacle(1,1)+LeftObstacle(1,1))];

sk = (plotis-real\_plot);

suma = sum(abs(sk)/936);

%Atstumai

text(Bumper(2,1),Bumper(2,2),[num2str(RigthObstacle)],'FontSize',10,'BackGroundColor','w');

text(Bumper(1,1),Bumper(2,2),[num2str(LeftObstacle)],'FontSize',10,'BackGroundColor','w');

%Procentai

text(HorMidlPoint,Bumper2(2)+Bumper2(4)/2,[num2str(Bumper2Proc) '%'],'FontSize',10,'BackGroundColor','w');

text(HorMidlPoint,Bumper1(2)+Bumper1(4)/2,[num2str(Bumper1Proc) '%'],'FontSize',10,'BackGroundColor','w');

hold off

title(['k:' num2str(k)])

subplot(1,2,2)

imshow(edge(ForBumper),[])

drawnow

end

plot(plotis);