LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

SPORTA SACENSĪBU FOTOGRĀFIJU ANOTĒŠANA

BAKALAURA DARBS

Autors: Dmitrijs Surženko
St. apl. nr.: ds07019
Darba vadītājs: Mg.dat. Rūdolfs Opmanis

Rīga, 2011

ANOTĀCIJA

Darba mērķis ir izpētīt iespējas veikt automātisku dalībnieku numuru atpazīšanu sporta notikumu fotogrāfijās.

Darba gaitā tiks izpētīti eksistējošie paņēmieni un algoritmi datora redzes īstenošanai, kā arī apgūti pieejamie zinātniskie raksti. Balstoties uz iegūto informāciju, autors izpētīja iespēju veidot sistēmu dalībnieku numuru atpazīšanai.

Darba rezultātā tiks apstiprināta iespēja veidot datora redzes sistēmu numuru atpazīšanai. Pēc teorētiskā materiāla izpētes autors veic datora redzes programmas izstrādi, izmantojot Java programmēšanas valodu. Ar programmas palīdzību ir iespējams atpazīt fotogrāfijās redzamos dalībnieku numurus.

ABSTRACT

The main purpose of this paper is to investigate the possibility of automatic participants' number recognition on sport event photos.

Existing techniques' and algorithms' of computer vision implementations are researched and studied while working on this paper. Based on theoretical research author investigates possibility to develop working computer-vision system that could automatically tag photos, by the number plates visible in the photo.

The successful implementation of computer vision powered number recognition system is the main achievement of this paper. As a result of theory studies author could develop solution to the described problem and create working prototype in Java programming language that could recognizes number plates on sport event photos.

ATSLĒGVĀRDI

Automātiskā anotēšana

Datora redze

Tēlu atpazīšana

OCR

Hafa transformācija

Malu meklēšana

Kanni malu detektors

KEYWORDS

Automatic annotation

Computer vision

Patten recognition

OCR

Hough transform

Edge detection

Canny edge detector

SATURS

TERMINI UN SAĪSINĀJUMI

RFID – Radiofrekvences identifikācija (*angļu val.*: Radio Frequency IDentification jeb RFID) ir tehnoloģija automātiskai identifikācijai, kurā datus nosūta un saņem ar radiosignālu palīdzību, izmantojot ierīces, ko sauc pa RFID tagiem jeb retranslatoriem[1].

RGB – saīsinājums no angļu val. Red, Green, Blue – Sarkans, Zaļš, Zils krāsu modelis, kurā krāsa tiek reprezentēta ar trim krāsu komponentiem.

YUV – krāsu modelis, kurā krāsa tiek reprezentēta ar spilgtuma kanālu un diviem krāsu kanāliem. Izstrādāts, ņemot vērā cilvēka redzes īpatnības, lai nodrošinātu kvalitatīvāku attēla pārraidi, samazinot datu plūsmu.

GPL – Vispārēja publiskā licence (*angļu val.*: General Public Licence jeb GPL) ir plaši izmantojama brīva licence. Saturs licencēts ar GPL licenci, var tikt brīvi izplatīts un modificēts.

JPEG - (akronīms: Joint Photographic Experts Group) ir digitālo attēlu kompresijas formāts.

OCR – optiskā simbolu atpazīšana (*angļu val.*: Optical character Recognition jeb OCR) ir mehāniskā vai elektroniskā skanētu rokrakstu vai drukātu attēlu translācija mašīnu kodētā veidā.

CAPTCHA- (IPA: /kæptʃə/) ir autentifikācijas tehnoloģija, kas paredzēta, lai noteiktu, vai datorsistēmas lietotājs ir cilvēks vai mašīna. Tajā lietotājam jāveic neliels tests, kas tiek uzskatīts par neizpildāmu datoram, bet viegli veicamu cilvēkam.

API - (*angļu val*: Application programming interface) ir lietojumprogrammas saskarne. Specifisku protokolu un/vai likumu kopums, kas apraksta lietojumu savstarpējās komunikācijas iespēju un veidu.

---

1 http://lv.wikipedia.org/wiki/Radiofrekvences_identifik%C4%81cija

IEVADS

Datora redzes sistēmām mūsdienās ir daudz pielietojumu. Tās plaši tiek izmantotas mūsu dzīvē: sacīkstes kontroles sistēmas atpazīst automobiļu numurus, plastmasas pudeļu pieņemšanas automāti atpazīst pārstrādes simbolus uz pudelēm, naudas iemaksas bankomāti spēj atšķirt banknotes, mobilais telefons un fotokameras automātiski sameklē cilvēku sejas fotogrāfijās un precīzāk fokusējas uz tām. Datora jaudai palielinoties, algoritmi, kas agrāk prasīja pamatīgu laiku izpildei, tagad veicami reālajā laikā: video konferenču kameras spēj automātiski sekot prezentētājam, lietotnes mobilajā telefonā tulko kameras filmētos tekstus ceļa zīmēs un veikalu izkārtnēs, rokrakstu atpazīst planšetdatoros.

Skanētus drukātus tekstus jau sen atpazīst datorprogrammas. Ar OCR programmu palīdzību apstrādā centralizētus eksāmenu datus, aptaujas anketas, pat skaita vēlētāju balsis vēlēšanās. Ar OCR programmām pamatā ir iespējams atpazīt tikai drukātus tekstus, bet ir programmas, kas spēj atpazīt pat rokrakstu.

Apkārt esošajiem priekšmetiem var piešķirt jaunas īpašības virtuālajā pasaulē. Ikdienišķi priekšmeti var kļūt par datora kontroles rīku. Microsoft Kinect sensorā īsteno ilgi gaidītu iespēju kontrolēt spēles procesu ar savu ķermeni, neizmanto nekādus papildu manipulatorus. Šādus paņēmienus sauc par paplašinātu realitāti. Virziens, kas nav iedomājams bez datora redzes realizācijas.

Tīmeklī plaši tiek izmantota CAPTCHA, lai pārliecinātos, ka darbības veic cilvēks, bet dažreiz fotogrāfiju grūti atšifrēt pat cilvēkam. Tas netieši parāda OCR algoritmu attīstības līmeni. Dažos uzdevumos tie jau pietuvinās cilvēka tēlu atpazīšanas prasmēm.

Galvenā iepriekš minēto sistēmu īpatnība ir tā, ka šīs sistēmas atpazīst simbolus no attēla – izmanto vizuālo informāciju. Tās interpretē cilvēkam ierastus simbolus, pārveido redzamu informāciju mašīnas kodētos simbolos. Lielākoties šādas sistēmas aprakstāmas ar terminu OCR plašā nozīmē. Bet visas līdz šim veidotās OCR sistēmas ir šauri specializētas. Universālu risinājumu trūkumu var paskaidrot ar risinājumu realizācijas sarežģītību un nepieciešamību to pielāgot ieejas datiem.

Šī darba mērķis ir izpētīt metodes un paņēmienus, kas palīdzētu automātiski atpazīt dalībnieka numuru sporta sacīkšu fotogrāfijās. Vairākos sporta veidos izmanto dalībnieku

numurus. Dalībnieki numuru piestiprina labi redzamā vietā. Tie paredzēti tam, lai skatītāji un tiesneši spētu ātri un viennozīmīgi identificēt konkrētu sportistu. Mūsdienās digitālās foto un video kameras kļuvušas par de facto standartu. Digitālās fotokameras ir plaši un viegli pieejamas. Fotografēšanas maksa gandrīz neeksistē (neskaitot amortizācijas izmaksas) un fotogrāfiju skaits ir ierobežots tikai ar pieejamo atmiņas apjomu. Notiekot masu sporta sacensībām, tiek veidoti vairāki simti fotogrāfiju ar sportistiem. Lielākā daļa no veidotajām fotogrāfijām tiek publicētas interneta vietnēs. Fotogrāfiju daudzuma dēļ rodas nepieciešamība fotogrāfijas kādā noteiktā veidā sakārtot un nodrošināt iespēju tās meklēt pēc dalībnieka numura. Lai atlasītu fotogrāfijas, kurās ir nofotografēts konkrēts dalībnieks, šādu informāciju vispirms jāiegūst. Automātiskā attēlā redzamo dalībnieku numuru anotēšana fotogrāfijai ir apskatīta šajā darbā.

Pirmajā nodaļā autors apraksta problēmas būtību un datu kvalitātes un īpatnības problēmas, apraksta iespējamos risinājumus un apraksta pētāmo datu kopu. Otrajā nodaļā apskata attēla matemātisko un attēla pētīšanas paņēmienus. Trešajā nodaļā aprakstītas grafikas pamata apstrādes tehnikas, tajā skaita trokšņu filtri (Gausa filtrs, Mediānu filtrs). Ceturtajā nodaļā tiek apskatīti kompleksāki apstrādes algoritmi un sarežģītākas tehnikas. Tiek aprakstītas vienkāršākās un sarežģītākās robežu atrašanas tehnikas, plašāk aprakstīts Kanni malu detektors. Piektajā nodaļā aprakstītas tehnikas attēlu segmentācijai. Aprakstīti paņēmieni numura plāksnes atrašanai attēlā ar Hafa transformācijas paņēmienu. Sestā nodaļa vērsta uz simbolu sadalīšanu, identifikāciju un atpazīšanu. Septītajā nodaļā aprakstīta darbā minēto metožu realizācija. Pamatota programmēšanas vides un konkrētu paņēmienu izvēle. Doti paskaidrojumi operatoru uzstādījumu izvēlei. Ir minēti pielietojumu scenāriji un iespējas. Darba beigās aprakstīti autora secinājumi un turpmākais darbu plāns. Pielikumā pievienoti šajā darbā aprakstītās redzes sistēmas realizācijas fragmenti programmēšanas valodā Java.

PROBLĒMAS APRAKSTS

1.1. Pieņēmumi un datu kopa

Darbā tiek apskatītas sporta sacensības profesionāļiem un amatieriem izvēlētā sporta veidu grupā. Ir paredzēts izstrādāt risinājumu, kas būtu pielietojams visiem tiem sporta veidiem, kur pie dalībnieka ķermeņa vai inventāra tiek piestiprināts numurs. Piemēram, skriešana, riteņbraukšana, triatlons, orientēšanās sports, skrituļslidu maratoni, distanču slēpošana un piedzīvojumu sacensības.

1.1. att. **Dalībnieka fotogrāfija distancē**

1.1. attēlā redzams sporta sacensību dalībnieks un šajā darbā tiek apskatīta sarkanā rāmī izmētātā numura atpazīšana.

.

Darba izstrādei izmantots fiksēts fotogrāfiju krājums. Krājumi un to apraksts pieejams 1.

pielikumā. Darba izstrādē tika izmantotas tikai tās fotogrāfijas, kuru autortiesību noteikumi to pieļauj. Darbā aplūkojamās fotogrāfijas aizsargātas ar GPL licenci.

1.2. Esošā situācija

Sporta sacensībās sportistus apskatāmos sporta veidos apzīmē ar individuālajiem numuriem. Sporta sacensības dalībnieki saņem numuru, kuru piestiprina redzamā vietā. Katrā apskatāmā sporta notikumā gan profesionāli preses fotogrāfi, gan skatītāji un dalībnieki dokumentē sacensību norisi. Veidojas liels klāsts ar fotogrāfijām, ko interneta lietotāji publicē koplietošanas fotogrāfiju galerijās un sociālajos tīklos. Tiek publicēti vairāki simti fotogrāfiju. Precīzāk fotogrāfiju skaitu var novērtēt 1. pielikuma 1. tabulā. Šādos fotogrāfiju daudzumos ir grūti orientēties un problemātiski sameklēt konkrētu dalībnieku. Ļoti aktuāla kļūst fotogrāfiju sistematizācija un meklēšana, izmantojot dalībnieka numuru. Praksē fotogrāfijas pēc sporta notikumiem netiek anotētas un sistematizācija notiek tikai notikumu līmenī – veidoti krājumi, kas attiecas uz noteiktu sporta notikumu.

Latvijā ar fotogrāfiju anotēšanu un piesaisti pie dalībnieka profila nodarbojas tikai viens sacensību organizators – Necom - Nordea Rīgas maratons. Autors sazinājās ar SIA Lattelekom BPO pārstāvjiem, lai saņemtu informāciju par to, kādā veidā notiek anotācija. Anotāciju Nordea Maratona fotogrāfijām veic desmit studenti, kuri 2 dienu laikā anotē fotogrāfijas no sacensībām.[2]

Galvenais iemesls tam, kāpēc automātiski risinājumi nav plaši izplatīti, ir tas, ka lielāko daļu fotogrāfiju veido pilnībā savstarpēji nesaistīti cilvēki. Katrs skatītājs veic savu atsevišķu foto krājumu. Cilvēki attēlu glabāšanai izmanto sev piemērotākās vietnes. Nav vienotas tīmekļa vietnes, kur varētu apvienot visus sporta fotogrāfiju krājumus. Visizplatītākie ir Google Picasa, Inbox Foto un Flicr. Bieži vien ir sastopami fotogrāfiju arhīvi vai privātās interneta vietnes, kuras fotogrāfiju publicēšanai izmanto tikai viens cilvēks. Organizatoru veidoto fotogrāfiju skaits ir pārāk mazs, salīdzinot ar kopējo fotogrāfiju skaitu. Un, ņemot vērā to, ka dažādas sporta sacensības organizē atsevišķi uzņēmumi, saprotams, ka izmantot automātiskās anotēšanas

2 Pilno e-pasta atbildi var apskatīt 3. Pielikumā.

sistēmas nav izdevīgi, jo neeksistē specializēts, viegli pielāgojams universāls risinājums.

Fotogrāfiju lielāko daļu veido nevis profesionāli fotogrāfi, bet līdzjutēji, izmantojot lētas digitālās fotokameras. Būtiska problēma, ar ko autors saskārās, meklējot problēmas risinājumu, ir datu kvalitāte. Procentuāli ir ļoti maz fotogrāfiju, kas ir pieejamas augstā izšķirtspējā. Bet fotogrāfijas kvalitāte būtiski ietekmē atpazīšanas kvalitāti. Jo mazāka ir fotogrāfijas izšķirtspēja, jo vairāk informācijas tiek pazaudēts un grūtāk atšķirt objektus un atpazīt tos.

.

1.3. Automātiskā anotācijas iespēja

Pastāv problēma, kad nepieciešams lielu fotogrāfiju klāstu anotēt ar dalībnieku numuriem. Ir paņēmieni, kas var īstenot pilno vai daļējo automātisko anotēšanu. Vairākumā no sacensībām laika kontrolei izmanto RFID tagu, kas ir piestiprināts pie dalībnieka numura vai citur pie dalībnieka, vai pie viņa transportlīdzekļa. Ir iespējams veidot "foto finišu" fotogrāfijās no finiša vai starpfiniša vietas. Šķērsojot RFID tagu nolasīšanas joslu, veidotā fotogrāfija automātiski tiek piesaistīta sportistam. Līdzīga tehnika tiek izmantota Rīgas Nordea Maratonā 2011. gadā, atrādot lielajā ekrānā sportistiem sveicienus, kurus pirms maratona un maratona laikā bija iespējams adresēt jebkuram maratona dalībniekam. Nosūtot sveicienu uz speciālo telefona numuru vai pievienojot to tīmekļa lietotnē, tas tika automātiski atrādīts uz lielā ekrāna, atbilstošam dalībniekam šķērsojot tagu nolasīšanas iekārtu. Bet
šādu paņēmienu izmanto ļoti reti, jo ar to iespējams anotēt tikai fotogrāfijas no fiksētām distances vietām (tikai finišu vai starpfinišu, jo nepieciešama tagu nolasīšanas iekārta).
Otra pieeja fotogrāfijas anotēšanai - izmantot vizuālo informācijas avotu, t.i. fotogrāfijas un video. Jau veidotajām fotogrāfijām tiek veikta analīze, meklējot numuru plāksnes un anotējot fotogrāfijas ar fotogrāfijās esošajiem dalībnieku numuriem. Tieši šādu anotēšanas tehniku autors aprakstīs šajā darbā.

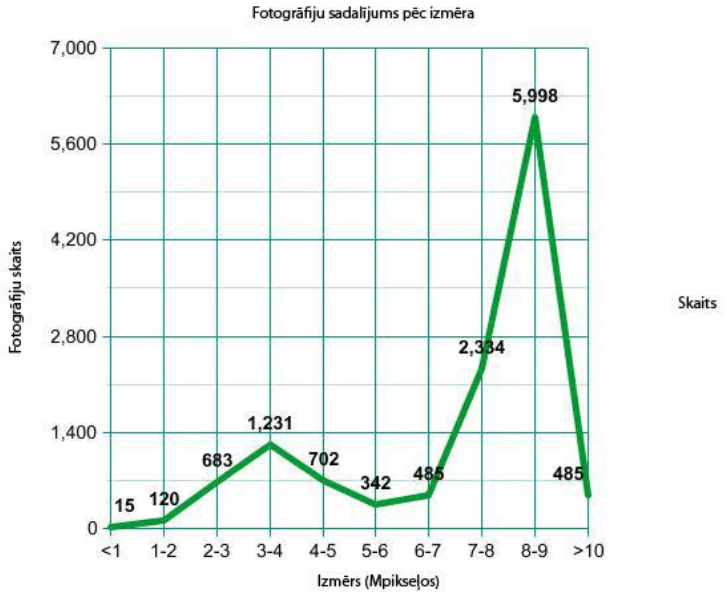
1.4. Datu kvalitātes un īpatnības problēmas

Jau tika minēts, ka fotogrāfiju krājumi atrodas vairākās interneta vietnēs un dažreiz ir

lejupielādēti sociālajos tīklos vai citos koplietošanas vortālos. Pirmā problēma, ar ko jāsaskaras, veidojot darbu, ir datu ieguve. Jābūt iespējai iegūt fotogrāfiju apstrādei un kādā noteiktā veidā sasaistīt šo fotogrāfiju ar dalībnieka numuru. Viens no variantiem ir lejupielādēt fotogrāfijas lokālā mapē datorā. Attēla analīze notiek lokāli, bet rezultāti tiek saglabāti tabulu veidā vai lokālā SQL datu bāzē. Blakus šim var attīstīt risinājumu, kas varētu tiešā veidā anotēt fotogrāfijas koplietošanas vortālos. Gandrīz visiem populārajiem vortāliem ir iespēja pievienot birkas fotogrāfijai vai labot fotogrāfijas redzamo nosaukumu. Potenciāli ir iespējams veidot spraudņus atsevišķiem vortāliem un piedāvāt automātiskās anotēšanas funkcionalitāti bez nepieciešamības pārvietot fotogrāfijas. Par līdzīga risinājuma attīstību un realizāciju pielietojumu sīkāk ir aprakstīts 7.4. nodaļā.

Vietnes, kur glabā fotogrāfijas, modificē fotogrāfijas pēc to ielādes un apskatei piedāvā jau samazinātu fotogrāfijas kopiju. Šis risinājums ir pieņemams tiem lietotājiem, kuri nevēlas fotogrāfijas drukāt, bet apskatei ir pietiekamas samazinātas fotogrāfijas. 1.4.1. attēlā ir redzams fotogrāfiju izmēru sadalījums. Var secināt, ka interneta vietnēs tiek glabātas fotogrāfijas mazā un vidējā izmērā. Fotogrāfijas vidējais izmērs ir 832*615.

.



1.4.1. att. **Fotogrāfiju sadalījums pa izmēriem**

Lielākajai daļai fotogrāfiju ir pielietota JPEG kompresija, kas apgrūtina objektu atrašanu

fotogrāfijās, jo daļa datu tiek zaudēta un grafiskie trokšņi apgrūtina robežu meklēšanu. Var secināt, ka apskatāmo risinājumu jāpielāgo darbam ar vidējā izmēra fotogrāfijām, kurām ir piemērota JPEG kompresija.

Apskatāmo sporta veidu un sacensību fotogrāfijās vienlaikus var atrasties vairāki sportisti. Numuri var pārklāties vai būt daļēji pārklāti. Numuru plāksne var atrasties plaknē, kas ir gandrīz perpendikulāra fotografēšanas virzienam, kas var izraisīt būtiskas numuru plāksnes nobīdes un deformācijas. Pati plāksne var būt fiziski deformēta vai nepareizi piestiprināta.

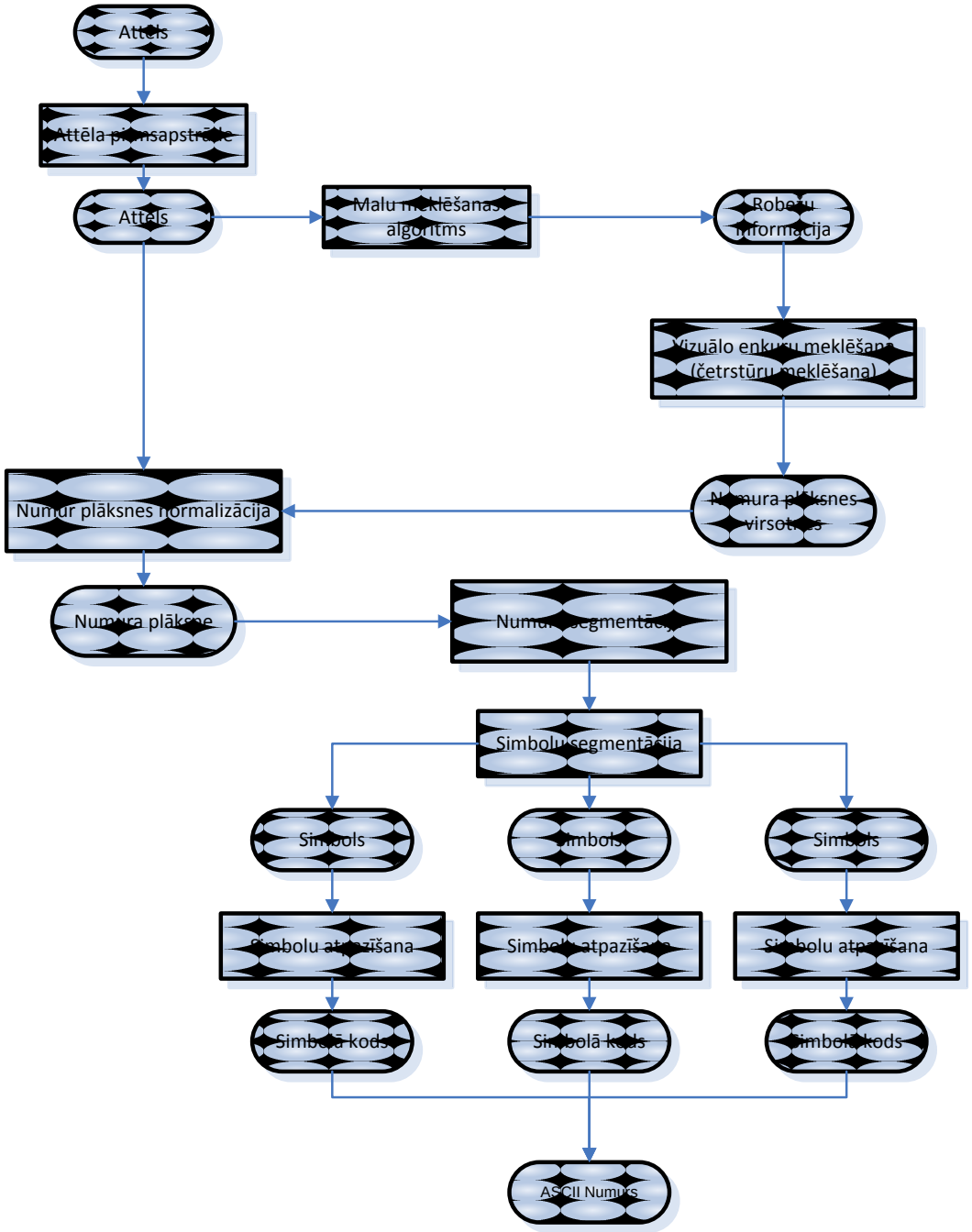
1.4.2. att. **Tipiskās numuru plāksnes deformācijas**

Tipiskās deformācijas ir apskatāmas 1.4.2. attēlā. Līdzīgas deformācijas var būtiski apgrūtināt uzdevumu, salīdzinot ar citiem numuru atpazīšanas vai OCR uzdevumiem. Apskatot pētāmo uzdevumu, var teikt, ka nav viegli atrādāmu vizuālu enkuru, kas var palīdzēt pētāmā apgabala definēšanā. Piemēram, automobiļu numuru atpazīšanā ir iespējams atrast automašīnas robežas un jau izvēlētajā pikseļu kopā meklēt numura zīmi. Apskatāmajās fotogrāfijās gandrīz nav iespējams atdalīt konkrētus sportistus, lai precīzāk definētu pētāmo apgabalu. Autora piedāvātā sākotnējā ideja meklēt cilvēka seju un pēc tam definēt pētāmos apgabalus arī nav realizējama, jo sportisti izmanto ķiveres (riteņbraukšanā, skrituļslidu maratonos) vai saulesbrilles (skriešanā). Ņemot vēra to, ka numura plāksnes vieta var atšķirties katram sporta veidam un katrām atsevišķām sacensībām, šāds paņēmiens nav pietiekami universāls

2. Metodes apraksts

Darbā tiek apskatīts metožu kopums, kura mērķis ir identificēt numuru plāksnes attēlā un segmentēt simbolus. Pēc tam ar OCR algoritmu atpazīt ciparus un saglabāt dalībnieka numuru.

Apstrādes paņēmieni veido kopīgu procesu, kura rezultātā ir iespēja saņemt dalībnieka numuru un numuru plāksnes koordinātes. Shematiski process attēlots 2.1. attēlā. Process sadalīts vairākos etapos, kuru secība ir stingri definēta. Katru etapu ir iespējams paveikt ar vairākiem paņēmieniem, dažas alternatīvas ir aprakstītas darbā.



2.1. att. **Atpazīšanas procesa shēma**

3. ATTĒLU ANALĪZS PAMATI

3.1. Attēlu matemātiskā reprezentācija

Attēlu var definēt kā pikseļu kopu, kur katrs pikselis ir minimāla nedalāma attēla daļa, kuras krāsa ir viendabīga. Pikseli definē divas koordinātes un intensitātes parametrs. To var pierakstīt kā daļēji definētu funkciju

Funkcija attēlo attēla pikseļu intensitāti naturālu skaitļu apgabalā. Intensitāti mēdz izteikt skaitliski intervālā [0,255], kas atbilst viena baita kodējamai vērtībai. Funkcija definēta apgabalos [0,W] pēc pirmā argumenta un [0,H], kur W un H ir attēlu attiecīgās dimensijas: W - platums un H – augstums.

Krāsainā attēla gadījumā tiek attēlota ne tikai intensitāte, bet vairākas intensitātes katram krāsas komponentam. Krāsu komponentu skaits ir atkarīgs no izvēlētā krāsu modeļa. Visplašāk datorgrafikā tiek izmantots RGB krāsu modelis. Tas sastāv no trim krāsu komponentiem: R - sarkana, G - zaļa un B - zila. Krāsu attēlu apraksta funkcija c. Funkcija attēlo nevis vienu intensitāti, bet kopumu no trim intensitātēm:

Attēla pikseli reprezentēs trijnieks p = (x, y, l); kur l – krāsas intensitātes trijnieks un l = (r, g, b); kur

Operācijas ar attēlu - tādas kā robežu meklēšana vai gradients var uztvert kā matemātiskās funkcijas c transformācijas. Pielietojot matemātiskās darbības, reizinot, atvasinot un integrējot, var iegūt attēla apkopotu vai modificētu informāciju, kas var būt lietderīga datora redzes realizācijā.

Definējot attēlu kā funkciju, jābūt iespējai salīdzināt divas funkciju vērtības. Krāsu attēlu gadījumā nav iespējams vienkārši salīdzināt vērtības, jo intensitātes vērtības reprezentē skaitļu trijnieks, nevis viens skaitlis. Krāsu attēliem izmanto matemātisko attāluma funkciju, ņemot vērā visus trīs krāsu komponentus. Autors piedāvā izmantot Eiklida attāluma funkciju:

Apstrādājot RGB kodēto attēlu, jāapstrādā trīs krāsu komponenti, jo krāsas ir līdzvērtīgas un katrs krāsu kanāls var saturēt līdzvērtīgi daudz informācijas, bet datorgrafikā ne vienmēr izmanto visus trīs krāsu komponentus. Lai optimizētu algoritmus, bieži vien izmanto melnbaltus attēlus. Melnbaltu attēlu veido viena matrica ar intensitātēm. Melnbalto attēlu iegūšanai iespējams izmantot citus krāsu modeļus, kas dažreiz ir labāk piemēroti datorapstrādei. Balstoties uz cilvēka redzes īpašībām, tika izstrādāts YUV krāsu modelis. Modelis ir izstrādāts, lai pārraidītu fotogrāfijas ar pēc iespējas mazākiem trūkumiem, upurējot krāsu komponentus, bet nezaudējot intensitātes informāciju. Datora redzē lietderīgi izmantot YUV modeļa intensitātes komponentu, jo tas satur pietiekami daudz informācijas, lai apstrādātu attēlu. Intensitātes komponents ir melnbalts attēls, kas veidots no krāsu attēliem, katrai krāsai lietojot savu konvertācijas indeksu. Lai konvertētu no RGB krāsu modeļa uz YUV krāsu modeli, izmanto formulu:

kur R, G, B ir attiecīgie krāsu komponenti – sarkana, zaļa un zila. Matemātiski YUV pikseli interpretē nevis kā struktūru (x,y, (r,g,b)), bet gan kā struktūru (x,y,l), kur l - YUV attēla intensitāte mērojama arī viena baita robežās

Datorprogrammās to mēdz pierakstīt kā:

Ja skatīt attēlu kā matricu, kur katrs matricas elements reprezentē pikseļa intensitāti, tad var definēt konvertācijas matricu:

;

Bieži vien izmanto arī vienkāršotu variantu:

Izmantojot šādu konvertāciju, zūd daļa no datiem, bet bieži vien ar to pietiek, jo melnbaltās interpretācijas veids īpaši stipri neietekmē algoritmu rezultātus [15].

Plaši tēlu atpazīšanas algoritmos izmanto bināros attēlus. Šādu attēlu paveidu izmanto masku glabāšanai. Katram attēla pikselim ir tikai divas iespējamās vērtības - 1 vai 0. Var viegli aprēķināt, ka uz vienu pikseli binārajam attēlam vajadzīgs tikai viens informācijas bits. Attēlu iegūšanā izmanto slieksni (*angļu val.*: threshold) - etalona intensitāti. Var aprakstīt konvertācijas funkciju melnbaltajam attēlam:

; kur

Vai krāsainajiem RGB attēliem:

Ja kopējā pikseļa intensitāte pārsniedz sliekšņa intensitāti (skaitot pēc Eiklida formulas), tad minētais pikselis tiek izmests. Sliekšņa pikseli izvēlas, bastoties uz kopējā attēla intensitātes sadalījumu vai intensitātes sadalījumu apskatāmajā apgabalā, jo binārā attēla pārveidošanas laikā nepareizas sliekšņa izvēles dēļ var zaudēt attēla informācijas lielāko daļu, kas ir nepieciešama tālākai apstrādei.

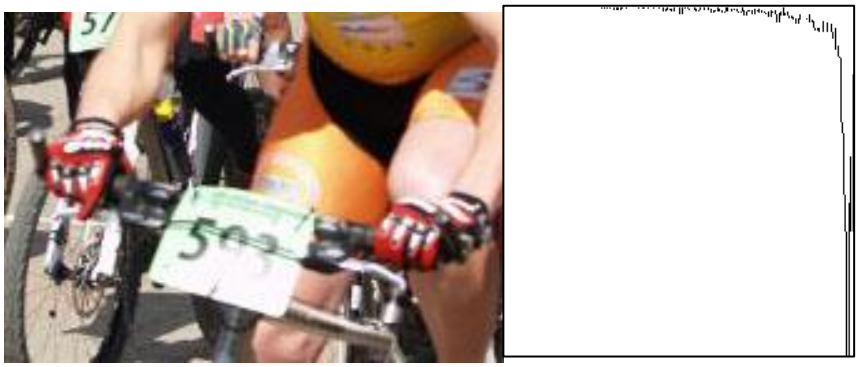
3.2. Attēla histogramma

Histogramma ir attēla analīzes paņēmiens, kas balstās uz statistikas pamatiem. Histogrammas vispārējā nozīmē attēlo notikuma iestāšanās biežumu apskatāmajā kopā. Attēlu gadījumā histogramma attēlo pikseļu kopējās intensitātes sadalījumu attēlā un to intensitātes sastapšanas biežumu. Ir iespējams vērot arī atsevišķu krāsu histogrammas, veidojot analīzi pēc izvelētās krāsas matricas (RGB vai cits krāsu modelis).

Histogramma palīdz novērtēt pikseļu intensitātes daudzveidību un sadalījumu attēlā, kā arī kopējo intensitātes līmeni. Histogramma var kalpot par nozīmīgu informācijas avotu sliekšņa izvēlē [11], jo histogrammas profils nemainīsies atkarībā no attēla rotācijas vai spilgtuma.(3.2.1 . att., 3.2.2. att.). Pāreksponētā fotogrāfijā būs palielināts līmeņu daudzums augstas intensitātes zonā un gandrīz nebūs vai būs minimāls līmeņu skaits zemas intensitātes zonā.

Vispārīgā gadījumā histogramma neapraksta objekta būtību (piemēram, objekta formu vai īpašību), bet vairākumā gadījumu palīdz definēt attēla korekcijas stiprumu un veidu, ja ir šāda nepieciešamība.

3.2.1. att. **Tumšas numura plāksnes attēls (a) un tā histogramma (b)**



3.2.2. att. **Gaišas numura plāksnes attēls (a) un tā histogramma (b)**

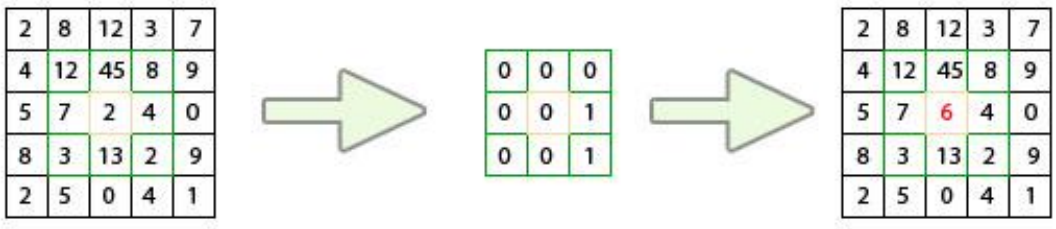
4. ATTĒLU PIRMSAPSTRĀDE

4.1. Trokšņu filtri

Viena no attēlu apstrādes problēmām ir troksņi. Trokšņu iemesli var būt vairāki: fotografēšanas tehnika un paņēmieni; foto vai video kameras iekšējie trokšņi; kompresijas trokšņi; pārraides trokšņi vai citi. Ja cilvēkam, nepalielinot fotogrāfiju sīki attēla trūkumi nav saredzami, tad attēla apstrādes laikā tie var radīt kļūdas apstrādē. Lai izvairītos no trokšņiem un samazinātu to iedarbību, izmanto trokšņu filtrus.

Grafikas apstrādē bieži vien izmanto matricu konvolūciju (*angļu val.:* matrix convolution), kur matricu pārveido ar citas matricas palīdzību. Maskas matricu mēdz saukt par kodolu (*angļu val.:* kernel). Mijiedarbība melnbaltu attēlu gadījumā notiek ar intensitātes matricu un ar katru no krāsu kanāliem RGB attēlu gadījumā (izņēmums, ja darbība veikta tikai konkrētam krāsu kanālam). Kodola vietā izmanto matricas izmēru 3x3 vai lielākus (5x5, 7x7 utt., bet 3x3 izmērs ir izplatīts, jo tas ir pietiekams, lai attēlotu trīs RGB pikseļus un 3x3 matricu reizināšanas operācijas tiek realizētas videokartes speciālo instrukciju līmenī). Konvolūcijas laikā kodola elementu vērtības apzīmē apkārtnē esošos matricas elementu (pikseļu) mijiedarbības spēku ar matricas apstrādājamo elementu. Apstrāde notiek, kārtējo apkārtnē esošo elementu reizinot ar matricas koeficientu un pēc tam summējot ar visiem kodola elementiem.

.



4.1.1. *att.* **Matricu pielietojuma piemērs**

4.1.1. attēlā apstrādātā elementa vērtība mainīta uz jaunu vērtību, jo tikai augšā esošajiem

pikseļiem ir ielikts reizināšanas koeficients, bet pārējie apkārtnes elementi tiek ignorēti.

Var veidoties situācija, kad pikseļa krāsas spilgtums iziet ārpus iespējamās robežas r >255, kas nav pieļaujams atbilstoši attēla matemātiskajai definīcijai. Lai normalizētu izejas rezultātu, filtriem ir paredzēti divi papildu mainīgie div (*angļu val.:* devider) – dalītais un offset – nobīde. Pati filtra pielietošana notiek šādi:

kur I – oriģinālā fotogrāfija, m un n - pielietojamās matricas dimensijas, A - pielietojamā matrica, div – dalītais, offest – nobīde. Izpildot matricas konvolūcijas operācijas, jāapstrādā situācijas, kad < 0. Filtrs mēģinās vērsties pie koordinātēm ārpus fotogrāfijas definētā apgabala. Atšķirīgās realizācijās to realizē dažādi, visbiežāk neapstrādā malas robežās vai paplašina fotogrāfiju, aizpildot jaunos apgabalus ar klāt esošajiem pikseļiem.

Izmantojot konvolūciju matricas, attēliem var pielietot vairākus filtrus, kas var būt lietderīgi, veicot attēlu segmentāciju un robežu meklēšanu.

4.2. Apgabala vidēja metode

Apgabala vidējā metode – vienkāršākais veids, kā izvairīties no trokšņiem. Metodes pamatā ir vidējais aritmētiskais. Katram pikselim tiek aprēķināta pikseļu intensitātes vidējā aritmētiskā vērtība, bieži vien šo filtru sauc par izpludināšanu (*angļu val.:* blur).

;

Kur def = 9, offset = 0

Izmantojot apgabala vidējo metodi, ne vienmēr var iegūt labākus trokšņu filtrēšanas rezultātus. Izlīdzināšana ir pārāk atkarīga no apkārtējiem trokšņiem un slikti apstrādā asas attēla robežas, kas slikti ietekmē atpazīšanas kvalitāti.

4.2.1. **Gausa filtrs**

Attēlu apstrādē izmanto Gausa filtru. Tas strādā līdzīgi apgabala vidējā metodei, bet mazāk ietekmīgs no trokšņiem, jo ir iespēja izmantot lielākas konvolūciju matricas, nezaudējot asas attēlu robežas, jo katra nākamā pikseļa mijiedarbības stiprums ir apgriezti proporcionāls pikseļa attālumam no pielietojuma vietas (kodola centra). Koeficienti konvolūcijas matricā atbilst Gausa normālā sadalījuma funkcijai. Gausa funkcija vienai dimensijai definēta šādi:

Attēlu apstrādei izmantosim Gausa funkciju, kas definēta divām dimensijām:

Gausa filtra konvolūcijas matrica 3x3 izskatās šādi:

;

Kur def = 16, offset = 0

Filtru iedarbību var ietekmēt, ne tikai izmantojot koeficientus. Lai sasniegtu labākus rezultātus, filtrus darbina vairākas reizes (Gausa filtru datora redzē izmanto simtiem reižu, lai sasniegtu vēlamo rezultātu). Izmanto arī lielākus kodola izmērus, kas palīdz izvairīties no trokšņa ietekmes, bet prasa vairāk skaitīšanas operāciju. Gausa filtra konvolūcijas kodols izmērā 5x5 aprēķināts ar =1 izskatās šādi:

Kur def = 273, offset = 0

4.2.1.1. att. **Gausa filtra pielietojuma piemērs**

Pielietojot matricas filtrus, ir iespējams apvienot trokšņu filtrus kopā ar citām konvolūciju darbībām, iegūstot jaunus operatorus. Apvienojot vairākas darbības vienā, var nozīmīgi palielināt algoritma ātrdarbību, atsakoties no liekām darbībām, jo bieži vien ir iespējams izmantot vienu darbību divu operatoru vietā.

4.2.2. **Mediānu metode**

Vēl viena populāra trokšņu filtrēšanas metode ir mediānu metode. Mediānu metode, kā arī citi trokšņu filtri, izmanto apskatāmā punkta apkārtni, lai aprēķinātu jauno punkta (pikseļa) vērtību. Klasiskā mediānu filtra apstrāde notiek NxN punkta apkārtnē, kur N ir nepāra skaitlis un c atrodas NxN apgabala centrā. No apgabala NxN tiek atrasts vidējās intensitātes punkts. Tātad, katrs punkts c ir aizvietots ar apgabala NxN vidējo vērtību. Apskatīsim punktu ar koordinātēm (i,j) un apkārtni NxN. Tiek veidota sakārtota rinda P izmērā N^2, kurā ietilpst visi apgabala NxN pikseļi. Pikseļi rindā P tiek kārtoti pēc intensitātes un ir acīmredzams, ka rindas vidū atradīsies pikselis ar apgabala vidējo intensitāti.

Mediānu filtru var pierakstīt kā funkciju:

kur k - algoritma parametrs, N – nepāra skaitlis atbildīgs par apkārtnes izmēru, I - izejas attēls, p – sakārtota apgabala NxN rinda.

4.2.2.1. att. **Mediānu filtra pielietojums a- oriģinālais attēls ar trokšņiem, b – nofiltrēts attēls.**

Mediānu filtrs labi filtrē trokšņus, bet tam ir arī dažas problēmas. Filtrs slikti apstrādā augsta kontrasta robežas. Mediāna filtrs var izpludināt robežas, šādā veidā zaudējot daļu no informācijas. Salīdzinot ar lineārajiem filtriem, algoritma realizācija paredz sakārtotas rindas veidošanu, kas paredz kārtošanu. Labākā kārtošana ir iespējamā $\Theta = n*\log(n)$, kas ir viennozīmīgi sliktāk nekā lineārie filtri [12].

5. ROBEŽU MEKLĒŠANA

Darbā apskatītā problēma paredz, ka analīze notiek jau statiskam, diskrētam attēlam, kurā nav iespējams viennozīmīgi atdalīt objektu no fona. Tāpēc ļoti nozīmīgs ir objekta robežu meklēšanas posms. Robežu meklēšana ir nepieciešama, lai pēc tam būtu iespējams identificēt objekta kontūras un atdalīt to pikseļu kopu, kura atbilst apskatāmajam objektam.

Ar robežām attēlā saprot krāsas vai apgaismojuma intensitātes izmaiņas, kā arī cilvēki mēdz ar robežām uztvert iedomājamas, pagarinātas robežas, balstoties uz savu pieredzi. Šādas robežas datoram ir grūti atrast, jo, neko nezinot par objektu kopumā, nav iespējams veikt secinājumu, ka vietai, kur ne krāsa, ne intensitāte nemainās, jābūt robežai un tikai īpašu apstākļu dēļ robeža nav redzama.

5.1. Gradientu metode

Gradients ir vektoriāla vērtība, kas norāda uz visātrāk funkcijas vērtību mainošo virzienu. Skaitliski vienāda ar izmaiņu lielumu. Gradientam datora redzē ir ļoti nopietna loma, jo augstā gradienta vērtība apzīmē tos attēla apgabalus, kur intensitātes maiņa notiek strauji. Straujas intensitātes izmaņas var norādīt uz robežas eksistenci šajā pikseļu kopā. Gradients ir viens no primitīvākajiem malu detektoriem, kas tomēr pietiekami labi strādā pamata situācijās. Funkciju gadījumā gradientu var izteikt kā:

$$\frac{\partial I}{\partial x} \quad \frac{\partial I}{\partial y}$$

kur I - izejas attēls. Taču praksē izmanto analogus, jo attēls nav nepārtraukta funkcija un nav iespējams rēķināt atvasinājumu:

Gradientus var pielietot arī kā operatoru kodolu matricas konvolūcijai. Gradientu rēķina vairākos virzienos. Pamatā tiek apskatīti ortogonālie virzieni un gradientu virzienu absolūtās vērtības tiek summētas, iegūstot kopējo attēla gradientu.

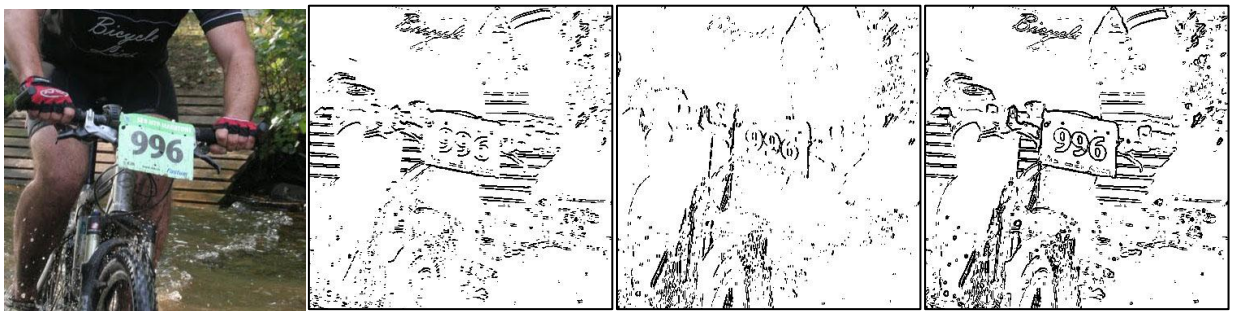
Virzienā 0° un 90°:

; ;

Gradientu var skaitīt arī citos virzienos tā saucamās Roberta Krosa konvolūcijas maskas. Gradienta aprēķins notiek ar citiem leņķiem, bet tomēr tie arī ir ortogonālie tāpat kā klasiskais attēla gradients [17].

Virzienā 45° un 135°:

; ;



5.1.1. att. **Gradientu aprēķināšana attēlam a - oriģinālais attēls, b – gradients virzienā 0, c – gradients virzienā 90, d - gradientu kompozīcija**

5.2. Sobeļa operators

Pastāv arī vairāki operatori malu meklēšanai attēlos. Bieži vien izmanto Sobeļa operatoru. Tas ir līdzīgs gradientu metodei un arī sastāv no diviem komponentiem – virziena x un virziena y. Operatora pielietošana notiek līdzīgi kā citiem konvolūcijas operatoriem.

Sobeļa operatoram tiek izmantotas šādas matricas:

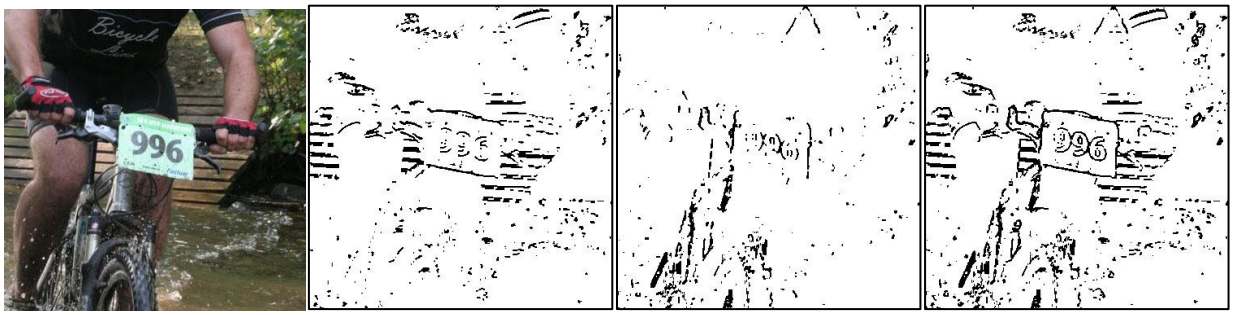
Lai meklētu vertikālās robežas, kuras ir tumšākas no labās puses.

;

Lai meklētu horizontālās robežas, kuras ir gaišākas no augšas.

Kopējo Sobeļa operatoru aprēķina kā: ; []

Sobeļa operatoram ir iespējams definēt konkrētas īpašības, ko tam jāmeklē attēlā. Ir iespējams meklēt intensitātes izmaiņas noteiktā virzienā un intensitātes izmaiņas robežas leņķī [14]. Leņķi maskām aprēķina kā:



5.2.1. att. **Sobeļa operatora aprēķināšana attēlam a - oriģinālais attēls, b – operators virzienā x, c – operators virzienā y, d - operatoru kompozīcija**

5.3. Kanni malu detektora algoritms

Kanni detektors (*angļu val.:* Canny Edge Detector) ir operatoru kopums optimizēts malu meklēšanai attēlos. Detektorā ir apkopoti paņēmieni, kā izvairīties no kļūdainām robežām un no dublējošām robežām. Darbā ir apskatīts vienkāršotais detektora variants, kas ir sadalīts vairākos posmos. Oriģināla rakstā piedāvāts operatorus pēc iespējas apvienot kopā, iegūstot sarežģītākus operatorus. Ir piedāvāts apvienot trokšņu filtrēšanu ar gradientu meklēšanu, iegūstot vienu operatoru, kuru bieži vien sauc par Kanni operatoru [2]. Izmantojot divu līmeņu vērtības aproksimāciju, var panākt trokšņu samazināšanu līdz 40%, tajā pašā laikā samazinot darbību skaitu.

Apskatāmā detektora realizācija tiek sadalīta piecos atsevišķos:

1. Izpludināšana – izlīdzina attēlu, lai samazinātu trokšņus.
2. Gradientu meklēšana – malas ir iezīmētas tur, kur gradienta absolūtās vērtības ir vislielākās homogēnu gradienta virzienu apgabalos.
3. Ne maksimumu nomākšana (*angļu val.:* Non-Maximum Suppression).
4. Dubultsliekšņu filtrācija – potenciālās robežas tiek sadalītas 3 prioritāšu grupās, izmantojot sliekšņus.

5. Neskaidrību izsekošana – izsekota vāju robežu piederība. Robežas, kas pieder stiprām robežām, tiek apvienotas, pārējās tiek ignorētas.

Katrs etaps ir operāciju kopa, kas balstās uz iepriekšējo etapu rezultātiem. Etapu secība ir stingri definēta un veido kopēju detektoru. Dažos etapos tiek izmantotas tehnikas, kas jau aprakstītas darba atsevišķās nodaļās, tāpēc etapu apraksti saturēs tikai norādi uz operatora aprakstu un izmantotās īpatnības to realizācijā.

Pirms apstrādāt fotogrāfiju ar Kanni algoritmu ir ieteicams konvertēt RGB krāsu fotogrāfiju citā krāsu modelī, iegūstot melnbalto fotogrāfiju. Tas, galvenokārt, ir nepieciešams, lai optimizētu algoritma darbību, jo, apstrādājot RGB fotogrāfijas, būs nepieciešams veikt katru darbību trīs reizes – atsevišķu reizi katrai krāsu matricai [2]. Autora piedāvātais variants ir izmantot YUV krāsas modeļa intensitātes komponentu, kas satur informāciju par krāsu intensitāti, balstoties uz cilvēka redzes īpatnībām. RGB uz YUV konvertācija sīkāk aprakstīta 3.1. nodaļā.

5.3.1. **Izpludināšana**

Pielietojot Kanni algoritmu un dažus citus robežu meklēšanas algoritmus, pirms apstrādes jāizpludina attēls. Neapstrādāts attēls satur vairākus trokšņus, kas var būt apstrādāti ar robežu detektoru kā potenciālās objektu malas. Grafiskie trokšņi var radīt traucējumus robežu atpazīšanā un palielināt attēla apstrādes laiku, jo būs nepieciešams apstrādāt ne tikai būtiskas intensitātes un krāsu izmaiņas raksturīgas objektiem, bet arī grafiskos trokšņus [5].

Kā trokšņu filtru var izmantot Gausa filtru vai citu filtru no 4.1. nodaļas Trokšņu filtri. Gausa filtrs izmantots tāpēc, ka tas dod labākus rezultātus, salīdzinot ar citiem trokšņu filtriem, tomēr paliekot ar lineāru sarežģītību.

Gausa filtrā pikseļu ietekme ir apgriezti proporcionāla attālumam starp tiem un arī izlīdzināšanas pakāpe ir atkarīga no sigmas. Izmantojot Gausa filtru, konvolūcijas masku jāpielāgo konkrētam uzdevumam, pēc iespējas sasniedzot optimālākas trokšņu filtrēšanas īpašības. Pielāgošanu veic, mainot sigmas vērtību un filtra kodola izmēru. Plašāk Gausa filtrs ir aprakstīts 4.2.1. nodaļā Gausa filtrs.

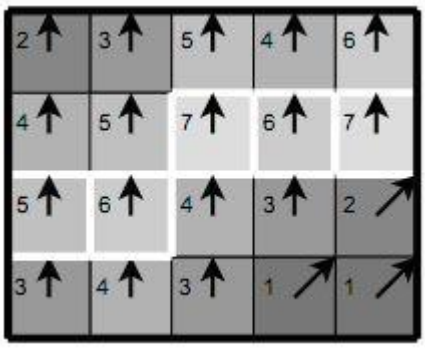
5.3.2. **Gradientu meklēšana**

Pēc trokšņu filtrēšanas iegūtais attēls ir izpludināts, bet tomēr satur pietiekami daudz informācijas, lai būtu iespējams atdalīt attēlā redzamos objektus. Otrais etaps ir robežu meklēšana. Kanni detektoru izmanto gradientu robežu identifikācijai, jo nākamos etapos būs nepieciešama informācija ne tikai par intensitātes izmaiņām, bet arī par intensitātes izmaiņu stiprumu un virzienu. Plašāk attēla gradients aprakstīts 5.1. nodaļā.

5.3.3. **Ne maksimumu nomākšana**

Viena no funkcionālajām īpašībām, kas atšķir Kanni detektoru no citiem robežu meklēšanas paņēmieniem, ir iespēja atdalīt tikai vienu robežu atkārtoto robežu kopā. Rēķinot gradientu, attēlā pastāv problēma, kas saistīta ar ātras intensitātes izmaiņām attēlā. Ļoti reti gadās, ka objektu atdalošo malu veido robeža viena pikseļa platumā. Intensitātes un krāsas pārejas attēlā tomēr notiek pakāpeniski kādā noteiktā apgabalā. Problēma ir ļoti aktuāla liela izmēra fotogrāfijām, kad jebkura robeža būs attēlota ar kādu pikseļu apgabalu platumā vairāk par vienu pikseli, kur intensitāte mainīsies ļoti ātri. Meklējot robežas ar attēla gradienta palīdzību ar augstām intensitātes izmaiņām būs iezīmēts pilnībā viss robežas apgabals, pa kuru izpludināta robeža. Tādas sekas rada nopietnas problēmas attēlu apstrādē. Kanni detektorā ir mēģināts izvairīties no līdzīgas situācijas.

Lai atdalītu robežas no tās dublikātiem, tiek izmantota gradienta virziena un intensitātes informācija. Gradienta vektors tiek normalizēts virzienos ar 45° soli, kopā izmantojot astoņus virzienus. Tiek meklēti apgabali ar homogēnu gradienta virzienu. Par robežu pikseļiem tiek izvēlēti tie pikseļi, kuros tiek iegūta gradienta maksimālā vērtība vienā gradienta virzienā. Gradientu maksimumi tiek meklēti starp pikseļiem, kuriem sakrīt gradienta vektors gradienta virzienā vai gradientam pretējā virzienā.



5.3.3.1. att. **Gradientu virziens un vērtība**

Gradientu aprēķināšana apskatāma 5.3.3.1. attēlā. Gandrīz visiem pikseļiem gradienta virziens ir uz augšu 90° virzienā, tāpēc šo pikseļu vērtības tiks salīdzinātas ar augšā un lejā stāvošajiem pikseļiem (visiem kolonnā, kuriem gradienta virziens sakrīt). Tikai tie pikseļi, kas ir marķēti ar balto rāmi, paliek attēlā, pārējie tiek nomākti.

5.3.4. **Dubultsliekšņu filtrācija**

Pēc robežu meklēšanas etapa fotogrāfijā pastāv informācija par intensitātes izmaiņu centriem un saglabāta informācija par intensitātes izmaiņu stiprumu. Etapa ietvaros potenciālās robežas tiek sadalītas trīs pamata grupās: stiprās „patiesās" robežas, potenciālās robežas un trokšņi. Stiprās robežas veido finmāla detektora robežu informāciju, potenciālās robežas tiek sadalītas vēl divās grupās nākošajā etapā un trokšņi tiek ignorēti. Robežu sadalījumam pielieto slieksni, lai noteiktu, vai konkrētais punkts pieder konkrētai robežu grupai vai nē. Jo mazāks ir slieksnis, jo vairāk robežu atradīs metode un stiprāk to ietekmēs grafiskais troksnis. Pretējā virzienā pārāk liels slieksnis var ignorēt apstrādē nepieciešamās robežas.
Kanni detektors izmanto divus sliekšņus: augšējo un apakšējo. Ja pikseļa intensitāte lielāka nekā augšējais slieksnis – pikseļiem tiek piešķirta maksimāli iespējamā vērtība baita robežās. Šādi pikseļi veido finmāla robežu attēlu. Ja pikseļa intensitāte ir zemāka nekā apakšējais slieksnis, pikselis tiks ignorēts, jo tas tiek uzskatīts par troksni un intensitātes izmaiņas šajā punktā nav pietiekami stipras, lai uzskatītu to par objekta robežu. Ja pikseļa intensitāte atrodas sliekšņa robežās, t.i., intensitāte lielāka par apakšējo slieksni, bet mazāka par augšējo slieksni, pikselim tiek piešķirta vidējā intensitāte – tas iezīmē sadalījumu otrajā grupā. Matemātiķi šādu sadalījumu

var izteikt kā funkciju:

;

kur i - pikseļa intensitāte (apskatām melnbalto attēlu), T_h – augšējais slieksnis, T_l – apakšējais slieksnis. Sliekšņu vērtības ir izteiktas absolūtas intensitātes mērvienībās, t.i. naturālais skaitlis baitu robežās.

Pēc etapa izpildes attēls ir sava veida trinārs attēls, jo tas nav binārs, bet kopumā pikselim ir tikai trīs iespējamās intensitātes: balts – noteiktā robeža, pelēks – potenciālā robeža un melns – fons. Ar fonu ir iekrāsota attēla lielākā daļa un visi tie apgabali, kuros intensitāte nemainās pietiekami ātri, lai tos uzskatītu par objektu robežām.

5.3.5. **Neskaidrību izsekošana**

No dudultsliekšņu filtrācijas etapa robežas tiek sadalītas divās grupās: stiprās robežas un potenciālās robežas. Kanni detektors neskaidrību izsekošanas etapā sadala potenciālās robežas divās grupās: robežas – tiek pieskaitītas klāt stiprām robežām un trokšņi – tie tiks ignorēti. Lai novērtētu grupas sadalījumu, tiek apskatīts pikseļu izvietojums attēlā. Ja potenciālās robežās pikselis atrodas blakus stiprai robežai, tad tas tiek pieskaitīts pie stiprām robežām, jo visticamāk pikselis veido stiprās robežas pagarinājumu. Ja potenciālā pikseļa apkārtnē neatrodas neviens stiprās robežas pikselis, tad tādu potenciālo pikseli vai potenciālo robežu pikseļu formāciju var uzskatīt par troksni un ignorēt.

Veicot neskaidrību izsekošanu, tiek pētīta pikseļa apkārtne attālumā -1 līdz +1 abu koordināšu ass virzienos. Tātad, katram pikselim tiek apskatīti astoņu kaimiņu pikseļi vai astoņi virzieni. Pētāmo apgabalu var aprakstīt ar virzienu matricas palīdzību [16]:

;

Virzienu matricu, var izmantot ka konvolūcijas masku, rēķinot gradienta virzienu.

Izejā tiek saņemts binārs attēls, kurā ar baltiem pikseļiem ir iezīmēti objektu robežu

centri. Šādu attēlu var izmantot tālākai apstrādei.

5.3.5.1. att. **Numuru plāksnes robežas meklēšana, a - oriģinālais attēls, b - atdalīto sliekšņu informācija**

6. SEGMENTACIJA

Viena no svarīgākajām darbībām datora redzē ir attēla segmentācija. Segmentācija ir attēla sadalījums atsevišķās pikseļu kopās, kur katra kopa atbilst konkrētā attēlā attēlotajam objektam. Darbā apskatīta segmentācija atbilstoši objektu kontūrām, robežām, kas ir atrodamas attēlā. Robežu meklēšana ir aprakstīta 5. nodaļā un, segmentējot attēlu, pieņemsim, ka ir pieejams oriģinālais attēls un attēla robežu binārs attēls.

Pētāmās problēmas ietvaros attēlā ir nepieciešams identificēt dalībnieka numurus. Dalībnieka numurs atrodas četrstūra numura plāksnēs, kas piespirinātas pie dalībnieka vai pie dalībnieka inventāra. Dalībnieka numura plāksni var raksturot kā paralelogrammu. Segmentācijas procesā nepieciešams sameklēt numura plāksnes apgabalu – sameklēto numura plāksni identificēsim ar četrām virsotnēm. Katra virsotne atbilst numura plāksnes paralelogramma virsotnei.

Darbā vispirms ir aprakstīti paņēmieni, kā nelielā attēlā segmentēt numura plāksni un iegūt numura plāksnes saturošo pikseļu kopu no oriģinālā attēla. Pēc tam tiek apskatīti paņēmieni, kā attēlā atrast vairākas numura plāksnes, un kandidātu attēlu atlase, filtrējot loģiskos trokšņus.

Uzdevumā meklētais objekts ir četrstūra plāksne, kas ir segmentējama ieejas attēlā. Pastāv vairāki paņēmieni, kā atrast noteiktas ģeometriskas figūras attēlā. Tie visi balstās uz ģeometrisko figūru īpašību meklēšanu. Viens no plaši pielietojamajiem variantiem ir masku filtrēšana. Ar izstieptas vienā dimensijā matricas palīdzību tiek meklētas robežas, kas var būt taisnes. Pēc taišņu izvietojuma tiek konstatēts, kur atrodas meklējamais objekts. Šādi operatori stādā līdzīgi Roberta Krosa operatoriem, Sobeļa operatoram vai jau minētajam Kanni malu detektoram.

Otrs izplatīts paņēmiens balstās uz citas četrstūra īpašības. Četrstūrim pastāv četri malu krustojumi. Ja attēla kvalitāte ir pietiekami laba, lai atpazītu taišņu krustojumus, tad ir iespējams ar masku palīdzību meklēt sakritības attēlā. Atrodot vismaz trīs malu krustojumus, var pilnībā rekonstruēt četrstūri attēlā, nezaudējot nekādu informāciju. Šādam paņēmienam ir arī problēmas. Tas jāpielāgo attēla izšķirtspējai – ja liela izmēra fotogrāfijās četrstūru stūri būs izpludināti un taisnes krustojuma vietās stūris būs apaļš. Bet apaļa stūra atpazīšana ir apgrūtināta ar stūra meklēšanas maskām, turklāt tie nav tām speciāli pielāgoti. Otra pamatīga problēma stūru meklēšanā ir attēla nobīde. Ja numura plāksnes malas neatrodas paralēli attēla malām, kas var būt

izraisīts ar kameras un dalībnieka savstarpējo novietojumu, četrstūru malas šķērsošana nenotiek perpendikulāri. Taču nobīdi ir iespējams labot, veicot dažas attēla transformācijas. Lai labotu nobīdi, ir nepieciešams atrast viennozīmīgas pazīmes, no kurām izriet, ka attēls ir nobīdīts un par cik lielu leņķi.

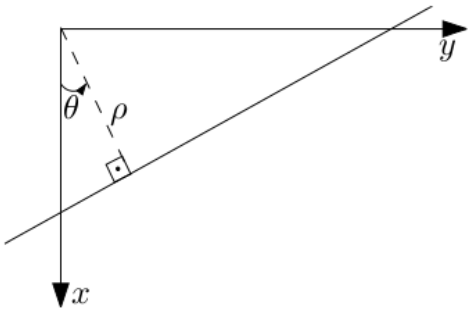
6.1. Hafa transformācija

Hafa transformācija ir metode, ko izmanto ģeometrisko objektu segmentācijai attēlā. Sākotnēji Hafa transformācija tika veidota, lai identificētu taisnes attēlā, bet vēlāk algoritms tika modificēts, pievienojot iespēju atpazīt patvaļīgas formas figūras. Hafa transformāciju izmanto parametriski aprakstāmo figūru meklēšanā, biežāk līniju un riņķa līniju. Apstrādājot attēlus, bieži vien rodas problēmas, ka pēc malu detektora darbības apstrādātajā attēlā trūkst daļas no figūru kontūras vai kontūru malas nav precīzas grafisko trokšņu vai attēla deformācijas dēļ. Hafa transformācijas priekšrocības ir tādas, ka algoritms ir tolerants pret šāda veida attēla trūkumiem un, ja problemātisko apgabalu skaits attiecībā pret korekti apstrādāto robežu skaitu ir neliels, tad tas nesagādā grūtības identificēt objektus attēlā.

Hafa transformāciju var apskatīt kā balsošanas algoritmu. Katrs attēla punkts pievieno balsi visām iespējamajām taisnēm, kas potenciāli var iet caur šo punktu. Balsošanas rezultāti tiek glabāti akumulatoru reģistros (*angļu val.:* accumulator registers), kas satur visas iespējamās taisnes reprezentācijas. Un pēc izpildes - jo lielāka vērtība konkrētajā balsošanas reģistrā, jo lielāka varbūtība, ka attēlā atrodas taisne, ko raksturo minētais punkts. Taisnes meklēšanas uzdevums tiek reducēts uz maksimālās vērtības meklēšanu plāksnē.

Hafa transformācijas princips tiek balstīts uz taišņu ģeometriskajām īpašībām. Taisni var izteikt ar vienādojumu . Un to var definēt katram punktam attēlā. Hafa transformācijas laikā tiek novērtētas taisnes parametru vērtības, tātad taisnei taisne var būt apzīmēta ar punktu (b,m) parametru telpā (saukta arī par Hafa telpu). Bet šāda veida apzīmējums neder visām taisnēm. Vertikālajām taisnēm, piemēram, x =2, b un m vērtības nav iespējams nodefinēt [18], tāpēc taisnes parametrus apzīmē ar citiem diviem parametriem: un , kur - vektora lielums no koordinātes sākumpunkta līdz taisnei un – leņķis starp vektoru un koordinašu asi. Aprakstot taisni šādā veidā, var definēt taisnes vienādojumu:

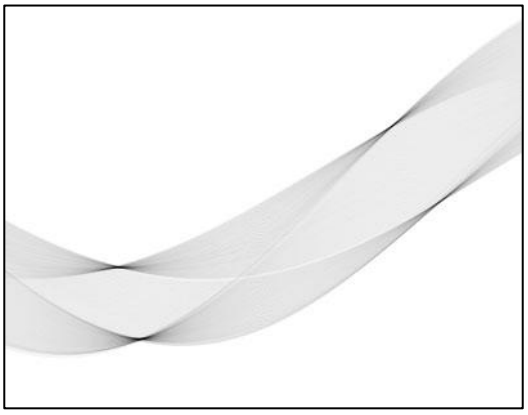
vai arī pārveidot vienādojumu:



6.1.1. *att.* **Hafa transformācijas taisnes parametri ģeometriskā nozīmē**

6.1.2.

Katram punktam apskatāmajā attēlā Hafa telpa atbilst unikālam sinusoīdam . Ja sinusoīdi pārklājas, tad punkts Hafa telpā, kur notiek pārklāšanās, atbilst taisnei oriģinālajā attēlā, kura iet cauri abiem punktiem oriģinālajā attēlā. Šādā gadījumā pikseļi, kas oriģinālajā attēlā veido taisni, tiek attēloti ar vairākiem sinusoīdiem, kas pārklājas vienā punktā Hafa telpā. Var apskatīt četrstūra četras līnijas Hafa telpā 6.1.2. attēlā, kur redzami tumšākie punkti.



6.1.2. att. **Hafa transformācijas piemērs**

Hafa transformācijas galvenā problēma ir ātrdarbība. Lai iegūtu attēla ģeometrisko aprakstu,

nepieciešams veikt vairākas operācijas. Hafa telpa tiek konstruēta visiem leņķiem tā, lai būtu iespējams glabāt informāciju par visām iespējamajām taisnēm. Ņemot vērā apskatāmo problēmu, var secināt, ka apskatāmos attēlos dalībnieka numura plāksnes malas nebūs izvietotas leņķī lielākā nekā 40°, rēķinot pret attēla malām. Tad var samazināt apskatāmo taišņu diapazonu un definēt ne robežās, bet — [8].

6.2. Četrstūru meklēšana

Uzdevumā aprakstītā problēma paredz četrstūru objektu meklēšanu attēlos. Četrstūru atrašanas problēma ir plaši izplatīta problēma, kas ir sastopama vairākās dzīves sfērās. Piemēram, to izmanto krion-elektronu mikroskopu izmantošanā, veicot atpazīšanu četrstūra vai apaļiem elementiem [1], ēku atpazīšanas kosmosa vai aerofotogrāfijas attēlos vai automobiļu numuru atpazīšanas programmās. Taču vairākums risinājumu ir pielāgots konkrētiem apstākļiem, piemēram, meklējot fiksēta izmēra četrstūrus vai meklējot četrstūrus fiksētā diapazonā ar zināmu malu attiecību. Darbā apskatītās problēmas definīcijā nav noteikti nekādi papildu nosacījumi par numura plāksnēm. Saprotams, ka dažādiem sporta veidiem un atsevišķiem organizatoriem numura plāksnes atšķirsies. Autors apskata piedāvāto [Jung, C.R., Schramm, R] nenoteiktā izmēra četrstūru atpazīšanas metodiku [1]. Metodika balstās uz četrstūra pazīmju meklēšanu Hafa akumulatoru telpā.

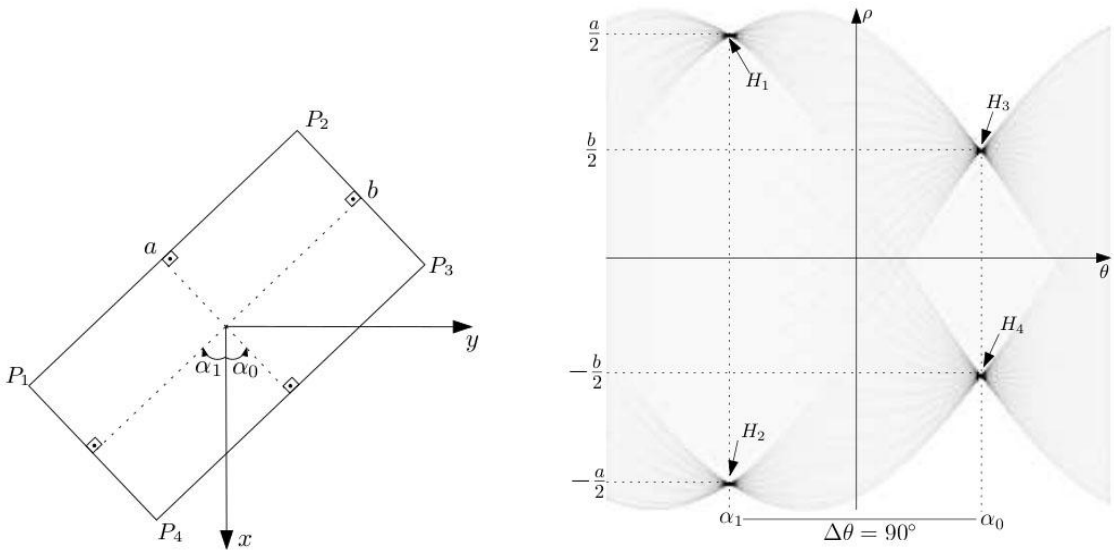
Pēc oriģināla formulējuma Hafa transformācijai parādījās vairākas variācijas, kas veidotas, lai samazinātu skaitļošanas operāciju skaitu, kā arī pielāgotu risinājumu konkrētiem datiem. Piemēram, eksistē Paraboliskā Hafa transformācija, Gadījuma Hafa transformācija, Hierarhiskā Hafa transformācija, Progresīvā Paraboliskā Hafa transformācija un citas. Atsevišķi ir formulēta Universālā Hafa transformācija, kura var būt pielietojama patvaļīgu formu atpazīšanai. Taču tai nav praktiskā pielietojuma algoritma sarežģītības dēļ.

Apskatot attēlu, kas satur četrstūri, var novērot dažas īpatnības, kas ir pastāvīgas noteiktas formas figūrām. Ir pierādīts, ka izliekto daudzstūri unikāli nosaka virsotnes Hafa telpā [6]. Tāpēc, novērojot noteiktas īpašības Hafa telpā, var apgalvot, ka oriģinālajā attēlā eksistē četrstūris. Taču četrstūrim blakus esošie elementi un trokšņi var radīt nopietnus traucējumus četrstūra atpazīšanas

procesā.

Apskatīsim četrstūri, kas ir kodēts ar četrām virsotnēm

Var apskatīt divus paralēlus nogriežņu pārus un garumā a, un un garumā b. Ģeometrisko interpretāciju var apskatīt 6.2.1.(a) attēlā. Pieņemsim arī to, ka koordinātu centrs sakrīt ar četrstūra ģeometrisko centru.



6.2.1. att. **Hafa transformācijas piemērs a - četrstūris, b - Hafa telpa**

Hafa telpā ir apzīmēti lokālie maksimumi: , , un , kas attiecīgi atbilst nogriežņiem izejas attēlā V_2V_3, V_1V_4,V_3V_4 un V_1V_2. Fotogrāfijā ir redzami tikai četri maksimumi. Katrs maksimums atbilst noteiktai taisnei izejas attēlā. Apskatot vairākus četrstūrus Hafa telpā, var veikt secinājumus par parametriem, kas paliek nemainīgi un korelē ar oriģinālajā attēlā redzamo četrstūri. Tad varam definēt nosacījumus, pie kuriem apskatāmie maksimumi ir četrstūru malas:

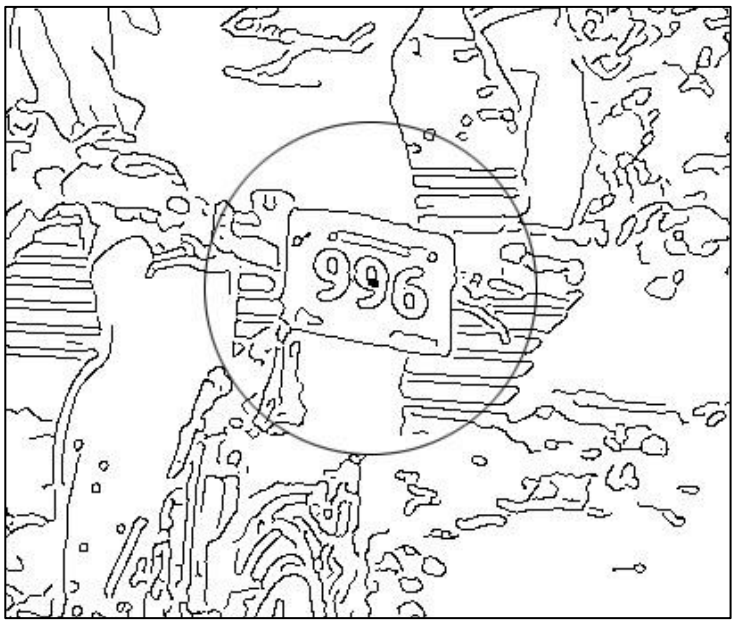
1. Maksimumi atrodas pāros. Ir redzams, ka katram maksimumam eksistē pāris H_1 pāris ir H_2 un H_3 ir H_4. Pāru maksimumiem ir vienāda parametra vērtība, tātad , apskatot H_1 un H_2, un , apskatot H_3 un H_4.
2. Pāru maksimumi ir simetriski attiecībā pret Hafa telpas asi. Šādu īpašību var apskatīt tikai tad, ja koordinātu centrs sakrīt ar četrstūra centru, citādi maksimumi atradīsies vienā vērtībā, bet attālumi p_1 un p_2 nebūs vienādi.

3. Leņķis starp maksimumu pāriem vienāds ar 90°. Tā ir četrstūra īpašība, ko var skaidri novērot Hafa telpā. Ja taisnes krustojas 90° leņķī, tad attālums starp šīm taisnēm Hafa telpā būs tieši 90°.
4. Ir arī iespējams novērtēt maksimumu stiprumu. Apskatot noteiktu maksimumu apkārtni, var secināt, ka maksimuma vērtība abiem maksimumiem, kas ir saistīti vienā pārī, ir vienādi, jo tie reprezentē nogriežņus ar vienādu garumu. Skaitliski tie būs vienādi ar meklējamā četrstūra malu garumiem. Zinot aptuvenās meklējamā četrstūra proporcijas, var izmantot šo īpašību kļūdaino datu filtrēšanai.
5. Četrstūra malu garumus reprezentē ne tikai maksimuma vērtība, bet arī Hafa telpas attālums uz p ass. Parametrs p Hafa parametru telpā ir vektora garums līdz oriģinālajam pikselim, skaitot no koordināšu sākuma. Koordināšu centrs atrodas četrstūra centrā, sajā gadījumā p_1 ir vektora garums līdz taisnei, uz kuras atrodas nogrieznis V_1V_4, un tā garums ir vienāds ar perpendikulāra garumu, vilktu no četrstūra centra līdz četrstūra malai. Zinot šādu īpašību, var secināt, ka malas garums ir vai , kur ir vektors līdz taisnei, uz kuras atrodas nogrieznis V_3V_4. Tad, ņemot vērā vektoru garumu zīmes, veidojas sakarība, ka un .

Jung, C.R. un Schramm, R. Rectangle [1] piedāvātais risinājums balstās uz piecu punktu pārbaudi visiem potenciālajiem maksimumiem. Piedāvātajos nosacījumos 2. un 5. punkts ir tieši atkarīgs no koordināšu centra izvietojuma. Lai atpazītu četrstūri, balstoties uz šiem punktiem, četrstūrim obligāti jāatrodas koordināšu centrā. Lai to īstenotu praksē, netiek apstrādāts viss attēls uzreiz. Tiek izvēlēts pētāmais apgabals, ko analizē un meklē atbilstošās sakarības. Apgabalam jābūt pietiekami lielam, lai tajā pilnībā ietilptu četrstūris, bet arī jābūt mazākam, lai blakusesošie elementi pēc iespējas mazāk ietekmētu apstrādes gaitu. Apgabalu var ierobežot ar riņķa līnijas palīdzību, jo riņķa līnija ir vienkāršākā aprakstāmā ģeometriskā figūra, kas vairāk atbilst četrstūra izliektai čaulai.

Pētāmo apgabalu apraksta ar aprobežojošas riņķa līnijas diametru. Diametra izmērs jāizvēlas intervālā starp D_{min} un D_{max}. Autors piedāvā šo izmēru noteikt imperatīvi. D_{min} – minimālais četrstūra izmērs, ko ir nepieciešams apstrādāt. D_{min} aptuveni jāsakrīt ar minimālā apgabala izmēru, kurā nākošajos etapos būs iespējams atpazīt dalībnieka numuru. Gadījumā, ja D_{min} būs pārāk mazs, tas ir $D_{min} < D_{a\text{-}min}$, kur $D_{a\text{-}min}$ ir minimālā apgabala izmērs, kurā ir iespējams

atpazīt simbolus ar ORC algoritmu, tad eksistē $\Delta D = D_{a\text{-}min} - D_{min}$, kad apstrādei nav jēgas, jo pat atrodot četrstūri izmērā D_{min}, tajā nebūs iespējams atpazīt simbolus. D_{max} parametru jāizvēlas, ņemot vērā attēla izmēru. D_{max} ir maksimāli iespējamais numura plates izmērs fotogrāfijā. Saprotams, ka var gadīties, ka attēlu pilnībā aizņems dalībnieka numurs, bet, apstrādājot šādus robežgadījumus, var ļoti būtiski zaudēt ātrdarbību.



6.2.2. att. **Četrstūru meklēšanas apgabala izvēle**

Veidojot Hafa telpu, jādefinē diskretizācijas pakāpe. Hafa telpā tiek reprezentētas visas iespējamās taisnes. Saprotams, ka iespējamo taišņu skaits ir bezgalīgs, tāpēc jāizvēlas diskretizācijas soļi. Shen un Wang [3] piedāvā izvēlēties diskretizācijas soli, balstoties uz oriģinālā attēla izmēru. Jo lielāks ir oriģinālais attēls, par tik diskretizācijas solim jābūt mazākam. Samazinot diskretizācijas soli, Hafa telpa palielinās, jo saturēs informāciju par vairākām iespējamajām taisnēm. Tiek piedāvāts izmantot Hafa transformācijas telpas platumu —un augstumu —, kur W un H – attēla dimensijas [4]. Apskatāmajā gadījumā, kad pētāmo robežu definē riņķa līnija $W = H = D_{max}$, diskretizācijas soli var izteikt:

———

uz ass, un uz ass:

Hafa telpā pikseļiem reālajā attēlā atbilstošie sinusoīdi pārklājas noteiktos punktos. Pēc transformācijas izpildes akumulatoros rezistoros tiek saglabāta informācija par attēlu. Hafa telpā ir vairāki punkti, kurus šķērso vairāki sinusoīdi. Katrs punkts Hafa telpā) skaitliski vienāds ar pikseļu skaitu, kas potenciāli var veidot punkta aprakstāmo taisni, un vienāds ar potenciālās taisnes garumu attēlā. Hafa punkta vērtība ir visu to punktu skaits, kas apmierina vienādojumu . Nākamais apstrādes uzdevums ir atlasīt tikai tos punktus, kas viennozīmīgi apzīmē taisnes attēlu – atlasīt Hafa telpas maksimumus.

Punktu atlasi var veidot pēc taisnes garuma [10]. Ar reālajām taisnēm var nosaukt visus tos Hafa telpas punktus, kas apmierina nosacījumu , kur ir taisnes mazākais iespējamais garums. Garuma atlase dod labus rezultātus mākslīgajos testos, bet nav veiksmīga stipru trokšņu gadījumā (par trokšņiem šeit var nosaukt ne tikai grafiskos trokšņus, bet arī citu objektu malas vai malu nogriežņus, kas ietilpst pētāmajā apgabalā). Šādu īpatnību var aprakstīt ar to, ka apskata visus pieejamos punktus, kas atrodas uz aprakstāmās taisnes. Līdz ar to jebkurš troksnis, kas atrodas pētāmās taisnes pagarinājumā, palielina akumulatora H vērtību. Diviem atsevišķiem nogriežņiem, kas atrodas uz vienas taisnes, H vērtība, kas raksturo šo taisni, būs skaitliski vienāda ar šo nogriežņu garumu summu, neskatoties uz nogriežņu savstarpējo izvietojumu. Ir iespējama situācija, kurā nogriežņi var piederēt atsevišķiem objektiem, bet informācija par to būs apkopota vienā Hafa telpas punktā. Garumu summu problēma vēlreiz apstiprina pētāmā apgabala pareizo izmēra izvēli.

Pētot Hafa telpu, var secināt, ka maksimumus Hafa telpā ir lietderīgi meklēt, analizējot tauriņveida struktūras. Ap maksimuma punktu veidojas sinusoīdu struktūra, kas vizuāli atgādina tauriņa spārnus. Sinusoīdi, šķērsojot maksimuma punktu, veido četrus „izaugumus", kas pakāpeniski sadalās atsevišķos sinusoīdos. Efekts vērojams diskretizācijas dēļ, kad sinusoīdi, kas raksturo dažādus punktus, krustojas ne tikai maksimuma punktos, bet arī blakus esošajos punktos. Matemātiskajā interpretācijā blakus krustošanās neeksistē, un katrs sinusoīds krustojas ar citiem tikai maksimuma punktos. Šādu diskretizācijas īpatnību var izmantot maksimumu meklēšanai. Hafa telpa būtiski neatšķiras no melnbaltā attēla. Katru telpas punktu raksturo divas koordinātes p un un punkta intensitāte (taisnes pikseļu skaits). Telpas apstrādei var izmantot līdzīgas attēlu apstrādes tehnikas. Izmantojot konvolūcijas maskas, var meklēt tauriņveida

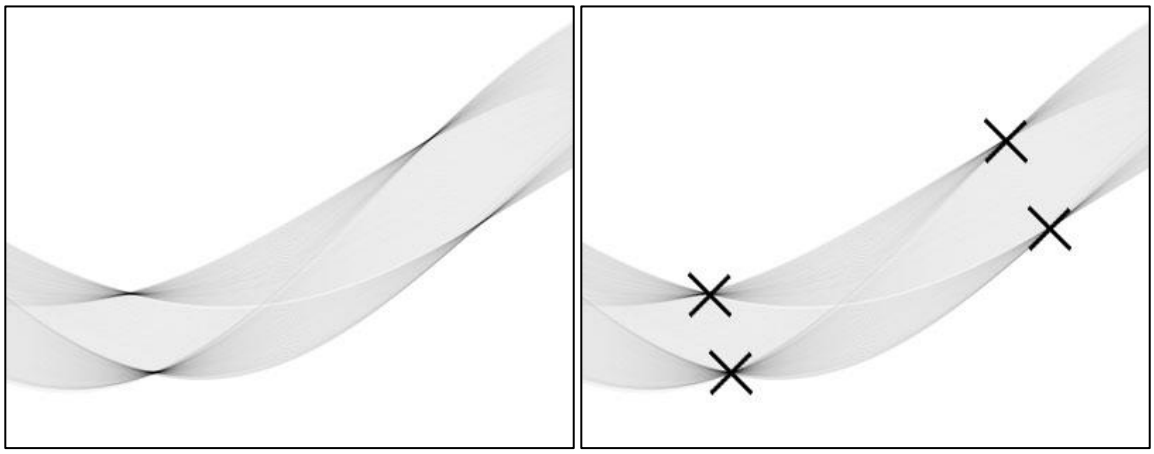
struktūras Hafa telpā [7]. Konvolūcijas maskas piemērs maksimumu meklēšanai:

Fukuwa un Shigawa piedāvātais maksimumu meklēšanas variants ir balstīts uz kopējo Hafa telpā iekļauto vērtību. Katru punktu pārveido:

;

kur h un w - Hafa telpas izmēri diskretizētās vienībās - punktos.

Iegūstot lokālos maksimumus, tos filtrē ar jau apskatīto slieksni ,. > ,. , lietderīgi aprēķināt, balstoties uz pētāmā apgabala izmēru. Mūsu gadījumā izmantojam riņķa līniju un , var definēt kā pusi no Rindas līnijas diametra, t.i. , = $0{,}5D_{min} = R_{min}$. Pielietojot šādu slieksni, tiek apskatītas taisnes, kas ir lielākas par minimālā pētāmā apgabala izmēra pusi. Šāda sliekšņa filtrācija padara Hafa transformāciju ļoti izturīgu pret datu zudumu. Iespējams zaudēt līdz 50% no nogriežņa malu taisnes punktiem, joprojām atrodot plāksnes robežas.



6.2.3. att. **Hafa transformācijas rezultāts a- Hafa telpa, b – lokālie maksimumi**

Atrodot lokālos maksimumus, nākamais posms ir atrast tos, kas apmierina četrstūra nosacījumus Hafa telpā (nosacījumi aprakstīti 6.2. nodaļas sākumā). Pirms veikt četrstūra pazīmes kontroli, nepieciešams sadalīt maksimumus pāros, kas varētu atbilst paralēlajām malām.

Katrs maksimumu pāris ir pārbaudīts, vai tas apmierina nosacījumus:

Leņķu identitātes nosacījums. Abām potenciālajām taisnēm jābūt paralēlām, kur ir leņķa slieksnis. Leņķa slieksnis norāda uz maksimālo nobīdi starp taisnēm. Ja taisnes ir paralēlas, . Ar tiek pieļauta neliela leņķa nobīde, kas var veidoties trokšņu vai attēla īpatnības dēļ.

Attāluma identitātes nosacījums. Katra potenciālā taisne atrodas vienādā attālumā no koordināšu centra. Pēc izpētes definējuma koordināšu centram jāatrodas četrstūra centrā, tāpēc attālumi līdz četrstūru malām sakritīs, jo sakrīt perpendikulāru garumi, kas vilkti no četrstūra ģeometriskā centra līdz tā paralēlajām malām .

.

—————|;

Garumu identitāte. Potenciālo taišņu garumiem jāsakrīt. – normalizēts slieksnis, kas apraksta lielāko iespējamo taišņu garumu atšķirību procentuālajā izteiksmē. slieksnis ir līdzīgs pieļaujamajam kļūdas procentam, kas parāda to, par cik procentiem taisnes var atšķirties savā starpā.

Sadalot maksimumus pāros, ar lielu varbūtību var teikt, ka minētie maksimumi ir paralēlas līnijas ar vienādu garumu, kas atrodas vienādā attālumā no koordināšu centra [13]. Turklāt pētāmā attēla apgabalā var atrasties vairāki četrstūri un vairākas paralēlās taisnes. Nākošais etaps ir pāru savstarpējā sasaiste. Jāatrod pāri, kas ir ortogonāli savā starpā un var veidot četrstūri. Informāciju par taišņu pāri var apkopot, aprēķinot vidējo vērtību piedāvā [1].

–

un

– ;

Mainīgie skaitliskie atbilst vidējam leņķim un vidējam izmēram. Aprēķinot vidējo leņķi

katram potenciālajam pārim, var ērti salīdzināt tos savā starpā. Hafa telpā leņķa ass reprezentē leņķi starp koordināšu asi un vektoru līdz attēla punktam. Var apgalvot, ka, ja taišņu pāru vidējo leņķu starpība vienāda ar 90 grādiem, tad pāri ir ortogonāli un var secināt, ka tie veido četrstūri.

Trokšņu un robežu meklēšanas neprecizitātes dēļ malas var nekrustoties taisnā leņķī. Malu krustošanās leņķis arī ir atkarīgs no sportista un fotogrāfa savstarpējā izvietojuma fotografēšanas brīdī. Ja plakne, uz kuras atrodas dalībnieka numura plāksne, veido leņķi, kas atbilst tuvu taisnam leņķim ar fotogrāfa fotografēšanas vektora perpendikulāro plakni (fotogrāfa fotografēšanas vektoram perpendikulārā plakne ir sava veida fotogrāfijas plakne, tā plakne, uz kuras tiek projicēta fotogrāfija), tad numura plāksne būs deformēta.

Autors piedāvā izvēlēties šķērsošanas slieksni, balstoties uz malu attiecībām. Veidojoties lielai nobīdei, numura simboli aizņems arvien mazāk horizontālās vietas un kādā brīdī izveidosies situācija, kurā grafisko trokšņu, izpludināšanas un nobīdes dēļ nebūs iespējams sadalīt numura simbolus savā starpā. Protams, iespējamā situācija ir ļoti atkarīga no attēla izmēra un absolūtās numura plāksnes izmēriem fotogrāfijā, tomēr 40 grādu lielo leņķi var uzskatīt par to slieksni, pie kāda četrstūri apskatīt nav jēgas.

Četrstūru meklēšana attēlā notiek pakāpeniski. Ierobežojošā riņķa līnija tiek pārvietota pa attēlu. Katrā pārvietošanas reizē tiek veikta apstrāde un četrstūru meklēšana. Pazīmju sakritības gadījumā rezultāts tiek saglabāts un apstrāde tiek turpināta. Izmantojot šādu metodiku, ir iespējams atpazīt četrstūrus ar visdažādākajiem izvietojumiem un ar patvaļīgiem malu garumiem. Izmantojot pārvietojamo apstrādes logu, var veidoties dublikāti. Nebūtiski pārvietojot apstrādes logu, tiek radīta situācija, kurā jau atpazītais četrstūris var būt atpazīts atkārtoti [1]. Lai izvairītos no dublikātu veidošanās, var samazināt pieļaujamās nobīdes – sliekšņus un , bet tam var sekot deformētu četrstūru neatpazīšana. Veicot apstrādi, ir iespējams noteikt vispareizāk atpazīto četrstūri. Faktiski veicot četrstūra atpazīšanu, atpazīšana notiek ar noteiktu mērāmu kļūdu. Tieši tad, kad kļūda ir sliekšņu ietvaros, četrstūra atpazīšana notiek sekmīgi. Par labāk atpazīto četrstūri var uzskatīt to četrstūri, kam kopējā atpazīšanas kļūda ir vismazākā. Kopējo atpazīšanas kļūdu var aprakstīt kā kvadrātsakni no kļūdu summas:

;

Pastāv iespēja kļūdas sadalīt pa grupām. Leņķu nobīdes kļūdas un izmēra kļūdas. Tad kļūdas

V_k,V_t,V_a pieder pie leņķu kļūdām un P_k, P_t - pie izmēra kļūdām. Vērtības, kas ir iekļautas kopējās kļūdas aprēķinā, ir mērāmas dažādās vērtībās (leņķos un pikseļos). Vizuāli viena pikseļa nobīde ir pamanāmāka nekā viena grāda nobīde, tāpēc ir ieteicams izmantot svaru koeficientus. Katrai kļūdu grupai var definēt savu svaru kopējās kļūdas aprēķinā:

;

kur a – leņķu kļūdas svars un b – izmēru kļūdas svars.

Sameklējot četrstūru dublikātu kopā četrstūri ar vismazāko kopējo kļūdu, varam apzīmēt to kā vislabāk atpazīto un īsto četrstūri, bet citus uzskatīt par šī četrstūra dublikātiem.

Četrstūra atpazīšana ir viena no sarežģītākajiem procesiem aprakstītajā algoritmā. Izvēlētais uzdevums neparedz daudz iespēju izmantot grafiskos enkurus. Numura plāksne ir visvieglāk atpazīstamais grafiskais enkurs, kas ir pietiekami universāls un relatīvi viegli atrodams algoritmiski.

Četrstūru meklēšanas algoritms ir pielāgojams ar 5 mainīgajiem:

1. Pētāmā apgabala diametrs D. Pētāmā „loga” – apgabala izmērs.
2. Līnijas minimālais garuma slieksnis .
3. Šo taišņu paralelitātes nobīde. Maksimāla leņķu nobīde paralēlajām četrstūru malām.
4. – attāluma nobīde. Iespējama četrstūra nobīde no koordināšu centra. Ietekmē dublikātu veidošanos.
5. nogriežņu identitātes nobīde.
6. Perpendikulārās krustošanās nobīde. Maksimāli pieļaujamā leņķa nobīde no taisnā leņķa, krustojoties četrstūru malām.

6.2.4. att. **Dalībnieku fotogrāfija ar iezīmētam dalībnieku numura plāksnēm**

6.3. Kandidātu attēlu atlase

Apstrādājot attēlu, esam nonākuši pie soļa, kurā numura plāksnes izvietojums attēlā ir zināms. Numura plāksnes atpazīšana notiek, meklējot četrstūra objektus attēla robežu informācijā. Šāds paņēmiens var radīt tā saucamos loģiskos trokšņus. Meklējot numura plāksnes, netika ņemts vērā tas, kas atrodas četrstūra robežās. Izvēlētais četrstūris ir tikai četrstūra objekts, ko nodala ātras intensitātes pāreju robežas. Pastāv iespēja, ka daļai no iepriekš sameklētajiem četrstūriem būs loģiskie trokšņi. Ar loģiskajiem trokšņiem apskatīsim jebkurus četrstūru objektus fotogrāfijā, kur nav dalībnieku numuru plāksnes. Jebkurš četrstūra objekts fotogrāfijā potenciāli var būt atpazīstams kā numura plāksne. Kā piemēru var izmantot ķieģeļu sienas attēlu. Katrs

ķieģelis ir labi segmentējams četrstūris, bet neviens ķieģelis nav dalībnieka numurs.

Veicot automobiļu numuru atpazīšanu, ir iespējams meklēt attēla apgabalus tur, kur intensitāte mainās vairākas reizes nelielā attēla daļā. Šāda īpašība var netieši norādīt uz numura eksistenci šajā apgabalā. Turklāt sporta sacensības numuri tiek piešķirti, skaitot no 1. numura, bet viens cipars neveido vairākas atkārtojošās intensitātes izmaiņas attēlā. Tāpēc paņēmienu ar vairākkārtējām intensitātes izmaiņām, ko plaši izmanto numuru atpazīšanas uzdevumu risināšanai, nav iespējams izmantot.

Viens no variantiem, kā var pārbaudīt to, vai potenciālais četrstūris ir vai nav numuru plāksne, ir meklēt tajā ciparus. Ja ignorē loģiskos trokšņus, tad numura atpazīšana beigsies ar kļūdu – neviena simbola sakritība netiks atrasta. Apskatot piemēru ar ķieģeļu sienu, var secināt, ka šāda rīcība var izraisīt lielas apstrādes ātrdarbības problēmas, jo katrs nākamais ķieģelis tiks apstrādāts ar simbolu segmentācijas un OCR algoritmiem.

Otrs variants ir apstrādāt attēlu un pētīt attēla histogrammu. Kā jau tika aprakstīts 3.2. nodaļā, attēla histogramma parāda pikseļu sadalījumu pēc intensitātēm. Ja plāksne ir loģiskais troksnis, tad intensitātes sadalījums neeksistēs vai būs citādāks nekā numura plāksnēm. Ar šādu paņēmienu uzreiz var labot attēla intensitāti, jo histogramma var kalpot kā informācijas avots attēla intensitātes labošanai un binārā attēla sliekšņa izvēlei. Attēla intensitātes labošana ir nepieciešama, lai nodrošinātu stabilāku simbolu atpazīšanu tālākajos etapos. Histogrammas analīzes paņēmiens nespēj filtrēt visus iespējamos loģiskos trokšņus. Pastāv situācijas, kad uz gaišā fona ir pietiekami zemas intensitātes trokšņi vai četrstūru atpazīšanas laikā četrstūris netiek atpazīts ar pietiekamu precizitāti. Tad tumšās apkārtesošās malas var kalpot par tumšo simbolu pikseļiem histogrammas sadalījumā. 5.3.1. attēlā ir parādīti šādu kļūdu piemēri.

6.3.1. att. **Histogrammu metodes kļūdas a - plāksnes attēls b - loģiskā trokšņa attēls c - histogrammas a un b (treknā) attēliem**

Trešais paņēmiens ir iespēja izmantot neironu tīklu. Neironu tīklus bieži vien izmanto arī

simbolu atpazīšanai. Neironu tīkla sarežģītību un apmācīšanas iespēju būtiski nosaka induktīvais pieņēmums, t.i. topoloģijas izvēle, ieejas kodēšana, ieeju skaits, izejas kodēšana, izeju skaits un normalizācijas veids. No neironu tīkla topoloģijas, galvenokārt, ir atkarīgs skaitļojumu apjoms, kas jāveic, darbinot neirona tīklu. Ir iespējams izmantot pēc iespējas vienkāršotu topoloģiju un veidot neirona tīklus, kur galvenais uzdevums būs atpazīt nevis konkrētus simbolus, bet to eksistenci attēlā. Ar diezgan lielu precizitāti var noteikt numura esamību plāksnē un šādā veidā filtrēt loģiskos trokšņus. Neironu tīkla vājā īpašība ir apmācīšanas procesā. Apmācot tīklu uz konkrētas datu kopas bāzes, nav iespējams pārliecināties par to, ka kādā citā datu paveidā darbība noritēs tikpat labi.

Autors piedāvā loģisko trokšņu filtrēšanā izmantot numuru plāksnes īpašības. Plāksnes numurs ir rakstīts ar tumšāku krāsu gaišākā fonā un numurs ir centrēts gan vertikāli, gan horizontāli. Filtrējot loģiskos trokšņus, var izmantot jau esošo robežu informāciju. Pēc Kanni malu detektora darbības ir pieejama informācija par intensitātes izmaiņām – robežām. Numurs ir domāts dalībnieku atpazīšanai, tāpēc var secināt, ka numura simbolu kontrasts ar plāksnes fonu būs vienmēr pietiekams, lai numura simboli tiktu iezīmēti ar robežām plāksnes apgabalā. Izmantojot robežu informāciju, jāuzbūvē robežu projekcijas abu asu virzienā. Robežu projekcijas ir samērā vienkārša, taču rezultatīva metode. Horizontālo malu projekciju x asi var apskatīt kā funkciju:

;

kur h – apstrādātās plāksnes augstums un i - pikseļa intensitāte. Robežu informācija ir pieejama kā binārs attēls, tāpēc ir pieejamas tikai divas vērtības {0,1}. Citiem vārdiem sakot - katrai diskrētai vērtībai tiek meklēts to pikseļu skaits, cik reizes ar vieninieku kodētie pikseļi būs atrodami attēlā uz taisnes, ko raksturo vienādojums .

Pirms izmantot robežu projekciju, jāveic datu normalizācija un pirmapstrāde. Apstrādājot projekciju, jāizslēdz projekcijas daļas, kur x vērtības ir minimālas un maksimālas, jo potenciālā plāksne var būt neprecīzi segmentēta un pašas plāksnes robežas var ietekmēt projekcijas analīzi. Ja projekcijas maksimums atrodas x vērtību centrālajā daļā, tad var apgalvot, ka kaut kāds atpazīstams objekts plāksnē ir centrēts plāksnes x ass virzienā. Robežu projekcijas centra nosacījums:

– , kur xmax – x vērtība, kur izvietojies robežu projekcijas maksimums uz x ass, w – plāksnes platums, – pieļaujamais nobīdes slieksnis.

Līdzīgu operāciju veido y ass virzienā:

– , kur ymax – y vērtība, kur izvietojies robežu projekcijas maksimums uz y ass, h – plāksnes augstums, – pieļaujamais nobīdes slieksnis.

Ja plāksne apmierina abus nosacījumus, tad ar lielu varbūtību var teikt, ka apskatāmais četrstūris ir numura plāksne.

6.4. Numura segmentācija

Numura segmentācijas uzdevums paredz numura simbolu izdalīšanu numura plāksnē. Segmentācijas uzdevumu pilda jau nodalītai numura plāksnei. No iepriekšējiem apstrādes etapiem ir pieejams oriģinālais attēls un robežu binārais attēls. Autora piedāvātais variants balstās uz 6.3. nodaļā ” Kandidātu attēlu atlase” aprakstīto autora piedāvāto loģisko trokšņu filtrēšanu. Filtrēšanā tiek izmantotas numura plāksnes īpašības: numura izvietojums plāksnes centrā un numura simbolu relatīvs kontrasts ar numura plāksnes fonu. Plašāk paņēmiens apskatīts 6.3. nodaļā

Segmentācijai nepieciešamā informācija tiek iegūta no numura plāksnes robežu informācijas. Robežu informācija ir iekodēta binārā attēla veidā un ir pieejama pēc Kanni malu detektora darbības. Veidojot y ass robežu projekciju, ir iespējams segmentēt to plāksnes attēla daļu, kur jāatrodas numuram. No projekcijas ir iespējams secināt, ka numurs vienmēr atrodas vienā no lokālajiem projekciju maksimumiem. No fotogrāfijas redzams, ka var eksistēt vairāki maksimumi un viena lokālā maksimuma atrašana negarantē to, ka tieši tajā atradīsies numurs. Būs nepieciešams apstrādāt visas potenciālās plāksnes joslas. Lokālo maksimumu meklēšanā var izmantot vairākus paņēmienus – pētīt funkcijas vērtības, veidot robežu projekciju atvasinājumu vai apskatīt joslas platumu. Apskatot robežu projekcijas vairākām numura plāksnēm, var definēt numura joslas īpašības, kas jāņem vērā, meklējot nepieciešamo joslu.

1. Joslas platumam jābūt pietiekami lielam, lai tajā izvietotu numuru. Pieņemsim, ka lielāka izmēra numurs aizņem ¾ no attēla augstuma, bet mazākais - 1/3 no attēla augstuma. T.i., W – numura joslas platums. – –

2. Numura robežu projekcijas maksimumu josla sākas ar minimumu un beidzas ar minimumu. Tas saistīts ar to, ka ap numuru ir brīva vieta, ko neaizpilda ar sponsoru reklāmu vai citu informāciju.

   , kur – minimālā trokšņa slieksnis, pieņemot, ka ir iespējami nelieli trokšņi.

3. Numura maksimumam jāsatur vismaz vienu punktu, kur projekcijas vērtība lielāka par vidējo projekcijas vērtību

   Ey p(y) <= pavg, kur - vidējā projekcijas vērtība, aprēķināta pēc

   —— , kur size - vertikālās projekcijas izmērs (visu to nogriežņu summa, kur vertikālā robežu projekcija ir lielāka par 0).

Ja vertikālās projekcijas josla apmierina visus trīs kritērijus, tad tā tiek uzskatīta par kandidāta numuru. Katram plāksnes kandidāta numuram tiek veikta nākamā apstrāde. Nākamā apstrādes posma mērķis ir sadalīt plāksnes numuru atsevišķos simbolos, kurus izmanto atpazīšanai.

Katram numura kandidātam tiek aprēķināta horizontālā robežu projekcija. Ja joslā ir numurs, tad projekcijas maksimumi atbilst atsevišķam numura ciparam.

Aprēķinot projekcijas maksimumu, joslas var aprakstīt ciparu izvietojumu numura joslā. Varam definēt slieksni T_x, lai nodalītu projekcijas joslas.

Ja $p(x) => T_x$, tad tā ir josla,

ja $p(x) < T_x$, tad tā ir tukša vieta starp cipariem.

Sliekšņa T_x vērtību ir vēlams piešķirt, balstoties uz projekcijas vidējo vērtību. Piemēram, T_x = ——, kur size - projekcijas garums, visu to pikseļu garums, kur projekcijas vērtība lielāka par 0.

Segmentācijas darbs ir izpildīts tad, kad visi cipari ir sadalīti atsevišķos četrstūru apgabalos.

6.5. Simbolu atpazīšana

Pēc iepriekšējo attēla apstrādes etapu veikšanas iegūti ciparu atsevišķie attēli. Šos attēlus jāizmanto simbolu atpazīšanai ar ORC algoritmu. OCR algoritmi ir plaši pielietota un labi izpētīta sfēra. Pētot atvērtā koda risinājumus un jau izveidotās programmas, autors secināja, ka nav nepieciešams veidot savu risinājumu un var izmantot jau esošās realizācijas.

Segmentēto numuru atpazīšanai tiek izmantota Tesseract OCR. Tesseract OCR – atvērtā koda atpazīšanas produkts, kas sevi labi parāda numuru atpazīšanas uzdevumos [8]. Plašāk par konkrēto risinājumu izvēli ir aprakstīts 7. nodaļā.

6.5.1. att. **Numura atpazīšanas bibliotēkas pielietojums a - segmentētā plāksne, b – segmentēta numuru josla, c – segmentēti un atpazīti simboli**

7. REALIZĀCIJAS APRAKSTS

Pēc teorētisko metožu apskata autors mēģināja realizēt aprakstītās metodes un izveidot datora redzes sistēmu. Sistēma realizē atvieglotu variantu un nav optimizēta plašai izmantošanai. Tās mērķis ir pierādīt atpazīšanas iespējamību un norādīt uz iespējamajām problēmām, kā arī izpētīt attīstības virzienu.

Visi ātrdarbības novērtējumi tiek veikti uz testa datora: Intel Core 2Duo 1.8 2Gb operētājatmiņas, Ubuntu 10.10 32 biti.

7.1. Programmēšanas valodas izvēle

Datora redzes sistēmas ir procesora resursietilpīgas. Attēlu apstrādes laikā jāizpilda vairākas matemātiskas darbības. Dažādas programmēšanas valodas ir atšķirīgi pielāgotas resursietilpīgajiem uzdevumiem. Veidotās realizācijas nolūks ir apskatīt algoritmu realizācijas iespēju ar ieskatu tālākajā attīstībā, tāpēc autors vispirms izvēlējās programmēšanas valodu no jau pazīstamajām valodām. Pirmā izvēle bija PHP programmēšanas valoda, jo risinājuma pielietojumu plānots cieši saistīt ar tīmekli. Tika izpētīta dažādu attēlu apstrādes ātrdarbība. Gandrīz uzreiz tika konstatēts, ka PHP valoda nav pielietojama grafikas apstrādei. Ir iespējams izmantot iebūvētās attēla apstrādes funkcijas un paplašinātās bibliotēkas, bet dažus paņēmienus no teorētiski aprakstītajiem nav iespējams īstenot, neizmantojot konvolūcijas matricas. Konvolūcijas matricas pielietošana attēlam notika ar ātrumu 2000 pikseļu sekundē. Ar šādu ātrdarbību varēja 30 sekunžu laikā apstrādāt attēlu ar 200 X 300 pikseļu izmēru. Šāda ātrdarbība neapmierina un tāpēc tika izvēlēta valoda Java. Java valodā līdzīgas operācijas izpilde notika vismaz 2000 reizes ātrāk.

Java valoda tika izvēlēta, jo autoram tā ir pazīstama un autoram ir plaša praktiskā pieredze, strādājot gan ar Java valodu, gan ar Java izstrādes rīkiem un Java tīmekļa platformu. Java platforma ir labi piemērota attēlu apstrādei. Standarta klasēs atrodamas klases grafisko failu interpretācijai, atvēršanai un saglabāšanai.

Tālākajā izpētē autors secināja, ka eksistē valodas, kas ir optimizētas matemātisko

pārveidojumu veikšanai un labāk noder tieši algoritmisko problēmu risināšanai. Tika apskatīta Mathlab pakotne un valoda M. Tā ir optimizēta matemātisko funkciju realizācijā. Mathlab pakotnē jau ir realizētas vairākas transformācijas (piemēram, Hafa transformācijas, gradienta meklēšana, utt..). M – valoda dod iespēju lakoniski aprakstīt algoritmus un strādāt ar attēlu kā ar matricu. Konkrēti M valodas pielietojumi nav aprakstīti darbā, bet autors atzina, ka tā būtu labākā programmēšanas valodas izvēle datora redzes problēmas pētījumiem.

7.2. Sistēmas prasību definēšana

Sistēmai ir izvirzītas dažas prasības, kas jāievēro, izvēloties apbilstošu risinājumu un algoritmus. Tie ir tikai pieņēmumi un atkāpe no šiem pieņēmumiem nav kritiska, tomēr nav vēlama. Par strādājošu sistēmu tiek uzskatīta lietotne, kas spēj kontrolkopā atpazīt dalībnieku numurus. Kontrolkopa tiek veidota no vairākiem attēlu krājumiem un satur dažāda izmēra un kvalitātes fotogrāfijas. Fotogrāfijas tiek ņemtas atbilstoši fotogrāfiju kvalifikācijai, kas aprakstīta 2. pielikumā:

- Atpazīšanas efektivitātei pirmās un otrās kvalitātes fotogrāfijas klašu fotogrāfijām jātuvinās 100%.
- Atpazīšanas efektivitātei trešās un ceturtās kvalitātes fotogrāfijas klašu fotogrāfijām jātuvinās 80%.
- Darbības laiks nedrīkst pārsniegt 5 sekunžu robežu 0.5 Mp (0.5 Mega pikseļu aptuvenais 800*600 fotogrāfijas ekvivalents) lielai fotogrāfijai.
- Apstrādes laikā nedrīkst paturēt vairāk nekā 400Mb operētājatmiņas uz 1Mp ieejas datu apjomu.

7.3. Realizācijas metodes apraksts

Realizācijas koncepcija tika formulēta 1.1. nodaļā, kad grafiski tiek parādīta atpazīšanas procesa plūsma. Kopējais process sastāv no līdzīgiem soļiem. Katru soli veido viens algoritms,

kas balstās uz iepriekš paveikto soļu izejas datiem.

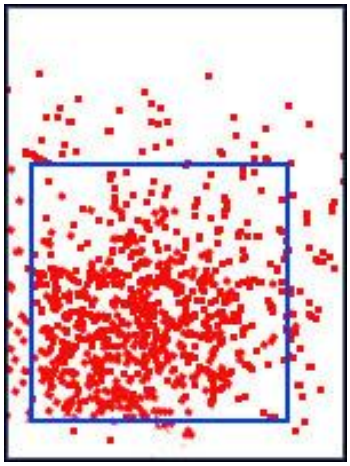
Katrā posmā ir izvēlēti risinājumi, kas sīkāk aprakstīti teorētiskā izklāsta nodaļās. Soļu realizācijas apskatā tiek piedāvātas norādes uz teorētisko aprakstu un aprakstītas pielietotās optimizācijas.

1. Pirmais solis - notiek attēla pirmapstrāde. Attēls tiek filtrēts ar Gausa filtru (plašāk aprakstīts 4.2.1. nodaļā), izmantojot konvolūcijas kodolu izmērā 3x3 ar sigma 1. Konvolūcijas matrica ir: Gausa konvolūciju matricas pielietojuma realizācija ir pievienota
2. Nākamais solis ir malu detektors. Tika realizēts vienkāršotais Kanni malu detektors. Kanni detektors sastāv no pieciem apakšposmiem (plašāk aprakstīts 5.3. nodaļā).
    2.1.Gausa izpludināšana.
    2.2.Gradientu aprēķins.
    2.3.Ne maksimumu nomākšana.
    2.4.Dubultsliekšņu filtrācija.
    2.5.Neskaidrību izsekošana.

    Kanni malu detektora realizācija ir pievienota 4. pielikumā CannyEdgeDetector.java.
3. Četrstūru meklēšana, izmantojot Hafa transformāciju (plašāk aprakstīts 6.1. nodaļā). Binārais robežu informāciju saturošais attēls tiek apstrādāts ar Hafa transformāciju, izmantojot apstrādes logu. Loga izmērs - D = 100 px, četrstūru atpazīšanas sliekšņi - .

    Hafa transformācijas realizācija pievienota 4. pielikumā HoughTransform.java.
4. Numura segmentācijai tiek izmantots autora piedāvātais variants ar robežu projekciju pielietošanu robežu binārajam attēlam (plašāk aprakstīts 3.1. nodaļā). Robežu meklēšanas un analīzes realizācija ir apvienota vienā klasē ar maksimumu meklēšanas algoritmu un simbolu segmentācijas algoritmu.

Numuru segmentācija plašāk tiek skatīta pētāmā apgabala izvēlē. Tiek meklēts optimāls pētāmā apgabala pārvietošanas solis. Teorētiskais izklāsts paredz apgabala pārbīdi par 1 pikseli. Apstrādājot attēlu tik bieži, tas prasīs pārāk daudz skaitļošanas resursu. Tika apskatīts atpazīstamu numuru plākšņu izvietojuma sadalījums attēlā.



7.3.2. att. **Numuru plāksnes izvietojuma sadalījums attēlā**

7.3.2. attēlā tiek parādīts atpazīstamu plākšņu izvietojuma sadalījums fotogrāfijās. Katrs sarkanais rāmis attēlā apzīmē plāksnes izvietojumu fotogrāfijā. Atšķirīgo izmēru fotogrāfijas tika apvienotas centru punktā un lielākā no dimensijām pielāgota rezultatīvā attēla dimensijai. Citiem vārdiem sakot - katra nākamā fotogrāfija tika palielināta vai samazināta tikmēr, kamēr fotogrāfija pilnībā neatradās rezultatīvā attēla robežās. Šādā veidā fotogrāfijas izmēri tika normalizēti un 6.3.2. attēlā redzamā attēlu kopsumma raksturo relatīvu plāksnes izvietojumu katrā fotogrāfijā. Apskatot datus, var izdalīt fotogrāfijas zonu, kurā atrodas 80% no atpazīstamajām numuru plāksnēm (7.3.2. attēlā iezīmēta ar zilo krāsu). Šajā apgabalā pētāmo apgabalu autors piedāvā pārvietot ar jau minēto 1 pikseļa soli, bet ārpus šī apgabala – ar 5px soli. Šādā veidā tiek ietaupīts ~10% skaitļojamo četrstūru meklēšanas etapā.

1. Pēdējais etaps ir simbolu atpazīšana. Autors nolēma neveidot savu OCR algoritma realizāciju, jo OCR algoritmi tekstu atpazīšanai ir labi izpētīti un eksistē vairākas labas realizācijas. Pētāmais uzdevums paredz tikai ciparu atpazīšanu, tāpēc būs pietiekami gandrīz ar jau strādājošo risinājumu. Segmentēto numuru atpazīšanai tiek izmantota Tesseract OCR. Tesseract OCR – atvērtā koda atpazīšanas produkts, kas sevi labi parāda numuru atpazīšanas uzdevumos [8].

Pēc 5. posma izpildes numura plāksnes koordinātes un atpazītais numurs tiek saglabāti teksta faila formātā:
<Faila nosaukums> <P1X> <P1Y> <P2X> <P2Y> <P3X> <P3Y> <P4X> <P4Y> <NUMURS>,
kur P1 – P4 ir atpazītās numura plāksnes virsotnes un NUMURS ir OCR atpazītais dalībnieka

numurs.

7.4. Tīmekļa vietņu spraudņi

Risinājuma izveides laikā tika apskatīta iespēja veidot risinājumu, kas palīdzētu anotēt lietotāju veidotos fotogrāfiju krājumus. Tīmekļa lietotāji lejupielādē savus krājumus koplietošanas vortālos. Tiek apskatīta iespēja veikt fotogrāfiju anotēšanu ar dalībnieka numuriem bez nepieciešamības lejupielādēt fotogrāfijas datora redzes sistēmā. Jāapskata iespēja, ka datora redzes sistēma pati pieslēdzas pie lietotāja krājuma un anotē bildes, pievienojot birkas vai pievienojot komentāru pie katras fotogrāfijas, kur izdevies atpazīt dalībnieka numurus. Lai veiktu anotēšanu, krājuma īpašniekam nepieciešams „pasūtīt" anotēšanu un iedot tiesības pievienot birkas krājuma fotogrāfijām.

No apskatītajiem fotogrāfiju koplietošanas vortāliem dažiem ir pieejams API. Tiek apskatīta iespēja iegūt konkrēta koplietošanas lietotāja fotogrāfiju krājumus un iespēja katrai fotogrāfijai piestiprināt birku. Vortāli, kuros ir pieejams API fotogrāfiju piekļuvei:

1. Google Picasa (http://code.google.com/apis/picasaweb/overview.html);
2. Facebook (http://developers.facebook.com/docs/reference/api/photo/);
3. Flickr (http://www.flickr.com/services/api/).

Minētajiem vortāliem ir iespējams veidot attālinātu anotēšanu. Autors izpētīja šādu risinājumu iespējamību, bet koplietošanas vortālu spraudņu realizācija netiek apskatīta šajā darbā.

SECINĀJUMI UN TURPMĀKĀ ATTĪSTĪBA

Pēdējo gadu laikā fotogrāfiju skaits tīmeklī strauji pieaug. Tas ir saistīts ar digitālo fotokameru pieejamību, kā arī ar tīmekļa koplietošanas vortālu attīstību. Cilvēki arvien aktīvāk izmanto tīmekļa galerijas, lai glabātu veidotās fotogrāfijas. Pastāv pieprasījums pēc attēlu klasifikācijas un anotēšanas, tomēr automātiskā klasifikācija un anotēšana joprojām slikti attīstīta. Neskatoties uz to, ka datora redze kļūst arvien populārāka un tiek plaši izmantota mūsu dzīvē, joprojām nav universāla un brīvi pieejama risinājuma, kas palīdzētu atrisināt datora redzes problēmas.

Darbā autors izpētījis populārākās numuru atpazīšanas tehnikas un algoritmus. Var apgalvot, ka pastāv vairāki paņēmieni un veidi objektu segmentācijai un atpazīšanai. Algoritmus var uztvert kā ķieģeļus, no kuriem ir iespējams veidot kompleksas datora redzes un tēlu atpazīšanas sistēmas.

Šī darba ietvaros apskatītajiem algoritmiem tika veidota implementācija programmēšanas valodā Java. Autors ir pierādījis, ka ir iespējams veidot risinājumu, kas automatizētu sporta sacīkšu fotogrāfiju anotēšanu.

Autors uzstāj uz piemērotas programmēšanas valodas izvēli atbilstošajiem uzdevumiem. Java valoda dažreiz nepiedāvā tik brīvas iespējas, ko dod zemāka līmeņa valodas (piemēram, C, C++), kā arī specializētās valodas (M).

Autors paredz risinājuma tālāko attīstību. Sistēma spēj paveikt uzstādīto uzdevumu, bet veidotā realizācija nav pietiekami ātra, lai to varētu pielietot tīmekļa krājumu anotēšanai. Viens no attīstības virzieniem ir ātrdarbības uzlabojums un grafiskās lietotnes izveide, lai risinājuma pielietojums būtu pēc iespējas vienkāršots. Kā otrs būtisks attīstības virziens tiek apskatīts tīmekļa vortālu spraudņu izveide, par kurām var būt liela interese no tīmekļa lietotāju puses.

IZMANTOTĀ LITERATŪRA

1. **Jung, C.R., Schramm, R**. Rectangle detection based on a windowed Hough transform. *Computer Graphics and Image Processing*, 2004, N 17, p. 113–120.
2. **Canny, J.** A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986, N 6, vol. 8.
3. **Shen, F., Wang, H.** Corner detection based on modified Hough transform. *Pattern Recognition Letters*, 2002, N 23, vol. 8, p. 1039–1049.
4. **Yasutaka Furukawa, Yoshihisa Shinagawa.** Accurate and Robust Line Segment Extraction by Analyzing Distribution around Peaks in Hough Space. *Computer Vision and Image Understanding, 2004*, N 1, vol. 92, p. 1-25.
5. **Doyle, W**. Operations useful for similarity invariant pattern recognition, *Journal ACM*, 1962, vol. 9, N 2, p. 259-267.
6. **Pasi Franti, Eugene Ageenko, Saku Kukkonen, Heikki Kalviainen.** Using Hough transform for context-based image compression in hybrid raster' vector applications. Journal of Electronic Imaging - JEI, 2002.
7. **Fei Shen, Han Wang**. A Local edge detector used for finding corners. **In:** the Third International Conference on Information, Communications and Signal Processing (ICICS), Phuket, Thailand, 2001.
8. **Andris Atteka**, Automātiska automašīnu numura zīmju atpazīšana: maģistra darbs. LU Fizikas un matemātikas fakultāte, Datorikas nodaļa, Rīga : Latvijas Universitāte, 2008. 56 lp.
9. **Ondrej Martinsky**. B.SC. thesis. Brno: Brno University of Technology, 2007. 83 p.
10. **Richard O., Duda, Peter E. Hart.** *Use of the Hough transformation to detect lines and curves in pictures*, 1971, 18 p.
11. **James Z. Wang, Jia Li, Gio Wiederhold, Oscar Firschein.** *System for Screening Objectionable Images*, 1998, 12 p.
12. **Manfred Kopp, Werner Purgathofer**. *Efficient 3x3 Median Filter Computations*, 1994, 4 p.
13. **Vito Di Gesù, Vito Di Ges`u, Cesare Valenti**. *Symmetry operators in computer vision*, 2005, 9 p.

14. **J.R. Parker, Pavol Federl.** An Approach to Licence Plate Recognition, 1996, 5 p.

15. **Gernot Hoffmann.** *Luminance Models for Grayscale Conversions*, 2008, 11 p.

16. *Canny's Edge Detector* [tiešsaiste] – [atsauce 30.05.2011.]. Pieejams internetā: http://suraj.lums.edu.pk/~cs436a02/CannyImplementation.htm.

17. *Gradient based methods* [tiešsaiste] – [atsauce 30.05.2011.]. Pieejams internetā: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARSHALL/node28.html

18. *Hough Transform* [tiešsaiste] – [atsauce 30.05.2011.]. Pieejams internetā: http://planetmath.org/encyclopedia/HoughTransform.html.

PIELIKUMI:

1. Pielikums. Darbā apskatāmo datu avotu saraksts

Darbam tiek pievienota daļa no pētāmās fotogrāfiju kopas. Pielikumā pievienots DVD disks, kas satur riteņbraukšanas un skriešanas sporta pasākumu fotogrāfijas. Katrā krājuma mapē atrodas fails ar nosaukumu ‘info.txt’. Failam ir pievienots krājuma nosaukums un krājuma tīmekļa adrese (tiešsaiste). Visas tīmekļa adreses ir aktuālas uz 30.05.2011..

1. tabula

Darbā apskatāmo datu avotu saraksts

Sporta veids	Sacensību nosaukums	Krājuma nosaukums	Attēlu skaits
Skriešana (maratons)	Nordea Rīgas Maratons 2011	Nordea Rīgas Maratons 2011 galerija	>10000
	Pieejams tīmeklī: http://www.1188.lv/nordea-rigas-maratons/foto/2011		
Riteņbraukšana	SEB Latvijas čempionāts kalnu divriteņu maratonā 2010 (1. posms Cēsis - Valmiera)	SEB Latvijas čempionāts kalnu divriteņu maratonā 1. posms (Cēsis – Valmiera)	1310
	Pieejams tīmeklī: http://www.failiem.lv/list.php?i=ixeogk		
Skriešana	SEB Latvijas čempionāts kalnu divriteņu maratonā 2010 (1. posms Cēsis - Valmiera)	Cesis - Valmiera by Kristaps Skapars	607
	Pieejams tīmeklī: http://evelo.lv/?id=3538		
Riteņbraukšana	SEB Latvijas čempionāts kalnu divriteņu maratonā 2010 (1. posms Cēsis - Valmiera)	Barošanas punkts - SEB MTB 2011 Cesis-Valmiera (S.Mainule-Meidropa)	99
	Pieejams tīmeklī: https://picasaweb.google.com/wwwvelolv/BarosanasPunktsSEBMTB2011C		

	esisValmieraSMainuleMeidropa		
Riteņbraukšana	SEB Latvijas čempionāts kalnu divriteņu maratonā 2010 (1. posms Cēsis - Valmiera)	SEB Cēsis-Valmiera	15
	Pieejams tīmeklī: http://www.fotoblog.lv/rep/24264/		
Riteņbraukšana	SEB Latvijas čempionāts kalnu divriteņu maratonā 2010 (1. posms Cēsis - Valmiera)	SEB Cēsis - Valmiera	304
	Pieejams tīmeklī: https://picasaweb.google.com/maris.ulnicans/SEBCesisValmiera?authkey=Gv1sRgCKfuntG9revpjwE		
Riteņbraukšana	Mežaparks Rullē 2011	Rezultāti un Bildes no Mežaparks Rullē 2011	112
	Pieejams tīmeklī: http://freeskate.lv/2011/05/rezultati-un-bildes-no-mezaparks-rulle-2011/		
Riteņbraukšana	SEB Latvijas čempionāts kalnu divriteņu maratonā 2010 (1. posms Cēsis - Valmiera)	SEB 1.posms Cēsis - Valmiera 2011	535
	Pieejams tīmeklī: https://picasaweb.google.com/105309383157698860739/SEB1PosmsCesisValmiera2011#		
Riteņbraukšana	SEB Latvijas čempionāts kalnu divriteņu maratonā 2010 (1. posms Cēsis - Valmiera)	Ceļā no Cēsīm uz Valmieru (SEB)	15
	Pieejams tīmeklī: http://www.fotoblog.lv/rep/24257/?cid=32589		
Riteņbraukšana	SEB Latvijas čempionāts kalnu	SEB maratons 1posms	155

	divriteņu maratonā 2010 (1. posms Cēsis - Valmiera)	Cēsis-Valmiera	
	Pieejams tīmeklī: https://picasaweb.google.com/gunarsslaide/SEBMaratons1posmsCesisValmiera#		
Riteņbraukšana	SEB Latvijas čempionāts kalnu divriteņu maratonā 2010 (1. posms Cēsis - Valmiera)	2011_05_01_cesis_valmiera.zip	213
	Pieejams tīmeklī: http://files.inbox.lv/ticket/c2dd0c7a729246a7eda384598b8c2cb929e80fb3/2011_05_01_cesis_valmiera.zip		
Riteņbraukšana	SEB Latvijas čempionāts kalnu divriteņu maratonā 2010 (1. posms Cēsis - Valmiera)	Seb MTB Cēsis - Valmiera 2011	116
	Pieejams tīmeklī: http://cid-ae3a1da454e920fa.photos.live.com/browse.aspx/Seb%20MTB%20C%C4%93sis%20-%20Valmiera%202011		
Skrituļslidu maratons	Mazsalacas Novada Kauss 2. posms	Mazsalacas Novada Kauss 2. posms	127
	Pieejams tīmeklī: https://picasaweb.google.com/107718285552551599806/MazsalacasNovadaKauss2Posms?authkey=Gv1sRgCOfPxI7a2rKQNQ		
Riteņbraukšana	Trek Philips MTB Maratons 1. posms	Trek Philips MTB Maratons - Pļaviņas 2011	329
	Pieejams tīmeklī: https://picasaweb.google.com/116725477694538052597/TrekPhilipsMTBMaratonsPlavinas2011?authkey=Gv1sRgCLSO3Ybl6-LPjwE		

2. Pielikums. Fotogrāfiju klasifikācijas grupu apraksts

1) Liela izmēra fotogrāfijas bez skaidri redzamiem trokšņiem. Fotogrāfijās parādās viens sporta notikuma dalībnieks un ir skaidri redzams dalībnieka numurs. Dalībnieka numurs atrodas fotogrāfijas centrā. Numura izmērs no attēla izmēra ir robežās 8% - 20% platumā un 5% - 10 % augstumā. Numura zīmei nav defektu. Numura plāksne nav aizsegta, izpludināta vai nobīdīta. Numura plāksnes nobīde nepārsniedz 5 leņķi. Šādas fotogrāfijas var saukt par portreta fotogrāfijām. Tās ir labākie atpazīšanas paraugi.
2) Liela vai vidēja izmēra fotogrāfijas bez skaidri redzamiem trokšņiem. Fotogrāfijās parādās viens vai divi sporta notikuma dalībnieki un ir skaidri redzami dalībnieku numuri. Dalībnieku numuri nepārklājas un atrodas fotogrāfijas centrā. Numuru izmērs no attēla izmēra ir robežās 8% - 20% platumā un 5% - 10 % augstumā. Numuru zīmei nav defektu. Numuru plāksne nav aizsegta, izpludināta vai nobīdīta. Numuru plāksnes nobīde nepārsniedz 5 leņķi.
3) Liela vai vidēja izmēra fotogrāfijas bez skaidri redzamiem trokšņiem. Fotogrāfijās parādās trīs vai vairāki sporta notikuma dalībnieki un ir skaidri redzami dalībnieku numuri. Dalībnieku numuri nepārklājas. Numuru izmēri dalībniekiem var atšķirties, bet visi ir skaidri apskatāmi, nav izpludināti vai kādā citā veidā pārklāti.
4) Liela vai vidēja izmēra fotogrāfijas bez skaidri redzamiem trokšņiem. Fotogrāfijās parādās vairāki sporta notikuma dalībnieki un ir redzami dalībnieku numuri. Daži dalībnieku numuri pārklājas, ir pārāk stipri izpludināti vai nav citā veidā atpazīstami. Var saturēt loģiskos vai nebūtiskus vizuālos trokšņus.
5) Vidēja vai maza izmēra fotogrāfijas bez skaidri redzamiem trokšņiem. Fotogrāfijās vairāki sporta notikuma dalībnieki un ir redzami dalībnieku numuri. Dalībnieku numuri nepārklājas. Numuru izmēri dalībniekiem var atšķirties, bet visi ir skaidri apskatāmi, nav izpludināti vai kādā citā veidā pārklāti.

6) Vidēja vai maza izmēra fotogrāfijas bez skaidri redzamiem trokšņiem. Fotogrāfijās vairāki sporta notikuma dalībnieki un ir redzami dalībnieku numuri. Daži dalībnieku numuri pārklājas, ir pārāk stipri izpludināti vai citā veidā nav atpazīstami. Var saturēt

loģiskos vai nebūtiskus grafiskos trokšņus.

7) Vidēja vai maza izmēra fotogrāfijas ar grafiskajiem trokšņiem. Fotogrāfijās ir vairāki sporta notikuma dalībnieki un ir redzami daži dalībnieku numuri. Vairākums dalībniekiem numuri pārklājas vai ir pārāk stipri izpludināti, vai citā veidā nav atpazīstami. Var saturēt loģiskos vai grafiskos trokšņus.

Jebkura izmēra fotogrāfijas. Nesatur dalībnieku numurus. Var saturēt loģiskos trokšņus vai citus atpazīšanu mulsinošus trokšņus.

3. Pielikums. Lattelekom BPO, SIA, epasta atbilde.

Daks Klave
1188 biznesa attīstības vadītājs,
Lattelekom BPO, SIA,
atbilde, par Nordea Maratona fotogrāfiju anotēšanu

Automatizēta numuru zīmju atpazīšanas programmatūras eksistē,
tomēr to pilnvērtīga ieviešana un pielāgošana noteiktam mērķim izmaksā relatīvi dārgi.
mūsu gadījumā - lai reizi gadā veiktu ~10.000 maratona dalībnieku foto apstrādi,
automatizētas sistēmas ieviešana neatmaksājas.

Bija alternatīvas opcijas:
a) nosūtīt foto failus apstrādāt kādam kantorim, kam ir specializēta automatizēta sistēma
b) apstrādāt bildes manuāli – pie katra faila pierakstot TAGS, kāds numurs ir redzams bildē

kā lētāko variantu izvēlējāmies manuālu apstrādi –
~10.000 fotogrāfiju apstrādi 2 dienu laikā veica 10 studenti Latvijā.
/Ja būtu lielāks skaits foto failu, tad droši vien vēl lētāk būtu nosūtīt apstrādei uz Indiju vai Filipīnām./

Te vēl daži linki par pieejamiem (maksas un bezmaksas) programmatūras risinājumiem,
kas tiek parasti izmantoti Automašīnu numuru zīmju automātiskai atpazīšanai,
kas ir pielāgojami arī sporta dalībnieku numuru atpazīšanai
(tomēr līdz šim neesmu atradis programmu kas vienā bildē atrod un piefiksē vienlaicīgi vairākus/visus redzamos numurus, nevis tikai vienu)
http://www.warelogic.com/
http://www.dtksoft.com/dtkanpr.php
http://javaanpr.sourceforge.net/
http://www.download32.com/anpr-i8299.html
http://en.wikipedia.org/wiki/Automatic_number_plate_recognition

4. Pielikums. Implementācija valodā Java.

```
 1: package lv.naut.number;
 2:
 3:
 4: import java.awt.image.BufferedImage;
 5: import java.awt.*;
 6: import java.util.Vector;
 7: import java.io.File;
 8: import java.net.URL;
 9:
10: import javax.imageio.ImageIO;
11:
12:
13: public class HoughTransform extends Thread {
14:
15:     public static void main(String[] args) throws Exception {
16:         String filename = "filename";
17:
18:         // load the file using Java's imageIO library
19:         BufferedImage image = javax.imageio.ImageIO.read(new
File(filename));
20:
21:         // create a hough transform object with the right dimensions
22:         HoughTransform h = new HoughTransform(image.getWidth(), im-
age.getHeight());
23:
24:         // add the points from the image (or call the addPoint method
separately if your points are not in an image
25:         h.addPoints(image);
26:
27:         // get the lines out
28:         Vector<HoughLine> lines = h.getLines(50);
29:
30:         // draw the lines back onto the image
31:         for (int j = 0; j < lines.size(); j++) {
32:             HoughLine line = lines.elementAt(j);
33:             line.draw(image, Color.RED.getRGB());
34:         }
35:         File outputfile = new File("filename");
36:         ImageIO.write(image, "png", outputfile);
37:     }
```

```
38:
39:     // The size of the neighbourhood in which to search for other local
maxima
40:     final int neighbourhoodSize = 4;
41:
42:     // How many discrete values of theta shall we check?
43:     final int maxTheta = 180;
44:
45:     // Using maxTheta, work out the step
46:     final double thetaStep = Math.PI / maxTheta;
47:
48:     // the width and height of the image
49:     protected int width, height;
50:
51:     // the hough array
52:     protected int[][] houghArray;
53:
54:     // the coordinates of the centre of the image
55:     protected float centerX, centerY;
56:
57:     // the height of the hough array
58:     protected int houghHeight;
59:
60:     // double the hough height (allows for negative numbers)
61:     protected int doubleHeight;
62:
63:     // the number of points that have been added
64:     protected int numPoints;
65:
66:     // cache of values of sin and cos for different theta values. Has a
significant performance improvement.
67:     private double[] sinCache;
68:     private double[] cosCache;
69:
70:
71:     public HoughTransform(int width, int height) {
72:
73:         this.width = width;
74:         this.height = height;
75:
76:         initialise();
77:
78:     }
79:
80:
81:     public void initialise() {
82:
83:         // Calculate the maximum height the hough array needs to have
```

```
 84:         houghHeight = (int) (Math.sqrt(2) * Math.max(height, width))
/ 2;
 85:
 86:         // Double the height of the hough array to cope with negative r
values
 87:         doubleHeight = 2 * houghHeight;
 88:
 89:         // Create the hough array
 90:         houghArray = new int[maxTheta][doubleHeight];
 91:
 92:         // Find edge points and vote in array
 93:         centerX = width / 2;
 94:         centerY = height / 2;
 95:
 96:         // Count how many points there are
 97:         numPoints = 0;
 98:
 99:         // cache the values of sin and cos for faster processing
100:         sinCache = new double[maxTheta];
101:         cosCache = sinCache.clone();
102:         for (int t = 0; t < maxTheta; t++) {
103:             double realTheta = t * thetaStep;
104:             sinCache[t] = Math.sin(realTheta);
105:             cosCache[t] = Math.cos(realTheta);
106:         }
107:     }
108:
109:     /**
110:      * Adds points from an image. The image is assumed to be greyscale
black and white, so all pixels that are
111:      * not black are counted as edges. The image should have the same di-
mensions as the one passed to the constructor.
112:      */
113:     public void addPoints(BufferedImage image) {
114:
115:         // Now find edge points and update the hough array
116:         for (int x = 0; x < image.getWidth(); x++) {
117:             for (int y = 0; y < image.getHeight(); y++) {
118:                 // Find non-black pixels
119:                 if ((image.getRGB(x, y) & 0x000000ff) != 0) {
120:                     addPoint(x, y);
121:                 }
122:             }
123:         }
124:     }
125:
126:     /**
```

```
127:      * Adds a single point to the hough transform. You can use this me-
thod directly
128:      * if your data isn't represented as a buffered image.
129:      */
130:     public void addPoint(int x, int y) {
131:
132:         // Go through each value of theta
133:         for (int t = 0; t < maxTheta; t++) {
134:
135:             //Work out the r values for each theta step
136:             int r = (int) (((x - centerX) * cosCache[t]) + ((y -
centerY) * sinCache[t]));
137:
138:             // this copes with negative values of r
139:             r += houghHeight;
140:
141:             if (r < 0 || r >= doubleHeight) continue;
142:
143:             // Increment the hough array
144:             houghArray[t][r]++;
145:
146:         }
147:
148:         numPoints++;
149:     }
150:
151:
152:     public Vector<HoughLine> getLines(int threshold) {
153:
154:         // Initialise the vector of lines that we'll return
155:         Vector<HoughLine> lines = new Vector<HoughLine>(20);
156:
157:         // Only proceed if the hough array is not empty
158:         if (numPoints == 0) return lines;
159:
160:         // Search for local peaks above threshold to draw
161:         for (int t = 0; t < maxTheta; t++) {
162:             loop:
163:             for (int r = neighbourhoodSize; r < doubleHeight - neigh-
bourhoodSize; r++) {
164:
165:                 // Only consider points above threshold
166:                 if (houghArray[t][r] > threshold) {
167:
168:                     int peak = houghArray[t][r];
169:
170:                     // Check that this peak is indeed the local maxima
```

```
171:                     for (int dx = -neighbourhoodSize; dx <= neighbour-
hoodSize; dx++) {
172:                         for (int dy = -neighbourhoodSize; dy <= neigh-
bourhoodSize; dy++) {
173:                             int dt = t + dx;
174:                             int dr = r + dy;
175:                             if (dt < 0) dt = dt + maxTheta;
176:                             else if (dt >= maxTheta) dt = dt - maxThe-
ta;
177:                             if (houghArray[dt][dr] > peak) {
178:                                 // found a bigger point nearby, skip
179:                                 continue loop;
180:                             }
181:                         }
182:                     }
183:
184:                     // calculate the true value of theta
185:                     double theta = t * thetaStep;
186:
187:                     // add the line to the vector
188:                     lines.add(new HoughLine(theta, r));
189:
190:                 }
191:             }
192:         }
193:
194:         return lines;
195:     }
196:
197:     /**
198:      * Gets the highest value in the hough array
199:      */
200:     public int getHighestValue() {
201:         int max = 0;
202:         for (int t = 0; t < maxTheta; t++) {
203:             for (int r = 0; r < doubleHeight; r++) {
204:                 if (houghArray[t][r] > max) {
205:                     max = houghArray[t][r];
206:                 }
207:             }
208:         }
209:         return max;
210:     }
211:
212:     /**
```

```
213:      * Gets the hough array as an image, in case you want to have a look
at it.
214:      */
215:     public BufferedImage getHoughArrayImage() {
216:         int max = getHighestValue();
217:         BufferedImage image = new BufferedImage(maxTheta, doubleHeight,
BufferedImage.TYPE_INT_ARGB);
218:         for (int t = 0; t < maxTheta; t++) {
219:             for (int r = 0; r < doubleHeight; r++) {
220:                 double value = 255 * ((double) houghArray[t][r]) /
max;
221:                 int v = 2
```

```
 1: package lv.naut.number;
 2:
 3: import java.awt.image.BufferedImage;
 4: import java.awt.image.ColorModel;
 5: import java.awt.image.DataBuffer;
 6: import java.awt.image.IndexColorModel;
 7: import java.awt.image.Raster;
 8: import java.awt.image.WritableRaster;
 9: import java.io.File;
10: import java.io.IOException;
11: import java.util.Arrays;
12:
13: import javax.imageio.ImageIO;
14:
15:
16:
17: public class CannyEdgeDetector {
18:
19:    // statics
20:
21:    private final static float GAUSSIAN_CUT_OFF = 0.005f;
22:    private final static float MAGNITUDE_SCALE = 100F;
23:    private final static float MAGNITUDE_LIMIT = 1000F;
24:    private final static int MAGNITUDE_MAX = (int) (MAGNITUDE_SCALE *
MAGNITUDE_LIMIT);
25:
26:    // fields
27:
28:    private int height;
29:    private int width;
30:    private int picsize;
31:    private int[] data;
32:    private int[] magnitude;
33:    private BufferedImage sourceImage;
34:    private BufferedImage edgesImage;
35:
36:    private float gaussianKernelRadius;
37:    private float lowThreshold;
38:    private float highThreshold;
39:    private int gaussianKernelWidth;
40:
41:
42:    private float[] xConv;
43:    private float[] yConv;
44:    private float[] xGradient;
```

```
45:     private float[] yGradient;
46:
47:     // constructors
48:
49:     /**
50:      * Constructs a new detector with default parameters.
51:      */
52:
53:     public CannyEdgeDetector() {
54:             lowThreshold = 2.5f;
55:             highThreshold = 7.5f;
56:             gaussianKernelRadius = 2f;
57:             gaussianKernelWidth = 16;
58:
59:
60:
61:     }
62:
63:
64:
65:
66:     public BufferedImage getSourceImage() {
67:             return sourceImage;
68:     }
69:
70:
71:
72:     public void setSourceImage(BufferedImage image) {
73:             sourceImage = image;
74:     }
75:
76:
77:
78:     public BufferedImage getEdgesImage() {
79:             return edgesImage;
80:     }
81:
82:
83:
84:     public void setEdgesImage(BufferedImage edgesImage) {
85:             this.edgesImage = edgesImage;
86:     }
87:
88:
89:
90:     public float getLowThreshold() {
91:             return lowThreshold;
92:     }
```

```
93:
94:
95:
96:     public void setLowThreshold(float threshold) {
97:             if (threshold < 0)
98:                     throw new IllegalArgumentException();
99:             lowThreshold = threshold;
100:    }
101:
102:    /**
103:     * The high threshold for hysteresis. The default value is 7.5.
104:     *
105:     * @return the high hysteresis threshold
106:     */
107:
108:    public float getHighThreshold() {
109:            return highThreshold;
110:    }
111:
112:
113:
114:    public void setHighThreshold(float threshold) {
115:            if (threshold < 0)
116:                    throw new IllegalArgumentException();
117:            highThreshold = threshold;
118:    }
119:
120:
121:
122:    public int getGaussianKernelWidth() {
123:            return gaussianKernelWidth;
124:    }
125:
126:
127:    public void setGaussianKernelWidth(int gaussianKernelWidth) {
128:            if (gaussianKernelWidth < 2)
129:                    throw new IllegalArgumentException();
130:            this.gaussianKernelWidth = gaussianKernelWidth;
131:    }
132:
133:
134:
135:    public float getGaussianKernelRadius() {
136:            return gaussianKernelRadius;
137:    }
138:
139:
```

```
140:    public void setGaussianKernelRadius(float gaussianKernelRadius) {
141:            if (gaussianKernelRadius < 0.1f)
142:                    throw new IllegalArgumentException();
143:            this.gaussianKernelRadius = gaussianKernelRadius;
144:    }
145:
146:
147:
148:    // methods
149:
150:    public void process() {
151:            width = sourceImage.getWidth();
152:            height = sourceImage.getHeight();
153:            picsize = width * height;
154:            initArrays();
155:            readLuminance();
156:
157:            computeGradients(gaussianKernelRadius, gaussianKernelWidth);
158:            int low = Math.round(lowThreshold * MAGNITUDE_SCALE);
159:            int high = Math.round(highThreshold * MAGNITUDE_SCALE);
160:            performHysteresis(low, high);
161:            thresholdEdges();
162:            writeEdges(data);
163:    }
164:
165:    // private utility methods
166:
167:    private void initArrays() {
168:            if (data == null || picsize != data.length) {
169:                    data = new int[picsize];
170:                    magnitude = new int[picsize];
171:
172:                    xConv = new float[picsize];
173:                    yConv = new float[picsize];
174:                    xGradient = new float[picsize];
175:                    yGradient = new float[picsize];
176:            }
177:    }
178:
179:
180:    private void computeGradients(float kernelRadius, int kernelWidth) {
181:
182:            // generate the gaussian convolution masks
183:            float kernel[] = new float[kernelWidth];
184:            float diffKernel[] = new float[kernelWidth];
```

```
185:            int kwidth;
186:            for (kwidth = 0; kwidth < kernelWidth; kwidth++) {
187:                    float g1 = gaussian(kwidth, kernelRadius);
188:                    if (g1 <= GAUSSIAN_CUT_OFF && kwidth >= 2)
189:                            break;
190:                    float g2 = gaussian(kwidth - 0.5f, kernelRadius);
191:                    float g3 = gaussian(kwidth + 0.5f, kernelRadius);
192:                    kernel[kwidth] = (g1 + g2 + g3) / 3f
193:                                    / (2f * (float) Math.PI * kernelRa-
dius * kernelRadius);
194:                    diffKernel[kwidth] = g3 - g2;
195:            }
196:
197:            int initX = kwidth - 1;
198:            int maxX = width - (kwidth - 1);
199:            int initY = width * (kwidth - 1);
200:            int maxY = width * (height - (kwidth - 1));
201:
202:            // perform convolution in x and y directions
203:            for (int x = initX; x < maxX; x++) {
204:                    for (int y = initY; y < maxY; y += width) {
205:                            int index = x + y;
206:                            float sumX = data[index] * kernel[0];
207:                            float sumY = sumX;
208:                            int xOffset = 1;
209:                            int yOffset = width;
210:                            for (; xOffset < kwidth;) {
211:                                    sumY += kernel[xOffset]
212:                                                    * (data[index - yOff-
set] + data[index + yOffset]);
213:                                    sumX += kernel[xOffset]
214:                                                    * (data[index - xOff-
set] + data[index + xOffset]);
215:                                    yOffset += width;
216:                                    xOffset++;
217:                            }
218:
219:                            yConv[index] = sumY;
220:                            xConv[index] = sumX;
221:                    }
222:
223:            }
224:
```

```
225:            for (int x = initX; x < maxX; x++) {
226:                    for (int y = initY; y < maxY; y += width) {
227:                            float sum = 0f;
228:                            int index = x + y;
229:                            for (int i = 1; i < kwidth; i++)
230:                                    sum += diffKernel[i]
231:                                                    * (yConv[index - i] -
yConv[index + i]);
232:
233:                            xGradient[index] = sum;
234:                    }
235:
236:            }
237:
238:            for (int x = kwidth; x < width - kwidth; x++) {
239:                    for (int y = initY; y < maxY; y += width) {
240:                            float sum = 0.0f;
241:                            int index = x + y;
242:                            int yOffset = width;
243:                            for (int i = 1; i < kwidth; i++) {
244:                                    sum += diffKernel[i]
245:                                                    * (xConv[index - yOff-
set] - xConv[index + yOffset]);
246:                                    yOffset += width;
247:                            }
248:
249:                            yGradient[index] = sum;
250:
```

Bakalaura darbs „Sporta sacensību fotogrāfiju anotēšana” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Dmitrijs Surženko 30.05.2011.

Rekomendēju darbu aizstāvēšanai

Vadītājs: Mg.dat. Rūdolfs Opmanis 30.05.2011.

Recenzents: profsors Dr.dat. Jānis Bičevskis

Darbs iesniegts Datorikas fakultatē 30.05.2011.

Metodiķe: Ārija Sproģe

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

09.06.2011. prot. Nr.____, vērtējums _____

Komisijas sekretāre: