KLAIPĖDOS UNIVERSITETAS
JŪROS TECHNOLOGIJŲ IR GAMTOS MOKSLŲ FAKULTETAS
INFORMATIKOS IR STATISTIKOS KATEDRA

AURIMAS VAITKUS

POVANDENINIŲ OBJEKTŲ APTIKIMO ALGORITMŲ TOBULINIMAS

Geoinformatikos magistro baigiamasis darbas
Geoinformatikos studijų programa 621F74001

Klaipėda, 2016

MAGISTRO BAIGIAMŲJŲ DARBŲ LYDRAŠČIO FORMA

Pildo magistro baigiamojo darbo autorius

..... **Aurimas Vaitkus**.....
(magistro baigiamojo darbo autoriaus vardas, pavardė)

Baigiamojo darbo tema: Baigiamojo darbo tema: Povandeninių objektų aptikimo algoritmų tobulinimas
……
(magistro baigiamojo darbo pavadinimas lietuvių kalba)

Patvirtinu, kad bakalauro/magistro baigiamasis darbas parašytas savarankiškai, nepažeidžiant kitiems asmenims priklausančių autorių teisių, visas baigiamasis bakalauro/magistro darbas ar jo dalis nebuvo panaudotas Klaipėdos universitete ir kitose aukštosiose mokyklose.

_____
(magistro baigiamojo darbo autoriaus ir parašas)

Sutinku, kad bakalauro/magistro baigiamasis darbas būtų naudojamas neatlygintinai 5 m. Klaipėdos universiteto studijų procese.

_____
(magistro baigiamojo darbo autoriaus ir parašas)

Pildo magistro baigiamojo darbo vadovas

Magistro baigiamąjį darbą ginti.....
..... (įrašyti - **leidžiu** arba **neleidžiu**)
....2016 .... prof. dr. O. Ramašauskas
(data) (magistro baigiamojo darbo vadovo vardas, pavardė ir parašas)

Pildo katedros, kuruojančios studijų programa, administratorius (sekretorius)

Baigimasis darbas įregistruotas katedroje 2016 .....
(data)

Laima Brazdeikienė.....
(katedros sekretorės vardas, pavardė ir parašas)

Pildo katedros, kuruojančios studiju programą, vedėjas

Magistro baigiamąjį darbą ginti.....
(įrašyti - **leidžiu** arba **neleidžiu**)

2016..... prof. dr. Vitalijus Denisovas.....
(data) (katedros vedėjo vardas, pavardė ir parašas)

Recenzentu(-ais) skiriu..... .....
....
....
(įrašyti recenzento(ų) vardą, pavardę)

.....2016..... prof. dr. Vitalijus Denisovas.....
(data) (katedros vedėjo vardas, pavardė ir parašas)

ANOTACIJA

Darbe nagrinėjami povandeninių objektų aptikimo algoritmai ir programinės priemonės jiems įgyvendinti. Povandeninių objektų atpažinimo algoritmai aktualūs ieškant paskendusių objektų, jūros dugno artefaktų, jūrinės augmenijos ir gyvūnijos pavyzdžių. Tiriami slenksčio metodai ir jų charakteristikos bei jiems daroma įtaka esant skirtingiems filtravimo parametrams. Programinės priemonės kurtos ir testuotos naudojantis Microsoft Visual Studio sistema kartu su OPENCV vaizdų apdorojimo biblioteka, panaudojant KU povandeninio stebėjimo roboto (ROV) vaizdo medžiagą.

PAGRINDINIAI ŽODŽIAI: objektų aptikimas, slenkstis, filtras

ABSTRACT

The work deals with underwater object detection algorithms and software tools to implement them. Underwater object recognition algorithms for finding relevant submerged object, seabed artifacts, marine flora and fauna examples. This work investigates thresholding methods and their characteristics when they are being influenced by different filtering parameters. Software tools developed and tested using Microsoft Visual Studio system, together with the OpenCV image processing library using KU Underwater tracking robot (ROV) video.

KEYWORDS: object detection, threshold, filter

TURINYS

PAVEIKSLŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

ĮVADAS

Spalvos daro didelę įtaką automatizuotame objektų aptikime, tai ypač matyti naudojantis specializuota programine įranga. Jeigu kompiuteris gali fiksuoti spalva, tampa įmanoma aptikti objektus, kurie pasižymi skirtingom spalvomis ir skiriasi nuo fono. Sakoma, kad spalva yra žmogaus sugalvota sąvoka siekiant atskirti kitaip atrodančius objektus ir juos charakterizuoti [1]. Programose spalvas galima apibrėžti kaip tam tikrus skaitinių reikšmių intervalus vaizdo matricose. Regimos šviesos spektras yra ištisinė juosta su tolygiai kintančiomis reikšmėmis. Tam tikra spalva atitinka užsibrėžtą reikšmių intervalą regimos šviesos spektre ir ją atitinkančioje matricoje. Reikšmių intervalas dar priklauso nuo imtuvo ypatybių: jautrumo, skiriamosios gebos, naudojamų technologijų, trikdžių lygio ir jų pobūdžio. Spalvoto objekto aptikimui reikia žinoti spalvos reikšmių intervalus, bei parinkti filtrų slenksčius ir atidžiai nustačius slenksčius, fiksuoti spalvotus objektus, išryškinti jų kontūrus.

Kituose, tolimesniuose skaitmeninio vaizdo apdorojimo etapuose (segmentavimo, atpažinimo, klasifikavimo ir kt.) tiriamojo vaizdo charakteristikos keičiasi ir ne visada lengvai prognozuojamos. Tai daro neigiamą įtaką tolesniems objektų aptikimo ir atpažinimo procesams. Be to, povandeniniuose vaizduose gausu trikdžių dėl vandenyje esančių ištirpusių kietųjų medžiagų, blogo apšvietimo arba tiriamoje vietoje augančios floros, kuri gali daryti įtakos aptinkant objektus. Ne mažesnė problema yra vaizdo priėmimo ir perdavimo aparatūros įnešami triukšmai, todėl ypatingą svarbą įgyja tinkamas filtrų ir jų esminių charakteristikų pasirinkimas.

Yra žinoma daug mokslinių tyrimų šia tema ir yra pasiekta neblogų rezultatų [1][2], tačiau dar nėra sukurta nė vieno tobulai veikiančio povandeninių objektų aptikimo algoritmo, kuris nedarytų klaidų spalvotų objektų aptikimo procese [2]. Šiame darbe bus bandoma prisidėti prie tos problemos sprendimo. Tyrimui, informacijai rinkti ir vertinti bus naudojami įvairių straipsnių ir literatūros analizės metodai. Sistemos kūrimo etape bus taikoma universalių algoritmų adaptacija prie turimos medžiagos ir algoritmų atnaujinimas unikalioms užduotims įgyvendinti. Darbe bus naudojami lyginamosios analizės, kontent (turinio) analizės ir eksperimentavimo metodai. Lyginamosios analizės metodas bus naudojamas įvairių algoritmų atrankai, aprašytai patirčiai apibendrinti.

Hipotezė: mažas pridėtinis triukšmas gali padėti pagerinti objektų aptikimą esant tam tikriems slenksčiams.

Tyrimo objektas: povandeninių objektų aptikimo algoritmai.

Darbo tikslas

Atlikti povandeninių objektų aptikimo algoritmų tyrimą esant skirtingiems jautrumo, atkirtos, intensyvumo ir kitiems vaizdų filtravimo slenksčiams bei nustatyti tiriamųjų algoritmų charakteristikas su netipiniais filtravimo parametrais.

Darbo uždaviniai:

1. Išanalizuoti moksliniuose jūros dugno vaizdų tyrimuose dažniau naudojamas povandeninių objektų aptikimo priemones ir algoritmus, aprašyti jų elgseną ir palyginti būdingas charakteristikas.
2. Pasiūlyti tam tikrų klasių povandeninių objektų aptikimo algoritmų patobulinimus, naudojant skirtingus slenksčio nustatymo metodus, vaizdų filtrų charakteristikas, patikslintas parametrų reikšmes.
3. Eksperimentiškai patikrinti, įvertinti ir pateikti atnaujintų algoritmų programinės įrangos pavyzdžių.

Darbo naujumas

Patobulinus povandeninių objektų aptikimo algoritmus siekiama sumažinti triukšmų įtaką tiriamame vaizde iki minimumo. Šio darbo naujumas pasižymi tuo, kad algoritmų tyrimas bus atliekamas pridedant mažus triukšmus skaitmeninių filtrų pagalba prieš atliekant tolimesnį vaizdų apdorojimą, o gauti rezultatai vykdant objektų aptikimą bus lyginami su duomenimis gautais naudojant kitus filtrus ir slenksčius. Povandeninių vaizdų aptikimo priemonės kurtos Visual C++ programavimo aplinkoje pasinaudojant OPENCV biblioteka.

1. POVANDENINIŲ VAIZDŲ TYRIMO METODAI

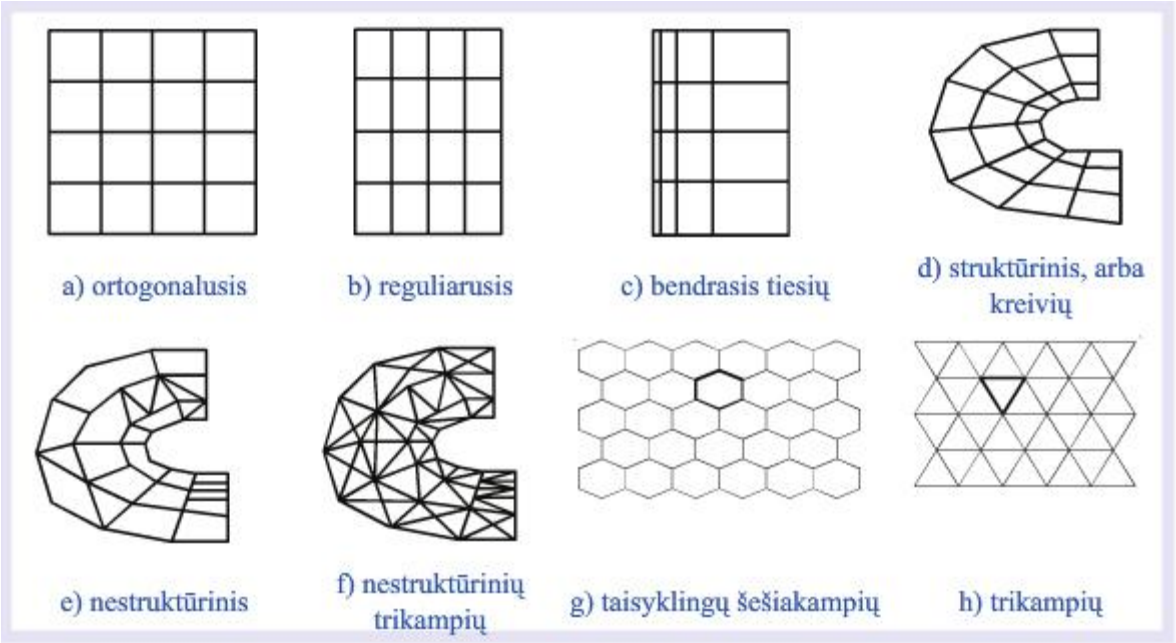
1.1. Skaitmeninis vaizdas

Vektorinės grafikos grafiniai duomenys pateikiami bazinių geometrinių elementų, vadinamų primityvais, aibe. Paprastai tai būna nesudėtingų formų figūros: taškai, stačiakampiai, daugiakampiai, lankai, elipsės, kubai, gretasieniai, sferos, kūgiai ir kt., kurių yra žinomi matematiniai modeliai, apibrėžiantys jų formą, padėtį ir dydį. Be to, vektorių faile saugomi primityvų vaizdavimo atributai: spalva, užpildymo tipas, ribinio kontūro storis ir kt. Kaip specifinę vektorinės grafikos sritį galima išskirti fraktalinę grafiką. Vektorinės grafikos pranašumai: maža duomenų apimtis, maža skaičiavimų apimtis, nesudėtinga keisti mastelį, didelis tikslumas, paprasta transformuoti į rastrinę grafiką, nesudėtinga redaguoti vaizdus, geresnė spausdinimo kokybė [4].

Jūros dugno atvaizduose gautos informacijos išskyrimo, identifikavimo ir transformavimo procesas gali būti priskiriamas mašininės regos procesams, todėl skirstomas į šešis pagrindinius etapus [3]:

- informacijos priėmimas (optinio signalo suvokimas ir regimojo atvaizdo formavimas);
- pirminis informacijos apdorojimas (sumažinamas triukšmų poveikis, pagerinamas atvaizdo kontrastiškumas, atliekama iškraipymų korekcija, informacijos sutankinimas ir kt.);
- segmentacija (atvaizdo reikalingų elementų, jų fragmentų arba būdingų ypatybių išskyrimo procesas);
- atvaizdo aprašymas (nustatomi būdingi parametrai, pavyzdžiui, matmenys, forma, orientacija, pagal kuriuos objektas išskiriamas kitų fone);
- vaizdo analizė ir atpažinimas (leidžia identifikuoti objektus, pavyzdžiui, veržlių raktą, varžtą, variklio bloką);
- darbo scenos interpretavimas (padeda nustatyti, kokiai grupei priklauso atpažįstamieji objektai).

Rastrinės grafikos grafiniai duomenys pateikiami abstrakčios diskrečiosios erdvės, vadinamos rastru, elementų aibe. Labiausiai paplitęs variantas – dvimatis rastras, nurodomas reguliariuoju ortogonaliuoju tinkleliu (stačiakampės matricos), sudarytų iš vienodų elementų (1 pav.).



1 pav. Rastinio tinklelio tipai

Kiekvienas diskretinis sistemos elementas vadinamas vaizdo elementu. Tarkim, kad $f(0,0)$ yra atvaizdo koordinačių pradžios vaizdo elementas, $f(0,1)$ – jo dešinysis vaizdo elementas ir t.t. Praktikoje *N*, *M* dydžiai ir kiekvieno vaizdo elemento diskretinis intensyvumas išreiškiamas skaičiais, kurie yra skaičiaus 2, pakelto sveikuoju laipsniu, rezultatas. Skiriamoji geba pagal skaistį – tai vaizdo elemento skaisčio diskretinių verčių skaičius. Jeigu $f(x,y)=16$, tai atvaizdas turi 16 intensyvumo (skaisčio) gradacijų arba 4 bitų skiriamąją gebą pagal intensyvumą.

Norint gauti skaitmeninį atvaizdą, tolydinis atvaizdas tolygiai diskretizuojamas į *N* eilučių ir *M* stulpelių ir kiekvienas diskretinis dydis kvantuojamas pagal intensyvumą. Tada skaitmeninį atvaizdą bendruoju atveju galima matematiškai išreikšti tokia forma:

$$f(x,y)=\begin{pmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & f(1,1) & \dots & f(1,M-1) \\ \dots & \dots & \dots & \dots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{pmatrix} \qquad (1)$$

čia *x* ir *y* – diskretiniai kintamieji, kurie nurodo atitinkamai eilutės ir stulpelio numerį $x-0,1,2,\dots N-1;\ y=0,1,2,\dots M-1$.[5]

Vaizdo komponentės, naudojant visuotinai paplitusį RGB spalvų modelį gali būti išskaidomos tokiu pavidalu [6]

$$\begin{cases} R = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1m} \\ r_{21} & r_{22} & \dots & r_{2m} \\ \dots & \dots & \dots & \dots \\ r_{n1} & r_{n2} & \dots & r_{nm} \end{pmatrix}, r_{i,j} \in [0,...,255] \\ G = \begin{pmatrix} g_{11} & g_{12} & \dots & g_{1m} \\ g_{21} & g_{22} & \dots & g_{2m} \\ \dots & \dots & \dots & \dots \\ g_{n1} & g_{n2} & \dots & g_{nm} \end{pmatrix}, g_{i,j} \in [0,...,255] \\ B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{pmatrix}, b_{i,j} \in [0,...,255] \end{cases} \quad (2)$$

Pilkos skalės vaizdas turi tik vieną komponentę – intensyvumą. Norint gauti pilkos skalės vaizdą, kiekvienos spalvinės komponentės (R, G, B) intensyvumas yra sumuojamas pagal formulę [7]

$$I = 0{,}2126R + 0{,}7152G + 0.0722B \quad (3)$$

Koeficientų reikšmės priskirtos pagal tipinį žmogaus akies jautrumą spalvoms, didžiausias jautrumas žaliai spalvai, o mažiausias mėlynai. Skaitmeninio vaizdo apdorojimo ir pateikimo praktikoje naudojami ir kiti spalvų modeliai, pvz.: CMYK, HSL, HSV, YUV, ... susiejami tarpusavyje sudėtinga vaizdo komponenčių ir funkcinių koeficientų tarpusavio maišymo struktūra. Priklausomai nuo pasirinkto spalvų modelio, jo atskirų komponenčių koeficientų reikšmės gali būti nustatomos skirtingai. Per didelis spalvų kiekis gali suklaidinti naudotoją [8].

Daugelis straipsnių autorių sutaria, kad specifinėms vaizdo detalėms ir dugno mozaikos fragmentams pavaizduoti bei atpažinti patogiau naudoti nespalvotą vaizdą, kuri patogu suformuoti naudojant ne RGB, o sintetinti iš kitų paminėtųjų modelių matricinio pavidalo. Čia gali būti naudojamos kitos skaitmeninio vaizdo modelio komponentės: sodrumas nurodytų, kiek chromatinė spalva gali būti plečiama, kol tampa nechromatine (kuo daugiau baltos, juodos ar pilkos spalvos yra primaišoma, tuo mažiau sodri ji tampa, pilkos spalvos sodrumas yra lygus nuliui); kontrastas yra objekto ir jo fono skleidžiamų šviesų santykis (ta pati pilka spalva kitaip atrodo skirtinguose fonuose); skaistis – tai šviesa, atspindėta nuo objekto paviršiaus ir matuojama kandelomis kvadratiniame metre (cd/m^2). Esant didesniam skaisčiui, akis geriau išskiria objekto detales.

1.2 Segmentacija

Teisingas vaizdo segmentavimas yra vienas iš svarbiausių uždavinių vaizdų tyrimo srityje. Šio proceso metu vaizdas yra suskaidomas į daugybę mažesniu segmentų, kurie yra reikšmingi

iškelto uždavinio sprendime. Segmentacijos užduotis yra supaprastinti esamą vaizdą taip palengvimanr tolesnius veiksmus objektų tyrimo srityje. Segmentavimas yra atliekamas prieš objektų aptikimą, t. y. vaizdas iš pradžių segmentuojamas, o po to apdorojami rezultatai. Segmentavimo algoritmus galima suskirstyti į 3 klases [9, 10]:

- Segmentavimas remiantis pikselių intensyvumu, tai atliekama analizuojant intensyvumo histogramas.
- Segmentavimas remiantis sritimis, kuris atliekamas skaidant vaizdą į sritis, turinčias panašias savybes, t. y., nagrinėjamos savybės tam tikroje pikselio aplinkoje ir sritis aplink pikselį auginama tol, kol gretimi pikseliai turi panašias savybes.
- Segmentavimas remiantis sričių kraštais, t. y. naudojantis anksčiau aprašytais kraštų radimo algoritmais randamas uždaras srities kraštas, kuris apibrėžia srities vidų ir išorę.

1.2.1. Priekinio plano segmentacijos algoritmas

„GrabCut“ algoritmas [11] yra vaizdų segmentavimo metodas, grįstas „GraphCut“ algoritmu [12]. Šis metodas suteikia galimybę išgauti priekinį vaizdo planą (angl. foreground) aplinkoje, iš kurios įprastai tai padaryti yra sudėtinga. Algoritmas iš pradžių atlieką grubią segmentaciją, panašią kaip ir „GraphCut“ algoritmas. Tuomet yra paskaičiuojamos alfa kanalo reikšmės segmentuojamos srities kraštuose.

Šiam metodui reikalingas vartotojo įsikišimas: vartotojas turi pats pažymėti sritį kurią nori segmentuoti, pavyzdžiui, kaip siūlo Lukas ir Ramašauskas (2011)[15]. Todėl šis algoritmas yra nėra geriausias pasirinkimas kuriant automatizuotą objektų atpažinimo sistemą. Taip pat šis metodas klaidingai segmentuoja vaizdus, kai segmentuojamas objektas yra susiliejęs su aplinka, pvz., stebimas objektas su vandens paviršiumi ar dangumi, o jei atliktume segmentacijas su daugiau triukšmo turinčiais (ROV kabelio triukšmas, aplinkos užterštumas, etc.) vaizdais, gaunami rezultatai būtų dar prastesni.

1.2.2. Foninės segmentacijos algoritmas

Šis algoritmas [14] gali atlikti vidutiniško sunkumo segmentacijas. Prieš pradedant segmentacija yra reikalinga nurodyti bent po kelis priekinio vaizdo objekto ir fono objekto taškus (angl. seeds). Tai dažniausiai padaroma spragtelėjus ant pasirinktų pikselių arba pažymint juos skirtingu brūkšniu, pvz. teptuku skirtingomis numatytomis spalvomis pažymint pagrindinį objektą ir foną. Deniz Kumlu bandymų metu nustatė, kad norint pasiekti geresnius rezultatus reikalinga parinkti bent 30 taškų [12].

Šis algoritmas pakankamai gerai segmentuoja daugumą vaizdų, tačiau kaip ir priekinio plano algoritmas, yra klaidinamas vaizdų su objektais, kurių fonas (angl. *backgroud*) yra dangus ir/ar vanduo. Taip pat tai, kad šis ir ankstesnis algoritmai prašo operatoriaus pagalbos atskleidžia

algoritmo trūkumą pačiam nustatyti objektus tiriamame vaizde. Šio algoritmo principą Denis Kumlu (2012) aiškina taip:

Algoritmas gali būti atskleistas intuityviu paaiškinimu. Šie taškai (angl. *seeds*) gali būti laikomos kaip bakterijos, kurios pradeda sklisti iš tų taškų ir bando užimti visą vaizdą po pikselį. Kiekvieną kartą „bakterija“ atakuoja savo kaimyną (gretimą pikselį), jei ji stipresnė – užimą jį. Galiausiai visi pikseliai užimti ir sugrupuoti pagal savo stiprumą – fono pikselis ar priekinio vaizdo pikselis [12].

1.3. Slenkstis

Daugybėje objektų aptikimo programų naudojančių skaitmeninius vaizdus slenkstis yra viena iš pagrindinių objektų aptikimo algoritmų sudedamųjų dalių. Slenkstis yra vienas iš paprasčiausių vaizdo segmentacijos metodų. Slenksčio metodo pagalba galima iš pustonių paveikslo gauti binarinį vaizdą [16]. Juodos spalvos pikseliai turintys reikšmę 0 sudaro foną. Baltos spalvos pikseliai turinys reikšmę 255 atstovauja aptiktus objektus. Slenksčio metodo pagalba galima lengvai atskirti kurie regionai tiriamame vaizde yra nustatytame diapazone, todėl galima lengvai išskirti norimos spalvos regionus o likusius ignoruoti ir paversti į foną. Pilkos skalės vaizdas turi tik vieną komponentę – intensyvumą pagal šios komponentės reikšmę ir nustatytą slenksčio reikšmę bei pasirinkto slenksčio tipą tiriamas pikselis gaus 0 arba jeigu netenkina iškeltos sąlygos arba 255 reikšmę jeigu tenkina iškelta sąlyga. Kuo didesnis vaizdo kontrastas tuo efektyvesnis slenksčio metodas.

Slenksčio metodus pagal jų analizuojamus duomenis galima skirtyti į šešias grupes[17]:

- Histogramos formos metodai. Šiais metodais yra analizuojamos tokios histogramos charakteristikos kaip viršūnės, slėniai arba histogramos kreivumas.
- Klasteriais paremti metodai. Šie metodai grupuoja pilko lygio mėginius į du klasterius: foną ir objektą arba gali būti modeliuojami kaip dviejų Gauso funkcijų mišinys.
- Entropija paremti metodai.
- Objekto atributais paremti metodai. Šie metodai ieško panašių formų, kraštų, t.t.
- Erdviniai metodai. Šie metodai naudojasi aukštesnės eilės tikimybės pasiskirstymą ir/arba koreliacija tarp pikselių.
- Vietiniai metodai. Šie metodai sugeba pritaikyti prie kiekvieno pikselio charakteristikų ir nustatyti jam skirtą slenksčio reikšmę.

Populiariausias ir dažniausiai naudojamas yra histogramos metodas. Yra įmanoma naudoti skirtingus slenksčio metodus atskiriems skaitmeninio vaizdo kanalams ir nustatyti tam tikrą erdvinę figūrą kurios ribose būtų tiriamo objekto spalva.

Slenksčio algoritmai pagal jų elgseną gali būti skirstomi į dvi klases:

- visuotinio slenksčio algoritmai
- vietiniai adaptyvūs algoritmai

Visuotinio slenksčio algoritmai naudoja vienoda slenksčio reikšmę visiems paveikslo pikseliams analizuoti. Pasirinkta slenksčio reikšmė gali daryti didelę įtaką objektų aptikime. Šis metodas puikiai tinka analizuoti vaizdams esant vienodam apšvietimui, bet susiduria su dideliais sunkumais naudojant aptikti objektus jeigu vaizdo apšvietimas skiriasi. Objektų aptikimas gali sumažėti iki nulio tuose zonose kuriose apšvietimo reikšmė skiriasi. Netgi mažas pokytis scenos apšvietime gali daryti įtakos objektų aptikimui esant tam tikriems slenksčiams.

Šia problema galima išspręsti pasinaudojant scenos apšvietimo suvienodinimo algoritmais arba naudojantis vietiniais adaptyvaus slenksčio algoritmais.

Vietiniai adaptyvūs slenksčio algoritmai apskaičiuoja atskira slenksčio reikšmę kiekvienam pikseliui. Tiriamas vaizdas yra suskirstomas į daugybę mažesnių plotų kuriems apskaičiuojamos atitinkamos slenksčio reikšmės priklausomai nuo esančio apšvietimo arba šešėlių.

1.3.1. Slenksčių tipai

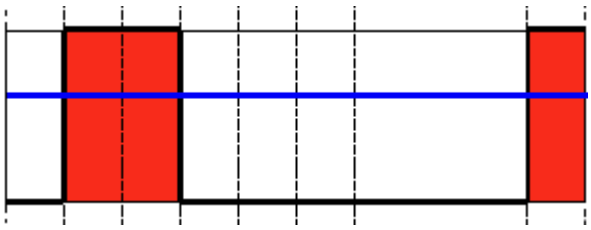
OpenCv vaizdų manipuliavimo biblioteka suteikia galimybę pasinaudoti penkiomis funkcijomis visuotiniam slenksčiui įvykdyti[18]:

- dvejetainis slenkstis
- dvejetainis atbulinis slenkstis
- nupjautinis slenkstis
- nulinis slenkstis
- nulinis atbulinis slenkstis

Dvejetainis slenkstis yra paprasčiausia slenksčio funkcija. Visi pikseliai kurių intensyvumo reikšmė yra didesnė už slenksčio nustatyta reikšmę gauna maksimalią galima reikšmę. Jeigu pikselio reikšmė yra mažesnė negu nustatyta slenksčio reikšmė, pikeliui yra priskiriama reikšmė 0. Slenksčio grafinę reprezentacija galima pamatyti žemiau pavaizduotame paveiksle 2. Slenkstį galima užrašyti tokiu pavidalu:

$$pikselis(x,y) = \begin{cases} maxVal & if\ pikselis(x,y) > slekstis \\ 0 & kitu\ atveju \end{cases} \quad (4)$$

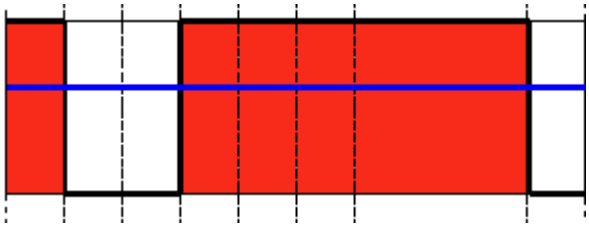
pikselis(x,y) yra tiriamo pikselio reikšmė, *slenkstis* – nustatyta slenksčio reikšmė, *maxVal* didžiausia galima reikšmė kuria pikselis gali įgyti. Jeigu ši reikšmė nėra atskirai nustatyta ji turi reikšmę 255.



2 pav. Dvejetainis slenkstis[18]

Dvejetainis atbulinis slenkstis veikia atvirkščiai negu dvejetainis slenkstis. Šio atveju jeigu tiriamo pikselio reikšmė yra didesnė negu nustatyta slenksčio reikšmė, pikseliui yra priskiriama reikšmė 0. Kitu atveju yra priskiriama maksimali galima reikšmė. Slenksčio grafinę reprezentacija galima pamatyti žemiau pavaizduotame paveiksle 3. Slenkstį galima užrašyti tokiu pavidalu:

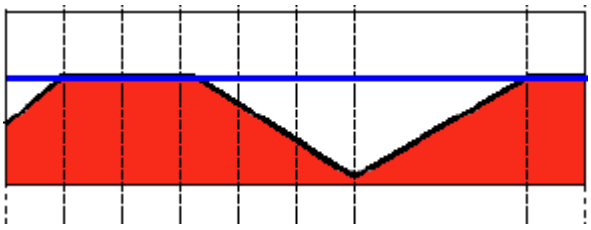
$$pikselis(x,y) = \begin{cases} 0 & if\ pikselis(x,y) > slekstis \\ maxVal & kitu\ atveju \end{cases} \quad (5)$$



3 pav. Dvejetainis atbulinis slenkstis[18]

Nupjautinis slenkstis išlaiko tiriamo pikelio reikšmę jeigu ji yra mažesnė už nustatyta slenksčio reikšmę. Jeigu tiriamojo pikselio reikšmė yra didesnė už slenksčio reikšmę, tiriamam pikseliui yra priskiriama slenksčio reikšmė. Slenksčio grafinę reprezentacija galima pamatyti žemiau pavaizduotame paveiksle 4. Slenkstį galima užrašyti tokiu pavidalu:

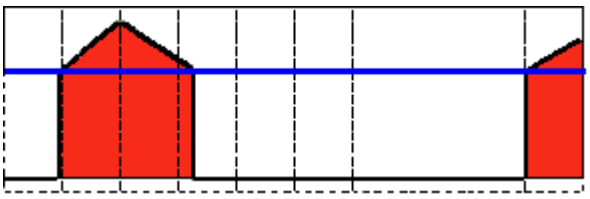
$$pikselis(x,y) = \begin{cases} slekstis & if\ pikselis(x,y) > slekstis \\ pikselis(x,y) & kitu\ atveju \end{cases} \quad (6)$$



4 pav. Nupjautinis slenkstis[18]

Nulinis slenkstis išlaiko tiriamo pikselio reikšmę jeigu ji yra didesnę už nustatytą slenksčio reikšmę. Jeigu tiriamojo pikselio reikšmė yra mažesnė už nustatyta slenksčio reikšmę, pikseliui bus priskirta reikšmė 0. Slenksčio grafinę reprezentacija galima pamatyti žemiau pavaizduotame paveiksle 5. Slenkstį galima užrašyti tokiu pavidalu:

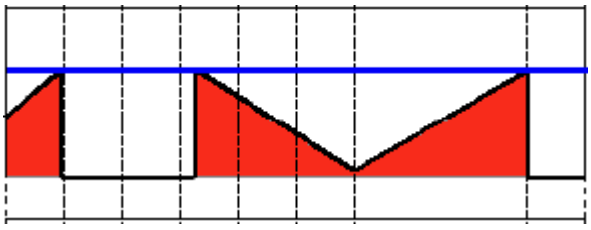
$$pikselis(x,y) = \begin{cases} pikselis(x,y) & if\ pikselis(x,y) > slekstis \\ 0 & kitu\ atveju \end{cases} \quad (7)$$



5 pav. Nulinis slenkstis[18]

Nulinis atbulinis slenkstis suteikia visiems pikseliams, kurių reikšmė yra didesnė už nustatytą slenksčio reikšmę, reikšmę 0. Jeigu tiriamojo pikselio reikšmė yra mažesnė už nustatytą slenksčio reikšmę, pikselio reikšmė nekinta. Slenksčio grafinę reprezentacija galima pamatyti žemiau pavaizduotame paveiksle 6. Slenkstį galima užrašyti tokiu pavidalu:

$$pikselis(x,y) = \begin{cases} 0 & if\ pikselis(x,y) > slekstis \\ pikselis(x,y) & kitu\ atveju \end{cases} \quad (8)$$



6 pav. Nulinis atbulinis slenkstis[18]

Atkirtos ir nuliniai slenksčiai gali būti panaudojami pradiniam vaizdo apdorojimui norint iššskirti objektus iš objektų visumos, kurių reikšmės yra intensyvumo spektro viduryje jeigu prieš arba už jų yra kiti objektai kurie gali daryti įtakos objektų aptikimui, prieš atliekant dvejetainio slenksčio operacija.

1.3.2. Histogramos metodas

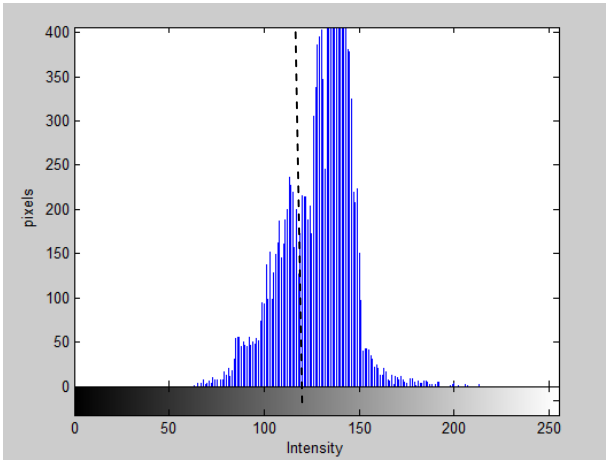
Histogramos pavidalu paremti metodai analizuoja turiamo vaizdo histogramą. Dažniausiai yra dirbama su vaizdais pervestais į pilkumo skalės formatą. Tokiu atveju yra analziuojamas pikselių intensyvumas ir sudaroma intensyvumo histograma. Pagal gautus duomenis galima spresti ar galima sėkmingai parinkti tinkamą slenksčio reikšmę norimėms objektams rasti. Tam kad šis metodas sėkmingai veiktų fono ir tiriamų objektų intensyvumo reikšmės turi žymiai skirtis.

Jeigu aukščiau išsakyta prielaida apie intensyvumų skirtumą tarp fono ir objektų intesyvumų galioja, tada histogramoje galima pamatyti viršūnę kuri žymi ieškomo objekto intensyvumo reikšmes. Tada galima pasirinkti tam tikrą slenkstį ir izoluoti šią viršūnę. Jeigu aukščiau išsakyta prielaida negalioje tada nėra galima nustatyti teisingos slenkščio reikšmės tiriamam objektui aptikti ir gali reikėti atlikti papildus vaizdo transformavimo žingsnius prieš bandant naudotis slenksčio metodu. Taip pat galima bandyti pasinaudoti adaptyvaus slenkčio metodu. Aukščiau išsakyta prielaida dažniausiai nėra tenkinama jeigu esamos scenos apšvietimas nėra pastovus ir kinta. Tokiu atveju dalis objekto gali būti kitokio apšvietimo zonoje ir tai darys jo characteristikoms sukurtoje vaizdo histogramoje. Tokiu atveju dažniausiai bus aliekamas tik dalinis aptikimas. Žemiau pateiktuose paveiksluose 7 ir 8 galima matyti tiriamąjį objektą ir jo histograma. Kontrastas tarp

objekto ir fono yra mažas. Tai taip pat galima pastebėti iš tiriamojo objekto histogramos, kuri yra centruota. Taip pat galima pastebėti dvi viena šalia kitos esančias viršunes. Pirma viršunė žyminti objektą nėra pilnai atskirta nuo antros viršunės todėl objekto aptikimas bus dalinis. Tai galima pastebėti ir iš pačio objekto vaizdo. Mažas kontrastingumas tarp objektų ir fono yra dažna charackteristika povandeninių objektų aptikime, nes tiriami objektai gali būti užnesti smėliu arba apauge vandens flora. Norima optimali slenksčio reikšmė šiam konkrečiam objektui bus slėnyje tarp tų dviejų viršūnių.

Priklausomai nuo tiramo objekto ir fono charackteristikų, tiriamą objektą dažniausiai atstovauja pirmoji viršūnė gautoje histogramoje. Jeigu tiriamąjį objektą atstovaujanti viršūnė yra histogramos centre, gali prireikti atlikti papildomas slenksčio operacijas norint pašalinti nereikalinga informacija iš vaizdo ja paverčiant fonu.

7 pav. Objektas 1, akmuo



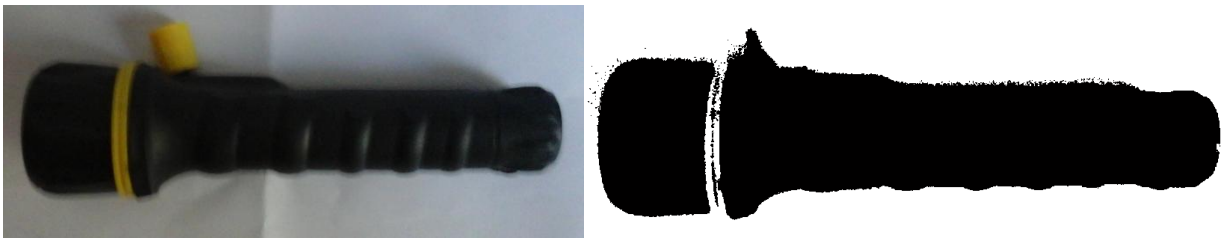
8 pav. Objekto 1 histograma

Kadangi negalima tiksliai nustatyti slenksčio ribos algoritmas atliks tik dalinį objekto aptikimą. Šio atveju gali būti aptikta tuščia vidurė figura arba tik jos dalis. Gauta rezultatą atlikus slenksčio operacija galima pamatyti žemiau esančiame paveiksle 9.

9 pav. Objekto 1 vaizdas po slenksčio panaudojimo

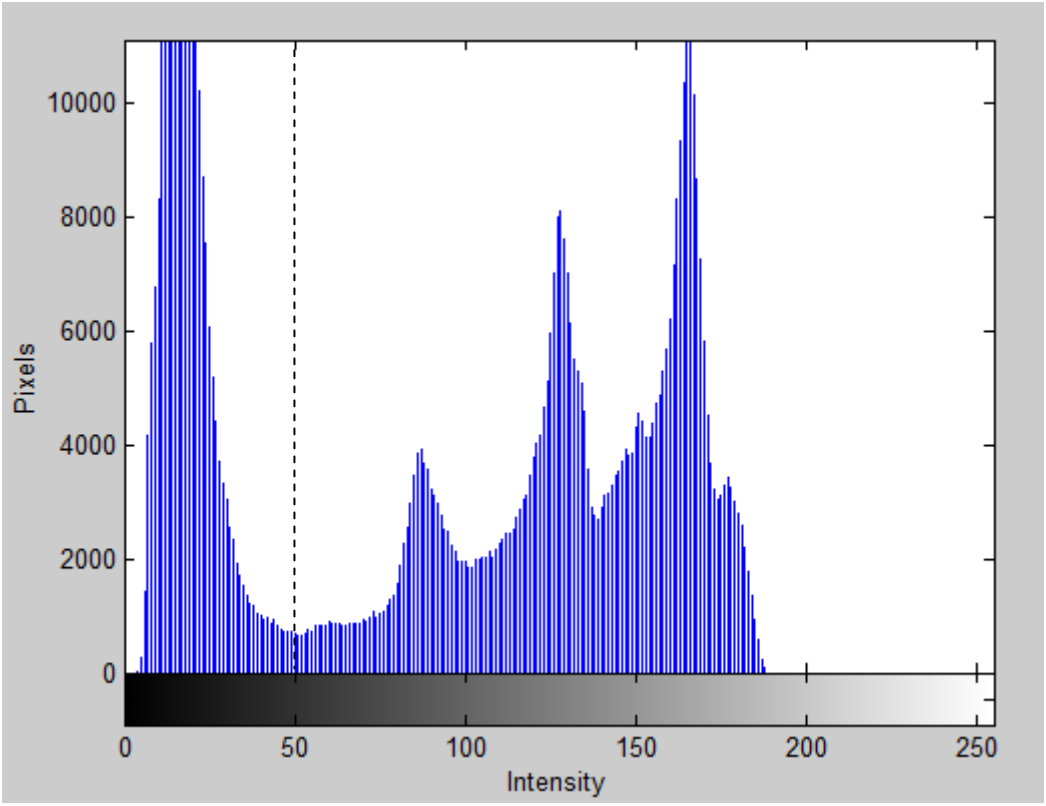
Šio atveju aptiktas objektas yra turi reikšmę 0 o fonui yra suteikta reikšmė 255. Kaip matoma iš paveikslų 7 ir 9 objekto vidus pasižymi beveik tokia pačia intesyvumo reikšme kaip ir fonas todėl yra priskiriamas fonui. Šio atveju geriau naudoti kitą metodą.

Jeigu objektas turi didelį kontrastą su fonu (paveikslas 10) tada jis bus pilnai aptiktas net jeigu fono intesyvumas nėra vienuodas. Kaip pavizdys paimtas juodas elektrinis žibintas.



10 pav. Objektas 2

Objektas 2 ir jo rezultatas atlikus sleksčio operaciją matomas aukščiau esančiame paveiksle 5. Dėl didelio kontrasto su fonu galima gerai atlikti slenksčio operacija nepaliekant jokių skylių tiriamame objekte. Objekto 2 histograma galima matyti žemiau patektame paveiksle 11. Matoma aiškiai išskirta pirma viršūnė kartu su kitomis trimis mažesnėmis viršūnėmis.



11 pav. Objektas 2 histograma

1.3.3. Otsu metodas

Otsu metodas yra klasteriais apremtas metodas naudojamas automatiniam slenksčio nustatymui esant visuotiniam slenksčiui. Šis metodas paremtas prielaida, kad tiriamas vaizdas yra bimodalinis t.y. vaizdo histograma turi tik dvi viršunes [17]. Šios viršunės atitinkamai atstovauja pikselius atitinkančius objektus ir foną. Metodas apskaičiuoja optimalią slenksčio reikšmę esančia

tarp dviejų klasių tokia, kad jų bendra slaida būtų mažiausia, o tarpusavio dispersija būtų didžiausia [19]. Šis metodas geriausiai veikia su pilkumo skalės vaizdai kai kontrastas tarp fono ir objektų yra didelis. Jeigu vaizdas nėra bimodalis, šis metodas bus neefektyvus arba veiks neoptimaliai. Tam kad padidinti šio metodo efektyvumą, galima pasinaudoti vaizdo filtravimo metodais kurie bus aprašyti vėlesniose skyriuose.

Otsu algoritmą galima užrašyti taip:

- suskaičiuojama tiriamo vaizdo histograma ir visas intensyvumo galimybes
- apskaičiuojamos pradinės reikšmės $q_i(0)$ $\mu_i(0)$
- ciklo operacija skirta apskaičiuoti visiems galimų slenksčių reikšmėms σ_i
- atnaujinam reikšmes $q_i(0)$ $\mu_i(0)$ konkrečiam slenksčiui
- surandama didžiausia reikšmė σ

Šio algoritmo matematinė forma yra užrašoma taip

$$\sigma_w^2(t) = q_1(t)\sigma_1^2 + q_2(t)\sigma_2^2(t) \tag{9}$$

$$q_1(t) = \sum_{i=1}^{t} P(i) \quad \& \quad q_1(t) = \sum_{i=t+1}^{I} P(i) \tag{10}$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \quad \& \quad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)} \tag{11}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \& \quad \sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_1(t)]^2 \frac{P(i)}{q_2(t)} \tag{12}$$

Šis metodas puikiai veikia naudojant aptikti į objektas 2 panašius objektus, nes jų histograma turi lengvai atskiriamas viršūnes, bet jis veikia vidutiniškai esant objekto 1 histogramai kai nėra galima tiksliai nustatyti ribos tarp foną ir objektą atstovaujančių viršunių. Jeigu negalima tiksliai nustatyti skirtumo tarp objekto ir fono arba tiriamame paveiksle esama trigdžių šis metodas netikslingai nustatys slenksčio reikšme ir padarys segmentacijos klaidų.

Egzistuoja daugybė šio metodo patobulinimų skirtingiems jo trūkumams sumažinti arba visai pašalinti. Populiariausias iš jų yra dviejų dimensijų Otsu metodas. Šio metodo metu kiekvieman pikseliui yra apskaičiuojamas vidutinė jo kaimynių reikšmė. Ši reikšmė yra suporuojama su esama pikselio reikšme. Porų kiekis padalintas iš visų pikselių skaičiaus apibriežia tikybės funkcija vęliau naudojama Otsu metode [20].

1.4. Filtrai

1.4.1. Vidurkio filtras

Vienas iš paprasčiausių ir dažniausiai naudojamų yra vidurkio filtras. Šis filtras yra paremtas aplinkui esančių pikselių vidurkiu. Filtras naudojamas pilkumo skalės vaizdams ir padeda sumažinti intensyvumo skirtumą tarp kaimyninių pikselių. Šis filtras pakeičia tiriamojo pikselio reikšmę į reikšmę visų aplink jį esančių pikselių įskantant tiriamąjį pikselį vidurkiu. Vidurkio filtras padeda sumažinti vaizde esantį triukšmą, nes eliminuoja dideles netinkamas reikšme tiriamame vaizde. Jis puikiai tinka pašalinti Gauso triukšmą. Vidurkio filtras yra tipiškas konvoliucijos filtras. Dažniausiai šis filtras yra naudojamas su 3x3 gardele, bet jis taip pat gali būti naudojamas su 5x5 arba didesne gardele. Esant didesnei gardelei galima pasiekti didesnį paveikslo švelninimo laipsnį. Šis filtras gali būti pritaikomas daugybę kartų, bet du arba daugiau kartų naudojant mažesnės gardelės filtrą nebus gauti tokie patys rezultatai kai panaudojus didesnės gardelės filtrą. Filtro grafinę reprezentacija galima pamatyti žemiau esančiame paveiksle 12. Gardelės reikšmės priklauso nuo filtro gardelės dydžio.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

12 pav. Vidurkio filtro branduolys

1.4.2. Medianos filtras

Medianos filtras veikia tokiu pačiu principu kaip ir vidurkio filtras, bet vietoj to kad skaičuotų kaimyninių pikelių vidurkį, jis skaičiuoja medianą. Visi kaimyniniai pikseliai yra surašomi į eilę didėjimo tvarka ir yra imama jų mediana. Jeigu pikselių kiekis yra lyginis skaičius tada yra paimamas vidurkis tarp dviejų vidurinių pikselių ir priklausomai nuo aplinkos kurioje dirba algoritmas, gautas rezultatas yra apvalinamas arba jo trupmeninė dalis yra pašalinama. Šis filtras yra konvoliucinis ir gali turėti skirtingas gardelės formas: 3x3, 5x5, ...

Šio filtro grafine reprezentaciją galima pamatyti žemiau esančiame paveiksle 13. Kaip matoma paveiksle 13 tiriamojo pikselio reikšmė 90 yra žymiai didesnė už visų jį supančių pikselių reikšmę beveik dvigubai. Panaudojant 3x3 gardelės formą ir suradus kaimyninių pikselių mediana jos reikšmė bus įrašyta į tiriamojo pikselio vietą.

Kaimyninių pikselių variacinė eilutė turi tokį pavidalą:

50, 50, 51, 51, 51, 52, 54, 54, 90. Medianos reikšmė yra 51 jis pakeis anksčiau buvusio pikselio reikšmę. Pradinė reikšmė 90. Reikšmė po filtro panaudojimo 51.

Medianos filtras turi du privalumus prieš vidurkio filtrą:

- Šis filtras naudoja kaimyninių pikselių reikšmes todėl atsitiktinė didelė reikšmė neturi įtakos galutiniam rezultatui.
- Šis filtras nesukuria naujų reikšmių ir padeda išsaugoti kraštų aštruma apdorojant tiriamo vaizdo kraštus.

Priklausomai nuo gauso triukšmo kiekio jis filtras gali turėti geresnius arba blogesnius rezultatus negu vidurkio filtras. Jeigu gauso triukšmo lygis yra mažas, šis filtras veiks geriau už vidurkio filtrą, bet jeigu gauso triukšmo lygis yra didelis, vidurkio filtras gali gauti geresnius rezultatus už medianos filtrą. Medianos filtras nėra tiesinis. Jis taip pat reikalauja daugiau skaičiavimo pajėgumų ir todėl yra lėtesnis negu vidurkio filtras. Tai ypač aiškiai galima pastebėti esant didesniam gardelės dydžiu negu 3x3.

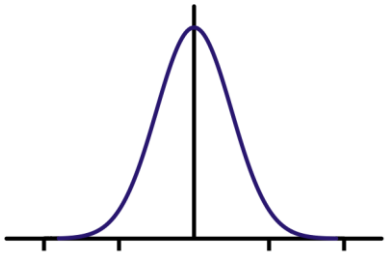
50	51	51
54	90	52
54	50	51

13 pav. Medianos filtro branduolys

1.4.3. Gauso filtras

Gauso švelninimo arba glotninimo filtras yra 2D konvoliucijos filtras. Jis savo veikimu yra panašus į vidurkio filtrą bet šio atveju naudojasi gauso forma. Nors gauso švelninimo filtras mažina triukšmų kiekį tiriamame vaizde, jis taip pat sumažinam detalių kiekį ir gali būti prarandama svarbi informacija. Gauso filtras naudojasi gauso forma duomenų manipuliacijai. Naudojamą gauso filtro matematine išraiška galima užrašyti tokiu pavidalu:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{13}$$



14 pav. Gauso funkcija

1.4.4. Sulyginimo filtras

Sulyginimo filtras geriausi veikia jeigu reikia pašalinti dideles triukšmo reikšmes tarp tiriamų pikseliu. Jis padeda pašalinti labai dideles arba mažas pikselių reikšmes kurios yra netipiškos kaimyniniams pikseliams (angl. *noise spikes*). Šio filtro veikimo principas yra paremtas prielaida, kad vaizde esantys triukšmai išsiskiria iš kaimyninių pikselių savo žymiai didesnėmis arba mažesnėmis reikšmėmis. Jis padeda iš lyginti pikselių reikšmes ir po jo panaudojimo, tiriamo vaizdo kaimyninės pikselių reikšmės mažai skiriasi viena nuo kitos. Šis filtras palygina kiekvieną tiriamojo vaizdo pikselį su kaimyninių pikselių reikšmėmis ir suranda minimalias ir maksimalias reikšmes. Jeigu tiriamojo pikselio reikšmė patenka į intervalą tarp minimalios ir maksimalios reikšmės, filtras nekeičia jo reikšmės. Jeigu tiriamojo pikselio reikšmė yra didesnė už maksimalią reikšmę tada filtras priskirs maksimalią reikšmę. Jeigu tiriama reikšmė yra mažesnė už minimalią reikšmę tada bus priskirta minimali reikšmė.

Paveikslui 13 pritaikius sulyginimo filtrą algoritmas nustatys, kad tiriama centrinė reikšmė yra didesnė už maksimalią kaimyninių pikselių reikšmę kuri yra 54 ir priskirs šia reikšmę tiriamam pikseliui. Sulyginimo filtras kitaip negu vidurkio arba medianos filtras padeda išlaikyti kraštų aštrumą ir padeda jiems neišblukti. Vidurkio filtras šiam pikseliui priskirs reikšmę 56. Medianos filtras priskirs reikšmę 51. Sulyginimo filtras priskirs reikšmę 54. Visi šie filtrai turės skirtingą įtaka tiriamai gardeliai ir tai gali daryti įtaka tiksliam kraštų aptikimui. Šios gardelės atveju geriausias atrodo medianos filtras.

Sulyginimas filtras pateikia blogesnius rezultatus jeigu reikia panaikinti gauso triukšmą nes jis neslopina triukšmo, bet kitaip negu vidurkio arba medianos filtrai jis padeda išlaikyti vaizdo detališkumą, kuris būtų prarastas panaudojant vidurkio arba medianos filtrus. Šis filtras suteikia geriausius rezultatus pašalinat intensyvumo triukšmą kurio lygis yra žemas. Jeigu tiriamos gardelės aplinkoje yra daugiau negu vienas sugadintas pikselis, šio filtro efektyvumas sumažės.

1.4.5. Crimmin‘s dėmelių šalinimo filtras

Crimmin‘s dėmelių šalinimo filtras naudojasi papildomu gvildenimo algoritmu [21]. Šis algoritmas buvo sukurtas intensyvumo tokio kaip druska ir pipirai triukšmui (angl. *salt and pepper noise*). Papildomo gvildenimo algoritmas padidiną pikselių kurie yra tamsesni už kaimyninius pikselius reikšmes bei sumažinant pikselių kurie yra ryškesni už kaimyninius pikselius reikšmes. Tai padeda sumažinti dėmių indeksą tiriamame vaizde. Algoritmas palygina kaimyninių tiriamojo pikselio reikšmes ir padidina arba sumažina jų reikšmes atitinkamai.

1.4.6. Gauso - Laplaso filtras

Laplasianas dažniausiai naudojamas algoritmuose, taikomuose kampų arba lašelių aptikimui ir sujungimui į atskirus objektus, nes gerai sugeba aptikti staigius intensyvumo pokyčius tiriamame

vaizde. Toks filtras yra labai jautrus triukšmui todėl dažniausiai yra naudojamas kartu su kitais filtrais, pavyzdžiui, Gauso švelninimo filtru, po to kai triukšmo lygis tiriamame paveiksle jau yra sumažintas.

Gauso laplasianas yra vaizdo funkcijos antros eilės išvestinė. Jo matematinė išraiška gali būti užrašyta tokiu pavidalu:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{14}$$

Dažniausias šio filtro naudojamas branduolys yra parodytas žemiau esančioje matricoje:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{15}$$

Jeigu pikselių intensyvumas nekinta Gauso laplasianas grąžina 0. Jeigu yra nustatytas intensyvumo pokytis, Gauso laplasianas grąžina teigiamą reikšmę mažesnio intensyvumo pusėje ir neigiamą – didesnio intensyvumo pusėje. Regionuose, kurie žymi kraštų buvimą, Gauso laplasianas grąžina šias reikšmes:

- nulį – einant tolyn nuo krašto;
- teigiamą reikšmę – vienam krašto gale ir neigiamą – priešingame krašto gale;
- gali gražinti nulį krašto riboje, nes intensyvumas nebepasikeičia.

1.5. Skyriaus apibendrinimas

Šiame skyriuje buvo išanalizuoti vaizdų apdorojimo ypatumai tokie kaip vaizdo segmentacija ir filtrai darantys įtaka tolesniuose algoritmų žingsniuose. Tai padeda žinoti savitus privalumus sprendžiant konkrečius povandeninių vaizdų apdorojimo uždavinius. Apžvelgtas bazinis segmentavimo ir filtrų sąryšis bei jų suderinamumas tarpusavyje. Numatyti galimi sprendimų keliai tikslui pasiekti.

Atlikta konkrečių povandeninių vaizdų apdorojimo uždavinių analizė padėjo įžvelgti esminius veiksnius turinčius įtakos objektų aptikimo algoritmuose. Po vizualinių povandeninių vaizdų savybių analizės aprašyti žinomų trūkumų šalinimui naudingi metodai ir numatyti atitinkami scenarijai jų taikymui.

Pagrindinės objektų aptikimo algoritmų dalys tūrinčios didžiausios įtakos, buvo nustatytos naudojamų filtrų tipai, slenksčių tipai ir reikšmės.

Pateikta apžvalga į galimas priemones savybių apiforminimui skaitmeniniu pavidalu.

Segmentavimo ir atpažinimo metodų apžvalga leido įvertinti kiekvienos technikos pritaikymo galimybes jūros dugno vaizdo apdorojimui. Tolimesniems tyrimams tinkamiausi pripažinti segmentavimas pagal pikselių intensyvumą.

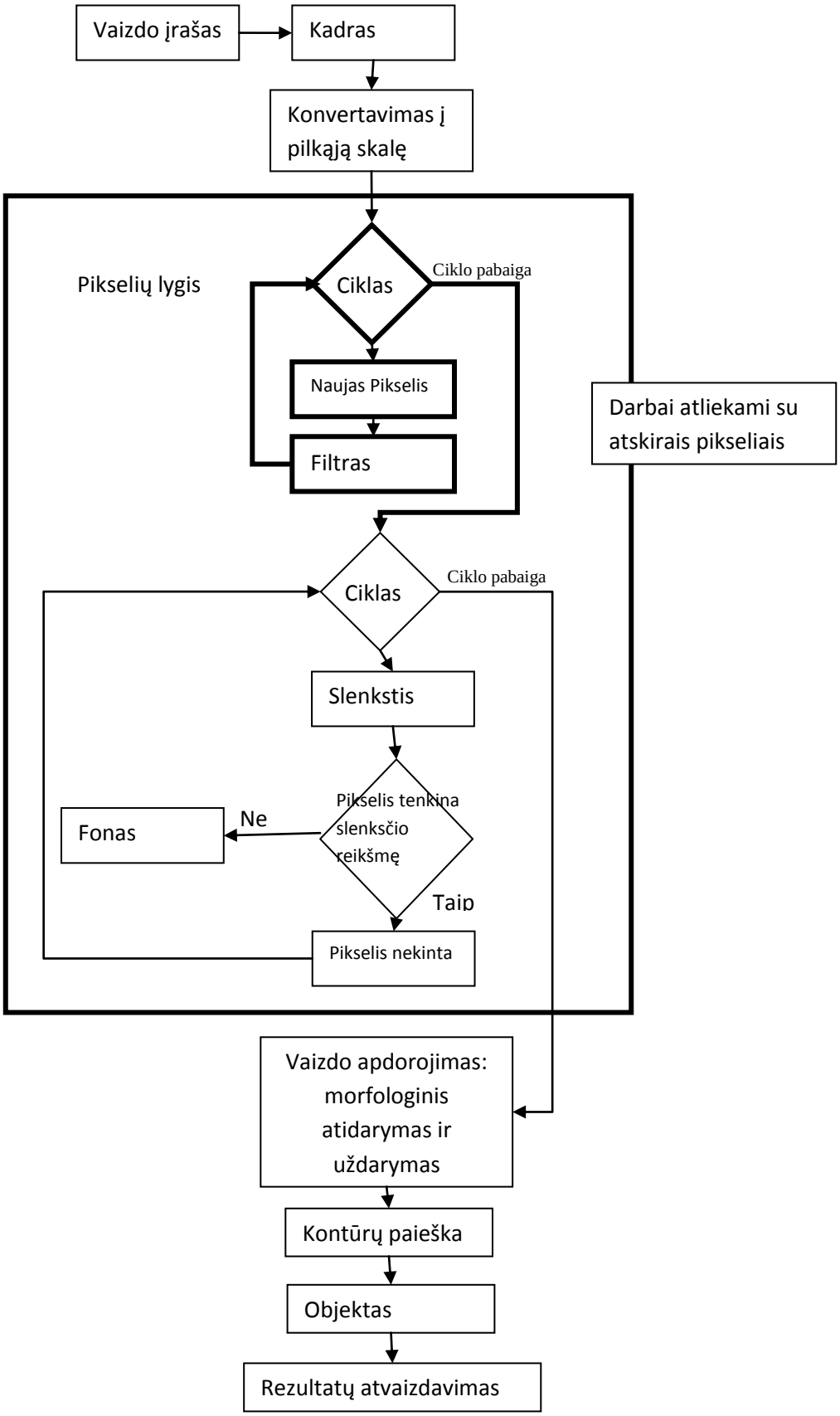
2. OBJEKTŲ APTIKIMO ALGORITMAI

2.1. Filtras

Objektų aptikimo algoritmai susideda iš filtro arba filtrų triukšmams sumažinti arba pašalinti, slenksčio nustatymo ir atlikimo metodų, vaizdo apdorojimo po slenksčio operacijos ir pikselių sujungimo į objektus metodų. Žemiau esančiame paveiksle 15 galima pamatyti algoritmo schema. Kiekvienas vaizdo įrašo kadras yra analizuojamas atskirai, bet taip pat egzistuoja priemonės rastiems objektams sujungti ir analizuoti viso vaizdo įrašo metu. Šiame darbe apie tokias priemones nebus kalbama.

Tiriamas vaizdas dažniausiai yra konvertuojamas į pilkumo skalės vaizdą paliekant tik intensyvumo kanalą, kuris bus panaudojamas filtruose sumažinti triukšmams ir/arba išryškinti objektų savybes.

Filtrai, priklausomai nuo jų prigimties pašalina triukšmą arba paryškina objektų savybes (intensyvumą) tam, kad jie būtų geriau aptinkami atlikus slenksčio operacija. Filtrai dirba pikselių lygmenyje naudodamiesi iš anksto nustatytas gardelės struktūras ir jų reikšmes. Dažniausiai yra naudojama 3x3 gardelės struktūra. Keli filtrai gali būti sujungiami į vieną filtrą arba panaudojami keletui iteracijų norint gauti geresnius rezultatus. Priklausomai nuo pasirinktų filtrų gali būti prarandama dalis informacijos. Norint gauti geresnį triukšmo slopinimą gali būti panaudoti keli skirtingi vidurkio, medianos, ...t.t. filtrai su skirtingais gardelės dydžiais.



15 pav. Objektų aptikimo algoritmas

Šiame darbe naudojami hibridiniai filtrai paremti vidurkio ir medianos principais kurie sumažina triukšmo daromą įtaka ir paryškina objektų vaizdą. Priklausomai nuo pasirinkto filtro gali būti aptinkami skirtingi objektai.

Priklausomai nuo naudojamo filtro, prieš atliekant filtravimo veiksmą gali būti sukurtas, klonuotas naujas identiškas vaizdas, kuris vėliau bus galutinis produktas gražintas kai filtras baigs darbą su pikseliais. Jeigu filtras ima pikselių reikšmes iš originalaus pateikto vaizdo ir yra sukurtas naujas vaizdas rezultatų talpinimui, gauti filtro rezultatai neturės jokios įtakos tolimesniam filtro veikimui ir jo gaunamiems rezultatams. Jeigu filtras ima rezultatus iš naujo vaizdo skirto rezultatams talpinti arba toks vaizdas nėra sukurtas, bet yra perrašomas pradinis vaizdas, tada jau esami filtro rezultatai darys įtaką tolesnėms rezultatams. Tai gali padaryti didelę įtaka kokie objektai gali būti aptikti tolesniuose algoritmo žingsniuose.

Šiame darbe naudojamų filtrų branduoliai pavaizduoti apačioje esančiame paveiksle 16.

1/8	1/8	1/8
1/8	1	1/8
1/8	1/8	1/8

a

-1/8	-1/8	-1/8
-1/8	1	-1/8
-1/8	-1/8	-1/8

b

1/9	1/9	1/9
1/9	1	1/9
1/9	1/9	1/9

c

-1/9	-1/9	-1/9
-1/9	1	-1/9
-1/9	-1/9	-1/9

d

16 pav. Filtravimo branduoliai

a atveju yra apskaičiuojamas visų kaimyninių pikselių vidurkis ir jis yra pridedamas prie esamos pikselio reikšmės. Tiriamojo pikselio reikšmė vidurkio operacijoje nedalyvauja. Didelę reikšmę turinčių pikselių reikšmė padidėja greičiau, negu maža reikšmę turintys pikseliai taip padidėja kontrastas tarp objektų. Jeigu iš tiriamojo pikselio reikšmės yra atimama iš kaimyninių pikselių vidurkio (b variantas), skirtumas tarp aplinkinių pikselių reikšmių sumažėja.

Jeigu yra leidžiama naudoti tiriamojo pikselio reikšmę (atvejai c ir d) tai gali daryti didelę įtaka galutiniam rezultatui. Didesnis pikselių kiekis naudojamas vidurkiui apskaičiuoti sumažina tikimybę kad labai didelę kaimyninė reikšmė darys įtakos vidurkio rezultatams. Priklausomai nuo rezoliucijos gauti rezultatai gali įtakoti tolesnį algoritmo veikimą. Tai ypač svarbu esant didės raiškos vaizdui.

2.2. Slenkstis

Tinkamas slenksčio parametrų parinkimas turi didelės įtakos objektų aptikimo algoritmų darbui. Blogai pasirinktos slenksčio charakteristikos darys drastiška įtaka tolesniam objektų aptikino algoritmo darbui: objektai nebus aptikti arba objektai bus klaidingai aptikti.

Svarbiausios slenksčio charakteristikos yra šios:

- slenksčio reikšmė
- slenksčio tipas
- slenksčio elgsena

Teisingai parinktos šios charakteristikos leidžia sėkmingai išskirti tiriamą objektą iš fono ir objektų visumos. Neteisingai parinktos charakteristikos gali sumažinti aptinkamų objektų kiekį ir formas.

2.2.1. Slenksčių tipai

Dvejetainiai slenksčiai

Slenkstį galima aprašyti kaip paprastą *if* funkciją darbuose su pikseliais. Svarbiausia slenksčio charakteristika yra slenksčio reikšmė. Slenksčio reikšmė ir slenksčio tipas nustato ar pikselis bus pripažintas naudingu. Slenksčio reikšmė yra skaliaras ir priklausomai nuo slenksčio elgsenos ši reikšmė yra konstanta arba gali laisvai kisti su kiekvienu tiriamu pikseliu. Slenksčio tipas nusako sąlyga, kuria tiriamasis pikselis turi tenkinti tam, kad jis būtų pripažintas naudingu. Slenksčio elgsena nustato ar slenksti yra pasyvus, visuotinis ir jo slenksčio reikšmė yra vienoda visiems tiriamo vaizdo pikseliams ar ji gali kisti ir prisitaikyti prie tiriamo pikselio reikšmės. Jeigu tiriamojo pikselio reikšmė tenkina slenksčio sąlyga, jo reikšmė nekinta arba kinta pagal nustatytą sąlyga susijusia su tiriamais objektais. Šio atveju pikselio reikšmė turi didelę tikimybę, kad tolesniuose objektų aptikimo žingsniuose ji bus priskirta objektą atitinkančių super pikselių sričiai. Jeigu tiriamojo pikselio reikšmė netenkina iškeltos slenksčio reikšmės, pikseliui yra priskiriama fono reikšmė ir šis pikselis neturi galimybės būti priskirtam objektus atstovaujantiems pikseliams.

OpenCv visuotinio slenksčio funkcija užrašoma tokiu pavidalu[18]:

```
threshold( src_gray, dst, threshold_value, max_BINARY_value,threshold_type );
```

- src_gray: tiriamas vaizdas pilkumo skalės formate
- dst: vaizdas gautas po slenksčio pritaikymo
- threshold_value: slenksčio reikšmė
- max_BINARY_value: reiškmė kuri priskiriama piksleliams tenkinantiems slenksčio sąlygą

- threshold_type: slenksčio tipas

Slenksčio reikšmė gali būti priskirta vartotojo arba apskaičiuota algoritmo veikimo pradžioje prieš atliekant slenksčio operacija. Tai dažniausiai yra atliekama analizuojant tiriamo vaizdo histogramą. Žinant aplinkos ir tiriamo objekto savybes galima nustatyti tokią reikšmę, kuri padėtų sėkmingai atskirti tiriamus objektus nuo fono. Vaizdai gauti po filtrų pritaikymo gali turėti pasikeitusias charakteristikas kurios gali paslėpti arba išryškinti tam tikrus objektus ir jų savybės gali keistis nuo originalaus vaizdo savybių.

17 pav. Visuotinio slenkstio rezultas 1

Priklausomai nuo nustatytos slenksčio reikšmės galima aptikti skirtingu intensyvumo pasižyminčius objektus. Paveikslai 17 ir 18 yra gauti naudojantis skirtingas slenksčio reikšmes.

18 pav. Visuotinio slenksčio rezultatas 2

Paveiksle 17 nustatyta slenksčio reikšmė yra 120. Paveiksle 18 nustatyta slenksčio reikšmė yra 65. Abiejų paveikslų gavybai buvo panaudoti dvejetainio slenksčio tipai, t.y. visi pikseliai tenkinantys slenksčio sąlyga gauną nustatytą max_BINARY_value, kurios reikšmė buvo nustatyta 255.

Panaudojus atvirkštinį dvejetainį slenkstį yra gaunamas toks pat vaizdas bet rezultatų vaizde pikselių reikšmės yra invertuotos, t.y. juoda spalva (pikselio intensyvumas 0) turintys pikseliai taps balti (pikselio intensyvumas 255), o balti pikseliai taps juodi. Kitaip tariant tai turi įtakos tik fono spalvai. Jeigu nėra atliekami jokie kiti papildomi slenksčio metodai, tai neturės įtakos tolesniuose objektų aptikimo algoritmo žingsniuose. Dvejetainio tipo slenksčių pritaikymas būna pagrindinė ir galutinė operacija vaizdų segmentacijos srityje prieš atliekant objektų aptikimus.

Nupjautiniai ir nuliniai slenksčiai

Slenksčiai pasižymintys nuopjovos arba nulio tipais gali būti jungiami su kitais slenksčiais norint išskirti tiriamus spalvotus objektus iš fono ir objektų gausos. Skirtingi objektai pasižymi skirtingu intensyvumu ir paprastas histogramos metodas gali netenkinti, jeigu tiriamo vaizdo viršūnė yra histogramos viduryje ir aplink ją yra kitų viršūnių. Tokių atveju geriausia naudoti nulinius arba nupjautinius slenksčius nereikalingiems objektams pašalinti iš tiriamo vaizdo.

Priklausomai nuo tipo slenkstis išsaugo reikšmes tenkinančias arba netenkinančias slenksčio sąlyga o visa kita nereikalinga informacija yra priskiriama fonui. Gali prireikti keleto slenksčių operacijų pilnai išskirti dominančius objektus.

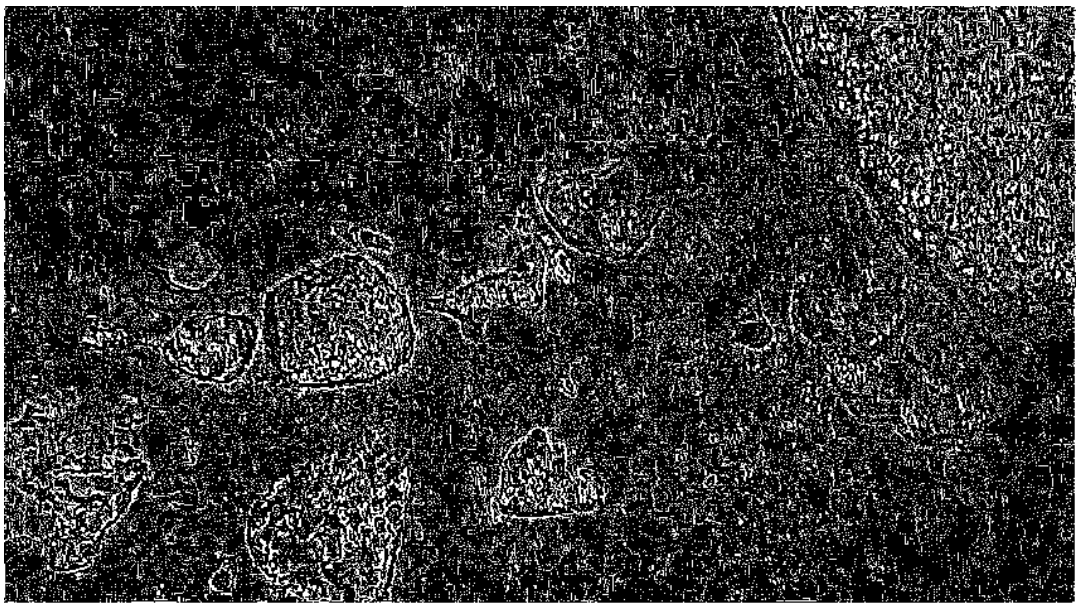
Visuotino slenksčio elgsenos metu vienoda slenksčio reikšmė yra priskiriama visiems pikseliams. Scenos apšvietimo vientisumas daro didelę įtaką objektų aptikime. Apšviesto objekto pikseliai turės didesnį intensyvumą, kuris gali keistis tolstant nuo šviesos šaltinio. Nuotoliniu būdu valdomi aparatai naudojantys vaizdo kamerą objektams aptikti, taip pat gali naudoti papildomus apšvietimo įrenginius tiriamos vietos apšvietimui ir tai gali turėti įtakos objektų aptikimui. Esant papildomam scenos apšvietimui scenos viduryje esantys objektai bus daugiau apšviesti ir tai darys įtakos objektų aptikime. Kraštuose esantys objektai, dėl mažesnio pikselių spalvinio intensyvumo gali būti ignoruojami ir netenkinti iškeltų slenksčio reikšmių. Paveiksle 17 galima matyti apskritimo formos zoną, kuri pasižymi didesniu apšvietimu centre. Šioje vietoje vėliau gali būti aptinkami objektai. Einant į kraštus pikselių intensyvumo reikšmės mažėja ir nebetenkina slenksčio iškeltų sąlygų. Paveiksle 18 skirtingas scenos apšvietimas turi minimalią įtaką objektų aptikimui, nes slenksčio reikšmė yra mažesnė ir visi didesnį intensyvumą turintys objektai netenkina slenksčio sąlygos. Jeigu scenos apšvietimas būtų vienodas vaizdo centre atsirastų pikselių, kurie tenkintų slenksčio sąlygą.

2.2.2. Adaptyvaus slenksčio metodas

Esančiu esant skirtingam scenos apšvietimui geriau naudoti adaptavaus slenksčio nustatymo elgsena pasižyminčius slenksčio metodus. Adaptyvaus slenksčio metodas segementuoja vaizdą į mažesnius nustatyto dydžio regionus ir apskaičiuoja tam regionui slenksčio reikšmę kuri gali būti skirtinga visiems regionams. Adaptyvaus slenksčio metodas OpenCv vaizdų apdorojimo sistemoje gali veikti paprasto `ADAPTIVE_THRESH_MEAN_C` arba svertinio `ADAPTIVE_THRESH_GAUSSIAN_C` vidurkio principu [18].

`adaptiveThreshold`(InputArray **src**, OutputArray **dst**, double **maxValue**, int **adaptiveMethod**, int **thresholdType**, int **blockSize**, double **C**)

- src – 8-bitų vieno kanalo paveikslas
- dst – operacijos rezultatas. Tokio pat dydžio kaip tiriamas vaizdas
- maxValue – reiškmė kuri priskiriama pikseliams tenkinantiems slenksčio sąlygą
- adaptiveMethod – slenksčio algoritmas, ADAPTIVE_THRESH_MEAN_C arba ADAPTIVE_THRESH_GAUSSIAN_C
- thresholdType – slenksčio tipas THRESH_BINARY arba THRESH_BINARY_INV .
- blockSize – gardelės dydis pikseliais slenksčio skaičiavimui: 3, 5, 7, t.t.
- C – konstanta skirta atimti iš gauto vidurkio rezultato



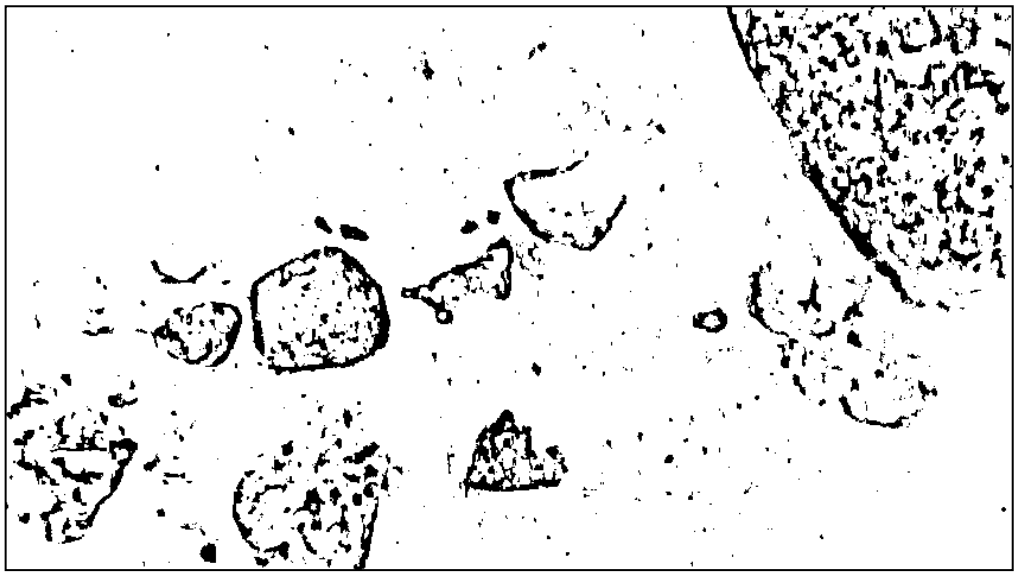
19 pav. Adaptyvaus slenksčio rezultatas 1

Priklausomai nuo nustatytos gardelės dydžio gautas vaizdas gali skirtis savo tikslumu, bet skirtingai negu vaizdas gautas naudojantis visuotiniu slenksčiu jame gali būti didesnis kiekis trikdžiui. Taip ypač pastebima esant mažam gardelės dydžiui ir didelės rezoliucijos vaizdams. Esant

didelės rezoliucijos vaizdui privaloma atitinkamai padidinti gardelės dydį arba gautas vaizdas bus sunkiai interpretuojamas. Padidinus gardelės dydį sumažėja dirbtinių trikdžių kiekis, kuris atsiranda dėl vaizdo segmentavimo į mažesnius regionus ir jiems apskaičiuojamos slenksčio reikšmės, kuri gali būti skirtinga. Paveiksle 19 pavaizduotas adaptyvaus slenksčio rezultatas esant gardelės dydžiui 3x3. Žemiau esančiame paveiksle 20 adaptyvaus slenksčio rezultatas esant gardelės dydžiui 43x43. Esant didesniam gardelės dydžiui galima pastebėti žymiai sumažėjusį trikdžių kiekį. Gardelės kurių dydis yra didesnis negu 43 nebeturi geresnių rezultatų ir tampa pertekliniai.

20 pav. Adaptyvaus slenksčio rezultatas 2

Didesnis gardelės dydis leidžia geriau atlikti vaizdo segmentacija ir išskirti objektus tiriamame vaizde. Trikdžių kiekį vaizde galima sumažinti dviem būdais: nustačius tam tikrą reikšmę C, kuri bus atimama iš gauto vidurkio prieš atliekant slenksčio operacija arba pasinaudojant logines erozijos difuzijos operacija kitame algoritmo žingsnyje kai slenksčio operacija jau buvo atlikta. Adaptyvaus slenksčio metodas funkcijos iškvietimo metu leidžia nustatyti reikšmę C, kuri bus atimama iš gauto vidurkio skirto slenksčio reikšmei apskaičiuoti. C reikšmė turi būti nustatyta atkreipiant dėmesį į pasirinktos gardelės dydį. Reikšmė C dažniausiai būna teigiama, bet priklausomai nuo aplinkybių ji taip pat gali būti neigiama.



21 pav. Adaptyvaus slenksčio rezultatas 3

Per didelė C reikšmė daro didelę įtaką slenksčio pritaikymo metu ir objekto pikseliai gali netenkinti slenksčio sąlygos. Paveikslui 21 gauti buvo panaudota C reikšmė lygi 5. Esant didesnei c reikšmei negu 7 dalis objektams priklausančių pikselių yra priskiriami fonui ir yra prarandama naudinga informacija.

2.3. Briaunų aptikimo algoritmai

Briaunių aptikimas yra vaizdų apdorojimo technologija skirta rasti objektų ribas tiriamose vaizduose. Šio metodo veikimas yra pagrįstas pikselių ryškumo netolydumu vaizde. Briaunų aptikimo metodai yra naudojami vaizdo segmentavimo ir duomenų išskyrimo srityje. Dažniausiai naudojami briaunų aptikimo algoritmai naudoja Sobel, Canny, Prewitt, Roberts bei neraiškios (*fuzzy*) logikos metodus [22].

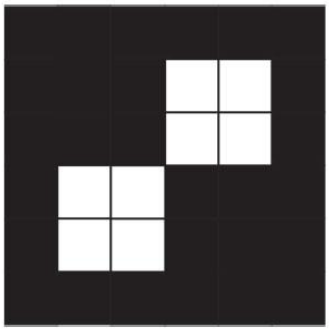
Briaunos gali būti trijų tipų:

- vertikalios
- horizontalios
- įstrižainės

Skirtingi briaunų aptikimo algoritmai sugeba aptikti atitinkamo tipo briaunas. Briaunų aptikimo algoritmai padeda išfiltruoti nereikalingą informacija ir surasti tiriamame vaizde esančius objektus, bei palengvina darbą juos atpažįstant. Triukšmai esantys tiriamame vaizde gali klaidinti briaunų detektorius ir turi būti pašalinti prieš atliekant aptikimą. Reikia pasirinkti gerą tarpinę reikšmę tarp triukšmų pašalinimo lygio ir briaunų aptikimo. Filtrai šalinantis triukšmus, kartu silpnina kraštų aptikimą ir kai kurios kraštinės gali būti ignoruojamos.

Briaunų aptikimo algoritmai gali būti klasifikuojami į šias penkias grupes[23].

- **Briaunų gradiento detektoriai:** Klasikiniai briaunų aptikimo algoritmai naudojantys pirmą kryptinės išvestinės operacija. Algoritmai savo sandara yra paprasti, bet netikslūs dėl didėlės trikdžių daromos įtakos. Šiai grupei priklauso šie algoritmai: Sobel, Prewitt, Roberts, ...
- **Nulio sankirtos:** Šios grupės algoritmai naudojasi antra krypine išvestine kartu su Laplaso operatoriumi. Algoritmai sugeba aptikti skirtingas briaunų kryptis, bet yra įtakojami triukšmo ir gali gražinti keletą aptikimų vienai briaunai.
- **Gauso Laplasianas (LoG):** Algoritmas sugeba tiksliai aptikti briaunų buvimo vietą, bet susiduria su problemomis aptinkant kampus ir įlenkius. Taip pat nesugeba aptikti briaunų krypties.
- **Gauso briaunų detektoriai:** Algoritmai naudojantys Gauso filtrą esamiems triukšmas pašalinti. Geras briaunų aptikimas esant triukšmams tiriamame vaizde. Tikslus briaunų vietos aptikimas, bet algoritmas yra sudėtingas ir jo veikimas užima daug laiko. Canny algoritmas priklauso šiai grupei.
- **Spalvinių briaunų detektoriai:** Tikslus objektų kraštų aptikimas, bet algoritmas yra sudėtingas ir briaunų aptikimas tiriamame vaizde trunka ilgai.



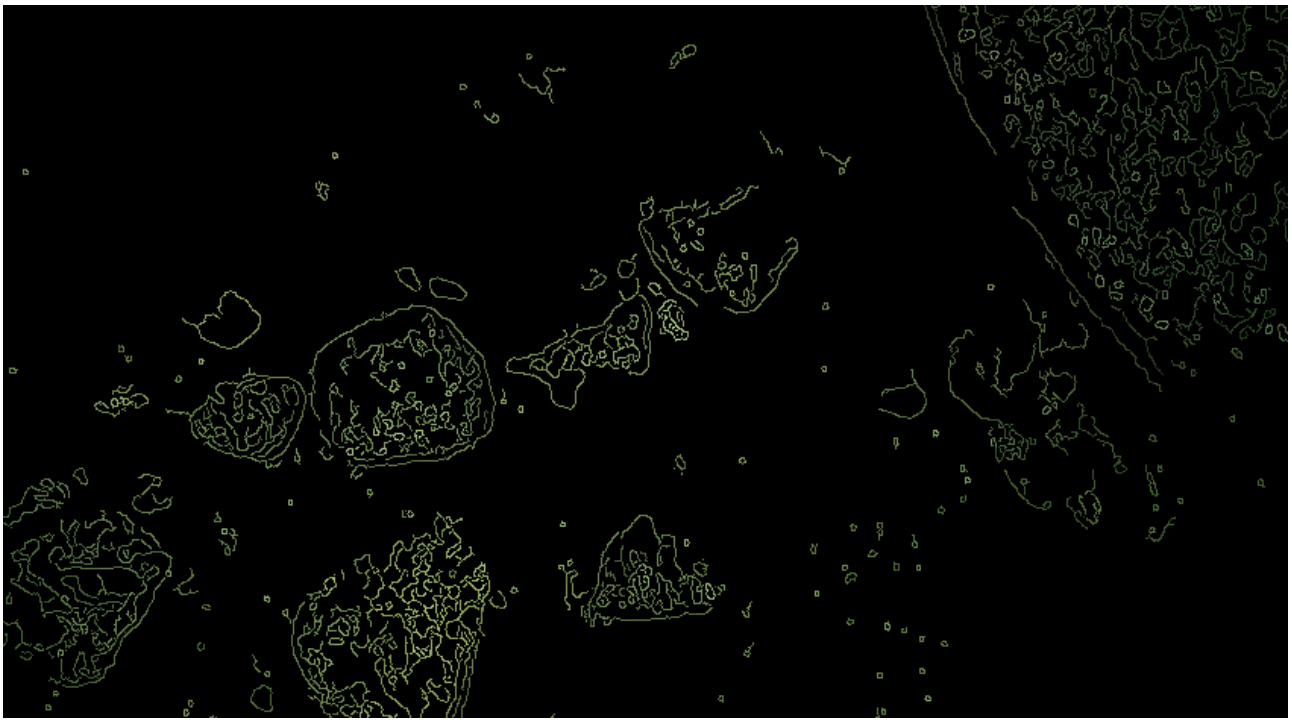
22 pav. Pikselių padėtis[24]

Priklausomai nuo to kaip briaunų aptikimo algoritmas veikia jis gali aptikti vieną arba dvi briaunas tiriamame plote (pav. 22). Algoritmai kurie tiria tik vertikaliai arba horizontaliai esančius pikselius gali aptiktu dvi atskiras briaunas. Tobulesni algoritmai galintys aptikti įstrižaines aptiks vieną briauną. Šiuolaikiniai algoritmai sugeba aptikti įstrižaines.

Canny algoritmas

Canny briaunų aptikimo algoritmas, kitaip dar žinomas kaip optimalus aptikimo algoritmas[47], buvo sukurtas John F. Canny ir pasižymi šiomis savybėmis [25]:

- žema klaidų koeficientas – geras briaunų aptikimas;
- gera lokalizacija – atstumas tarp aptiktų ir tikrų briaunų yra minimalus;
- minimalus atsakymas – vienintelė reikšmė per briauną.



23 pav. Canny briaunų aptikimo algoritmas

Canny algoritmą galima aprašyti taip:

- Triukšmų pašalinimas pasinaudojantis Gauso filtru;
- Vaizdo intensyvumo gradiento suradimas;
- Nemaksimalių reikšmių pašalinimas;
- Pikselių filtravimas pašalinant visus pikselius, kurie nepriklauso briaunoms.
- Vaizdo segmentavimas naudojantis dviem slenksčiais.

Jeigu pikselio gradientas yra didesnis už viršutinio slenksčio ribą, jis yra priskiriamas briaunai. Jeigu ši reikšmė patenka tarp viršutinio ir apatinio slenksčių, pikselis priskiriamas briaunai tik tada jeigu jis jungiasi su pikseliu turinčiu didesnį gradientą negu viršutinis slenkstis. Kitu atveju pikselis yra priskiriamas fonui.

2.4. Lašelių analizatorius

Lašelių analizatorius (angl. *blob analyzer*) yra kitas metodas veikiantis su segmentuotu vaizdu. Tai dažniausiai kompiuterinėje regoje naudojamas metodas objektams aptikti ir jų charakteristikoms analizuoti. Lašelių analizatorius veikia su uždarais objektais po slenksčio

operacijos. Šis metodas remiasi prielaida, kad objektai skiriasi nuo aplinkos savo spalva ir pikselių intensyvumu, bet šis intensyvumas nekinta objekto viduje. Lašelių analizatorius yra lankstus ir greitas metodas, bet jis reikalauja švaraus ir ryškaus fono norint pasiekti gerų rezultatų bei pikselių tikslumo [26]. Visi objektą sudarantys taškai turi intensyvumo reikšmes kurios priklauso mažai kintančiam intervalui.

Priklausomai nuo lašelių analizatoriaus tipo ir elgsenos galimi du pikselių jungimo metodai. Tai daro didelę įtaką aptinkamų lašelių kiekiui[16][24]. Sujungiant pikselius naudojantis keturių šalių taisykle arba sujungiant visus kaimynystėje esančius pikselius – aštuonių šalių taisyklė. Keturių šalių taisyklė skelbia, kad galima priskirti tiriamą pikselį objektui jeigu viršutinis centrinis arba kairysis centrinis pikseliai turi tokią pačią intensyvumo reikšmę kaip ir tiriamas pikselis. Pikseliai yra sujungiami horizontalia arba vertikalia kryptimi.

Aštuonių šalių taisyklė papildo keturių šalių taisyklę galimybę priskirti tiriamą pikselį objektui jeigu viršutinis kairysis pikselis turi tokia pačią reikšmę kaip ir tiriamas pikselis. T.y. pikseliai yra sujungiami netik horizontaliai arba vertikaliai bet ir įstrižai. Priklausomai nuo esamos pikselių sujungimo taisyklės gali būti aptinkamas vienas arba du objektai tame pačiame plote.

Lašelių analizatorius dažniausiai yra naudojamas kartu su Kalman filtru, bet apie šiame darbe nebus kalbama.

2.4. Morfologinės operacijos

Morfologinės operacijos yra atliekamos dvejetainiuose paveiksluose norint pašalinti mažus triukšmus atsiradusius po slenksčio operacijos. Dažniausiai naudojamos operacijos yra erozija ir išplėtimas (diliacija). Priklausomai nuo operacijų atlikimo tvarkos tai gali būti atidarymo arba uždarymo operacijos.

24 pav. Pikselių sujungimas

Erozijos operacijos metu tiriamas pikselis turintis reikšmę 1 išlaikys ja tik tuo atveju jeigu visi pikseliai branduolyje taip pat turės ta pačią reikšmę. Kito atveju jam bus priskirta reikšmė 0. Erozijos operacija sumažina objektų storumą bei pašalina baltą triukšmą.

Išplėtimo operacijos metu tiriamas pikselis gauna reikšmę 1 jeigu bent vienas pikselis branduolyje turi reikšmę 1. Išplėtimo operacija padeda atstatyti objektų formas po erozijos operacijos panaudojimo

Morfologinio atidarymo operacija yra erozija ir po jos sekantis išplėtimas. Ši operacija padeda sumažinti triukšmus fone. Uždarymo operacija yra išplėtimas ir po jo sekanti erozija. Ši operacija padeda užpildyti tuščias skyles objektų viduje.

2.5. Skyriaus apibendrinimas

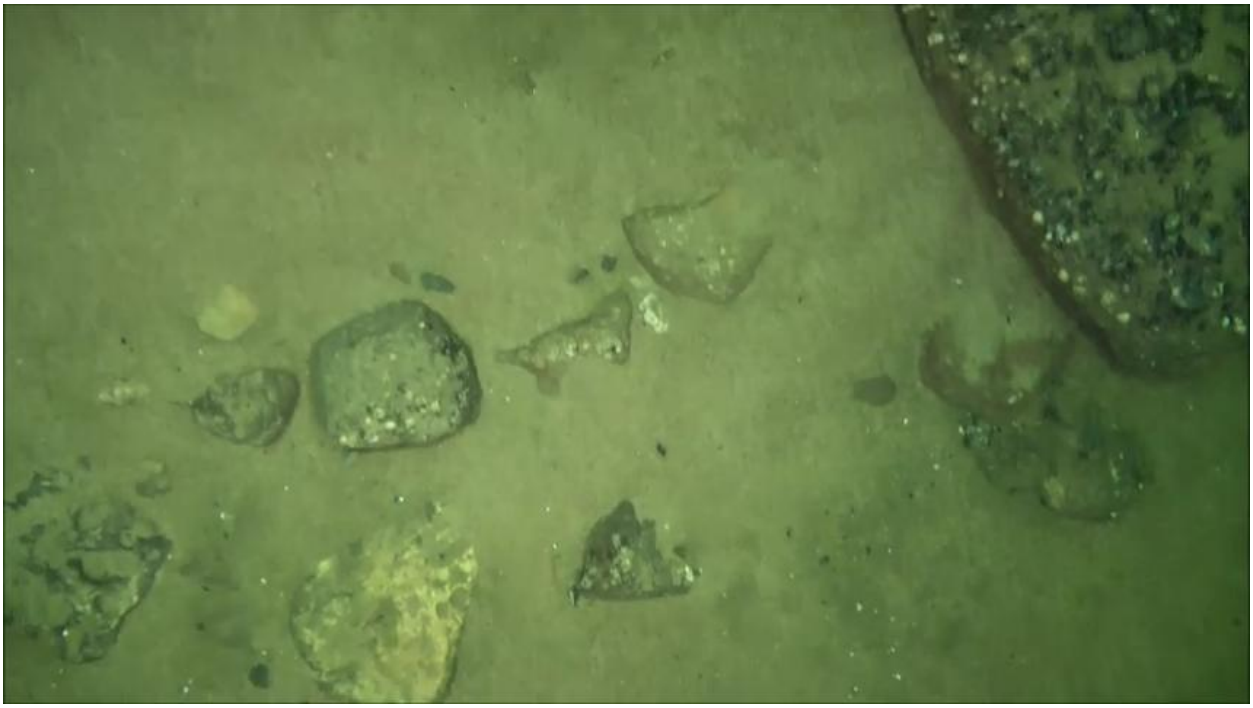
Objektų aptikimo algoritmai susideda iš filtro arba filtrų triukšmams sumažinti arba pašalinti, slenksčio nustatymo ir atlikimo metodų, vaizdo apdorojimo po slenksčio operacijos ir pikselių sujungimo į objektus metodų. Pritaikomi filtrai turi didelę įtaką triukšmų pašalinimui arba objektų savybių išryškinimui. Svarbiausios slenksčio charakteristikos yra slenksčio reikšmė, tipas ir elgsena. Scenos apšvietimo ne vientisumas daro didelę įtaką slenksčiams pasižymintiems visuotino slenksčio elgsena, todėl geriau naudoti adaptyvaus slenksčio elgsena pasižyminčiais slenksčiais. Jeigu tiriamas vaizdas pasižymi didelę rezoliucija reikia atitinkamai pakeisti, padidinti adaptyvaus slenksčio gardelės dydį, o norint sumažinti triukšmų kiekį vaizde reikia parinkti atitinkamą reikšmę c. Per didelė c reikšmė turės neigiamą įtaką objektų aptikime.

3. ALGORITMŲ TYRIMO (FILTRŲ KALIBRACIJŲ) REZULTATAI

Atlikto darbo tyrimų ir uždavinių modeliavimui bei eksperimentinių duomenų patikrai buvo sukurtos taikomosios programos. Sukurtosios programos pateiktos priede nr. 1 psl. 58.
Visi povandeninių objektų skaitmeniniai vaizdai gauti iš Jūros tyrimų atviros prieigos centro [28].

3.1. Tiriamas objektas

Visi naudoti filtrai gali veikti tiek su statinėmis nuotraukomis tiek su vaizdo bylomis. Vaizdo bylos atveju visu pirma yra pasirenkamas atskiras kadras ir jis yra analizuojamas kaip paveikslas. Prieš atliekant veiksmus su filtrais tiriamas paveikslas yra pervedamas į pilkumo skalės pavidalą. Tokiu būdu yra pašalinami spalvų kanalai ir yra paliekamas vienintelis intensyvumo kanalas. To kanalo pikselių reikšmės yra naudojamos veiksmuose su filtrais. Tiriamasis paveikslas pavaizduotas žemiau esančiame paveiksle 25. Tyrimo metu yra sukuriamas naujas paveikslas kuriame yra atliekami visi veiksmai su pikseliais. Kad būtų lengviau palyginti rezultatus, visi tyrimų rezultatų vaizdai pateikti žemiau esančiose paveiksluose yra gaunami esant tam pačiam slenksčiui. Visi vaizdai yra gauti esant spalviniam slenksčiui kurio reikšmės: `Scalar(20, 1, 1), Scalar(250, 255, 255)`. Filtrai yra panaudojami prieš atliekant slenksčio operacijas.



25 pav. Jūros dugnas

Tiriami vaizdai turėjo tokias rezoliucijas: rezoliucija1 – 1926x1080 ir rezoliucija 2 – 860x480. Rezoliucija 1 yra originali naudoto ROV fiksuojamo vaizdo rezoliucija. Tiriamo vaizdo rezoliucijos dydis gali turėti didelę įtaka objektų aptikime. Apie tai plačiau bus kalbama rezultatų apibendrinime.

3.2.Pirmasis (kaimynų vidurkio) filtras

Prieš atliekant veiksmus su filtrais tiriamas paveikslas yra pervedamas į pilkumo skalės pavidalą. Tokiu būdu yra pašalinami spalvų kanalai ir yra paliekamas vienintelis intensyvumo kanalas. To kanalo pikselių reikšmės yra naudojamos veiksmuose su filtrais. Filtras 1 veikimas paremtas kaimynų vidurkio algoritmo principu. Šis filtras surenka visus pikselių balsus esančius vieno pikselio atstumu nuo tiriamojo pikselio. Tiriamos gardelės dydis 3x3. Visų greta esančių pikselių reikšmės yra sumuojamos ir gaunamas reikšmių vidurkis. Žinant gauta reikšmę jį yra pridedama prie esamos tiriamojo pikselio reikšmės taip suteikiant skirtingą reikšmę negu tiriamas paveikslo pikselis turėjo ankščiau. Pikselių palyginimo formulę galima užrašyti tokiu pavidalu:

```
image2.at<uchar>(i,j)=image.at<uchar>(i,j)+(image.at<uchar>(i-1,j-1)+
image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-1,j)+
image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-1)+
image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/8;
```

Žemiau esančiame paveiksle 26 galima matyti gautą rezultatą pritaikius panaudotą filtrą (Filtras 1).

26 pav. Filtro 1 rezultatas

Centre esantys objektai viršija slenksčio ribą ir yra aptinkami. Aptikimo zona galima aiškiai atskirti nuo kitų objektų, kurios algoritmas priskiria fonui. Reikia pastebėti, kad centre esantys objektai dažniausiai gali turėti didesnį apšvietimą. Bet koks mažas apšvietimo pakitimas gali daryti didelės įtakos objektų aptikimui. Kadangi šiame vaizde centre esančio smėlio sudarančio foną spalvos intensyvumas yra didesnis už objektus, bet einant į kraštus kinta. Dalis kraštuose esančio smėlio yra klaidingai aptinkama kaip vientisas objektas, todėl algoritmas nesugeba aptikti krašte esančio objekto. Efektyvus algoritmo veikimo plotas yra pasiskirstęs vaizdo centre. Priklausomai

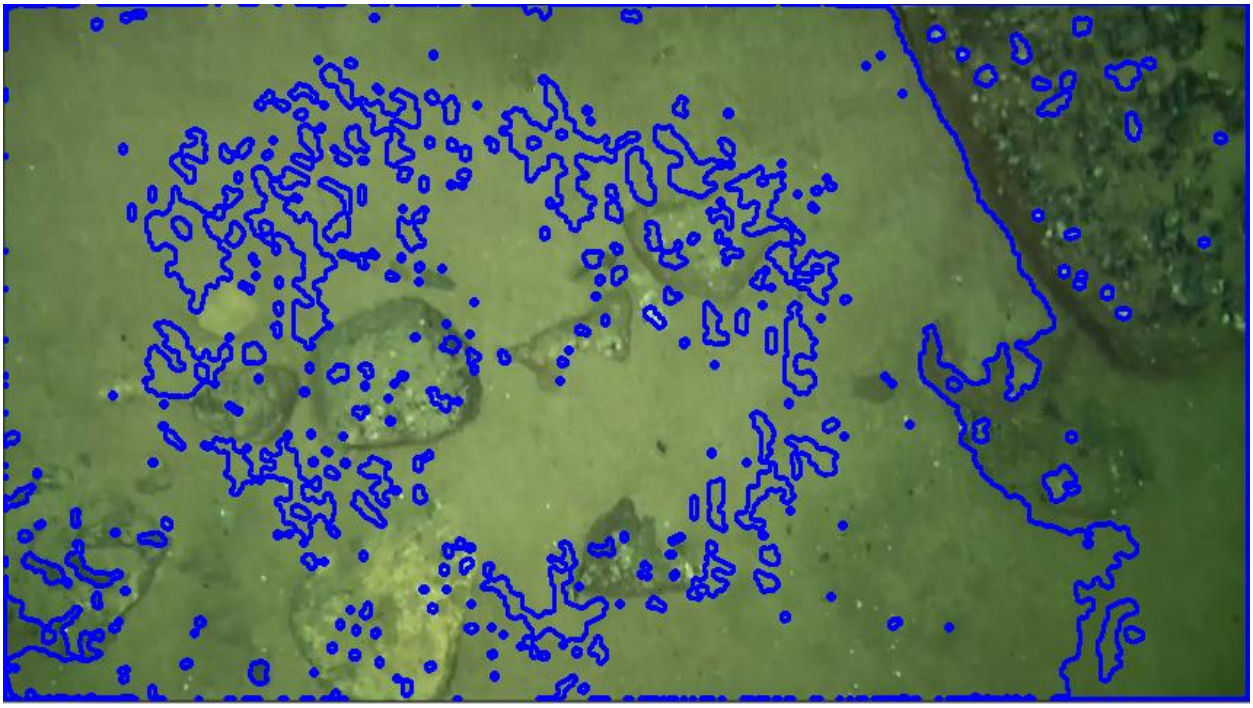
nuo scenos apšvietimo ir naudojamų slenksčio parametrų: t.y. jo tipo ir slenksčio reikšmės algoritmo efektyvaus veikimo plotas gali didėti arba mažėti.

3.3. Antrasis (pikselių aproksimavimo) filtras

Filtras 2 paremtas tuo pačių principu kaip Filtras 1, bet turi vieną esminį skirtumą tame, kad jis atlieka veiksmus atsižvelgiant į ankstesnius skaičiavimo rezultatus jau atliktus tiriamame paveiksle. T.y. Filtras 2 kitaip negu Filtras 1 ima pikselių reikšmes ne iš originalaus pradinio paveikslo, bet iš paveikslo kuris yra galutinis variantas. Tokiu būdu visi naujai apskaičiuoti pikseliai daro įtaką tolesnių pikselių reikšmių skaičiavimuose. Filtras 2 gauto paveikslo pikseliai mažiausiai skiriasi nuo pradinių reikšmių viršutiniame kairiame kampe ir daugiausiai apatiniame dešiniajame kampe. Pikselių palyginimo formule galima užrašyti tokiu pavidalu:

```
image2.at<uchar>(i,j)=image2.at<uchar>(i,j)+(image2.at<uchar>(i-1,j-1)
+image2.at<uchar>(i,j-1)+image2.at<uchar>(i+1,j-1)+image2.at<uchar>(i-1,j)
+image2.at<uchar>(i-1,j+1)+image2.at<uchar>(i+1,j-1)
+image2.at<uchar>(i+1,j)+image2.at<uchar>(i+1,j+1))/8;
```

Žemiau esančiame paveiksle 27 galima matyti gautą rezultatą pritaikius panaudotą filtrą Filtras 2. Galima pastebėti aiškius skirtumus tarp Filtras1 ir Filtras 2 sukurtų vaizdų. Pagrindinė forma paveikslo viduryje yra panaši, bet Filtras 2 atveju ji yra smulkiai išskaidyta. Taip pat Filtras 2 padeda aptikti kitus objektus kurių panaudojant Filtras 1 nebuvo galima atskirti.



27 pav. Filtro 2 rezultatas

Skirtingai nuo Filtras 1 gautų rezultatų, Filtras 2 daro didesnę įtaką slenksčio operacijos metu gaunamam vaizdui todėl šio atvejų yra aptinkama netik daugiau objektų, bet aptikimo zona yra

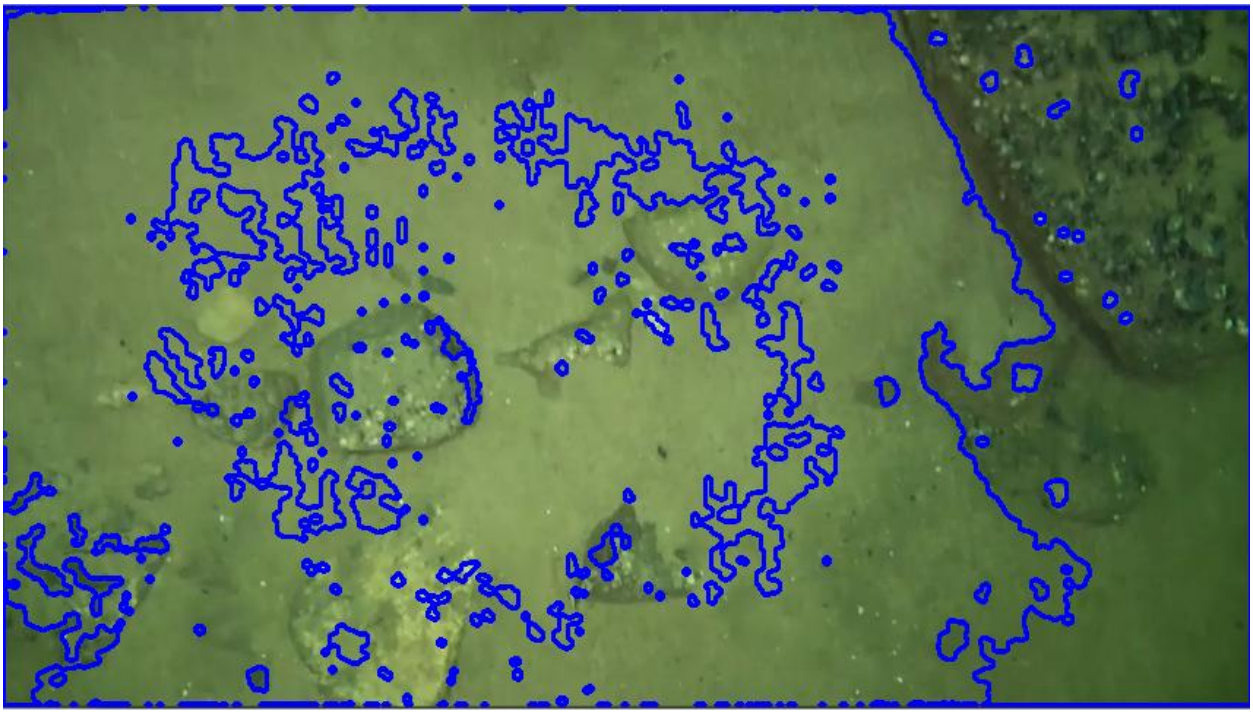
didesnė, tai taip daro įtakos objektų aptikimo kokybei. Algoritmas aptinka daugiau objektų, bet jų dydis yra mažas. Dauguma aptiktų objektų yra dalinai arba klaidingai aptikti.

3.4. Kombinaciniai (vidurkio) filtrai

Filtras 3 ir Filtras 4 atlieką skaičiavimus tokiu pačiu principu kaip ir Filtras 2. Į skaičiavimus įtraukiant tiriamojo pikselio reikšmę galima gauti geresnius rezultatus. Aptikti regionai turi labiau vientisą pavidalą ir yra mažiau mažesnių atskirų regionų, nes mažesni regionai yra sujungiami į didesnius regionus. Priklausomai nuo koeficiento, su kurio yra dauginama tiriamojo pikselio reikšmė, galima gauti skirtingus rezultatus. Pirmojo koeficiento (1) atveju gaunamas vaizdas yra panašus į vaizdą gautą naudojus Filtras 2, bet paveiksle mažiau mažų regionų. Pikselių palyginimo formule galima užrašyti tokiu pavidalu:

```
      image.at<uchar>(i,j)=image.at<uchar>(i,j)+(image.at<uchar>(i,j)+image.at<uchar>(i-1,j-
1)
      +image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-1,j)
      +image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-1)
      +image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/9;
```

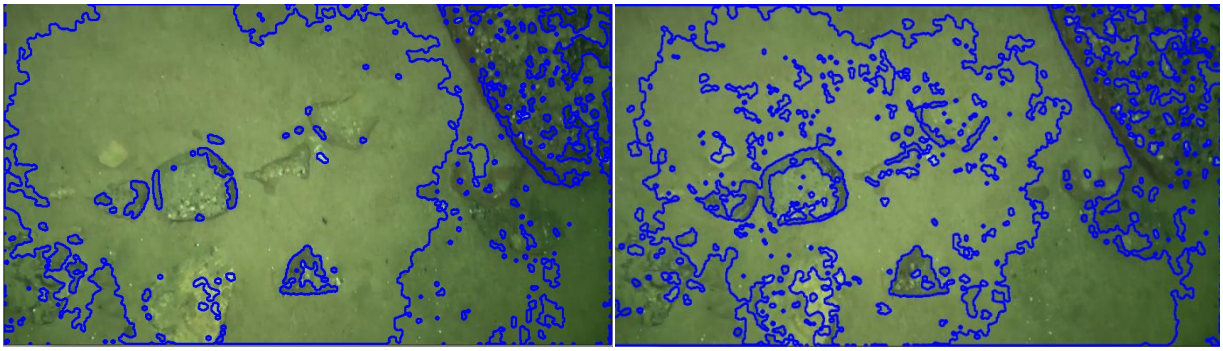
Gautą vaizdą galima pamatyti žemiau esančiame paveiksle 28.



28 pav. Filtro 4 rezultatas

Visai kitoks vaizdas yra gaunamas jeigu yra panaudojama pikselio reikšmė pakelta antruoju koeficientu (2). Tuo atveju visi maži regionai, kurie buvo aptikti pirmuoju atveju, yra sujungiami į visumą ir yra aptinkami nauji regionai kurie nebuvo aptikti ankščiau. Sudauginus tiriamojo pikselio reikšmę su trečiuoju koeficientu (3) yra gaunamas sudėtinis vaizdas sudarytas iš vaizdų su pikseliais

naudojantis pirmojo ir antrojo koeficientu. Gauti vaizdai pavaizduoti žemiau esančiame paveiksle 29. Kairėje esantis paveikslas yra gautas naudojant tiriamojo pikselio reikšmę su antruoju koeficientu, o dešinėje esantis paveikslas yra gautas naudojant tiriamojo pikselio reikšmę su trečiuoju koeficientu.



29 pav. Filtrų 3 ir 3.3 rezultatai

Reikia atkreipti dėmesį, kad šio atveju t.y. dauginat tiriamojo pikselio reikšmę svarbų vaidmenį gali vaidinti reikšmės ribų viršijimas. T.y. gauta naujai apskaičiuota reikšmė gali viršyti maksimalią leistiną reikšmę ir programinė įranga apskaičiuos naują reikšmę tenkinančią duomenų tipo nustatytas ribas.

3.5. Normalizavimo filtras

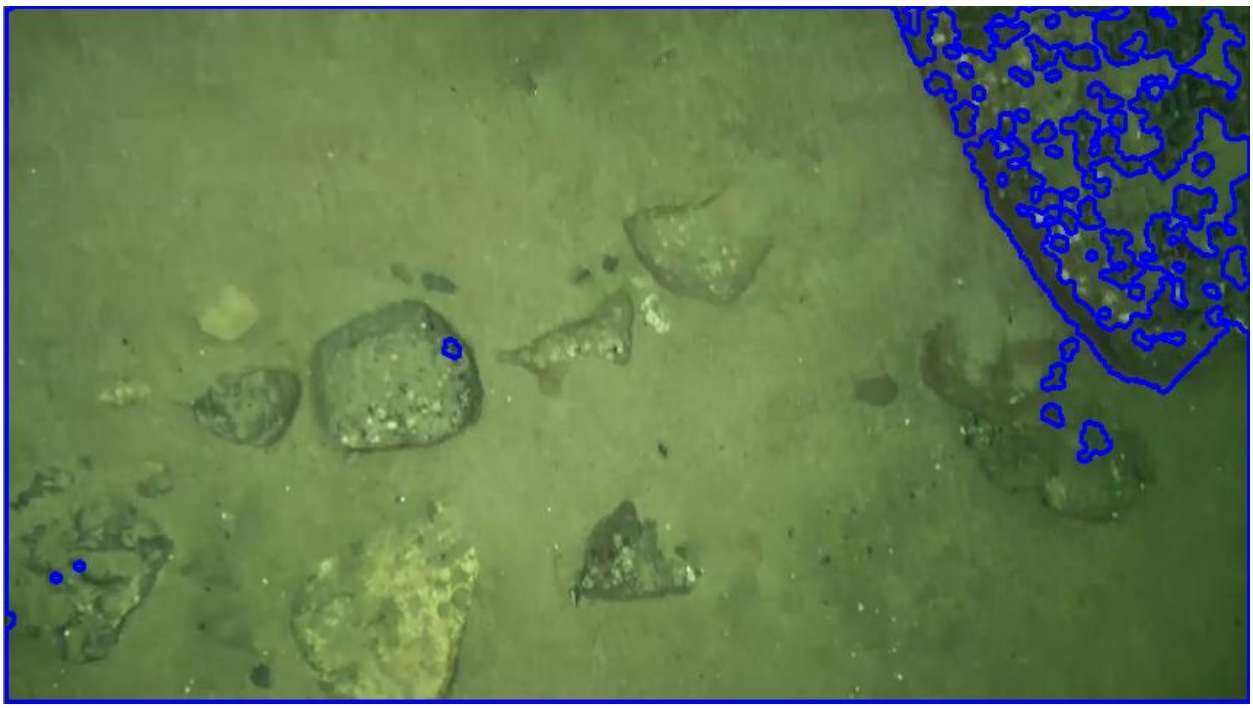
Filtras 5 veikia panašių principu kaip Filtras 3 arba Filtras 4, bet vietoj prie pradinės reikšmės pridedant visu greta esančių pikseliu vidurkį, vidurkis yra atimamas. Programa sugeba aptikti skirtingus regionus.

Šio atveju programa didesnį dėmesį skiria dešinėje viršutiniame kampe esančiam objektui, nes jo charakteristikos yra labiau matomos programai ir ji ignoruoja centre esančius objektus.

Pikselių palyginimo formule galima užrašyti tokiu pavidalu:

```
image.at<uchar>(i,j)=image.at<uchar>(i,j)-(image.at<uchar>(i,j)
+image.at<uchar>(i-1,j-1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)
+image.at<uchar>(i-1,j)+image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-1)
+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/9;
```

Gautą vaizdą galima pamatyti žemiau esančiame paveiksle 30.



30 pav. Filtro 5 rezultatas

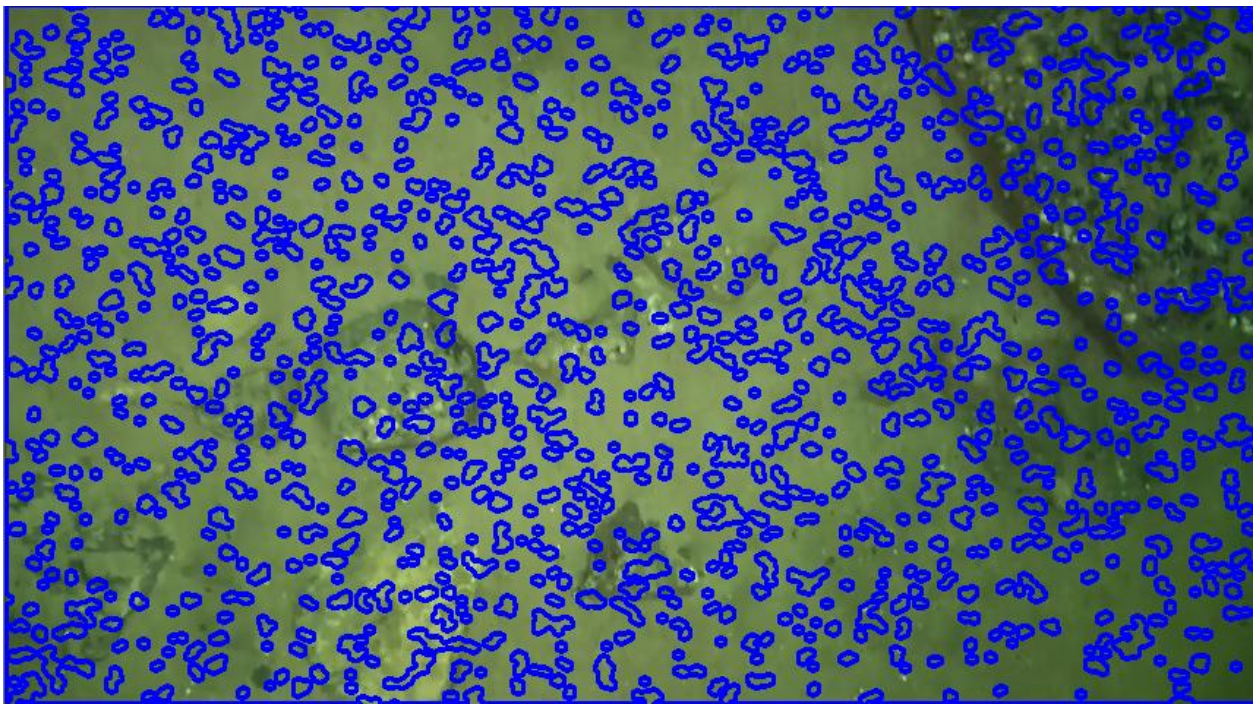
Didelę reikšmę turinčių pikselių reikšmė yra sumažinta daugiau negu mažą reikšmę turinčių pikselių reikšmė. Todėl šio atvejų yra aptinkamas objektas, kuris turėjo mažesnį intensyvumą prieš atliekant filtro operacija. Naudojamas algoritmas visiems šviesiems pikseliams priskiria fono reikšmę.

3.6. Vidurkio daugybos filtras

Filtras 6 atlieka tiriamojo pikselio reikšmės daugyba su kaimyninių pikselių vidurkiu. Reikšmės persipildymo faktorius pasiekia kritinę ribą ir paveikslas tampa nesuprantamas tiek žmogui tiek programai. Nors programa stengiasi sujungti atskirus regionus, bet paveikslo duomenys yra per daug pakeisti, kad juos būtų galima tinkamai interpretuoti.

```
image.at<uchar>(i,j)=image.at<uchar>(i,j)*(image.at<uchar>(i,j)
+image.at<uchar>(i-1,j-1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)
+image.at<uchar>(i-1,j)+image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-1)
+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/9;
```

Gautą vaizdą galima pamatyti žemiau esančiame paveiksle 31. Matoma daugybė mažų kontūrų. Visi kontūrai yra vienodo arba skiriančio dydžio esant spalviniam slenksčiui. Aptiktų kontūrų išsidėstymas yra tvarkingas.



31 pav. Filtro 6 rezultatas

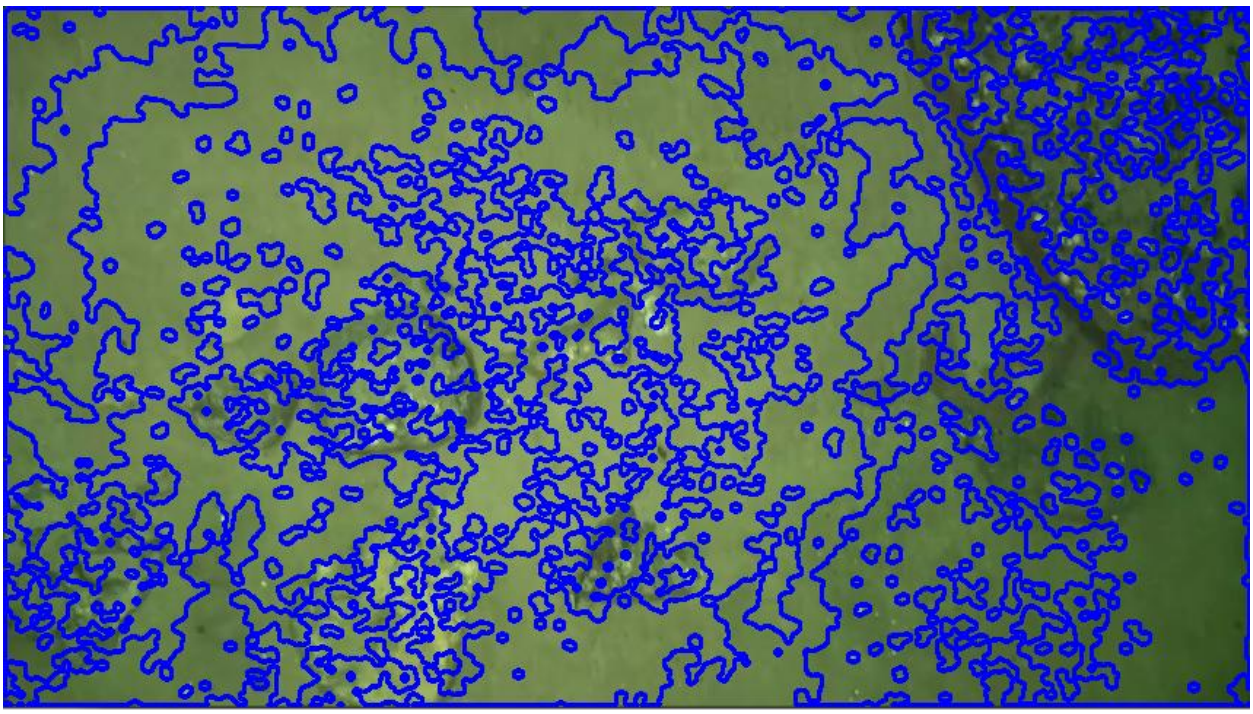
Priklausomai nuo naudojamo slenksčio charakteristikų rezultatai gali drastiškai kisti šio filtro naudojimo metu. Objektų aptikimo algoritmas neaptiks jokių objektų arba bus klaidingai aptikta daugybė objektų. Algoritmai priklausomai nuo naudojamų slenksčio charakteristikų grąžins minimalias arba maksimalias reikšmes. Aukščiau pateiktame paveiksle 31 matomas rezultatų vaizdas gerai iliustruoja esančia problemą. Vaizde yra aptinkama daugybė objektų, net jeigu jų ten nėra.

3.7. Kėlimo laipsniu filtras

Filtras 7 atlieka tokius pačius veiksmus kaip Filtras 4, bet gautas rezultatas dar yra pakeliamas antruoju laipsnius. Gautas rezultatas yra panašus į tarpinį varianta tarp Filtras 4 ir Filtras 6 gautų vaizdų. Vaizdas yra sunkiai suprantamas.

```
image2.at<uchar>(i,j)=pow((image.at<uchar>(i,j)+image.at<uchar>(i-1,j-1)
+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-1,j)
+image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-1)
+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/8, 2);
```

Gautą vaizdą galima pamatyti žemiau esančiame paveiksle 32.



32 pav. Filtro 7 rezultatas

Filtru 7 gautas rezultatų vaizdas (32 pav.) nepasižymi didelių mažų objektų išsibarstymu negu paveiksle 31 pavaizduotas filtro 6 rezultatas, bet šio atveju algoritmui yra sunkiau aptikti objektus. Dauguma pikselių yra sujungiami į didesnius regionus, bet jeigu filtro 6 rezultatai atstovavo kraštines reikšmes (min arba max), filtro 7 rezultatai yra tarpinis rezultatas tarp vidutinių ir maksimalių reikšmių. Filtro 6 gautas vaizdas yra tvarkingas, o filtro 7 vaizdas yra chaotiškas; rasti kontūrai yra išsidėstę netvarkingai.

3.8. Alternatyvus (šviesumo keitimo, daugybos) filtras

Filtras 8 atlieka paprasta pikselių reikšmės daugyba. Pagerėjus kontrastui tarp objektų jų aptikimas pagerėja. Galima gerai aptikti centre esančius akmenis, bet šone esantis akmuo yra ignoruojamas dėl skirtingo kontrasto su aplinka. Algoritmas naudojantis šį filtrą gerai aptinka vidutinio šviesiumo objektus. Gautą vaizdą galima pamatyti žemiau esančiame paveiksle 33.

33 pav. Filtro 8 rezultatas

Šis filtras gerai veikia esant visuotinam slenksčiui. Adaptyvus slenkstis esant didelės rezoliucijos vaizdui gali susidurti su sunkumais aptikti objektus. Šis filtras daro didesnę įtaka vaizdui kaip pikselių visumai negu atskiriems pikseliams.

3.9. Povandeninių objektų filtrų palyginimo rezultatai

Gauti rezultatai pateikti žemiau esančioje lentelėje. Darbo eigoje buvo pastebėta, kad sumažinus tiriamojo vaizdo rezoliucija bent du kartus, randamų objektų kiekis gali pasikeisti daugiau negu dešimt kartų. Lentelėje patektas aptiktų kontūrų kiekis.

Buvo naudoti tokios rezoliucijos vaizdai.

Rezoliucija 1 – 1926x1080. Rezoliucija 2 – 860x480.

Naudotų slenksčių reikšmės:

- slenkstis 1 – visuotinis spalvinis slenkstis su parametrais Scalar(120, 255, 1), Scalar(250, 255, 255);
- slenkstis 2 – visuotinis spalvinis slenkstis su parametrais Scalar(20, 255, 1), Scalar(250, 255, 255);
- slenkstis 3 – Adaptyvus dvejetainis slenkstis su 5x5 gardele ir vidurkio funkcija.

Lentelė 1. Slenksčių rezultatai

	Slenkstis1		Slenkstis 2		Slenkstis3	
	Rezoliucija 1	Rezoliucija 2	Rezoliucija 1	Rezoliucija 2	Rezoliucija 1	Rezoliucija 2
Filtras 1	848	117	1138	102	2712	470
Filtras 2	625	58	1784	299	1140	128
Filtras 3	891	119	1326	202	1716	401
Filtras 4	695	56	1559	228	1314	141
Filtras 5	54	1	121	46	3459	224
Filtras 6	27	5	5476	1025	17	5
Filtras 7	775	383	3410	501	1508	527
Filtras 8	832	127	702	86	17779	437

Kaip matoma iš aukščiau pateiktos lentelės tiriamo vaizdo rezoliucija daro didelę reikšmę objektų aptikimui. Minimalus skirtumas tarp aptiktų objektų kontūrų yra du kartai naudojant filtrą 7 slenksčių 1 ir 3 atveju ir filtro 5 naudojant slenkstį 2. Maksimali reikšmė yra skirtinga naudojant skirtingus slenksčius. Ji gali siekti 41. Nors esant Filtrui 5 su slenksčiu 1 ši reikšmė gali siekti 56, ši reikšmė yra klaidinga, nes esant mažesnei rezoliucijai buvo aptiktas tik vienas taškinis kontūras. Didesnės rezoliucijos metu kontūras yra išskaidomas į daugybę mažesnių kontūrų, nes pikselių reikšmė nėra vienoda. Slenksčiai 1 ir 2 parodė panašius rezultatus esant mažesnei rezoliucijai, bet naudojant specifinius filtrus tie rezultatai ryškiai skyrėsi. Reikia atkreipti dėmesį į tai kad esant slenksčiui 2 su rezoliucija 1 buvo aptikta daugiau kontūrų negu esant slenksčiui 1 su tokia pačia rezoliucija. Tai galima paaiškinti, tuo kad slenkstis 2 turi didesnę tolerancijos sritį, bet tai galioje tik esant didelei rezoliucijai (rezoliucija 1). Esant mažesnei rezoliucijai juo rezultatai artėja prie slenksčio 1 rezultatų esant tam tikriems filtrams.

Filtro 6 rezultatai skiriasi priklausomai nuo naudojamų slenksčių. Esant slenksčiams 1 ir 3 rezultatai yra panašūs, bet esant slenksčiui 2 pastebimas staigus rezultato reikšmių padidėjimas, kuris pasiekia maksimalią ribą. Slenksčio 2 rezultatai esant filtrui 6 parodo, kad šis slenkstis nesugeba susidoroti su iškilusia problema, kai skirtumai tarp pikselio reikšmių yra dideli. Objektų aptikimo algoritmas klaidingai aptinka daugybę objektų.

Naudojantis adaptyvaus slenksčio nustatymo metodu gauti rezultatai aiškiai skiriasi nuo rezultatų gautų naudojantis visuotinio slenksčio metodu. Visuotinio slenksčio metodu gauti rezultatai yra panašūs tarpusavyje. Adaptyvaus slenksčio atveju rezultatai skiriasi, nes slenksčio reikšmė yra apskaičiuojama kiekvienam pikseliu atskirai o aplink esantys pikseliai gali daryti didelę įtaką šiai reikšmei. Šis slenkstis aptiko mažiausiai kontūrų esant filtrui 6. Tai yra sėkmingas aptikimas su mažiausiu klaidų kiekiu. Vaizdas gautas panaudojus filtrą 6 yra netvarkingas, o jo pikselių reikšmės gali drastiškai kisti tarp kaimynų. Aptikti objektai buvo taškiniai, arba artimi jiems, kontūrai. Slenkstis 1 pasižymėjo panašiais rezultatais, bet aptiktų taškinių objektų kiekis buvo didesnis.

Svarbu pastebėti, kad tiriamo vaizdo rezoliucija daro didelę įtaką objektų aptikime naudojantis adaptyvaus slenksčio nustatymo metodu. Tai galima matyti lentelėje 1 palyginus aptiktų kontūrų kiekį esant skirtingoms rezoliucijos naudojant adaptyvų slenkstį. Aptiktų kontūrų santykis tarp rezoliucijos 1 ir rezoliucijos 2 kinta nuo 2,86 esant filtrui 7 iki 15, 44 esant filtrui 5. Jeigu nekreipiama dėmesio į filtro 6 rezultatus, aptiktų kontūrų kiekis priklauso intervalui [1140; 3459]. Rezultatų reikšmės gautos naudojantis filtrais 6 ir 8 parodo slenksčio lūžius esant ypatingoms situacijoms. Abejais atvejais tiriamas vaizdas turi pikselių su didele intensyvumo reikšme. Jeigu tiriamo vaizdo pikselių intensyvumo reikšmės pasižymi didelių skirtumu tarpusavyje, šis slenkstis aptinka mažiau objektų, bet jeigu pikselių reikšmės mažai skiriamasi tarpusavyje, algoritmas naudojantis adaptyvaus slenksčio nustatymo metodu nesugebės sėkmingai aplinkti vaizdo segmentacijos ir bus daugybė dalinių aptikimų. Adaptyvaus slenksčio reikšmė esant filtrui 8 yra 17779. Ši reikšmė yra 41 kartą didesnė už reikšmę gautą naudojantis mažos rezoliucijos vaizdu. Naudojantis adaptyvaus slenksčio metodu kartu su filtru 5, algoritmas susiduria su ta pačia problema. Naudojantis adaptyvaus slenksčio metodu ir esant mažos rezoliucijos vaizdui, filtrų rezultatai nepateikia tokiu staigių reikšmių negu naudojantis aukštos rezoliucijos vaizdu.

Žemiau esančiose lentelėse pateikti kontūrų aptikimo rezultatai esant skirtingiems slenksčio tipams. Lentelėje 2 yra pateikti rezultatai naudojantis rezoliucija 2. Lentelėje 3 yra pateikti rezultatai naudojantis rezoliucija 1.

Buvo naudojami tokie visuotinio slenksčiu tipai:

- slenkstis 1 – binarinis slenkstis;
- slenkstis 2 – binarinis atvirkštinis slenkstis;
- slenkstis 3 – nupjautinis slenkstis;
- slenkstis 4 – nulinis slenkstis
- slenkstis 5 – nulinis atvirkštinis slenkstis.

Visų slenksčių slenksčio reikšmė buvo vienoda. Jos reikšme 120.

Lentelė 2. Slenksčių rezultatai su rezoliucija 2

	Slenkstis1	Slenkstis 2	Slenkstis 3	Slenkstis 4	Slenkstis 5
Filtras 1	88	97	28	88	101
Filtras 2	53	19	2	53	19
Filtras 3	133	123	0	113	120
Filtras 4	57	19	0	57	19
Filtras 5	1	1	0	1	1
Filtras 6	165	2	0	165	0
Filtras 7	391	355	0	391	362
Filtras 8	14	21	0	14	21
Be filtro	46	50	0	46	50

Slenkstis 3 pateikė mažiausiai rezultatų. Jo gautų rezultatų kiekis esant rezoliucijai 2 siekė 0 esant visiems filtrams išskyrus filtrus 1 ir 2. Tai galima paaiškinti tuo, kad šis slenkstis yra dažniausiai naudojamas norimiems objektas išskirti iš objektų gausumuos prieš naudojant slenksčius su kitaip tipais pvz.: dvejetainį slenkstį. Vaizdai apdoroti filtrais 2-8 neturėjo pikselių atitinkančių slenksčio sąlygą. Taip pat iš gautų rezultatų galima spręsti, kad esama slenksčio reikšmė nebuvo optimali šio slenksčio tipui. Šio tipo slenkstis pateikė vienodus rezultatus esant abiems rezoliucijų tipams. Reikia pastebėti, kad rezultatai su šio slenksčio tipu ir filtru 1 yra vieninteliai kurie turėjo daugiau aptiktų kontūrų esant mažesnei rezoliucijai, negu esant didesnei rezoliucijai. Nors kontūrų kiekis esant mažesnei rezoliucijai yra dvigubai didesnis negu esant didesnei rezoliucijai, bet tai greičiausiai yra dėl klaidingai aptiktų objektų.

Kitas svarbus pastebėjimas yra rezultatai gauti naudojantis filtru 5. Šio atveju yra aptiktas tik vienas kontūras, bet šis kontūras apima visą objektą esantį viršutiniame dešiniajame kampe. Esant rezoliucijai 2 tyrimo rezultatai nekinta esant skirtingiems slenksčio nustatymo tipams išskyrus slenkstį 3, bet kodėl taip yra jau buvo kalbėta anksčiau. Kontūras yra vientisas esant rezoliucijai 2, bet yra išskaidomas į keletą atskirų kontūrų esant didesnei rezoliucijai (rezoliucija 1). Esant didesnei rezoliucijai daugiau pikselių sudaro kontūrą. Riboje tarp objekto ir fono gali būti pikselių, kurie priklauso objektui, bet jų intensyvumas gali nežymiai skirtis todėl nevisi pikseliai gali būti priskirti objektui ir mažesni objektai gali būti patikti net jeigu jų ten nėra.

Lentelė 3. Slenksčių rezultatai su rezoliucija 1

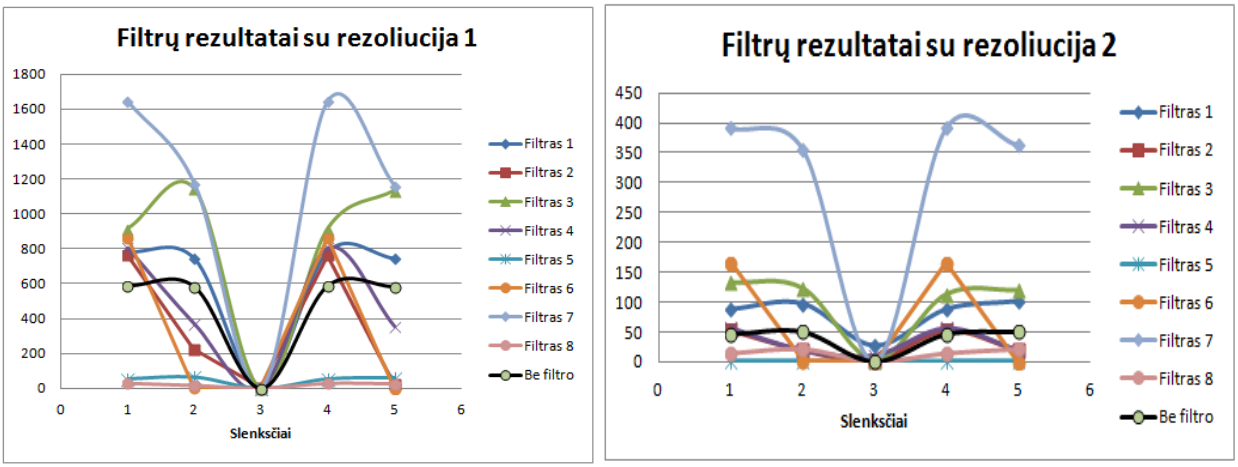
	Slenkstis1	Slenkstis 2	Slenkstis 3	Slenkstis 4	Slenkstis 5
Filtras 1	783	744	14	783	745
Filtras 2	762	223	3	762	21
Filtras 3	918	1145	0	918	1137
Filtras 4	806	369	0	806	356
Filtras 5	54	64	0	54	61
Filtras 6	861	3	0	861	0
Filtras 7	1646	1174	0	1646	1163
Filtras 8	31	18	0	31	28
Be filtro	587	581	0	587	581

Daugiausiai kontūrų nepriklausant nuo pasirinktos rezoliucijos buvo aptikta naudojantis filtru 7. Šio filtru gauti rezultatai tarpusavyje gali skirtis iki trejų kartų nepriklausomai nuo esamos rezoliucijos. Vaizdas yra pakeistas taip, kad yra sunku nustatyti tinkamą slenksčio reikšmę. Filtras 6 pateikia panašius rezultatus esant spalviniam slenksčiui, bet skirtingai negu rezultatai gauti filtru 7, aptikti objektai yra išsidėstę tvarkingai. Visi aptikti objektai naudojant filtrą 6 yra taškiniai. Filtras 7

padaro daugiausiai klaidų, jame daug taškinių arba dalinai aptiktų objektų, dauguma iš jų buvo dirbtinai sukurti ir originaliame vaizde neegzistuoja. Vaizde gausu trikdžių.

Esant didesnei rezoliucijai aptiktų objektų kiekis yra padidėjęs dėl didesnio pikselių kiekio, bet rezultatų kitimo tendencija išlieka nekintanti. Santykinis pokytis tarp rezultatų gautų su ir be filtrų yra išlaikomas, bet yra išimčių. Rezultatai gauti naudojant filtrą 7 naudojant rezoliucija 1 turi didžiausią reikšmę, bet santykinis rezultatas yra žymiai mažesnis už rezultatus gautus esant mažesnei rezoliucijai. Filtro 7 santykinis rezultatas (rezultatų santykis tarp reikšmės gautos naudojant filtrą ir be filtro) rezoliucijos 1 metu yra 1,98, o rezoliucijos 2 metu – 7,86. Didesnis pikselių kiekis padeda geriau aptikti objektus.

Filtro 8 rezultatai žymiai skiriasi nuo kitų rezultatų, nes yra daugybė pikselių turinčių panašia arba kintančia reikšmę ir algoritmas juos sujungia į žymiai didelius plotus negu esant kitiems filtrams arba nebūnant filtro.



34 pav. Filtrų kalibracijos rezultatų diagramos

Paveiksle 34 pavaizduota gautų rezultatų diagrama. Filtro 7 klaidingi rezultatai aiškiai išsiskiria iš kitų filtrų rezultatų. Juoda linija žymi rezultatus gautus be filtrų. Filtrų 5 ir 8 rezultatų reikšmės yra mažesnės už reikšmes gautas be filtro. Filtrų 2 ir 4 reikšmės yra artimiausios reikšmėms gautoms be filtrų esant slenksčiams 1 ir 4, kitais atvejais jos yra mažesnės už reikšmes gautas be filtrų. Rezultatai gauti naudojant filtrus 1 ir 3 visada yra didesni už rezultatus gautus be filtro.

IŠVADOS

Darbe nagrinėjami povandeninių objektų aptikimo algoritmai ir programinės priemonės jiems įgyvendinti. Sistemos kūrimo etape bus taikoma universalių algoritmų adaptacija prie turimos medžiagos ir algoritmų atnaujinimas unikalioms užduotims įgyvendinti. Povandeninių objektų atpažinimo algoritmai aktualūs ieškant paskendusių objektų, jūros dugno artefaktų, jūrinės augmenijos ir gyvūnijos pavyzdžių. Tiriami slenksčio metodai ir jų charakteristikos bei jiems daroma įtaka esant skirtingiems filtravimo parametrams. Programinės priemonės kurtos ir testuotos naudojantis Microsoft Visual Studio sistema kartu su OPENCV vaizdų apdorojimo biblioteka, panaudojant KU povandeninio stebėjimo roboto (ROV) vaizdo medžiagą.

Iškelta hipotezė galuoja esant filtrams 1, 2, 3, 5, 8 su slenksčiais 1, 2, 4, 5 bet negalioja esant slenksčiui 3 ir filtrams 6 ir 7. Kiekvienas filtras išskyrus filtrus 6 ir 7 padeda aptikti skirtingo intensyvumo objektus.

1. Atlikta konkrečių povandeninių vaizdų apdorojimo uždavinių analizė padėjo įžvelgti esminius veiksnius turinčius įtakos objektų aptikimo algoritmuose. Po vizualinių povandeninių vaizdų savybių analizės aprašyti žinomų trūkumų šalinimui naudingi metodai ir numatyti atitinkami scenarijai jų taikymui. Slenksčio tipas, reikšmė ir elgsena daro didelę įtaką objektų aptikime. Teisingai pasirinkta slenksčio reikšmė nustato kokie objektai bus aptikti ir kokie bus ignoruojami. Atitinkami pritaikyti filtrai sumažina triukšmų kiekį ir daro įtaka objektų aptikimui tolesniuose algoritmų žingsniuose.
2. Esant skirtingam scenos apšvietimui geriausia naudoti adaptyvaus slenksčio metodą su 43x43 gardelę. Didesnis gardelės dydis nekeičia rezultatų, mažesnis padidina triukšmų kiekį. Norint pašalinti triukšmus patartina naudoti c reikšmę kurios vertė 5. Gerai veikia Filtras 5 su visuotinius slenksčiu. Jis gerai padeda išryškinti ir aptikti mažesnio intensyvumo objektus kurie yra tiriamo vaizdo kraštuose jeigu yra padidėjęs centro apšvietimas. Filtrai 6 ir 7 netinka objektų aptikimui, nes vaizdą padaro sunkiai suprantamu.
3. Filtrai 6 ir 7 padarytų klaidingų aptikimų kiekis yra didelis ir netinkamas naudoti. Filtrų 5 ir 8 rezultatų reikšmės yra mažesnės už reikšmes gautas be filtro. Filtrų 2 ir 4 reikšmės yra artimiausios reikšmėms gautoms be filtrų esant slenksčiams 1 ir 4, kitais atvejais jos yra mažesnės už reikšmes gautas be filtrų. Rezultatai gauti naudojant filtrus 1 ir 3 visada yra didesni už rezultatus gautus be filtro.

LITERATŪRA

1. Berlin, B., Kay, P., „Basic Color Terms: Their Universality and Evolution“, Berkeley: University of California Press, 1969.
2. Williams D. P., „On adaptive underwater object detection“ IEEE/RSJ International 2011 Conference on Intelligent Robots and Systems (IROS), San Francisco, California, September 25-30, 2011
3. Bakšys, B., Fedaravičius, A. Robotų technika. Kaunas: Technologija, 2004, 373 p.
4. Lenkevičius A. Kompiuterinė grafika ir vizualizacija. Vilnius: TEV leidykla, 2011.
5. Lukas M. Jūros dugno video mozaikos spalvinės korekcijos ir objektų identifikacijos metodas, Klaipėda 2011.
6. Stabingienė L. Vaizdų analizė naudojant Bajeso diskriminantines funkcijas: daktaro disertacija, Vilnius, 2012
7. Stokes M., Anderson M., Chandrasekar S., Motta R., A Standard Default Color Space for the Internet – sRGB, [žiūrėta 2016 m. kovo 26 d.] prieiga per internetą http://www.w3.org/Graphics/Color/sRGB.
8. Moroz-Lapin K. Žmogaus ir kompiuterio sąveikia, Vilnius, VU, 2006.
9. Gonzalez R., Woods R.. Digital Image Processing. Prentice Hall, 2002.
10. A. Watt, F. Policarpo. The Computer Image. Addison-Wesley, Harlow, Eng., 1998.
11. Y. Y. Boykov ir M.-P. Jolly, 2001. Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images, In Proc. IEEE Int. Conf. on Computer Vision.
12. Kamlu D. Autonomous ship recognition from color images. University of south California, 2012.
13. Rother C., Blake A., ir Kolmogorov V., 2004. GrabCut — Interactive Foreground Extraction using Iterated Graph Cuts, ACM Transactions on Graphics SIGGRAPH’04.
14. Vezhnevets V., Konouchine V., GrowCut - Interactive Multi-Label N-D Image Segmentation By Cellular Automata, Proc. of Graphicon 2005.
15. Lukas M., Ramašauskas O., Methods of color correction and identification in seafloor video mosaic, Open Readings 2011, 54th Science Conference for Young Students of Physics and Natural Sciences, Vilnius, VU, 2011, p. 80-81, ISSN 2029-4425.
16. Shapiro, L., and Stockman, G. 2002. Computer Vision. Prentice Hall. p. 69–73. [žiūrėta 2016 m. gegužės 2 d.] prieiga per internetą http://www.cse.msu.edu/~stockman/Book/2002/Chapters/ch3.pdf.
17. Sezgin M., Sankur B., Survey over image thresholding techniques and quantitative performance evaluation, Journal of Electronic Imaging 13(1), 146–165, January 2004 [žiūrėta 2016 m. gegužės 2 d.] prieiga per internetą http://pequan.lip6.fr/~bereziat/pima/2012/seuillage/sezgin04.pdf
18. Kaehler A., Bradski G. R., Learning OpenCv, Basic Thresholding Operations, 2008, O'Reilly Media ISBN 978-0-596-51613-0

19. Otsu N., "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber. 9 (1): 62–66, 1979 [žiūrėta 2016 m. gegužės 2 d.] prieiga per internetą http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4310076
20. Vala, HJ., Baxi, Astha "A review on Otsu image segmentation algorithm". International Journal of Advanced Research in Computer Engineering \& Technology (IJARCET), 2013, 2 (2): 387.
21. Crimmins R., T., "Geometric filter for speckle reduction,", Appl. Opt. 24, 1438-1443, 1985, [žiūrėta 2016 m. gegužės 2 d.] prieiga per internetą https://www.osapublishing.org/ao/abstract.cfm?uri=ao-24-10-1438
22. Matlab Edge detection methods for finding object boundaries in images [žiūrėta 2016 m. gegužės 2 d.] prieiga per internetą http://se.mathworks.com/discovery/edge-detection.html
23. Sharifi, M., Fathy, M., & Mahmoudi, M. T. (2002, April). A classified and comparative study of edge detection algorithms. In Information Technology: Coding and Computing, 2002. Proceedings. International Conference on (pp. 117-120). IEEE, [žiūrėta 2016 m. gegužės 2 d.] prieiga per internetą http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1000371
24. Sookman S. Blob Analysis and Edge Detection In the Real World 2006 [žiūrėta 2016 m. gegužės 2 d.] prieiga per internetą http://www.evaluationengineering.com/articles/200608/blob-analysis-and-edge-detection-in-the-real-world.php.
25. Moeslund, T. (2009), Canny Edge Detection [žiūrėta 2016 m. gegužės 2 d.] prieiga per internetą http://www.cse.iitd.ernet.in/~pkalra/csl783/canny.pdf
26. Adaptive vision Blob Analysis [žiūrėta 2016 m. kovo 26 d.] prieiga per internetą https://www.adaptive-vision.com/en/technical_data/documentation/3.2/machine_vision_guide/BlobAnalysis.html.
27. Matlab Computer Vision System Toolbox 2014 [žiūrėta 2016 m. gegužės 2 d.] prieiga per internetą http://www.mathworks.se/products/computer-vision/.
28. Klaipėdos universitetas. Jūros tyrimo atviros prieigos centras. [žiūrėta 2015 m. gruodį 2 d.] prieiga per internetą http://apc.ku.lt.

Aurimas Vaitkus

Povandeninių objektų aptikimo algoritmų tobulinimas.: Geoinformatikos magistro darbas. Vadovas prof. dr. Olegas Ramašauskas. Klaipėdos universitetas, Jūros technologijų gamtos mokslų fakultetas, Informatikos katedra. Klaipėda, 2016 57 p.

SANTRAUKA

Spalvos daro didelę įtaką automatizuotame objektų aptikime, tai ypač matyti naudojantis specializuota programine įranga. Jeigu kompiuteris gali fiksuoti spalva, tampa įmanoma aptikti objektus, kurie pasižymi skirtingom spalvomis ir skiriasi nuo fono. Sakoma, kad spalva yra žmogaus sugalvota sąvoka siekiant atskirti kitaip atrodančius objektus ir juos charakterizuoti [41]. Programose spalvas galima apibrėžti kaip tam tikrus skaitinių reikšmių intervalus vaizdo matricose. Regimos šviesos spektras yra ištisinė juosta su tolygiai kintančiomis reikšmėmis. Tam tikra spalva atitinka užsibrėžtą reikšmių intervalą regimos šviesos spektre ir ją atitinkančioje matricoje. Reikšmių intervalas dar priklauso nuo imtuvo ypatybių: jautrumo, skiriamosios gebos, naudojamų technologijų, trikdžių lygio ir jų pobūdžio. Spalvoto objekto aptikimui reikia žinoti spalvos reikšmių intervalus, bei parinkti filtrų slenksčius ir atidžiai nustačius slenksčius, fiksuoti spalvotus objektus, išryškinti jų kontūrus.

Darbe nagrinėjami povandenių objektų aptikimo algoritmai; programinės priemonės jiems įgyvendinti. Povandeninių objektų atpažinimo algoritmai aktualūs ieškant paskendusių objektų, jūros dugno artefaktų, jūrinės augmenijos ir gyvūnijos pavyzdžių.

Išanalizuoti povandeninių objektų algoritmai atskleidė, kad naudojamų filtrų ir slenksčių teisingos charakteristikos turi didelės įtakos objektų aptikime. Tiriamos vietovės apšvietimo vientisumas daro didžiulę įtaką objektų aptikimui po vandeniu, o fono nevientisumas gali smarkiai sumažinti objektų aptikimo galimybę. Programinės priemonėms sukurti pasitelkta Microsoft Visual Studio sistema.

PAGRINDINIAI ŽODŽIAI: povandeninių objektų atpažinimas, slenkstis, filtras

Aurimas Vaitkus

Underwater objects detection algorithms improvement.: Master Thesis of Geoinformatics. Supervisor: Assoc. prof. dr. O. Ramašauskas. Klaipeda University, Faculty of Marine Technologies and Natural Sciences, Department of Informatics. Klaipeda, 2016, 57 p.

SUMMARY

Colors have a significant impact on the automated object detection, this is especially apparent using specialized software. If the computer can capture the color, it becomes possible to detect objects that are characterized by two different colors and different from the background. It is said that color is a human invented the term to distinguish different-looking objects and characterization [41]. The programs colors can be defined as a set of numerical ranges for video matrices. Visible light spectrum is a continuous band with uniformly changing values. Some color corresponds to the set values of the range of the visible light spectrum and in its appropriate matrix. The range of values still depends on the characteristics of the receiver: sensitivity, resolution, technology used, interference levels and their nature. Colored object detection need to know the color ranges of values, and the selection of filters and thresholds after careful thresholds, capture colored objects, highlighting their contours.

The work deals with underwater object detection algorithms; software tools to implement them. Underwater object recognition algorithms for finding relevant submerged object, seabed artifacts, marine flora and fauna examples.

To analyze the underwater algorithms showed that the use of filters and thresholds correct characteristics have a significant impact on the objects detected. The survey area lighting makes a huge impact on the integrity of detecting objects under water, and the background fragmentation can greatly reduce the possibility of detection of objects. Software measures invoked to create a Microsoft Visual Studio system.

KEYWORDS: Underwater object detection, threshold, filter

TERMINŲ IR SANTRUMPŲ ŽODYNĖLIS

Duomenų bazė – susistemintas ar metodiškai sutvarkytas duomenų rinkinys, kuriuo galima individualiai naudotis elektroniniu ar kitu būdu.

Geodezinis tinklas – Žemės paviršiuje įtvirtintų ir geodeziniais matavimais susietų geodezinių ženklų visuma. Pagal nustatomus parametrus skirstomas į GPS (erdvinį), planimetrinį, vertikalųjį, gravimetrinį, magnetometrinį, o pagal užimamą teritoriją – į pasaulinį, kontinentinį, valstybinį, savivaldybių, vietinį, specialios paskirties.

GIS – geografinių objektų, jų charakteristikų ir kitos informacijos, turinčios sąsają su Žeme, kaupimo, tvarkymo, apdorojimo, saugojimo, paieškos ir pateikimo kompiuterizuota informacinė sistema, skirta projektavimo, modeliavimo, analizės, mokslo ir kitiems geografinės erdvės uždaviniams spręsti.

GPS – specializuotų dirbtinių Žemės palydovų ir prietaisų visuma, skirta padėtims nustatyti, pasauliniams ir valstybiniams geodeziniams tinklams sudaryti, atnaujinti ir kitiems teoriniams bei praktiniams uždaviniams spręsti.

Magnetometras – prietaisas, skirtas matuoti magnetinei indukcijai.

Multirezoliucinis vaizdas – vaizdas sudarytas iš daugiau nei vieno skirtingo dydžio vaizdų.

RGB, HCI, CMYK – spalvų maišymo sistemos, kuriose naudojamos trys, žmogaus akių receptorius atitinkančios spalvos: raudona (*Red*), žalia (*Green*) ir Mėlyna (*Blue*).

Pilkumo skalė, pustonių skalė (angl. *greyscale*) – skaitmeninio vaizdo formatas kuriame visi pikseliai turi tik vieną parametrą: intensyvumą. Spalvų gama apima visus pilkumo atspalvio reikšmes nuo juodos iki baltos. Kitaip žinoma kaip monochromatinis arba vienspalvis vaizdo formatas.

RLE (angl. *Run Length Encoding*) – duomenų suspaudimo algoritmas pagrįstas daugybės vienodų pasikartojančių elementų sutraukimu nurodant viena elementą ir numerį kiek kartų jis kartojasi.

ROV – povandeninis nuotolinio valdymo įrenginys, skirtas dugno tyrimams atlikti.

SCC – Segmentacijos ir spalvinės korekcijos algoritmas.

Sonaras, echolotas – prietaisas, naudojamas po vandeniu esantiems kūnams aptikti.

T o p o g r a f i j a – geodezijos ir matavimų inžinerijos veiklos sritis, apimanti Žemės paviršiaus gamtinių ir fizinių (reljefo, hidrografijos, augmenijos), antropogeninių bei pavienių topografinių objektų erdvinius matavimus bei vaizdavimą topožemėlapiuose ir planuose standartizuotais metodais.

Ž e m ė l a p i s – sumažintas ir apibendrintas Žemės paviršiaus objektų bei gamtinių arba socialinių-ekonominių reiškinių vaizdas plokštumoje, išreikštas matematine projekcija, nustatytu masteliu, sutartiniais ženklais.

Ž e m ė l a p i s , s k a i t m e n i n i s – vietovės modelis, kurį sudaro užkoduotų vietovės taškų erdvinių koordinačių ir charakteristikų visuma, užrašyta informacijos nustatytos struktūros laikmenoje vektoriniu arba rastriniu pavidalu.

PRIEDAS 1

PROGRAMINIS KODAS

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <opencv/cv.h>
#include <math.h>

using namespace cv;
using namespace std;

Mat filter1(Mat image){
            Mat image2=image.clone();
                        for (int i=1; i<image.rows-1;i++)
            {
                        for(int j=1;j<image.cols-1;j++)
                        {

            image2.at<uchar>(i,j)=image.at<uchar>(i,j)+(image.at<uchar>(i-1,j-
1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-
1,j)+image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-
1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/8;
                        }
            }
                        return image2;
}

Mat filter2(Mat image){
            Mat image2=image.clone();
                        for (int i=1; i<image.rows-1;i++)
            {
                        for(int j=1;j<image.cols-1;j++)
                        {

            image2.at<uchar>(i,j)=image2.at<uchar>(i,j)+(image2.at<uchar>(i-1,j-
1)+image2.at<uchar>(i,j-1)+image2.at<uchar>(i+1,j-1)+image2.at<uchar>(i-
1,j)+image2.at<uchar>(i-1,j+1)+image2.at<uchar>(i+1,j-
1)+image2.at<uchar>(i+1,j)+image2.at<uchar>(i+1,j+1))/8;
                        }
            }
                        return image2;
}

Mat filter3(Mat image){
                        for (int i=1; i<image.rows-1;i++)
            {
                        for(int j=1;j<image.cols-1;j++)
                        {

            image.at<uchar>(i,j)=image.at<uchar>(i,j)*2+(image.at<uchar>(i-1,j-
1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-
1,j)+image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-
1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/9;
                        }
            }
                        return image;
}
```

```
Mat filter4(Mat image){
                        for (int i=1; i<image.rows-1;i++)
            {
                        for(int j=1;j<image.cols-1;j++)
                        {

            image.at<uchar>(i,j)=image.at<uchar>(i,j)+(image.at<uchar>(i,j)+image.at<
uchar>(i-1,j-1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-
1,j)+image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-
1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/9;
                        }
            }
                        return image;
}

Mat filter5(Mat image){
                        for (int i=1; i<image.rows-1;i++)
            {
                        for(int j=1;j<image.cols-1;j++)
                        {
                                    image.at<uchar>(i,j)=image.at<uchar>(i,j)-
(image.at<uchar>(i,j)+image.at<uchar>(i-1,j-1)+image.at<uchar>(i,j-
1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-1,j)+image.at<uchar>(i-
1,j+1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/9;
                        }
            }
                        return image;
}

Mat filter6(Mat image){
                        for (int i=1; i<image.rows-1;i++)
            {
                        for(int j=1;j<image.cols-1;j++)
                        {

            image.at<uchar>(i,j)=image.at<uchar>(i,j)*(image.at<uchar>(i,j)+image.at<
uchar>(i-1,j-1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-
1,j)+image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-
1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/9;
                        }
            }
                        return image;
}
Mat filter7(Mat image){
            Mat image2=image.clone();
                        for (int i=1; i<image.rows-1;i++)
            {
                        for(int j=1;j<image.cols-1;j++)
                        {

            image2.at<uchar>(i,j)=pow((image.at<uchar>(i,j)+image.at<uchar>(i-1,j-
1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-
1,j)+image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-
1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/8, 2);
                        }
            }
                        return image2;
```

```
}
Mat filter8(Mat image){
            Mat image2=image.clone();
            image2=image2*2;
                        return image2;
}

Mat nofilter(Mat image){
                        return image;
}
void konturai(Mat frame)
{
                        vector<vector<cv::Point> > contours2; //holds all found
contours
                        vector<Vec4i> hierarchy;


            findContours(frame,contours2,hierarchy,CV_RETR_LIST,CV_CHAIN_APPROX_SIMPL
E, cvPoint(0,0));

                        for( int i = 0; i< contours2.size(); i++ )
                        {
                        Scalar color = Scalar(255,0,0); // blue color
                        drawContours(frame, contours2, i, color, 2, 8,hierarchy, 0);
                        }
}
Mat image;
Mat filteredImage;
int main( int argc, char** argv )
{
    if( argc != 2)
    {
     cout <<" Nepasirinktas paveikslas. No image." << endl;
             waitKey(43);
     return -1;
    }
            image=imread( argv[1], 1 );
    if(! image.data )
    {
        cout << "Could not open or find the image" << std::endl ;
                        waitKey(43);
        return -1;
    }
            Mat gray_image, mean_div;
            cvtColor( image, gray_image, CV_BGR2GRAY );
    namedWindow( "Display window", WINDOW_AUTOSIZE );
            namedWindow( "MD", WINDOW_AUTOSIZE );
//          namedWindow( "MD2", WINDOW_AUTOSIZE );
            filteredImage=nofilter(gray_image);
            //filteredImage=filter5(filteredImage);
             /* THRESHOLD CODES FOR GLOBAL THRESHOLDING!
             0: Binary
     1: Binary Inverted
     2: Threshold Truncated
     3: Threshold to Zero
     4: Threshold to Zero Inverted
   */
```

```
            //adaptiveThreshold(filteredImage, filteredImage, 255,
ADAPTIVE_THRESH_GAUSSIAN_C,  THRESH_BINARY, 113, 5);
                        inRange(filteredImage, Scalar(120, 255, 1), Scalar(250, 255,
255), filteredImage); //green
                        //inRange(filteredImage, Scalar(20, 255, 1), Scalar(250, 255,
255), filteredImage); // blue
             //threshold(filteredImage, filteredImage, 120, 255, 0);
             //threshold(filteredImage, filteredImage, 90, 0, 2);
             //threshold(filteredImage, filteredImage, 60, 255, 0);
            //morphological opening (removes small objects from the foreground)
                        erode(filteredImage, filteredImage,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
                        dilate( filteredImage, filteredImage,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
                        //morphological closing (removes small holes from the
foreground)
                        dilate( filteredImage, filteredImage,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
                        erode(filteredImage, filteredImage,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
                        vector<vector<cv::Point> > contours2; //holds all found
contours
                        vector<Vec4i> hierarchy;

            findContours(filteredImage,contours2,hierarchy,CV_RETR_LIST,CV_CHAIN_APPR
OX_SIMPLE, cvPoint(0,0));
                        cout << " found "<<contours2.size()<<" \n";
                        for( int i = 0; i< contours2.size(); i++ )
                        {
                        Scalar color = Scalar(255,0,0); // blue color
                        drawContours(image, contours2, i, color, 2, 8,hierarchy, 0);
                        }
    imshow( "Display window", image );
            imshow( "MD",  filteredImage);
//          imshow( "MD2", filteredImage2);
    waitKey(0);
    return 0;
}
```

PROGRAMA 2

```
#pragma once
#include <iostream>
#include <opencv2/opencv.hpp>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <msclr\marshal_cppstd.h>
namespace Project3 {

            using namespace System;
            using namespace System::ComponentModel;
            using namespace System::Collections;
            using namespace System::Windows::Forms;
            using namespace System::Data;
            using namespace System::Drawing;
            using namespace cv;
```

```
		/// <summary>
		/// Summary for MainForm
		/// </summary>
		public ref class MainForm : public System::Windows::Forms::Form
		{
		public:
			MainForm(void)
			{
				InitializeComponent();
				//
				//TODO: Add the constructor code here
				//

			}

			Mat filter1(Mat image){
		Mat image2=image.clone();
			for (int i=1; i<image.rows-1;i++)
		{
			for(int j=1;j<image.cols-1;j++)
			{

		image2.at<uchar>(i,j)=image.at<uchar>(i,j)+(image.at<uchar>(i-1,j-
1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-1,j)+image.at<uchar>(i-
1,j+1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/8;
			}
		}
			return image2;
}

Mat filter2(Mat image){
		Mat image2=image.clone();
			for (int i=1; i<image.rows-1;i++)
		{
			for(int j=1;j<image.cols-1;j++)
			{

		image2.at<uchar>(i,j)=image2.at<uchar>(i,j)+(image2.at<uchar>(i-1,j-
1)+image2.at<uchar>(i,j-1)+image2.at<uchar>(i+1,j-1)+image2.at<uchar>(i-
1,j)+image2.at<uchar>(i-1,j+1)+image2.at<uchar>(i+1,j-
1)+image2.at<uchar>(i+1,j)+image2.at<uchar>(i+1,j+1))/8;
			}
		}
			return image2;
}

Mat filter3(Mat image){
			for (int i=1; i<image.rows-1;i++)
		{
			for(int j=1;j<image.cols-1;j++)
			{

		image.at<uchar>(i,j)=image.at<uchar>(i,j)*2+(image.at<uchar>(i-1,j-
1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-1,j)+image.at<uchar>(i-
1,j+1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/9;
			}
		}
			return image;
}

Mat filter4(Mat image){
			for (int i=1; i<image.rows-1;i++)
		{
			for(int j=1;j<image.cols-1;j++)
			{
```

```
            image.at<uchar>(i,j)=image.at<uchar>(i,j)+(image.at<uchar>(i,j)+image.at<uchar>
(i-1,j-1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-
1,j)+image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-
1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/9;
                        }
            }
                        return image;
}

Mat filter5(Mat image){
                        for (int i=1; i<image.rows-1;i++)
            {
                        for(int j=1;j<image.cols-1;j++)
                        {
                                    image.at<uchar>(i,j)=image.at<uchar>(i,j)-
(image.at<uchar>(i,j)+image.at<uchar>(i-1,j-1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-
1)+image.at<uchar>(i-1,j)+image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-
1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/9;
                        }
            }
                        return image;
}

Mat filter6(Mat image){
                        for (int i=1; i<image.rows-1;i++)
            {
                        for(int j=1;j<image.cols-1;j++)
                        {

            image.at<uchar>(i,j)=image.at<uchar>(i,j)*(image.at<uchar>(i,j)+image.at<uchar>
(i-1,j-1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-
1,j)+image.at<uchar>(i-1,j+1)+image.at<uchar>(i+1,j-
1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/9;
                        }
            }
                        return image;
}
Mat filter7(Mat image){
            Mat image2=image.clone();
                        for (int i=1; i<image.rows-1;i++)
            {
                        for(int j=1;j<image.cols-1;j++)
                        {

            image2.at<uchar>(i,j)=pow((image.at<uchar>(i,j)+image.at<uchar>(i-1,j-
1)+image.at<uchar>(i,j-1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i-1,j)+image.at<uchar>(i-
1,j+1)+image.at<uchar>(i+1,j-1)+image.at<uchar>(i+1,j)+image.at<uchar>(i+1,j+1))/8, 2);
                        }
            }
                        return image2;
}

Mat filter8(Mat image){
            Mat image2=image.clone();
            /*          for (int i=1; i<image.rows-1;i++)
            {
                        for(int j=1;j<image.cols-1;j++)
                        {
                                    image2.at<uchar>(i,j)=image2.at<uchar>(i,j)*2;
                        }
            }*/
            image2=image2*2;
                        return image2;
}
```

```
		protected:
			/// <summary>
			/// Clean up any resources being used.
			/// </summary>
			~MainForm()
			{
				if (components)
				{
					delete components;
				}
			}
		private: System::Windows::Forms::Label^  label1;
		private: System::Windows::Forms::Button^  button1;
		private: System::Windows::Forms::OpenFileDialog^  openFileDialog1;
		private: System::Windows::Forms::Label^  label2;
		private: System::Windows::Forms::CheckBox^  checkBox1;
		private: System::Windows::Forms::TrackBar^  trackBar1;
		private: System::Windows::Forms::TrackBar^  trackBar2;
		private: System::Windows::Forms::Label^  label3;
		private: System::Windows::Forms::Label^  label5;
		private: System::Windows::Forms::Label^  trackBar2Value;
		private: System::Windows::Forms::Label^  trackBar1Value;
		private: System::Windows::Forms::Panel^  panel1;
		private: System::Windows::Forms::Label^  label4;
		private: System::Windows::Forms::TabControl^  tabControl1;
		private: System::Windows::Forms::TabPage^  tabPage1;
		private: System::Windows::Forms::TabPage^  tabPage2;
		private: System::Windows::Forms::GroupBox^  groupBox1;
		private: System::Windows::Forms::RadioButton^  threshToZeroInv_radio;

		private: System::Windows::Forms::RadioButton^  threshToZero_radio;


		private: System::Windows::Forms::RadioButton^  threshBinaryInv_radio;


		private: System::Windows::Forms::RadioButton^  threshBinary_radio;


		private: System::Windows::Forms::RadioButton^  threshColor_radio;
		private: System::Windows::Forms::RadioButton^  threshTrunc_radio;
		protected:

		private:
			/// <summary>
			/// Required designer variable.
			/// </summary>
			System::ComponentModel::Container ^components;



#pragma region Windows Form Designer generated code
			/// <summary>
			/// Required method for Designer support - do not modify
			/// the contents of this method with the code editor.
			/// </summary>
			void InitializeComponent(void)
			{
				this->label1 = (gcnew System::Windows::Forms::Label());
				this->button1 = (gcnew
System::Windows::Forms::Button());
				this->openFileDialog1 = (gcnew
System::Windows::Forms::OpenFileDialog());
				this->label2 = (gcnew System::Windows::Forms::Label());
```

```
                                    this->checkBox1 = (gcnew
System::Windows::Forms::CheckBox());
                                    this->trackBar1 = (gcnew
System::Windows::Forms::TrackBar());
                                    this->trackBar2 = (gcnew
System::Windows::Forms::TrackBar());
                                    this->label3 = (gcnew System::Windows::Forms::Label());
                                    this->label5 = (gcnew System::Windows::Forms::Label());
                                    this->trackBar2Value = (gcnew
System::Windows::Forms::Label());
                                    this->trackBar1Value = (gcnew
System::Windows::Forms::Label());
                                    this->panel1 = (gcnew System::Windows::Forms::Panel());
                                    this->label4 = (gcnew System::Windows::Forms::Label());
                                    this->tabControl1 = (gcnew
System::Windows::Forms::TabControl());
                                    this->tabPage1 = (gcnew
System::Windows::Forms::TabPage());
                                    this->tabPage2 = (gcnew
System::Windows::Forms::TabPage());
                                    this->groupBox1 = (gcnew
System::Windows::Forms::GroupBox());
                                    this->threshTrunc_radio = (gcnew
System::Windows::Forms::RadioButton());
                                    this->threshToZeroInv_radio = (gcnew
System::Windows::Forms::RadioButton());
                                    this->threshToZero_radio = (gcnew
System::Windows::Forms::RadioButton());
                                    this->threshBinaryInv_radio = (gcnew
System::Windows::Forms::RadioButton());
                                    this->threshBinary_radio = (gcnew
System::Windows::Forms::RadioButton());
                                    this->threshColor_radio = (gcnew
System::Windows::Forms::RadioButton());

            (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this-
>trackBar1))->BeginInit();

            (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this-
>trackBar2))->BeginInit();
                                    this->panel1->SuspendLayout();
                                    this->tabControl1->SuspendLayout();
                                    this->tabPage1->SuspendLayout();
                                    this->tabPage2->SuspendLayout();
                                    this->groupBox1->SuspendLayout();
                                    this->SuspendLayout();
                                    //
                                    // label1
                                    //
                                    this->label1->AutoSize = true;
                                    this->label1->Location = System::Drawing::Point(12, 9);
                                    this->label1->Name = L"label1";
                                    this->label1->Size = System::Drawing::Size(55, 13);
                                    this->label1->TabIndex = 0;
                                    this->label1->Text = L"File name:";
                                    //
                                    // button1
                                    //
                                    this->button1->Location = System::Drawing::Point(15,
26);
                                    this->button1->Name = L"button1";
                                    this->button1->Size = System::Drawing::Size(75, 23);
                                    this->button1->TabIndex = 1;
                                    this->button1->Text = L"Choose file";
                                    this->button1->UseVisualStyleBackColor = true;
```

```
                                    this->button1->Click += gcnew
System::EventHandler(this, &MainForm::button1_Click);
                                    //
                                    // openFileDialog1
                                    //
                                    this->openFileDialog1->FileName = L"openFileDialog1";
                                    //
                                    // label2
                                    //
                                    this->label2->AutoSize = true;
                                    this->label2->Location = System::Drawing::Point(73, 9);
                                    this->label2->Name = L"label2";
                                    this->label2->Size = System::Drawing::Size(0, 13);
                                    this->label2->TabIndex = 2;
                                    //
                                    // checkBox1
                                    //
                                    this->checkBox1->AutoSize = true;
                                    this->checkBox1->Location = System::Drawing::Point(15,
55);
                                    this->checkBox1->Name = L"checkBox1";
                                    this->checkBox1->Size = System::Drawing::Size(132, 17);
                                    this->checkBox1->TabIndex = 3;
                                    this->checkBox1->Text = L"Histogram equalisation";
                                    this->checkBox1->UseVisualStyleBackColor = true;
                                    //
                                    // trackBar1
                                    //
                                    this->trackBar1->Location = System::Drawing::Point(9,
83);
                                    this->trackBar1->Maximum = 255;
                                    this->trackBar1->Name = L"trackBar1";
                                    this->trackBar1->Size = System::Drawing::Size(260, 45);
                                    this->trackBar1->TabIndex = 4;
                                    this->trackBar1->Value = 255;
                                    this->trackBar1->Scroll += gcnew
System::EventHandler(this, &MainForm::trackBar1_Scroll);
                                    //
                                    // trackBar2
                                    //
                                    this->trackBar2->Location = System::Drawing::Point(3,
19);
                                    this->trackBar2->Maximum = 255;
                                    this->trackBar2->Name = L"trackBar2";
                                    this->trackBar2->Size = System::Drawing::Size(260, 45);
                                    this->trackBar2->TabIndex = 5;
                                    this->trackBar2->Value = 1;
                                    this->trackBar2->Scroll += gcnew
System::EventHandler(this, &MainForm::trackBar2_Scroll);
                                    //
                                    // label3
                                    //
                                    this->label3->Location = System::Drawing::Point(6, 3);
                                    this->label3->Name = L"label3";
                                    this->label3->Size = System::Drawing::Size(35, 13);
                                    this->label3->TabIndex = 0;
                                    this->label3->Text = L"Low";
                                    //
                                    // label5
                                    //
                                    this->label5->Location = System::Drawing::Point(6, 67);
                                    this->label5->Name = L"label5";
                                    this->label5->Size = System::Drawing::Size(35, 13);
                                    this->label5->TabIndex = 0;
                                    this->label5->Text = L"High";
```

```
                                    //
                                    // trackBar2Value
                                    //
                                    this->trackBar2Value->AutoSize = true;
                                    this->trackBar2Value->Location =
System::Drawing::Point(47, 3);
                                    this->trackBar2Value->Name = L"trackBar2Value";
                                    this->trackBar2Value->Size = System::Drawing::Size(13,
13);
                                    this->trackBar2Value->TabIndex = 6;
                                    this->trackBar2Value->Text = L"1";
                                    //
                                    // trackBar1Value
                                    //
                                    this->trackBar1Value->AutoSize = true;
                                    this->trackBar1Value->Location =
System::Drawing::Point(47, 67);
                                    this->trackBar1Value->Name = L"trackBar1Value";
                                    this->trackBar1Value->Size = System::Drawing::Size(25,
13);
                                    this->trackBar1Value->TabIndex = 7;
                                    this->trackBar1Value->Text = L"255";
                                    //
                                    // panel1
                                    //
                                    this->panel1->Controls->Add(this->trackBar1Value);
                                    this->panel1->Controls->Add(this->trackBar1);
                                    this->panel1->Controls->Add(this->trackBar2);
                                    this->panel1->Controls->Add(this->trackBar2Value);
                                    this->panel1->Controls->Add(this->label5);
                                    this->panel1->Controls->Add(this->label3);
                                    this->panel1->Location = System::Drawing::Point(6, 6);
                                    this->panel1->Name = L"panel1";
                                    this->panel1->Size = System::Drawing::Size(276, 140);
                                    this->panel1->TabIndex = 8;
                                    //
                                    // label4
                                    //
                                    this->label4->AutoSize = true;
                                    this->label4->Location = System::Drawing::Point(12,
75);
                                    this->label4->Name = L"label4";
                                    this->label4->Size = System::Drawing::Size(94, 13);
                                    this->label4->TabIndex = 9;
                                    this->label4->Text = L"Threshold controls";
                                    //
                                    // tabControl1
                                    //
                                    this->tabControl1->Controls->Add(this->tabPage1);
                                    this->tabControl1->Controls->Add(this->tabPage2);
                                    this->tabControl1->Location =
System::Drawing::Point(15, 91);
                                    this->tabControl1->Name = L"tabControl1";
                                    this->tabControl1->SelectedIndex = 0;
                                    this->tabControl1->Size = System::Drawing::Size(304,
201);
                                    this->tabControl1->TabIndex = 10;
                                    //
                                    // tabPage1
                                    //
                                    this->tabPage1->Controls->Add(this->panel1);
                                    this->tabPage1->Location = System::Drawing::Point(4,
22);
                                    this->tabPage1->Name = L"tabPage1";
```

```
                                    this->tabPage1->Padding =
System::Windows::Forms::Padding(3);
                                    this->tabPage1->Size = System::Drawing::Size(296, 175);
                                    this->tabPage1->TabIndex = 0;
                                    this->tabPage1->Text = L"Controls";
                                    this->tabPage1->UseVisualStyleBackColor = true;
                                    //
                                    // tabPage2
                                    //
                                    this->tabPage2->Controls->Add(this->groupBox1);
                                    this->tabPage2->Location = System::Drawing::Point(4,
22);
                                    this->tabPage2->Name = L"tabPage2";
                                    this->tabPage2->Padding =
System::Windows::Forms::Padding(3);
                                    this->tabPage2->Size = System::Drawing::Size(296, 175);
                                    this->tabPage2->TabIndex = 1;
                                    this->tabPage2->Text = L"Threshold choices";
                                    this->tabPage2->UseVisualStyleBackColor = true;
                                    //
                                    // groupBox1
                                    //
                                    this->groupBox1->Controls->Add(this-
>threshTrunc_radio);
                                    this->groupBox1->Controls->Add(this-
>threshToZeroInv_radio);
                                    this->groupBox1->Controls->Add(this-
>threshToZero_radio);
                                    this->groupBox1->Controls->Add(this-
>threshBinaryInv_radio);
                                    this->groupBox1->Controls->Add(this-
>threshBinary_radio);
                                    this->groupBox1->Controls->Add(this-
>threshColor_radio);
                                    this->groupBox1->Location = System::Drawing::Point(6,
8);
                                    this->groupBox1->Name = L"groupBox1";
                                    this->groupBox1->Size = System::Drawing::Size(266,
161);
                                    this->groupBox1->TabIndex = 0;
                                    this->groupBox1->TabStop = false;
                                    this->groupBox1->Text = L"Choices";
                                    //
                                    // threshTrunc_radio
                                    //
                                    this->threshTrunc_radio->AutoSize = true;
                                    this->threshTrunc_radio->Location =
System::Drawing::Point(7, 138);
                                    this->threshTrunc_radio->Name = L"threshTrunc_radio";
                                    this->threshTrunc_radio->Size =
System::Drawing::Size(114, 17);
                                    this->threshTrunc_radio->TabIndex = 5;
                                    this->threshTrunc_radio->Text = L"THRESH_TRUNC";
                                    this->threshTrunc_radio->UseVisualStyleBackColor =
true;
                                    //
                                    // threshToZeroInv_radio
                                    //
                                    this->threshToZeroInv_radio->AutoSize = true;
                                    this->threshToZeroInv_radio->Location =
System::Drawing::Point(7, 115);
                                    this->threshToZeroInv_radio->Name =
L"threshToZeroInv_radio";
                                    this->threshToZeroInv_radio->Size =
System::Drawing::Size(145, 17);
```

```
                                    this->threshToZeroInv_radio->TabIndex = 4;
                                    this->threshToZeroInv_radio->Text =
L"THRESH_TOZERO_INV";
                                    this->threshToZeroInv_radio->UseVisualStyleBackColor =
true;
                                    //
                                    // threshToZero_radio
                                    //
                                    this->threshToZero_radio->AutoSize = true;
                                    this->threshToZero_radio->Location =
System::Drawing::Point(7, 91);
                                    this->threshToZero_radio->Name = L"threshToZero_radio";
                                    this->threshToZero_radio->Size =
System::Drawing::Size(121, 17);
                                    this->threshToZero_radio->TabIndex = 3;
                                    this->threshToZero_radio->Text = L"THRESH_TOZERO";
                                    this->threshToZero_radio->UseVisualStyleBackColor =
true;
                                    //
                                    // threshBinaryInv_radio
                                    //
                                    this->threshBinaryInv_radio->AutoSize = true;
                                    this->threshBinaryInv_radio->Location =
System::Drawing::Point(7, 67);
                                    this->threshBinaryInv_radio->Name =
L"threshBinaryInv_radio";
                                    this->threshBinaryInv_radio->Size =
System::Drawing::Size(140, 17);
                                    this->threshBinaryInv_radio->TabIndex = 2;
                                    this->threshBinaryInv_radio->Text =
L"THRESH_BINARY_INV";
                                    this->threshBinaryInv_radio->UseVisualStyleBackColor =
true;
                                    //
                                    // threshBinary_radio
                                    //
                                    this->threshBinary_radio->AutoSize = true;
                                    this->threshBinary_radio->Location =
System::Drawing::Point(7, 43);
                                    this->threshBinary_radio->Name = L"threshBinary_radio";
                                    this->threshBinary_radio->Size =
System::Drawing::Size(116, 17);
                                    this->threshBinary_radio->TabIndex = 1;
                                    this->threshBinary_radio->Text = L"THRESH_BINARY";
                                    this->threshBinary_radio->UseVisualStyleBackColor =
true;
                                    //
                                    // threshColor_radio
                                    //
                                    this->threshColor_radio->AutoSize = true;
                                    this->threshColor_radio->Checked = true;
                                    this->threshColor_radio->Location =
System::Drawing::Point(7, 20);
                                    this->threshColor_radio->Name = L"threshColor_radio";
                                    this->threshColor_radio->Size =
System::Drawing::Size(113, 17);
                                    this->threshColor_radio->TabIndex = 0;
                                    this->threshColor_radio->TabStop = true;
                                    this->threshColor_radio->Text = L"THRESH_COLOR";
                                    this->threshColor_radio->UseVisualStyleBackColor =
true;
                                    //
                                    // MainForm
                                    //
```

```
                                    this->AutoScaleDimensions = System::Drawing::SizeF(6,
13);
                                    this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
                                    this->ClientSize = System::Drawing::Size(325, 296);
                                    this->Controls->Add(this->tabControl1);
                                    this->Controls->Add(this->label4);
                                    this->Controls->Add(this->checkBox1);
                                    this->Controls->Add(this->label2);
                                    this->Controls->Add(this->button1);
                                    this->Controls->Add(this->label1);
                                    this->Name = L"MainForm";
                                    this->Text = L"MainForm";
                                    this->Load += gcnew System::EventHandler(this,
&MainForm::MainForm_Load);

            (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this-
>trackBar1))->EndInit();

            (cli::safe_cast<System::ComponentModel::ISupportInitialize^  >(this-
>trackBar2))->EndInit();
                                    this->panel1->ResumeLayout(false);
                                    this->panel1->PerformLayout();
                                    this->tabControl1->ResumeLayout(false);
                                    this->tabPage1->ResumeLayout(false);
                                    this->tabPage2->ResumeLayout(false);
                                    this->groupBox1->ResumeLayout(false);
                                    this->groupBox1->PerformLayout();
                                    this->ResumeLayout(false);
                                    this->PerformLayout();

                        }
#pragma endregion
            private: System::Void button1_Click(System::Object^  sender, System::EventArgs^
e) {
                                                    if(openFileDialog1->ShowDialog() ==
System::Windows::Forms::DialogResult::OK)
      {
                          System::String^ byla=openFileDialog1->FileName;
                         label2->Text=byla;
                          msclr::interop::marshal_context context;
                        std::string fileName = context.marshal_as<std::string>(byla);
                        //TODO this code is really bad and should be rewritten
                            VideoCapture cap(fileName);
    if ( !cap.isOpened() )
    {
                        MessageBox::Show("Failed to open file", "Failed to open
file",MessageBoxButtons::OK);
                        return;
    }

    while (true)
    {
        Mat imgOriginal;

        bool bSuccess = cap.read(imgOriginal);

         if (!bSuccess)
        {
                                    MessageBox::Show("End of video! No more frames to
read.", "End of video",MessageBoxButtons::OK);
             break;
        }

                         Mat imgGrayScale;
```

```
                        cvtColor(imgOriginal, imgGrayScale, CV_BGR2GRAY);
                        if(checkBox1->Checked)
                        equalizeHist(imgGrayScale, imgGrayScale);

                        Mat imgThresholded=filter3(imgGrayScale);
                        IplImage* img2;
                        //if(threshColor_radio->Checked==true){
                        inRange(imgThresholded, Scalar(trackBar2->Value, 1, 1),
Scalar(trackBar1->Value, 255, 255), imgThresholded); //Threshold the image



                        }
                        else
                        if(threshBinary_radio->Checked==true){
                         threshold(imgThresholded,imgThresholded,trackBar2-
>Value,trackBar1->Value,CV_THRESH_BINARY);
                        }else
                        if(threshBinaryInv_radio->Checked==true){
                         threshold(imgThresholded,imgThresholded,trackBar2-
>Value,trackBar1->Value,CV_THRESH_BINARY_INV);
                        }else
                        if(threshToZero_radio->Checked==true){
                        threshold(imgThresholded,imgThresholded,trackBar2->Value,trackBar1-
>Value,CV_THRESH_TOZERO);
                        }else
                        if(threshToZeroInv_radio->Checked==true){
                         threshold(imgThresholded,imgThresholded,trackBar2-
>Value,trackBar1->Value,CV_THRESH_TOZERO_INV);
                        }else
                        if(threshTrunc_radio->Checked==true){
                         threshold(imgThresholded,imgThresholded,trackBar2-
>Value,trackBar1->Value,CV_THRESH_TRUNC);
                        }

                        //morphological opening (removes small objects from the foreground)
                        erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
                        dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );

                        //morphological closing (removes small holes from the foreground)
                        dilate( imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );
                        erode(imgThresholded, imgThresholded,
getStructuringElement(MORPH_ELLIPSE, cv::Size(5, 5)) );


                        vector<vector<cv::Point> > contours2; //holds all found contours
                        vector<Vec4i> hierarchy;



            findContours(imgThresholded,contours2,hierarchy,CV_RETR_LIST,CV_CHAIN_APPROX_SI
MPLE, cvPoint(0,0));

                        for( int i = 0; i< contours2.size(); i++ )
     {
       Scalar color = Scalar(255,0,0); // blue color
       drawContours(imgOriginal, contours2, i, color, 2, 8,hierarchy, 0);
     }
                        //
```

```
                                        imshow("Thresholded Image", imgThresholded); //show the
thresholded image

                            }
                            imshow("Original", imgOriginal); //show the original image

          if (waitKey(30) == 27)
         {
                               MessageBox::Show("User terminated execution", "End of
video",MessageBoxButtons::OK);
              break;
         }
     }
              destroyAllWindows();
       }
                                        }

private: System::Void trackBar2_Scroll(System::Object^  sender, System::EventArgs^  e) {
                                        trackBar2Value->Text=trackBar2->Value+"";
                             }
private: System::Void trackBar1_Scroll(System::Object^  sender, System::EventArgs^  e) {
                                        trackBar1Value->Text=trackBar1->Value+"";
                             }
private: System::Void MainForm_Load(System::Object^  sender, System::EventArgs^  e) {
                             }
};
}
```